Floyd (WA)rshall

Contents

1 Binary Search and Ternary Search

1.1 Aplications

```
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pii pair<int, pi>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 1001

// 1 - ts para double
long double ts()
{
  long double l = 0, r = DBL_MAX;
  for (int i = 0; i < 2000; i++)
  {
    long double l1 = (l * 2 + r) / 3.0;
    long double l2 = (l + 2 * r) / 3.0;
    if (possible(l1))
      r = l2;
    else
      l = l1;
  }
  return l;
}
// 2- bb para double
long double bb()
{
  long double i = 0, f = DBL_MAX, m;
  while (f - i > 0.000000001)
  {
    m = (i + f) / 2.0;
    if (possible(m))
      f = m;
    else
      i = m;
  }
  return i;
}
// 3 - bb pra int
lli bb()
{
  lli i = 0, f = INT_MAX, m;
  while (i < f)
  {
    m = (i + f) / 2;
    if (possible(m))
      f = m;
    else
      i = m + 1;
  }
  return i;
}
// 4 - ts pra int (valor minimo da funcao f(x)), sendo x um inteiro
int l = 1, r = INT_MAX;
while (r - l > 15)
{
  int l1 = (l * 2 + r) / 3;
  int l2 = (l + 2 * r) / 3;
  (calc(l1) < calc(l2)) ? r = l2 : l = l1;
}
for (int i = l; i <= r; i++)
// vejo qual a melhor opcao de l ate r em o(n)


// busca ternaria para int, usando busca binaria:
int l = 0, r = 1e9;
while (l < r)
{
  int mid = (l + r) >> 1;
  (calc(mid) < calc(mid + 1)) ? r = mid : l = mid + 1;
}
return calc(l);
```

1.2 BS

```
#include <bits/stdc++.h>
using namespace std ;

#define lli long long int
#define pb push_back

vector <int> v;
int binarysearch (int n , int x)
{
    int i = 0 ;
    int f = n - 1 ;
    int m ;

    while(i <= f)
    {
        m = (i + f) / 2 ;

        if(v[m] == x) return m + 1 ;
        if(v[m] < x)  i = m + 1 ;
        if(v[m] > x)  f = m - 1 ;
    }

    return 0 ;
}
int main ()
{
    int n , aux , m ;

    cin >> n ;

    for (int i = 0 ; i < n ; i++)
    {
        cin >> aux ;
        v.pb(aux);
    }
```

```
    sort(v.begin() , v.end());

    cin >> m ;
    cout << binarysearch(n , m) << endl ;

    return 0 ;
}
```

1.3 LowerBound

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 1000001
#define mod 1000000007
// first element >= x
vector<int> k(MAXN);
int lower(int l, int r, int x) // first element >= x
{
  while (l < r)
  {
    int mid = (l + r) >> 1;
    (x <= k[mid]) ? r = mid : l = mid + 1;
  }
  return k[l];
}
```

1.4 parallel binary search

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300006
#define mod 1000000007

int n;
int b[MAXN];

void reset()
{
  for (int i = 0; i < MAXN; i++)
    b[i] = 0;
}
int sum(int r)
{
  int ret = 0;
  for (; r >= 0; r = (r & (r + 1)) - 1)
    ret += b[r];
  return ret;
}
void add(int idx, int delta)
{
  for (; idx < MAXN; idx = idx | (idx + 1))
    b[idx] += delta;
}
void update(int l, int r, int x)
{
  add(l, x);
  add(r + 1, -x);
}
void upd(int l, int r, int x)
{
  if (l <= r)
  {
    update(l, r, x);
    return;
  }
  update(l, MAXN - 2, x);
  update(0, r, x);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  vector<vector<int>> adj(n);
  for (int i = 0; i < m; i++)
  {
    int x;
    cin >> x;
    x--;
    adj[x].pb(i);
  }
  vector<int> need(n);
  for (int i = 0; i < n; i++)
  {
    cin >> need[i];
  }
  int q;
  cin >> q;
  vector<pii> qry(q);
  for (int i = 0; i < q; i++)
  {
    cin >> qry[i].sec.fir >> qry[i].sec.sec >> qry[i].fir;
    qry[i].sec.fir--, qry[i].sec.sec--;
  }
  vector<int> l(n);
  vector<int> r(n);
  vector<vector<int>> on(q);
  for (int i = 0; i < n; i++)
  {
    l[i] = 0;
    r[i] = q;
  }
  while (1)
  {
    bool ok = 1;
    for (int i = 0; i < n; i++)
    {
      if (l[i] < r[i])
      {
        ok = 0;
        int mid = (l[i] + r[i]) >> 1;
        on[mid].pb(i);
      }
    }
    if (ok)
      break;
    reset();
    for (int mid = 0; mid < q; mid++)
    {
```

```
      upd(qry[mid].sec.fir, qry[mid].sec.sec, qry[mid].fir);
      for (auto const &j : on[mid])
      {
        int val = 0;
        for (auto const &k : adj[j])
        {
          val += sum(k);
          if (val >= need[j])
            break;
        }
        (val >= need[j]) ? r[j] = mid : l[j] = mid + 1;
      }
      on[mid].clear();
    }
  }
  for (int i = 0; i < n; i++)
  {
    if (l[i] >= q)
      cout << "NIE\n";
    else
      cout << l[i] + 1 << endl;
  }
  return 0;
}
// busca binaria paralela
// https://www.spoj.com/problems/METEORS/

// tem n member states e m sectors
// cada sector ta associado a uma member state
// cada query incrementa o range [l[i], r[i]] de sectors por a[i]
// seja q[i] a soma de todos os v[j], sendo j um sector associado ao member
    state i
// qual o primeiro momento no qual q[i] >= min_qt[i]
// para todos os i

// a sagacidade vai ser fzr uma busca binaria pra cada resposta
// primeiro vc faz todas a primeira iteracao de cada busca binaria
// depois cada segunda iteracao de cada bb
// e assim vai

// ai a bb eh so tipo
// a soma apos a query mid ja deu bom pra aquele member state?
```

1.5 recursive parallel binary search

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define pb push_back
#define fir first
#define sec second
#define MAXN 300006
#define mod 1000000007

int n, ptr;
int b[MAXN];
int ll[MAXN];
int rr[MAXN];
int qryl[MAXN];
int qryr[MAXN];
int qryg[MAXN];
int need[MAXN];
vector<int> adj[MAXN];

int sum(int r)
{
  int ret = 0;
  for (; r >= 0; r = (r & (r + 1)) - 1)
    ret += b[r];
  return ret;
}
void add(int idx, int delta)
{
  for (; idx < MAXN; idx = idx | (idx + 1))
    b[idx] += delta;
}
void update(int l, int r, int x)
{
  add(l, x);
  add(r + 1, -x);
}
void upd(int l, int r, int x)
{
  if (l <= r)
  {
    update(l, r, x);
    return;
  }
  update(l, MAXN - 2, x);
  update(0, r, x);
}
void solve(vector<int> &v, int l, int r)
{
  if (!(l < r) || !v.size())
  {
    return;
  }
  int mid = (l + r) / 2;
  while (ptr <= mid)
  {
    upd(qryl[ptr], qryr[ptr], qryg[ptr]);
    ptr++;
  }
  while ((ptr - 1) > mid)
  {
    ptr--;
    upd(qryl[ptr], qryr[ptr], -qryg[ptr]);
  }
  vector<int> to_left, to_right;
  for (auto const &j : v)
  {
    int val = 0;
    for (auto const &k : adj[j])
    {
      val += sum(k);
      if (val >= need[j])
        break;
    }
    if (val >= need[j])
    {
      to_right.pb(j);
      rr[j] = mid;
    }
    else
    {
      to_left.pb(j);
      ll[j] = mid + 1;
    }
  }
  v.clear();
  solve(to_left, mid + 1, r);
  solve(to_right, l, mid);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cout.tie(NULL);
  int n, m;
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int x;
    cin >> x;
    x--;
    adj[x].pb(i);
  }
  for (int i = 0; i < n; i++)
  {
    cin >> need[i];
  }
  int q;
```

```
  cin >> q;
  for (int i = 0; i < q; i++)
  {
    cin >> qryl[i] >> qryr[i] >> qryg[i];
    qryl[i]--, qryr[i]--;
  }
  vector<int> vec;
  for (int i = 0; i < n; i++)
  {
    ll[i] = 0;
    rr[i] = q;
    vec.pb(i);
  }
  solve(vec, 0, q);
  for (int i = 0; i < n; i++)
  {
    if (ll[i] >= q)
      cout << "NIE\n";
    else
      cout << ll[i] + 1 << endl;
  }
  return 0;
}
// busca binaria paralela
// https://www.spoj.com/problems/METEORS/
// so que a recursiva
```

1.6 STL

```
// lower - primeiro maior ou igual a x
// upper - ultimo menor ou igual a x

#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back

vector <int> v ;
int main()
{
    int n , aux ;
    cin >> n ;

    for (int i = 0 ; i < n ; i++)
    {
        cin >> aux ;
        v.pb(aux);
    }

    sort(v.begin() , v.end());

    int q ;
    cin >> q ;

    while (q--)
    {
        cin >> aux ;
        vector <int> :: iterator low = lower_bound (v.begin() , v.end() , aux) ;
        vector <int> :: iterator up = upper_bound (v.begin() , v.end() , aux) ;

        cout << (low - v.begin()) << " " << (up - v.begin()) - 1 << endl ;
    }

    return 0 ;
}
```

1.7 TS

```
// achar valor maximo da funcao
for (int i = 0; i < 400; i++)
{
  double m1 = l + (r - l) / 3;
  double m2 = r - (r - l) / 3;
  double f1 = f(m1);
  double f2 = f(m2);
  if (f1 < f2)
    l = m1;
  else
    r = m2;
}
```

1.8 UpperBound

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 1000001
#define mod 1000000007
// last element <= x
vector<int> k(MAXN);
int upper(int l, int r, int x)
{
  while (l < r)
  {
    int mid = (l + r + 1) >> 1;
    (k[mid] <= x) ? l = mid : r = mid - 1;
  }
  return k[l];
}
```

2 Dynamic programming and common problems

2.1 aliens trick

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

int n, k, l;
string s;
```

```
pi solve(vector<int> &v, int lambda)
{
  // associar um custo lambda para ser subtraido quando realizamos uma operacao
  // dp[i] - melhor profit que tivemos considerando as i primeiras posicoes
  // cnt[i] - quantas operacoes utilizamos para chegarno valor de dp[i]
  vector<int> dp(n + 1);
  vector<int> cnt(n + 1);
  dp[0] = 0;
  cnt[0] = 0;
  for (int i = 1; i <= n; i++)
  {
    dp[i] = dp[i - 1];
    cnt[i] = cnt[i - 1];
    int id = i - 1;
    dp[i] += v[id];
    int lo = max(0ll, id - l + 1);
    int s = dp[lo] + (id - lo + 1) - lambda;
    if (s > dp[i])
    {
      dp[i] = s;
      cnt[i] = cnt[lo] + 1;
    }
  }
  return {dp[n], cnt[n]};
}
int aliens_trick(vector<int> &v)
{
  int l = 0, r = n;
  while (l < r)
  {
    int mid = (l + r) >> 1;
    pi ans = solve(v, mid);
    (ans.sec > k) ? l = mid + 1 : r = mid;
  }
  pi ans = solve(v, l);
  return ans.fir + (l * k);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> k >> l >> s;
  vector<int> a(n);
  vector<int> b(n);
  for (int i = 0; i < n; i++)
  {
    a[i] = 1, b[i] = 0;
    if (s[i] >= 'A' && s[i] <= 'Z')
    {
      a[i] ^= 1;
      b[i] ^= 1;
    }
  }
  cout << n - max(aliens_trick(a), aliens_trick(b)) << endl;
  return 0;
}
// https://codeforces.com/contest/1279/problem/F
```

2.2 bitwise digit dp

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 65
#define mod 1000000007

int a, b;
// current bit / i is bigger than l / i is lower than r / j is bigger than l / j
     is lower than r
int dp[MAXN][2][2][2][2];

int solve(int i, int j, int k, int l, int m)
{
  if (i < 0)
    return (j && k && l && m) ? 1 : 0;
  if (dp[i][j][k][l][m] != -1)
    return dp[i][j][k][l][m];
  int ret = 0;
  int ll = a & (1LL << i);
  int rr = b & (1LL << i);
  if ((j || !ll) && (l || !ll))
    ret += solve(i - 1, j, (rr) ? 1 : k, l, (rr) ? 1 : m);
  if ((k || rr) && (l || !ll))
    ret += solve(i - 1, (!ll) ? 1 : j, k, l, (rr) ? 1 : m);
  if ((m || rr) && (j || !ll))
    ret += solve(i - 1, j, (rr) ? 1 : k, (!ll) ? 1 : l, m);
  return dp[i][j][k][l][m] = ret;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    cin >> a >> b;
    a--, b++;
    memset(dp, -1, sizeof(dp));
    if (a == -1)
      cout << solve(60, 1, 0, 1, 0) << endl;
    else
      cout << solve(60, 0, 0, 0, 0) << endl;
  }
  return 0;
}
// https://codeforces.com/contest/1245/problem/F
// https://codeforces.com/blog/entry/88064
// count the number of pairs (i, j) which (i & j) == 0
```

2.3 broken profile

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pair<int, pi>>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 1001
#define mod 1000000007

int n;
vector<int> validmasks;
int dp[MAXN][1 << 4];
```

```
void init() // preprocess valid masks
{
  for (int mask = 0; mask < (1 << 7); mask++)
  {
    int nxt_mask = 0, prev_mask = 0, valid = true;
    for (int k = 0; k < 7; k++)
    {
      if (mask & (1 << k))
      {
        if (k <= 3)
        {
          int idx = k, idx2 = k;
          if (nxt_mask & (1 << idx) || prev_mask & (1 << idx2))
            valid = false;
          prev_mask = prev_mask | (1 << idx);
          nxt_mask = nxt_mask | (1 << idx2);
        }
        else
        {
          int idx = k - 4, idx2 = idx + 1;
          if (nxt_mask & (1 << idx) || nxt_mask & (1 << idx2))
            valid = false;
          nxt_mask = nxt_mask | (1 << idx);
          nxt_mask = nxt_mask | (1 << idx2);
        }
      }
    }
    if (valid)
      validmasks.pb(mask);
  }
}
int solve(int i, int j)
{
  if (i == n)
    return (j == ((1 << 4) - 1)) ? 1 : 0;
  if (dp[i][j] != -1)
    return dp[i][j];
  int ret = 0;
  for (auto const &mask : validmasks)
  {
    int nxt_mask = 0, prev_mask = j, valid = true;
    for (int k = 0; k < 7; k++)
    {
      if (mask & (1 << k))
      {
        if (k <= 3)
        {
          int idx = k, idx2 = idx;
          if (prev_mask & (1 << idx) || nxt_mask & (1 << idx2))
            valid = false;
          prev_mask = prev_mask | (1 << idx);
          nxt_mask = nxt_mask | (1 << idx2);
        }
        else
        {
          int idx = k - 4, idx2 = idx + 1;
          if (nxt_mask & (1 << idx) || nxt_mask & (1 << idx2))
            valid = false;
          nxt_mask = nxt_mask | (1 << idx);
          nxt_mask = nxt_mask | (1 << idx2);
        }
      }
    }
    if (valid && prev_mask == ((1 << 4) - 1))
      ret += solve(i + 1, nxt_mask);
  }
  return dp[i][j] = ret;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  init();
  for (int i = 1; i <= q; i++)
  {
    cin >> n;
    memset(dp, -1, sizeof(dp));
    cout << i << " " << solve(0, (1 << 4) - 1) << endl;
  }
  return 0;
}
// broken profile dp
// if you can fully fill an area with some figures
// finding number of ways to fully fill an area with some figures
// finding a way to fill an area with minimum number of figures
// ...
// https://www.spoj.com/problems/GNY07H/
// We wish to tile a 4xN grid with rectangles 2x1 (in either orientation)
// dp[i][mask]
// i denotes the current column
// mask denotes the situation of the previous column
// our mission is to fill all of the units of
// the previous column in a state [i][mask]
```

2.4 cht

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000005
#define mod 1000000007

struct line
{
  int m, b, p;
  line(int m, int b) : m(m), b(b) {}
  bool operator<(const line &o) const
  {
    if (m != o.m)
      return m < o.m;
    return b < o.b;
  }
  bool operator<(const int x) const { return p < x; }
  int eval(int x) const { return m * x + b; }
  int inter(const line &o) const
  {
    int x = b - o.b, y = o.m - m;
    return (x / y) - ((x ^ y) < 0 && x % y);
  }
};
struct cht
{
  int ptr;
  vector<line> a;
  cht() { ptr = 0; }
  void add(line l)
  {
    while (1)
    {
      if (a.size() >= 1 && a.back().m == l.m && l.b > a.back().b)
      {
        a.pop_back();
      }
      else if (a.size() >= 1 && a.back().m == l.m && l.b <= a.back().b)
      {
        break;
```

```
      }
      else if (a.size() >= 2 && a.back().inter(l) >= a[a.size() - 2].inter(a.
          back()))
      {
        a.pop_back();
      }
      else
      {
        a.pb(l);
        break;
      }
    }
  }
  int get(int x)
  {
    if (!a.size())
      return -inf;
    while (ptr + 1 < a.size() && a[ptr].eval(x) <= a[ptr + 1].eval(x))
      ptr++;
    return a[ptr].eval(x);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// cht
// queries ordenadas em ordem decrescente
// linhas ordenadas em ordem decrescente
```

2.5 Digitdp

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define fir first
#define sec second
#define MAXN 2001
#define mod 1000000007

int dp[20][20 * 9][2]; // a,b <= 10^18
vector<int> dig;

int solve(int i, int j, int k)
{
  if (i == dig.size())
    return (k) ? dp[i][j][k] = j : dp[i][j][k] = 0;
  if (dp[i][j][k] != -1)
    return dp[i][j][k];
  int sum = 0;
  if (k)
    for (int f = 0; f <= 9; f++)
      sum += solve(i + 1, j + f, k);
  if (!k)
    for (int f = 0; f <= dig[i]; f++)
      sum += solve(i + 1, j + f, (dig[i] != f) ? 1 : 0);
  return dp[i][j][k] = sum;
}
void get_digits(int n)
{
  dig.clear();
  while (n)
  {
    dig.pb(n % 10);
    n = n / 10;
  }
  reverse(dig.begin(), dig.end());
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int a, b;
  cin >> a >> b;
  get_digits(a);
  memset(dp, -1, sizeof(dp));
  int aa = solve(0, 0, 0);
  get_digits(b + 1);
  memset(dp, -1, sizeof(dp));
  int bb = solve(0, 0, 0);
  cout << bb - aa << endl;
  return 0;
}
```

2.6 divideandconquer

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007

int s[8005];
int dp[3005][8005];

int cost(int l, int r)
{
  return (s[r + 1] - s[l]) * (r - l + 1);
}
void compute(int l, int r, int optl, int optr, int i)
{
  if (l > r)
    return;
  int mid = (l + r) >> 1;
  pair<int, int> ans = {1e18, -1}; // dp, k
  for (int q = optl; q <= min(mid, optr); q++)
  {
    if (q > 0)
      ans = min(ans, {dp[i - 1][q - 1] + cost(q, mid), q});
    else
      ans = min(ans, {cost(q, mid), q});
  }
  dp[i][mid] = ans.fir;
  compute(l, mid - 1, optl, ans.sec, i);
  compute(mid + 1, r, ans.sec, optr, i);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, g;
  cin >> n >> g;
  for (int i = 1; i <= n; i++)
  {
    cin >> s[i];
    s[i] += s[i - 1];
  }
  for (int i = 0; i <= g; i++)
  {
    for (int j = 0; j <= n; j++)
      dp[i][j] = 1e18;
```

```
  }
  for (int i = 1; i <= g; i++)
    compute(0, n - 1, 0, n - 1, i);
  cout << dp[g][n - 1] << endl;
  return 0;
}
// https://codeforces.com/gym/103536/problem/A
// https://codeforces.com/contest/321/problem/E

// otimizacao de dp usando divide and conquer
// para dps do tipo:
// dp[i][j] = min(dp[i - 1][k] + c(k, j)), para algum k <= j
// considerando opt(i, j) o menor valor de k que minimiza dp[i][j]
// podemos calcular opt(i, j) usando divide and conquer
// isso diminuiria a complexidade para O(k * n * log(n))
```

2.7 dynamic cht

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pf push_front
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000005
#define mod 1000000007

struct line
{
  mutable int m, b, p;
  bool operator<(const line &o) const
  {
    if (m != o.m)
      return m < o.m;
    return b < o.b;
  }
  bool operator<(const int x) const { return p < x; }
  int eval(int x) const { return m * x + b; }
  int inter(const line &o) const
  {
    int x = b - o.b, y = o.m - m;
    return (x / y) - ((x ^ y) < 0 && x % y);
  }
};
struct cht
{
  int INF = 1e18;
  multiset<line, less<>> l;
  void add(int m, int b)
  {
    auto y = l.insert({m, b, INF});
    auto z = next(y);
    if (z != l.end() && y->m == z->m)
    {
      l.erase(y);
      return;
    }
    if (y != l.begin())
    {
      auto x = prev(y);
      if (x->m == y->m)
        x = l.erase(x);
    }
    while (1)
    {
      if (z == l.end())
      {
        y->p = INF;
        break;
      }
      y->p = y->inter(*z);
      if (y->p < z->p)
        break;
      else
        z = l.erase(z);
    }
    if (y == l.begin())
      return;
    z = y;
    auto x = --y;
    while (1)
    {
      int ninter = x->inter(*z);
      if (ninter <= x->p)
        x->p = ninter;
      else
      {
        l.erase(z);
        break;
      }
      if (x == l.begin())
        break;
      y = x;
      x--;
      if (x->p < y->p)
        break;
      else
        l.erase(y);
    }
  }
  int get(int x)
  {
    if (l.empty())
      return 0;
    return l.lower_bound(x)->eval(x);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// sources:
// https://github.com/pauloamed/Training/blob/master/PD/cht.cpp
// https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/DP/CHT-Dinamico
    .cpp

// cht dinamico
// dado uma coordenada x
// e um conjunto com varias equacoes lineares da forma: y = mx + c
// retorna o maior valor de y entre as equacoes do conjunto

// para o menor valor, multiplicar m e c de cada equacao por -1
// e multiplicar o resultado da query por -1

// problemas iniciais:
// https://atcoder.jp/contests/dp/tasks/dp_z
// https://codeforces.com/contest/1083/problem/E
```

2.8 exchange arguments

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
```

```cpp
template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1001
#define mod 1000000009

const int inf = 1e18;

int n;
vector<pi> v;
int dp[MAXN][MAXN];
bool vis[MAXN][MAXN];

int solve(int i, int j)
{
  if (j == 0)
    return inf;
  if (i == n)
    return -inf;
  if (vis[i][j])
    return dp[i][j];
  int ans = -inf;
  ans = max(ans, solve(i + 1, j));
  int ot = min(v[i].sec, solve(i + 1, j - 1) - v[i].fir);
  ans = max(ans, ot);
  vis[i][j] = 1;
  return dp[i][j] = ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n;
  v.resize(n);
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].fir >> v[i].sec;
    v[i].sec -= v[i].fir;
  }
  auto cmp = [&](pi a, pi b)
  {
    return (a.sec - b.fir) < (b.sec - a.fir);
  };
  sort(v.begin(), v.end(), cmp);
  memset(dp, -1, sizeof(dp));
  int ans = 0;
  for (int i = n; i >= 0; i--)
  {
    if (solve(0, i) >= 0)
    {
      ans = i;
      break;
    }
  }
  cout << ans << endl;
  return 0;
}
// problema:
// existem n caixas, cada uma tem um peso w[i] e uma resistencia r[i]
// voce deve escolher um subset de caixas e empilhar na ordem que vc quiser
// tal que: a soma dos pesos de todas as caixas acima de uma caixa seja menor ou
    igual a resistencia dessa caixa

// dp[i][j] - estou na caixa i e quero escolher mais j caixas para botar na
    pilha
// qual a maior resistencia restante que eu posso obter escolhendo essas j
    caixas

// a grande sacada pra achar a ordenacao otima antes da dp:
// para duas caixas a e b
// quando vai ser stonks botar a antes de b?
// r[a] - w[b] > r[b] - w[a]
// pois a resistencia reestante vai ser maior

// pra demais problemas de exchange argument, essa ideia pode se aplicar
// do tipo, ver o jeito otimo de resolver pro n = 2
// e fazer a ordenacao baseada nisso
```

2.9 expected value

```cpp
//https://atcoder.jp/contests/dp/tasks/dp_j
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define mp make_pair
#define pi pair<int, int>
#define pii pair<pi, int>
#define pci pair<char, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 301
#define mod 1000000007

int n;
vector<int> v;
vector<int> cnt(3);
double dp[MAXN][MAXN][MAXN];

double solve(int i, int j, int k)
{
  if (!i && !j && !k)
    return dp[i][j][k] = 0;
  if (dp[i][j][k] != -1)
    return dp[i][j][k];
  /*
  It is well-known from statistics that for the geometric distribution
  (counting number of trials before a success, where each independent trial is
      probability p)
  the expected value is i / p
  */
  double p = ((double)(i + j + k) / n);
  double ret = 1 / p; // expected number of trials before a success
  if (i)
  {
    double prob = (double)i / (i + j + k); // probabilidade de ser um prato com
        um sushi
    ret += (solve(i - 1, j, k) * prob);
  }
  if (j)
  {
    double prob = (double)j / (i + j + k); // probabilidade de ser um prato com
        dois sushis
    ret += (solve(i + 1, j - 1, k) * prob);
  }
  if (k)
  {
    double prob = (double)k / (i + j + k); // probabilidade de ser um prato com
        tres sushis
    ret += (solve(i, j + 1, k - 1) * prob);
  }
  return dp[i][j][k] = ret;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
```

```
  cin >> n;
  v.resize(n);
  for (int i = 0; i < n; i++)
    cin >> v[i], cnt[v[i] - 1]++;
  for (int i = 0; i < MAXN; i++)
    for (int j = 0; j < MAXN; j++)
      for (int k = 0; k < MAXN; k++)
        dp[i][j][k] = -1;
  cout << setprecision(15) << solve(cnt[0], cnt[1], cnt[2]) << endl;
  return 0;
}
```

2.10 inclusion exclusion

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 1000000007

struct modint
{
  int val;
  modint(int v = 0) { val = v % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};

modint f[MAXN];
modint invfat[MAXN];

void calc()
{
  f[0] = 1;
  invfat[0] = 1;
  for (int i = 1; i < MAXN; i++)
  {
    f[i] = f[i - 1] * i;
    invfat[i] = f[i].inv();
  }
}
modint ncr(int n, int k) // combinacao
{
  modint ans = f[n] * invfat[k];
  ans *= invfat[n - k];
  return ans;
}
modint arr(int n, int k) // arranjo
{
  modint ans = f[n] * invfat[n - k];
  return ans;
}

int h, w, n;
pi v[3005];
modint m1 = 1000000006; // -1
modint dp[3005][3005][2];
bool vis[3005][3005][2];

modint solve(int i, int j, int par)
{
  if (i > n)
  {
    modint ans = ncr((h - v[j].fir) + (w - v[j].sec), h - v[j].fir);
    if (!par)
      return ans;
    return ans * m1;
  }
  if (vis[i][j][par])
  {
    return dp[i][j][par];
  }
  modint ans = solve(i + 1, j, par);
  if (v[i].sec >= v[j].sec)
  {
    modint nxt = solve(i + 1, i, (par + 1) % 2);
    modint curr = ncr((v[i].fir - v[j].fir) + (v[i].sec - v[j].sec), v[i].fir -
         v[j].fir);
    curr *= nxt;
    ans = ans + curr;
  }
  vis[i][j][par] = 1;
  dp[i][j][par] = ans;
  return ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  calc();
  cin >> h >> w >> n;
  // tenho que fazer essa inclusao exclusao numa dp
  // for (int i = 0; i < n; i++)
  //   cin >> v[i].fir >> v[i].sec;
  // sort(v, v + n);
  // modint ans = 0;
  // for (int mask = 0; mask < (1 << n); mask++)
  // {
  //   modint at = 1;
  //   pi prv = {1, 1};
  //   for (int j = 0; j < n; j++)
  //   {
  //     if (mask & (1 << j))
  //     {
  //       if (v[j].sec < prv.sec)
  //       {
  //         at = 0;
  //         break;
  //       }
  //       at *= ncr((v[j].fir - prv.fir) + (v[j].sec - prv.sec), v[j].fir - prv
     .fir);
  //       prv = v[j];
  //     }
  //   }
  //   at *= ncr((h - prv.fir) + (w - prv.sec), h - prv.fir);
  //   cout << mask << " " << at.val << endl;
  //   ans = (__builtin_popcount(mask) % 2) ? ans - at : ans + at;
  // }
```

```
  // cout << ans.val << endl;
  v[0] = {1, 1};
  for (int i = 1; i <= n; i++)
    cin >> v[i].fir >> v[i].sec;
  sort(v, v + n + 1);
  cout << solve(1, 0, 0).val << endl;
}
// https://atcoder.jp/contests/dp/tasks/dp_y
// dado um grid com h linhas e w colunas
// h, w <= 10^5
// existem n <= 3000 posicoes com wall

// quantos caminhos tem do square (1, 1) ate o (h, w)

// com isso, podemos fzr inclusao exclusao
// calculando quantos caminhos passam por um conjunto de walls
// mas nao podemos fazer 2^30
// maaas, podemos fazer uma dp pra computar essa inclusao exclusao
// e dale
```

2.11 Knapsack

```
//O problema mais classico de Programacao Dinamica talvez seja o Knapsack.
//De maneira geral, um ladrao ira roubar uma casa com uma mochila
//que suporta um peso s. Ele ve n objetos na casa e sabe estimar o peso pi    e
     o valor vi
//de cada objeto i. Com essas informacoes, qual o maior valor que o ladrao pode
     roubar sem rasgar sua mochila?
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pii pair<int, pi>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 1001
#define INF 1000000000

int n, l;
int value[MAXN];
int peso[MAXN];
int dp[MAXN][MAXN];

int knapsack(int i, int limit)
{
  if (dp[i][limit] >= 0) // se ja foi calculado
  {
    return dp[i][limit];
  }

  if (i == n or !limit) // se chegou no fim do array ou chegou no limite
  {
    return dp[i][limit] = 0;
  }

  int nao_coloca = knapsack(i + 1, limit); // recursivamente pra caso eu nao
       coloque o objeto i

  if (peso[i] <= limit) // se eu consigo botar o objeto i
  {
    int coloca = value[i] + knapsack(i + 1, limit - peso[i]);
    return dp[i][limit] = max(coloca, nao_coloca);
  }

  return dp[i][limit] = nao_coloca;
}
signed main()
{
  cin >> l >> n;
  for (int i = 0; i < n; i++)
  {
    cin >> peso[i] >> value[i];
  }
  memset(dp, -1, sizeof(dp));
  cout << knapsack(0, l) << endl;
  return 0;
}
```

2.12 largest-sum-contiguous-subarray

```
// dada uma sequencia s qual a maior soma que podemos obter escolhendo um
     subconjunto de termos adjacentes de s
// nesse caso o temos apenas duas opcoes
// nao usar o elemento v[i]
// ou
// usamos, adicionando a maior soma possivel que antes dele
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200001
#define mod 1000000007

int kadane(vector<int> v)
{
  int n = v.size(), ans = 0, max_here = 0;
  for (int i = 0; i < n; i++)
  {
    max_here += v[i];
    if (ans < max_here)
      ans = max_here;
    if (max_here < 0)
      max_here = 0;
  }
  return ans;
}
int kadane_circular(vector<int> v)
{
  int n = v.size(), max_kadane = kadane(v);
  int max_wrap = 0, i;
  for (i = 0; i < n; i++)
  {
    max_wrap += v[i];
    v[i] = -v[i];
  }
  max_wrap += kadane(v);
  return max(max_wrap, max_kadane);
}
signed main()
{
  int n;
  cin >> n;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  cout << kadane_circular(v) << endl;
  return 0;
}
```

2.13 largest square

```cpp
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define double long double
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 1001
#define mod 1000000007

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  int v[n][n];
  int dp[n][n];
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      cin >> v[i][j];
  int ans = 0;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      dp[i][j] = v[i][j];
      if (i && j && dp[i][j])
        dp[i][j] = min({dp[i][j - 1], dp[i - 1][j], dp[i - 1][j - 1]}) + 1;
      ans = max(ans, dp[i][j]);
    }
  }
  cout << ans * ans << endl;
  return 0;
}
```

2.14 lcs

```cpp
//Dadas duas sequencias s1 e s2, uma de tamanho n e outra de tamanho m, qual a
    maior subsequencia comum as duas?

// uma subsequencia de s e um subconjunto dos elementos de s na mesma ordem em
    que apareciam antes.
// isto significa que {1, 3, 5} e uma subsequencia de {1, 2, 3, 4, 5}, mesmo 1
    nao estando do lado do 3.
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pii pair<int, pi>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 1001
#define INF 1000000000

int v1[MAXN];
int v2[MAXN];
int dp[MAXN][MAXN];

void lcs(int m, int n)
{
  for (int i = 0; i <= m; i++)
  {
    for (int j = 0; j <= n; j++)
    {
      if (i == 0 || j == 0) //se uma das sequencias for vazia
        dp[i][j] = 0;
      else if (v1[i - 1] == v2[j - 1]) // se eh igual, adiciono a lcs e subtraio
          dos dois
        dp[i][j] = dp[i - 1][j - 1] + 1;
      else
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]); // se nao retorno o maximo
            entre tirar um dos dois caras
    }
  }
  cout << dp[m][n] << endl;
}
signed main()
{
  int n, m;
  cin >> n >> m;
  for (int i = 0; i < n; i++)
    cin >> v1[i];
  for (int i = 0; i < m; i++)
    cin >> v2[i];
  lcs(n, m);
  return 0;
}
```

2.15 lichao

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define pci pair<char, int>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

const int inf = 1e18;

struct line
{
  int a, b, id;
  int ch[2];
  line()
  {
    a = 0, b = inf, id = -1;
    ch[0] = -1, ch[1] = -1;
  }
  line(int aa, int bb, int i)
  {
    a = aa, b = bb, id = i;
    ch[0] = -1, ch[1] = -1;
  }
  int f(int x) { return a * x + b; }
};
struct save
{
  int a, b, id, new_id, p;
};
struct lichao
{
  int lo, hi, curr;
  vector<line> t;
  stack<save> st; // se nao precisar de rollback, pode tirar a stack (pra nao
      usar tanta memoria)

  lichao(int ll, int rr)
  {
```

```
    lo = ll, hi = rr;
    t.emplace_back();
  }
  int child(int p, int d)
  {
    if (t[p].ch[d] == -1)
    {
      t[p].ch[d] = t.size();
      t.emplace_back();
    }
    return t[p].ch[d];
  }
  bool cmp(line a, line b, int x)
  {
    if (a.f(x) != b.f(x)) // menor valor em x
      return a.f(x) < b.f(x);
    return a.id > b.id; // desempata pelo maior id
  }
  void add(int l, int r, line s, int p)
  {
    int mid = (l + r) >> 1;
    bool fl = cmp(s, t[p], l);
    bool fm = cmp(s, t[p], mid);
    bool fr = cmp(s, t[p], r);
    if (fm)
    {
      st.push({t[p].a, t[p].b, t[p].id, curr, p});
      swap(t[p], s);
      swap(t[p].ch, s.ch);
    }
    if (s.b == inf)
      return;
    if (fl != fm)
      add(l, mid - 1, s, child(p, 0));
    else if (fr != fm)
      add(mid + 1, r, s, child(p, 1));
  }
  pi query(int l, int r, int x, int p)
  {
    int mid = (l + r) >> 1;
    pi ans = {t[p].f(x), -t[p].id}; // como eu quero o maior id, basta negar e
        pegar o menor
    if (ans.fir == inf)
      return ans;
    if (x < mid)
      return min(ans, query(l, mid - 1, x, child(p, 0)));
    return min(ans, query(mid + 1, r, x, child(p, 1)));
  }
  void add(line s)
  {
    curr = s.id;
    add(lo, hi, s, 0);
  }
  pi qry(int x)
  {
    return query(lo, hi, x, 0);
  }
  void rollback(int id)
  {
    while (!st.empty() && st.top().new_id == id)
    {
      int p = st.top().p;
      t[p].a = st.top().a;
      t[p].b = st.top().b;
      t[p].id = st.top().id;
      st.pop();
    }
  }
};
signed main()
{
  lichao lt(0, 1e9 + 2);
  lt.add(line(3, 2, 0));
  lt.add(line(5, -6, 1));
  cout << lt.qry(10).fir << " " << -lt.qry(10).sec << endl;
}
// li-chao tree
// dado uma coordenada x
// e um conjunto com varias equacoes lineares da forma: y = ax + b
// retorna o menor valor de y entre as equacoes do conjunto
// O(log(hi - lo))
// no qual:
// lo -> menor valor possivel de um x que vai ser passado pra uma query
// hi -> maior valor possivel de um x que vai ser passado pra uma query
```

2.16 lis

```
// dada uma sequencia s qualquer, descobrir o tamanho da maior subsequencia
     crescente de s
// uma subsequencia de s e qualquer subconjunto de elementos de s.
// Para cada novo numero, voce tem duas operacoes possiveis:
// 1 - Colocar o novo numero no topo de uma pilha se ele nao superar o que ja
     esta em seu topo;
// ou
// 2 - Criar uma nova pilha a direita de todas as outras e colocar o novo numero
     la.
// ao final do processo a nossa pilha tera os elementos da lis.
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pd pair<double, int>
#define pib pair<pi, bool>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 200001
#define MAXL 1000001
#define mod 1000000007

vector<int> v;

int lis()
{
  vector<int> q;
  for (int i = 0; i < v.size(); i++)
  {
    vector<int>::iterator it = lower_bound(q.begin(), q.end(), v[i]);
    if (it == q.end())
      q.pb(v[i]);
    else
      *it = v[i];
  }
  for (int i = 0; i < q.size(); i++)
    cout << q[i] << " ";
  cout << endl;
  return q.size();
}
signed main()
{
  int n;
  cin >> n;
  v.resize(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  cout << lis() << endl;
  return 0;
}
```

2.17 max matrix path

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define mp make_pair
```

```cpp
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 301
#define MAXL 20
#define mod 1000000007
#define INF 1000000001

int n;
int grid[MAXN][MAXN];
int dp[MAXN][MAXN];

int solve(int i, int j)
{
  if (i == n - 1 && j == n - 1)
    return grid[i][j];
  if (dp[i][j] != -1)
    return dp[i][j];
  if (i + 1 < n && j + 1 < n)
    return dp[i][j] = grid[i][j] + max(solve(i + 1, j), solve(i, j + 1));
  if (i + 1 < n)
    return dp[i][j] = grid[i][j] + solve(i + 1, j);
  if (j + 1 < n)
    return dp[i][j] = grid[i][j] + solve(i, j + 1);
}
signed main()
{
  cin >> n;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      cin >> grid[i][j];
  memset(dp, -1, sizeof(dp));
  cout << solve(0, 0) << endl;
  return 0;
}
```

2.18 optimization with fft

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// #define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

struct modint
{
  int val;
  modint(int v = 0) { val = ((v % mod) + mod) % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * 1ll * o.val) % mod); }
  int operator/(modint o) { return (val * 1ll * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};
namespace fft
{
  int n;
  int root = -31;
  int root_1 = 128805723;
  int pw = __builtin_ctz(mod - 1);
  int root_pw = (1 << pw);

  void ntt(vector<modint> &a, bool invert)
  {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++)
    {
      int bit = n >> 1;
      for (; j & bit; bit >>= 1)
        j ^= bit;
      j ^= bit;
      if (i < j)
        swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1)
    {
      modint wlen = (invert) ? root_1 : root;
      for (int i = len; i < root_pw; i <<= 1)
        wlen *= wlen;
      for (int i = 0; i < n; i += len)
      {
        modint w = 1;
        for (int j = 0; j < len / 2; j++)
        {
          modint u = a[i + j];
          modint v = a[i + j + len / 2] * w;
          a[i + j] = u + v;
          a[i + j + len / 2] = u - v;
          w *= wlen;
        }
      }
    }
    if (invert)
    {
      modint n_1 = modint(n).inv();
      for (int i = 0; i < a.size(); i++)
        a[i] *= n_1;
    }
  }
  vector<modint> mul(vector<modint> &a, vector<modint> &b)
  {
    n = 1;
    while (n < 2 * max(a.size(), b.size()))
      n <<= 1;
    a.resize(n);
    b.resize(n);
    ntt(a, false);
    ntt(b, false);
    for (int i = 0; i < n; i++)
      a[i] *= b[i];
    ntt(a, true);
    return a;
  }
}

// int dp[3005][3005 + 3005];

signed main()
{
```

```
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  // dp naive:
  // auto get = [&](int i, int j)
  // {
  //   return dp[i][j + 3005];
  // };
  // dp[1][3005] = 1;
  // for (int i = 2; i < 3005; i++)
  // {
  //   for (int j = -i; j <= i; j++)
  //   {
  //     dp[i][j + 3005] = get(i - 1, j - 1);
  //     dp[i][j + 3005] += get(i - 1, j + 1);
  //     dp[i][j + 3005] += (i - 2) * get(i - 1, j);
  //     dp[i][j + 3005] %= mod;
  //   }
  // }
  // for (int i = 0; i <= 5; i++)
  // {
  //   for (int j = -i; j <= i; j++)
  //     cout << dp[i][j + 3005] << ", ";
  //   cout << endl;
  // }
  int n, k;
  cin >> n >> k;
  queue<vector<modint>> vecs;
  vecs.push({0, 1, 0}); // dp[1][-1], dp[1][0], dp[1][1]
  for (int i = 2; i <= n; i++)
  {
    // dp[i] = fft::conv(dp[i - 1], {1, i - 2, 1})
    vecs.push({1, i - 2, 1});
  }
  // faz as convolucoes pegando smp os dois de menor tamanho
  while (vecs.size() > 1)
  {
    vector<modint> v = vecs.front();
    vecs.pop();
    vector<modint> v2 = vecs.front();
    vecs.pop();
    vector<modint> v3 = fft::mul(v, v2);
    vecs.push(v3);
  }
  // quero o valor de dp[n][k], o k dado na entrada pode ser negativo
  cout << vecs.back()[k + n].val << endl;
}
// https://atcoder.jp/contests/abc385/tasks/abc385_g
// a dp eh tipo:
// dp[i][j] = dp[i - 1][j] * (i - 2) + dp[i - 1][j - 1] + dp[i - 1][j + 1]
// considerando um certo i, j ta no intervalo [-i, i]
// mas n eh na casa de 10^5
```

2.19 permutations

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 1000000007

int n;
string s;
int dp[3005][3005];
int sum[3005][3005];

int solve(int i, int j);

void solve2(int i)
{
  if (sum[i][0] != -1)
    return;
  sum[i][0] = 0;
  for (int j = 0; j <= i; j++)
    sum[i][j + 1] = (sum[i][j] + solve(i, j)) % mod;
}
int get(int i, int l, int r)
{
  return (sum[i][r + 1] - sum[i][l] + mod) % mod;
}
int solve(int i, int j)
{
  if (i == s.size())
    return 1;
  if (dp[i][j] != -1)
    return dp[i][j];
  int nums = i + 1, ans = 0;
  solve2(i + 1);
  if (s[i] == '>')
    ans = get(i + 1, 0, min(nums, j));
  else
    ans = get(i + 1, j + 1, nums);
  return dp[i][j] = ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> s;
  memset(dp, -1, sizeof(dp));
  memset(sum, -1, sizeof(sum));
  cout << solve(0, 0) << endl;
}
// dado uma string s de tamanho n - 1 e um inteiro n
// conte quantas permutacoes de tamanho n satisfazem as seguintes restricoes
// se s[i] == '>', logo p[i] > p[i + 1]
// ou se s[i] == '<', logo p[i] < p[i + 1]

// se eu calculei dp[i][j]
// significa que eu preenchei o prefixo de tamanho i com uma permutacao
// e o ultimo elemento que eu coloquei foi o j (note que j <= i)

// se o proximo elemento precisa ser > j
// entao dp[i][j] = dp[i + 1][j + 1] + ... dp[i + 1][i + 1]
// caso contrario

// note que essa dp eh correta pq
// se eu tou botando um elemento x < n na transicao
// pro prefixo que eu tinha resolvido, posso incrementar todo mundo que for >= x
    que ta tudo certo
// tipo se eu tinha construido a permutacao 2 1 4 3
// tou calculando dp[4][3] entao pq o 3 foi o ultimo elemento
// mas na transicao tou considerando dp[4][3] = ... dp[5][2] ...
// a permutacao do dp[5][2] seria a 3 1 5 4 2 que continua sendo valida

// soh codar com soma de prefixo pra otimizar a dp e dale nessa questao
```

2.20 sos dp

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

```cpp
#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100001
#define mod 1000000007

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  // exemplos de sos dp para calcular f[x] para cada mask x
  // a[x] eh o valor de uma funcao a para uma mask x
  // complexidade: O(M * 2^M), M = numero de bits

  // Exemplo 1:
  // nesse caso, f[x] eh a funcao que soma:
  // todos os a[i], tal que, (x & i) == i)
  // isso eh, i eh uma "mask filha" de x
  // pois todos os bits de i estao setados em x
  for (int mask = 0; mask < (1 << m); mask++)
  {
    f[mask] = a[mask];
  }
  for (int i = 0; i < m; ++i)
  {
    for (int mask = 0; mask < (1 << m); mask++)
    {
      if (mask & (1 << i))
        f[mask] += f[mask ^ (1 << i)];
    }
  }

  // Exemplo 2:
  // nesse caso, f[x] eh a funcao que soma:
  // todos os a[i], tal que, (x & i) == x)
  // isso eh, i eh uma "mask pai" de x
  // pois todos os bits de x estao setados em i
  for (int mask = 0; mask < (1 << m); mask++)
  {
    f[mask] = a[mask];
  }
  for (int i = 0; i < m; ++i)
  {
    for (int mask = 0; mask < (1 << m); mask++)
    {
      if (!(mask & (1 << i)))
        f[mask] += f[mask ^ (1 << i)];
    }
  }

  return 0;
}
// https://codeforces.com/blog/entry/45223
```

2.21 steiner tree

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 4007
#define mod 998244353

const int inf = 1e18;

int n, m, k, sz;

int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, 1, -1};
int a[101][101];

vector<int> g[201];
int w[201][201];
int dist[201][201];
int par_floyd[201][201];
int dp[201][1 << 8];
char anss[201][201];
vector<pi> par[201][1 << 8];
int pos[10];

int steiner()
{
  // floyd warshall
  for (int i = 0; i < sz; i++)
  {
    for (int j = 0; j < sz; j++)
    {
      if (i == j)
      {
        dist[i][j] = 0;
      }
      else
      {
        dist[i][j] = w[i][j];
        par_floyd[i][j] = j;
      }
    }
  }
  for (int k = 0; k < sz; k++)
  {
    for (int i = 0; i < sz; i++)
    {
      for (int j = 0; j < sz; j++)
      {
        if (dist[i][j] > dist[i][k] + dist[k][j])
        {
          dist[i][j] = dist[i][k] + dist[k][j];
          par_floyd[i][j] = par_floyd[i][k];
        }
      }
    }
  }
  for (int i = 0; i < sz; i++)
  {
    for (int j = 0; j < (1 << k); j++)
      dp[i][j] = inf;
  }
  for (int i = 0; i < k; i++)
  {
    for (int j = 0; j < sz; j++)
    {
      dp[j][1 << i] = dist[pos[i]][j];
      par[j][1 << i] = {{pos[i], 0}};
    }
  }
  for (int mask = 2; mask < (1 << k); mask++)
  {
    for (int i = 0; i < sz; i++)
    {
      for (int mask2 = mask; mask2 > 0; mask2 = (mask2 - 1) & mask)
      {
        int mask3 = mask ^ mask2;
        if (dp[i][mask] > dp[i][mask2] + dp[i][mask3])
        {
          dp[i][mask] = dp[i][mask2] + dp[i][mask3];
          par[i][mask] = {{i, mask2}, {i, mask3}};
        }
      }
      for (int j = 0; j < sz; j++)
      {
        if (dp[j][mask] > dp[i][mask] + dist[i][j])
        {
          dp[j][mask] = dp[i][mask] + dist[i][j];
          par[j][mask] = {{i, mask}};
        }
      }
```

```
    }
  }
  // preciso somar a[i / m][i % m] pq tou usando a[i][j]
  // como peso de uma aresta (i, j) -> (x, y)
  // mas eh especifico para esse problema do garden
  int ans = inf, best = -1;
  for (int i = 0; i < sz; i++)
  {
    int curr = a[i / m][i % m] + dp[i][(1 << k) - 1];
    if (curr < ans)
    {
      ans = curr;
      best = i;
    }
  }
  // recuperar resposta
  queue<pi> q;
  q.push({best, (1 << k) - 1});
  while (!q.empty())
  {
    auto [i, mask] = q.front();
    q.pop();
    anss[i / m][i % m] = 'X';
    for (auto [j, mask2] : par[i][mask])
    {
      // marcar o caminho de j para i feito pelo floyd warshall
      int st = j, en = i;
      while (st != en)
      {
        st = par_floyd[st][en];
        anss[st / m][st % m] = 'X';
      }
      q.push({j, mask2});
    }
  }
  return ans;
}
signed main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);
  cin >> n >> m >> k;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++)
    {
      cin >> a[i][j];
      anss[i][j] = '.';
    }
  }
  sz = (n * m);
  for (int i = 0; i < sz; i++)
  {
    for (int j = 0; j < sz; j++)
      w[i][j] = inf;
  }
  // montando o grafo
  // (i, j) -> (x, y) com peso a[i][j]
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++)
    {
      for (int d = 0; d < 4; d++)
      {
        int x = i + dx[d];
        int y = j + dy[d];
        if (x >= 0 && x < n && y >= 0 && y < m)
          w[(i * m) + j][(x * m) + y] = a[i][j];
      }
    }
  }
  // posicoes importantes
  for (int i = 0; i < k; i++)
  {
    int x, y;
    cin >> x >> y;
    x--, y--;
    pos[i] = (x * m) + y;
  }
  cout << steiner() << endl;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++)
      cout << anss[i][j];
    cout << endl;
  }
  return 0;
}
// https://codeforces.com/problemset/problem/152/E
// dada uma matriz representando um jardim
// a posicao (i, j) tem a[i][j] flores
// com isso, quero cobrir algumas posicoes com concreto
// quando cobrimos uma posicao (i, j) com concreto, "matamos" as a[i][j] flores
    daquela posicao
// existem k <= 7 posicoes importantes, que devem ser cobertas com concreto
// alem disso, posso cobrir qualquer outra posicao com concreto
// alem disso, para duas posicoes a e b que sao importantes, deve existir um
    caminho
// de a ate b passando somente por posicoes cobertas por concreto.
// quero minimizar o numero de flores mortas, satisfazendo essas condicoes

// o que queremos nesse caso, eh uma steiner tree
// dado um grafo, com peso nas arestas
// e um subconjunto de vertices
// queremos achar uma arvore de menor peso que contenha todos os vertices do
    subconjunto
// mas essa arvore pode conter tambem outros vertices que nao estao no
    subconjunto
// minimizando o peso total das arestas da arvore

// um outro problema de steiner tree: https://codeforces.com/gym/101908/problem/
    J
```

2.22 subsequences string

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define mp make_pair
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100
#define MAXL 20
#define mod 998244353

void count(string a, string b)
{
  int m = a.size();
  int n = b.size();
  int dp[m + 1][n + 1] = {{0}};
  for (int i = 0; i <= n; ++i)
    dp[0][i] = 0;
  for (int i = 0; i <= m; ++i)
    dp[i][0] = 1;
  for (int i = 1; i <= m; i++)
  {
    for (int j = 1; j <= n; j++)
    {
      if (a[i - 1] == b[j - 1])
        dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
      else
        dp[i][j] = dp[i - 1][j];
    }
  }
  cout << dp[m][n] << endl;
}
signed main()
{
  string a, b;
```

```
  cin >> a >> b;
  count(a, b);
  return 0;
}
```

2.23 subset sum

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

// subset sum com bitset de tamanho variado
// usada no https://codeforces.com/contest/1856/problem/E2
// with n <= 10^6
template <int len = 1>
int subset_sum(int n, int h)
{
  if (n >= len)
  {
    return subset_sum<std::min(len * 2, (int)MAXN)>(n, h);
  }
  bitset<len> dp;
  dp[0] = 1;
  for (auto const &x : w)
  {
    dp = dp | (dp << x);
  }
  return dp._Find_next(max(0ll, h - 1)); // retorna o proximo bit setado apos a
       posicao passada como parametro
}
int solve(vector<int> &w, int tot, int h)
{
  // tot -> soma de todos os elementos de w
  // h -> valor desejado
  // quero retornar o menor valor x >= h, tal que existe um subset com soma x em
       w
  if (!w.size())
    return 0;
  sort(w.rbegin(), w.rend());
  if (w[0] * 2 >= tot)
    return w[0];
  int n = w.size();
  w.pb(0);
  vector<int> aux;
  int p = 0;
  for (int i = 1; i <= n; i++)
  {
    if (w[i] != w[i - 1])
    {
      int cnt = i - p;
      int x = w[i - 1];
      int j = 1;
      while (j < cnt)
      {
        aux.pb(x * j);
        cnt -= j;
        j *= 2;
      }
      aux.pb(x * cnt);
      p = i;
    }
  }
  swap(aux, w);
  return subset_sum(tot, h);
}

int f[MAXN];      // f[i] -> quantos "itens" com valor i tem
bitset<MAXN> dp; // dp[i] = 1, se existe um subset com soma i
// garantir que a soma de todo mundo seja < MAXN
void subset_sum(vector<int> &v)
{
  for (auto const &i : v)
  {
    f[i]++;
  }
  dp[0] = 1;
  for (int i = 1; i < MAXN; i++)
  {
    while (f[i] > 2)
    {
      f[i * 2]++;
      f[i] -= 2;
    }
    while (f[i]--)
      dp |= (dp << i);
  }
}

// https://github.com/gabrielpessoa1/ICPC-Library/blob/master/code/Miscellaneous
    /SubsetSum.cpp
/*
 Given N non-negative integer weights w and a non-negative target t,
 computes the maximum S <= t such that S is the sum of some subset of the
      weights.
 O(N * max(w[i]))
*/
int knapsack(vector<int> w, int t)
{
  int a = 0, b = 0;
  while (b < w.size() && a + w[b] <= t)
  {
    a += w[b++];
  }
  if (b == w.size())
  {
    return a;
  }
  int m = *max_element(w.begin(), w.end());
  vector<int> u, v(2 * m, -1);
  v[a + m - t] = b;
  for (int i = b; i < w.size(); i++)
  {
    u = v;
    for (int x = 0; x < m; x++)
    {
      v[x + w[i]] = max(v[x + w[i]], u[x]);
    }
    for (int x = 2 * m; --x > m;)
    {
      for (int j = max(0ll, u[x]); j < v[x]; j++)
        v[x - w[j]] = max(v[x - w[j]], j);
    }
  }
  a = t;
  while (v[a + m - t] < 0)
  {
    a--;
  }
  return a;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
```

2.24 suffix sum

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

int n, k;
int a[101];
int dp[101][100005];
int sum[101][100005];

int solve(int i, int j);

void calc(int i)
{
  sum[i][0] = solve(i, 0);
  for (int j = 1; j <= k; j++)
    sum[i][j] = (sum[i][j - 1] + solve(i, j)) % mod;
}
int solve(int i, int j)
{
  if (i == n)
    return (j == 0) ? 1 : 0;
  if (dp[i][j] != -1)
    return dp[i][j];
  int ans = 0, limit = min(j, a[i]);
  if (sum[i + 1][j] == -1)
    calc(i + 1);
  ans = (ans + sum[i + 1][j]) % mod;
  if (j - limit - 1 >= 0)
    ans = (ans - sum[i + 1][j - limit - 1] + mod) % mod;
  return dp[i][j] = ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> k;
  for (int i = 0; i < n; i++)
    cin >> a[i];
  memset(dp, -1, sizeof(dp));
  memset(sum, -1, sizeof(sum));
  cout << solve(0, k) << endl;
  return 0;
}
// uma dp que tem uma recorrencia do tipo:
// dp[i][j] = dp[i + 1][l] + dp[i + 1][l + 1] + ... + dp[i + 1][r]
// o jeito de fazer isso com soma de sufixo sem ter que codar a dp iterativa :)
```

2.25 tip

```cpp
// dados os valores de moedas v1, v2, ... vn e possivel formar um valor m como
     combinacao de moedas
// para isso basta montar uma dp inicializada com -1
// nesse caso a dp so precisa de um parametro q e = valor restante ate o limite
// mas podem existir variacoes do problema q precise de mais coisas
// se em achar alguma combinacao valida retorna 1, se nao retorna 0
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pd pair<double, int>
#define pib pair<pi, bool>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 200001
#define MAXL 10001
#define mod 1000000007

int dp[MAXN];
vector<int> v;

int solve(int rem)
{
  if (rem == 0)
    return 1;
  if (rem < 0)
    return 0;
  if (dp[rem] >= 0)
    return dp[rem];
  for (int i = 0; i < v.size(); i++)
    if (solve(rem - v[i]))
      return dp[rem - v[i]] = 1;
  return dp[rem] = 0;
}
signed main()
{
  int n, m;
  cin >> n >> m;
  v.resize(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  memset(dp, -1, sizeof(dp));
  (solve(m)) ? cout << "Yes\n" : cout << "No\n";
  return 0;
}
```

3 Geometry

3.1 ConvexHull

```cpp
#include <bits/stdc++.h>

using namespace std;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define MAXN 100005

struct point
{
  int x, y, id;
  point(int x, int y, int id) : x(x), y(y), id(id) {}
  point() {}
  point operator-(point const &o) const { return {x - o.x, y - o.y, -1}; }
  bool operator<(point const &o) const
  {
    if (x == o.x)
      return y < o.y;
    return x < o.x;
  }
  int operator^(point const &o) const { return x * o.y - y * o.x; }
};
int ccw(point const &a, point const &b, point const &x)
{
  auto p = (b - a) ^ (x - a);
  return (p > 0) - (p < 0);
}
vector<point> convex_hull(vector<point> P) // sem colineares
```

```
{
  sort(P.begin(), P.end());
  vector<point> L, U;
  for (auto p : P)
  {
    while (L.size() >= 2 && ccw(L.end()[-2], L.end()[-1], p) == -1)
      L.pop_back();
    L.push_back(p);
  }
  reverse(P.begin(), P.end());
  for (auto p : P)
  {
    while (U.size() >= 2 && ccw(U.end()[-2], U.end()[-1], p) == -1)
      U.pop_back();
    U.push_back(p);
  }
  L.insert(L.end(), U.begin(), U.end() - 1);
  return L;
}
vector<point> convex_hull_no_collinears(vector<point> P) // com colineares
{
  sort(P.begin(), P.end());
  vector<point> L, U;
  for (auto p : P)
  {
    while (L.size() >= 2 && ccw(L.end()[-2], L.end()[-1], p) <= 0)
      L.pop_back();
    L.push_back(p);
  }
  reverse(P.begin(), P.end());
  for (auto p : P)
  {
    while (U.size() >= 2 && ccw(U.end()[-2], U.end()[-1], p) <= 0)
      U.pop_back();
    U.push_back(p);
  }
  L.insert(L.end(), U.begin(), U.end() - 1);
  return L;
}
signed main()
{
  // int n;
  // cin >> n;
  // vector<point> v(n);
  // for (int i = 0; i < n; i++)
  // {
  //   cin >> v[i].x >> v[i].y;
  //   v[i].id = i;
  // }
  // vector<point> ans = convex_hull(v);
  // vector<int> ids;
  // for (auto const &i : ans)
  // {
  //   ids.pb(i.id);
  // }
  // sort(ids.begin(), ids.end());
  // ids.erase(unique(ids.begin(), ids.end()), ids.end());
  // for (auto const &i : ids)
  //   cout << i + 1 << " ";
  // cout << endl;
  int n;
  while (cin >> n)
  {
    if (n == 0)
      break;
    vector<point> v(n);
    for (int i = 0; i < n; i++)
    {
      cin >> v[i].x >> v[i].y;
      v[i].id = i;
    }
    vector<point> ans = convex_hull_no_collinears(v);
    vector<pi> resp;
    for (auto const &i : ans)
    {
      if (!resp.size() || (pi(i.x, i.y) != resp.back()))
        resp.pb({i.x, i.y});
    }
    cout << resp.size() << endl;
    for (auto [x, y] : resp)
      cout << x << " " << y << endl;
  }
}
// https://codeforces.com/gym/104555/problem/G
// https://open.kattis.com/problems/convexhull (sem colinear)
```

3.2 convex hull point location

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 5005
#define mod 998244353

struct pt
{
  int x, y;
  pt operator+(pt p) { return {x + p.x, y + p.y}; }
  pt operator-(pt p) { return {x - p.x, y - p.y}; }
  bool operator==(pt p) { return (x == p.x && y == p.y); }
  int cross(pt p) { return x * p.y - y * p.x; }
  int cross(pt a, pt b) { return (a - *this).cross(b - *this); }
  int dot(pt p) { return x * p.x + y * p.y; }
};
bool cmp_x(pt a, pt b)
{
  if (a.x != b.x)
    return a.x < b.x;
  return a.y < b.y;
}
// acha o convex hull
vector<pt> convex_hull(vector<pt> pts)
{
  if (pts.size() <= 1)
    return pts;
  sort(pts.begin(), pts.end(), cmp_x);
  vector<pt> h(pts.size() + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(pts.begin(), pts.end()))
  {
    for (auto const &p : pts)
    {
      while (t >= s + 2 && h[t - 2].cross(h[t - 1], p) <= 0)
        t--;
      h[t++] = p;
    }
  }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
int sgn(int x)
{
  return (x > 0) - (x < 0);
}
int side_of(pt s, pt e, pt p)
{
  return sgn(s.cross(e, p));
}
bool on_segment(pt s, pt e, pt p)
{
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
// retorna se o ponto p esta dentro ou nao do convex hull l
// caso strict = true, entao considera true se tiver na borda
```

```
// caso strict = false, entao considera false se tiver na borda
bool is_hull(vector<pt> &l, pt p, bool strict = true)
{
  int a = 1, b = l.size() - 1, r = !strict;
  if (l.size() < 3)
    return r && on_segment(l[0], l.back(), p);
  if (side_of(l[0], l[a], l[b]) > 0)
    swap(a, b);
  if (side_of(l[0], l[a], p) >= r || side_of(l[0], l[b], p) <= -r)
    return false;
  while (abs(a - b) > 1)
  {
    int c = (a + b) / 2;
    (side_of(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
signed main()
{
  int n;
  cin >> n;
  vector<pt> v(n);
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].x >> v[i].y;
  }
  vector<pt> ans = convex_hull(v);
  for (int i = 0; i < n; i++)
  {
    if (!is_hull(ans, v[i]))
      cout << i + 1 << " ";
  }
  cout << endl;
}
// h da subregional - https://codeforces.com/gym/104555/problem/G
```

3.3 dynamic ch

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define double long double
#define mod 1000000007

const double eps = 1e-9;

struct pt
{
  double x, y;
  pt operator-(pt p) { return {x - p.x, y - p.y}; }
  bool eq(double a, double b) const
  {
    return abs(a - b) <= eps;
  }
  double operator^(const pt p) const { return x * p.y - y * p.x; }
  bool operator<(const pt p) const
  {
    if (!eq(x, p.x))
      return x < p.x;
    if (!eq(y, p.y))
      return y < p.y;
    return 0;
  }
  bool operator==(const pt p) const
  {
    return eq(x, p.x) and eq(y, p.y);
  }
};
double sarea(pt p, pt q, pt r)
{
  return ((q - p) ^ (r - q)) / 2;
}
bool ccw(pt p, pt q, pt r)
{
  return sarea(p, q, r) > eps;
}
// https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/Problemas/
    dynamicHull.cpp
struct upper
{
  set<pt> se;
  set<pt>::iterator it;
  // 0 - fora
  // 1 - dentro
  // 2 - na borda
  int is_under(pt p)
  {
    it = se.lower_bound(p);
    if (it == se.end())
      return 0;
    if (it == se.begin())
      return p == *it ? 2 : 0;
    if (ccw(p, *it, *prev(it)))
      return 1;
    return ccw(p, *prev(it), *it) ? 0 : 2;
  }
  void insert(pt p)
  {
    if (is_under(p))
      return;
    if (it != se.end())
      while (next(it) != se.end() and !ccw(*next(it), *it, p))
        it = se.erase(it);
    if (it != se.begin())
      while (--it != se.begin() and !ccw(p, *it, *prev(it)))
        it = se.erase(it);
    se.insert(p);
  }
};
struct dyn_hull
{
  upper U, L;
  int is_inside(pt p)
  {
    int u = U.is_under(p), l = L.is_under({-p.x, -p.y});
    if (!u || !l)
      return 0;
    return max(u, l);
  }
  void insert(pt p)
  {
    U.insert(p);
    L.insert({-p.x, -p.y});
  }
  int size()
  {
    int ans = U.se.size() + L.se.size();
    return ans <= 2 ? ans / 2 : ans - 2;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// convex hull dinamico
// problema para usar: https://open.kattis.com/problems/hiringhelp
```

3.4 halfplane intersection

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

const long double eps = 1e-9;
const long double inf = 1e9;

struct pt
{
  long double x, y;
  pt(long double x = 0, long double y = 0) : x(x), y(y) {}
  friend pt operator+(pt p, pt q)
  {
    return pt(p.x + q.x, p.y + q.y);
  }
  friend pt operator-(pt p, pt q)
  {
    return pt(p.x - q.x, p.y - q.y);
  }
  friend pt operator*(pt p, long double k)
  {
    return pt(p.x * k, p.y * k);
  }
  friend long double dot(pt p, pt q)
  {
    return p.x * q.x + p.y * q.y;
  }
  friend long double cross(pt p, pt q)
  {
    return p.x * q.y - p.y * q.x;
  }
};
struct halfplane
{
  pt p, pq;
  long double angle;
  halfplane() {}
  halfplane(pt a, pt b) : p(a), pq(b - a)
  {
    angle = atan2l(pq.y, pq.x);
  }
  bool out(const pt &r)
  {
    return cross(pq, r - p) < -eps;
  }
  bool operator<(halfplane e) const
  {
    return angle < e.angle;
  }
  friend pt inter(halfplane s, halfplane t)
  {
    long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
    return s.p + (s.pq * alpha);
  }
};
vector<pt> hp_intersect(vector<halfplane> &h)
{
  pt box[4] = {pt(inf, inf), pt(-inf, inf), pt(-inf, -inf), pt(inf, -inf)}; //
      Bounding box in CCW order
  for (int i = 0; i < 4; i++)
  {
    halfplane aux(box[i], box[(i + 1) % 4]);
    h.pb(aux);
  }
  sort(h.begin(), h.end());
  deque<halfplane> dq;
  int len = 0;
  for (int i = 0; i < h.size(); i++)
  {
    while (len > 1 && h[i].out(inter(dq[len - 1], dq[len - 2])))
    {
      dq.pop_back();
      --len;
    }
    while (len > 1 && h[i].out(inter(dq[0], dq[1])))
    {
      dq.pop_front();
      --len;
    }
    if (len > 0 && fabsl(cross(h[i].pq, dq[len - 1].pq)) < eps)
    {
      if (dot(h[i].pq, dq[len - 1].pq) < 0.0)
      {
        return vector<pt>();
      }
      if (h[i].out(dq[len - 1].p))
      {
        dq.pop_back();
        --len;
      }
      else
      {
        continue;
      }
    }
    dq.push_back(h[i]);
    ++len;
  }
  while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2])))
  {
    dq.pop_back();
    --len;
  }
  while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1])))
  {
    dq.pop_front();
    --len;
  }
  if (len < 3)
  {
    return vector<pt>();
  }
  vector<pt> ret(len);
  for (int i = 0; i + 1 < len; i++)
  {
    ret[i] = inter(dq[i], dq[i + 1]);
  }
  ret.back() = inter(dq[len - 1], dq[0]);
  return ret;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q; // quantidade de poligonos
  vector<halfplane> h;
  while (q--)
  {
    int n;
    cin >> n;
    vector<pt> v(n);
    for (int i = 0; i < n; i++)
    {
      cin >> v[i].x >> v[i].y;
    }
    for (int i = 0; i < n; i++)
```

```
    {
      int j = (i + 1) % n;
      h.pb(halfplane(v[i], v[j]));
    }
  }
  vector<pt> ans = hp_intersect(h);
  if (ans.size() == 0)
  {
    cout << "0.0\n";
    return 0;
  }
  long double res = 0;
  for (int i = 0; i < ans.size(); i++) // area da interseccao
  {
    pt p = (i) ? ans[i - 1] : ans.back();
    pt q = ans[i];
    res += (p.x - q.x) * (p.y + q.y);
  }
  double resp = abs(res) / 2;
  cout << fixed << setprecision(15) << resp << endl;
  return 0;
}
// half-plane intersection

// definicoes:
// half-plane - regiao planar que consiste de todos os pontos que estao de um
     lado de uma reta
// geralmente podem ser descritos da seguninte forma
// conjuntos dos pontos (x, y) que satisfazem algo do tipo:
// ax + by + c <= 0 ou ax + by + c >= 0
// da pra representar as retas e os half-planes atraves de um ponto (que ta na
     reta) e o vetor de direcao
// e dai pros half-planes, considerando que e a regiao da esquerda em relacao ao
      vetor de direcao

// alem disso, considerar uma bounding box sendo um retangulo, pra caso a
     inserseccao dos halfplanes nao seja "fechada"

// https://open.kattis.com/problems/bigbrother
// qual a area que voce pode botar uma camera dentro do poligono
// tal que de um ponto escolhido, e possivel ver todos o poligono
// dai considerar todos os halfplanes de arestas do poligono
// e achar a interseccao de todos esses halfplanes

// https://www.codechef.com/problems/CHN02
// achar a area da interseccao de varios poligonos convexos
// considerar todos os halfplanes de arestas do poligono
// e achar a interseccao de todos esses halfplanes
```

3.5 kd tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 998244353
#define inf LLONG_MAX

struct pt
{
  int x, y, id;
  pt() {}
  pt(int xx, int yy) { x = xx, y = yy; }
  pt operator-(pt p) const { return pt(x - p.x, y - p.y); }
  bool operator<(pt p) const { return x < p.x; }
  int dist() const { return x * x + y * y; }
};
bool on_x(const pt &a, const pt &b) { return a.x < b.x; }
bool on_y(const pt &a, const pt &b) { return a.y < b.y; }
struct node
{
  pt pp;
  int id;
  int x0 = inf, x1 = -inf, y0 = inf, y1 = -inf;
  node *first = 0, *second = 0;
  int distance(const pt &p)
  {
    int x = (p.x < x0 ? x0 : p.x > x1 ? x1
                                      : p.x);
    int y = (p.y < y0 ? y0 : p.y > y1 ? y1
                                      : p.y);
    return (pt(x, y) - p).dist();
  }
  node(vector<pt> &&vp) : pp(vp[0])
  {
    for (pt p : vp)
    {
      x0 = min(x0, p.x);
      x1 = max(x1, p.x);
      y0 = min(y0, p.y);
      y1 = max(y1, p.y);
    }
    if (vp.size() > 1)
    {
      sort(vp.begin(), vp.end(), x1 - x0 >= y1 - y0 ? on_x : on_y);
      int half = vp.size() / 2;
      first = new node({vp.begin(), vp.begin() + half});
      second = new node({vp.begin() + half, vp.end()});
    }
  }
};
struct kd_tree
{
  node *root;
  kd_tree(const vector<pt> &vp) : root(new node({vp.begin(), vp.end()})) {}
  pi search(node *n, const pt &p)
  {
    if (!n->first)
    {
      if (n->pp.x == p.x && n->pp.y == p.y)
        return make_pair(inf, n->pp.id); // distancia infinita pra pontos iguais
      return make_pair((p - n->pp).dist(), n->pp.id);
    }
    node *f = n->first, *s = n->second;
    int bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec)
      swap(bsec, bfirst), swap(f, s);
    auto best = search(f, p);
    if (bsec < best.first || (!f->first))
      best = min(best, search(s, p));
    return best;
  }
  pi nearest(const pt &p)
  {
    return search(root, p);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<pt> v(n);
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].x >> v[i].y;
    v[i].id = i;
  }
  kd_tree t(v);
```

```
  pii ans = {inf, {inf, inf}};
  for (int i = 0; i < n; i++)
  {
    pi curr = t.nearest(v[i]);
    ans = min(ans, {curr.fir, {i, curr.sec}});
  }
  cout << fixed << setprecision(6) << ans.sec.fir << " " << ans.sec.sec << " "
       << sqrt(ans.fir) << endl;
  return 0;
}
// closest pair of points com kdtree
// da pra ser adaptado pro 3d tbm
// quando um ponto (x, y) pode aparecer em mais de um indice, tratar antes
// fonte: https://github.com/kth-competitive-programming/kactl/blob/main/kactl.
    pdf

// testei em:
// https://codeforces.com/contest/429/problem/D
// https://www.spoj.com/problems/CLOPPAIR/
// https://vjudge.net/problem/UVA-10245
// https://codeforces.com/gym/104020/problem/L (3D)
```

3.6 LineSweep

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007
#define PI acos(-1)

const double EPS = 1e-9;

struct pt
{
  double x, y;
};
struct seg
{
  pt p, q;
  int id;
  double get_y(double x) const
  {
    if (abs(p.x - q.x) < EPS)
      return p.y;
    return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
  }
};
bool intersect1d(double l1, double r1, double l2, double r2)
{
  if (l1 > r1)
    swap(l1, r1);
  if (l2 > r2)
    swap(l2, r2);
  return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt &a, const pt &b, const pt &c)
{
  double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
  return abs(s) < EPS ? 0 : s > 0 ? +1
                                  : -1;
}
bool intersect(const seg &a, const seg &b)
{
  return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
         intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
         vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
         vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg &a, const seg &b)
{
  double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
  return a.get_y(x) < b.get_y(x) - EPS;
}
struct event
{
  double x;
  int tp, id;
  event() {}
  event(double x, int tp, int id) : x(x), tp(tp), id(id) {}
  bool operator<(const event &e) const
  {
    if (abs(x - e.x) > EPS)
      return x < e.x;
    return tp > e.tp;
  }
};

set<seg> s;

set<seg>::iterator prev(set<seg>::iterator it)
{
  return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it)
{
  return ++it;
}
pi line_sweep(vector<seg> v)
{
  vector<event> e;
  for (int i = 0; i < v.size(); i++)
  {
    e.push_back({min(v[i].p.x, v[i].q.x), 1, i});
    e.push_back({max(v[i].p.x, v[i].q.x), 0, i});
  }
  sort(e.begin(), e.end());
  for (int i = 0; i < e.size(); i++)
  {
    int id = e[i].id;
    if (e[i].tp == 1)
    {
      auto nxt = s.lower_bound(v[id]), prv = prev(nxt);
      if (nxt != s.end() && intersect(*nxt, v[id]))
        return {(*nxt).id, id};
      if (prv != s.end() && intersect(*prv, v[id]))
        return {(*prv).id, id};
      s.insert(nxt, v[id]);
    }
    else
    {
      auto where = s.lower_bound(v[id]);
      auto nxt = next(where), prv = prev(where);
      if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
        return {(*prv).id, (*nxt).id};
      s.erase(where);
    }
  }
  return {-1, -1};
}
signed main()
{
  int n;
  cin >> n;
  vector<seg> v(n);
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].p.x >> v[i].p.y >> v[i].q.x >> v[i].q.y;
    v[i].id = i;
```

```
  }
  pi ans = line_sweep(v);
  if (ans.fir == -1)
  {
    cout << "NO\n";
  }
  else
  {
    cout << "YES\n";
    cout << ans.fir + 1 << " " << ans.sec + 1 << endl;
  }
  return 0;
}
// https://cp-algorithms.com/geometry/intersecting_segments.html
// https://acm.timus.ru/problem.aspx?space=1&num=1469
```

3.7 line trick

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 998244353

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

pi get_line(pi x, pi y)
{
  int xx = x.fir - y.fir;
  int yy = x.sec - y.sec;
  int g = __gcd(abs(xx), abs(yy));
  if (g != 0)
  {
    xx /= g, yy /= g;
  }
  if (xx < 0)
  {
    xx *= -1;
    yy *= -1;
  }
  return {xx, yy};
}
void solve()
{
  int n;
  cin >> n;
  vector<pi> v(n);
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].fir >> v[i].sec;
  }
  map<pi, int> mp;
  map<pi, int> repr;
  int cc = 0;
  // pega 2 indices i e j, e acha a reta que passa por esses 2 pontos
  while (cc < (2 * n))
  {
    cc++;
    int x = rng() % n;
    int y = rng() % n;
    if (x == y)
      continue;
    pi l = get_line(v[x], v[y]);
    mp[l]++;
    repr[l] = x;
  }
  vector<pii> vec;
  for (auto const &i : mp)
  {
    vec.pb({i.sec, i.fir});
  }
  sort(vec.rbegin(), vec.rend());
  // agora considerando as duas retas de maior frequencia
  // veja quantos pontos do conjunto estao nela
  int ans = n;
  for (int i = 0; i < min(2ll, (int)vec.size()); i++)
  {
    // guardo um "representante" dessa reta, que eu sei que ta nessa reta
    // pq dai eh so iterar por cada ponto e ver se o get_line bate com a reta
        que queremos
    int guy = repr[vec[i].sec];
    int cnt = 0;
    for (int j = 0; j < n; j++)
    {
      if (j == guy || get_line(v[guy], v[j]) == vec[i].sec)
        cnt++;
    }
    ans = min(ans, n - cnt);
  }
  cout << ans << endl;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  for (int i = 1; i <= q; i++)
  {
    cout << "Case #" << i << ": ";
    solve();
  }
  return 0;
}
// problema dahora
// https://www.facebook.com/codingcompetitions/hacker-cup/2024/practice-round/
    problems/C
// tem n pontos (x, y)
// ache a quantidade maxima de pontos, tal que, existe uma reta que passa por
    todos eles
// e imprima n - essa quantidade

// mas nao quero printar a resposta exata
// se a resposta otima for igual a m
// posso printar ate m * 2
// o que significa que da pra fazer umas heuristicas
```

3.8 minkowski

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 15
#define mod 1000000007
```

```cpp
struct pt
{
  int x, y;
  bool operator<(pt ot)
  {
    if (x != ot.x)
      return x < ot.x;
    return y < ot.y;
  }
  void operator=(pt p) { x = p.x, y = p.y; }
  bool operator==(pt p) { return (x == p.x && y == p.y); }
  bool operator!=(pt p) { return (x != p.x || y != p.y); }
  pt operator+(const pt &p) { return {x + p.x, y + p.y}; }
  pt operator-(const pt &p) { return {x - p.x, y - p.y}; }
  pt operator*(int d) { return {x * d, y * d}; }
  pt operator/(int d) { return {x / d, y / d}; }
  int cross(pt ot) const { return x * ot.y - y * ot.x; }
  int cross(pt a, pt b) const { return (a - *this).cross(b - *this); }
};
enum type
{
  outside,
  inside,
  boundary
};
int cross(pt v, pt w)
{
  return v.x * w.y - v.y * w.x;
}
bool ccw(pt a, pt b, pt c)
{
  return cross(b - a, c - b) > 0;
}
void radial_sort(vector<pt> &a)
{
  pt pivot = *min_element(a.begin(), a.end());
  auto cmp = [&](pt p, pt q)
  {
    if (p == pivot || q == pivot)
      return q != pivot;
    return ccw(pivot, p, q) > 0;
  };
  sort(a.begin(), a.end(), cmp);
}
vector<pt> trata(vector<pt> p)
{
  vector<pt> ans;
  for (int i = 0; i < p.size(); i++)
  {
    while (ans.size() >= 2 && ans.back().cross(p[i], ans.end()[-2]) == 0)
      ans.pop_back();
    ans.pb(p[i]);
  }
  if (ans.size() > 2 && ans.back().cross(p[0], ans.end()[-2]) == 0)
    ans.pop_back();
  return ans;
}
void prepare(vector<pt> &p)
{
  radial_sort(p); // sort points in counter-clockwise order
  p = trata(p);   // and the polygon dont have 3 colinear points
}
int sgn(int val)
{
  if (val > 0)
    return 1;
  else if (val < 0)
    return -1;
  return 0;
}
bool in_seg(pt p, pt a, pt b)
{
  // check if point p is in the line segment formed by a and b
  if (a.cross(b, p) == 0)
    return (p.x >= min(a.x, b.x) && p.x <= max(a.x, b.x) && p.y >= min(a.y, b.y)
            && p.y <= max(a.y, b.y));
  return 0;
}
bool in_tri(pt p, pt a, pt b, pt c)
{
  // check if point p is in the triangle formed by a, b and c
  int a1 = abs(a.cross(b, c));
  int a2 = abs(p.cross(a, b)) + abs(p.cross(a, c)) + abs(p.cross(b, c));
  return a1 == a2;
}
int in_polygon(vector<pt> &poly, pt p)
{
  int n = poly.size();
  if (n == 1)
    return (p == poly[0]) ? type::boundary : type::outside;
  if (n == 2)
    return (in_seg(p, poly[0], poly[1])) ? type::boundary : type::outside;
  if (poly[0].cross(poly[1], p) != 0 && sgn(poly[0].cross(poly[1], p)) != sgn(
      poly[0].cross(poly[1], poly[n - 1])))
    return type::outside;
  if (poly[0].cross(p, poly[n - 1]) != 0 && sgn(poly[0].cross(p, poly[n - 1]))
      != sgn(poly[0].cross(poly[1], poly[n - 1])))
    return type::outside;
  int l = 2, r = n - 1;
  if (poly[0].cross(poly[l], p) > 0)
  {
    while (l < r)
    {
      int mid = (l + r) >> 1;
      (poly[0].cross(poly[mid], p) <= 0) ? r = mid : l = mid + 1;
    }
  }
  if (!in_tri(p, poly[0], poly[l - 1], poly[l]))
    return type::outside;
  if (in_seg(p, poly[l - 1], poly[l]))
    return type::boundary;
  if (in_seg(p, poly[0], poly[1]))
    return type::boundary;
  if (in_seg(p, poly[0], poly[n - 1]))
    return type::boundary;
  return type::inside;
}
vector<pt> minkowski(vector<pt> a, vector<pt> b)
{
  prepare(a);
  prepare(b);
  a.push_back(a[0]);
  a.push_back(a[1]);
  b.push_back(b[0]);
  b.push_back(b[1]);
  vector<pt> ans;
  int i = 0, j = 0;
  while (i < a.size() - 2 || j < b.size() - 2)
  {
    ans.pb(a[i] + b[j]);
    auto c = cross(a[i + 1] - a[i], b[j + 1] - b[j]);
    if (c >= 0)
      i++;
    if (c <= 0)
      j++;
  }
  return ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  vector<pt> v;
  for (int _ = 0; _ < 3; _++)
  {
    int n;
    cin >> n;
    vector<pt> p(n);
    for (int i = 0; i < n; i++)
      cin >> p[i].x >> p[i].y;
    if (_ == 0)
      v = p;
    else
      v = minkowski(v, p);
  }
```

```
  prepare(v);
  int q;
  cin >> q;
  while (q--)
  {
    pt p;
    cin >> p.x >> p.y;
    p.x *= 3, p.y *= 3;
    // ve se o ponto (3x, 3y) esta na bora, dentro ou fora do poligono v
    (in_polygon(v, p) != type::outside) ? cout << "YES\n" : cout << "NO\n";
  }
  return 0;
}
// problema exemplo:
// https://codeforces.com/contest/87/problem/E
```

3.9 points and vectors

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007
#define PI acos(-1)

namespace p
{
  struct pt
  {
    double x, y;
    pt operator+(pt p) { return {x + p.x, y + p.y}; } // soma de pontos
    pt operator-(pt p) { return {x - p.x, y - p.y}; } // subtracao de pontos
    pt operator*(double d) { return {x * d, y * d}; } // multiplicacao por um
        double
    pt operator/(double d) { return {x / d, y / d}; } // divisao por um double
  };
  double dot(pt v, pt w) // produto escalar (dot product)
  {
    return v.x * w.x + v.y * w.y;
  }
  bool is_perp(pt v, pt w) // retorna se dois vetores sao perpendiculares (
      angulo 90 graus)
  {
    return dot(v, w) == 0;
  }
  double cross(pt v, pt w) // produto vetorial (cross product)
  {
    return v.x * w.y - v.y * w.x;
  }
  double dist(pt a, pt b) // distancia entre 2 pontos
  {
    pt c = a - b;
    return sqrt(c.x * c.x + c.y * c.y);
  }
  double dist2(pt a, pt b) // retorna o quadrado da distancia entre dois pontos
  {
    pt c = a - b;
    return c.x * c.x + c.y * c.y;
  }
  bool is_colinear(pt a, pt b, pt c) // retorna se os pontos a, b e c sao
      colineares
  {
    return cross(b - a, c - a) == 0;
  }
  bool ccw(pt a, pt b, pt c) // retorna se os pontos a,b e c estao no sentido
      anti horario
  {
    return cross(b - a, c - b) > 0;
  }
  bool cw(pt a, pt b, pt c) // retorna se os pontos a,b e c estao no sentido
      horario
  {
    return cross(b - a, c - b) < 0;
  }
  double modulo(pt v) // |v| = sqrt(x2 + y2)
  {
    return sqrt(v.x * v.x + v.y * v.y);
  }
  double angle(pt a, pt b, pt c) // angulo entre os vetores ab e ac
  {
    // dot(ab , ac) / |ab| * |ac|
    pt ab = b - a; // vetor ab
    pt ac = c - a; // vetor ac
    double m1 = modulo(ab);
    double m2 = modulo(ac);
    double m3 = m1 * m2;
    return (dot(ab, ac) / m3); // retorna o cos do angulo em graus
  }
  pt rotate(pt p, double a) // rotacionar o ponto p em relacao a origem, em a
      graus, no sentido anti-horario
  {
    a = (a * PI) / 180;
    double xx = (cos(a) * p.x) + ((sin(a) * -1) * p.y);
    double yy = (sin(a) * p.x) + (cos(a) * p.y);
    pt ans = {xx, yy};
    return ans;
  }
  double polar(pt p) // polar angle
  {
    return atan2l(p.y, p.x);
  }
  bool cmp(pt a, pt b) // ordenar pontos pelo polar angle
  {
    return polar(a) < polar(b);
  }
  bool cmp_x(pt a, pt b) // ordenar os pontos pela coordenada x
  {
    if (a.x != b.x)
      return a.x < b.x;
    return a.y < b.y;
  }
  pt polar_to_cartesian(double r, double theta) // r - distancia do centro,
      theta - polar angle
  {
    pt ans;
    ans.x = r * cos(double(theta) / 180 * PI); // assumindo que theta ta em
        graus, transforma pra radiano
    ans.y = r * sin(double(theta) / 180 * PI);
    return ans;
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
```

3.10 polygons distance

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define int long long int
```

```
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 5005
#define mod 998244353

struct pt
{
  double x, y;
  pt operator+(const pt p) const { return pt(x + p.x, y + p.y); }
  pt operator-(const pt p) const { return pt(x - p.x, y - p.y); }
  pt operator*(const double c) const { return pt(x * c, y * c); }
  pt operator/(const double c) const { return pt(x / c, y / c); }
  double operator*(const pt p) const { return x * p.x + y * p.y; }
  double operator^(const pt p) const { return x * p.y - y * p.x; }
};
struct line
{
  pt p, q;
  line() {}
  line(pt p_, pt q_) : p(p_), q(q_) {}
};
struct building
{
  // 0 - circulo, 1 - quadrado, 2 - triangulo
  int type, r;
  vector<pt> v;
  building()
  {
    v.clear();
    r = 0;
    type = 0;
  }
  void find_vertices()
  {
    pt a, b;
    double vx = v[1].x - v[0].x;
    double vy = v[1].y - v[0].y;
    a.x = (v[0].x + v[1].x) / 2 + (-vy) / 2;
    a.y = (v[0].y + v[1].y) / 2 + (vx) / 2;
    b.x = (v[0].x + v[1].x) / 2 - (-vy) / 2;
    b.y = (v[0].y + v[1].y) / 2 - (vx) / 2;
    v.pb(a);
    swap(v[1], v[2]);
    v.pb(b);
  }
};

int c, q, t, n;
vector<building> v;
vector<int> vc, vq, vt;

double sarea(pt p, pt q, pt r)
{
  return ((q - p) ^ (r - q)) / 2;
}
double dist(pt p, pt q)
{
  return hypot(p.y - q.y, p.x - q.x);
}
double disttoline(pt p, line r)
{
  return 2 * abs(sarea(p, r.p, r.q)) / dist(r.p, r.q);
}
double disttoseg(pt p, line r)
{
  if ((r.q - r.p) * (p - r.p) < 0)
    return dist(r.p, p);
  if ((r.p - r.q) * (p - r.q) < 0)
    return dist(r.q, p);
  return disttoline(p, r);
}
double dist_circ_seg(pt p1, pt p2, pt p, int r)
{
  double dist = disttoseg(p, line(p2, p1));
  dist -= r;
  return dist;
}
double dist_seg_seg(line a, line b)
{
  double ret = DBL_MAX;
  ret = min(ret, disttoseg(a.p, b));
  ret = min(ret, disttoseg(a.q, b));
  ret = min(ret, disttoseg(b.p, a));
  ret = min(ret, disttoseg(b.q, a));
  return ret;
}
double dist_square_tri(int i, int j)
{
  double ans = DBL_MAX;
  for (int x = 0; x < 4; x++)
  {
    int y = (x + 1) % 4;
    for (int x2 = 0; x2 < 3; x2++)
    {
      int y2 = (x2 + 1) % 3;
      ans = min(ans, dist_seg_seg(line(v[i].v[x], v[i].v[y]), line(v[j].v[x2], v
          [j].v[y2])));
    }
  }
  return ans;
}
double dist_square_circ(int i, int j)
{
  double ans = DBL_MAX;
  for (int x = 0; x < 4; x++)
  {
    int y = (x + 1) % 4;
    ans = min(ans, dist_circ_seg(v[i].v[x], v[i].v[y], v[j].v[0], v[j].r));
  }
  return ans;
}
double dist_tri_circ(int i, int j)
{
  double ans = DBL_MAX;
  for (int x = 0; x < 3; x++)
  {
    int y = (x + 1) % 3;
    ans = min(ans, dist_circ_seg(v[i].v[x], v[i].v[y], v[j].v[0], v[j].r));
  }
  return ans;
}
double dist_circ_circ(int i, int j)
{
  double dist = (v[i].v[0].x - v[j].v[0].x) * (v[i].v[0].x - v[j].v[0].x);
  dist += (v[i].v[0].y - v[j].v[0].y) * (v[i].v[0].y - v[j].v[0].y);
  dist = sqrtl(dist);
  dist -= (v[i].r + v[j].r);
  return (dist < 0) ? 0 : dist;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> c >> q >> t;
  int n = c + q + t;
  int id = 0;
  for (int i = 0; i < c; i++)
  {
    int x, y, r;
    cin >> x >> y >> r;
    building b;
    b.v.pb({x, y});
    b.r = r;
    b.type = 0;
    v.pb(b);
    vc.pb(id);
    id++;
  }
  for (int i = 0; i < q; i++)
  {
    building b;
    b.type = 1;
    for (int j = 0; j < 2; j++)
    {
```

```
    int x, y;
    cin >> x >> y;
    b.v.pb({x, y});
  }
  b.find_vertices();
  v.pb(b);
  vq.pb(id);
  id++;
}
for (int i = 0; i < t; i++)
{
  building b;
  b.type = 2;
  for (int j = 0; j < 3; j++)
  {
    int x, y;
    cin >> x >> y;
    b.v.pb({x, y});
  }
  v.pb(b);
  vt.pb(id);
  id++;
}
vector<vector<pair<double, int>>> adj(n + 2);
double ans = DBL_MAX;
for (auto const &i : vq)
{
  for (auto const &j : vt)
  {
    ans = min(ans, dist_square_tri(i, j));
  }
}
for (auto const &i : vq)
{
  for (auto const &j : vc)
  {
    double curr = dist_square_circ(i, j);
    adj[i].pb({curr, j});
    adj[j].pb({curr, i});
  }
}
for (auto const &i : vt)
{
  for (auto const &j : vc)
  {
    double curr = dist_tri_circ(i, j);
    adj[i].pb({curr, j});
    adj[j].pb({curr, i});
  }
}
for (auto const &i : vc)
{
  for (auto const &j : vc)
  {
    double curr = dist_circ_circ(i, j);
    adj[i].pb({curr, j});
    adj[j].pb({curr, i});
  }
}
int src = n, sink = n + 1;
for (auto const &i : vt)
{
  adj[src].pb({0, i});
}
for (auto const &i : vq)
{
  adj[i].pb({0, sink});
}
vector<double> dist(n + 2, 1e18);
vector<bool> vis(n + 2, 0);
dist[src] = 0;
priority_queue<pair<double, int>, vector<pair<double, int>>, greater<pair<
    double, int>>> pq;
pq.push({dist[src], src});
while (!pq.empty())
{
  int x = pq.top().sec;
  pq.pop();
  if (vis[x])
    continue;
  vis[x] = 1;
  for (auto [d, y] : adj[x])
  {
    if (dist[y] > dist[x] + d)
    {
      dist[y] = dist[x] + d;
      pq.push({dist[y], y});
    }
  }
}
ans = min(ans, dist[sink]);
cout << fixed << setprecision(15) << ans << endl;
return 0;
}
// solution for: https://codeforces.com/gym/104603/problem/I
```

3.11 polygon area

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

double area(vector<pi> fig)
{
  double res = 0;
  for (unsigned i = 0; i < fig.size(); i++)
  {
    pi p = i ? fig[i - 1] : fig.back();
    pi q = fig[i];
    res += (p.fir - q.fir) * (p.sec + q.sec);
  }
  return fabs(res) / 2;
}
int cross(pi a, pi b)
{
  return a.fir * b.sec - a.sec * b.fir;
}
double area2(vector<pi> fig)
{
  double res = 0;
  for (unsigned i = 0; i < fig.size(); i++)
  {
    pi p = i ? fig[i - 1] : fig.back();
    pi q = fig[i];
    res += cross(p, q);
  }
  return fabs(res) / 2;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// achar area de um poligono
// tomar cuiddado com a ordem
// percorrer os vertices em sentido horario ou anti-horario
```

3.12 polygon isomorfism

```
#include <bits/stdc++.h>
using namespace std;
```

```
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 4007
#define mod 998244353

struct pt
{
  int x, y;
  pt operator+(pt p) { return {x + p.x, y + p.y}; }
  pt operator-(pt p) { return {x - p.x, y - p.y}; }
  bool operator==(pt p) { return (x == p.x && y == p.y); }
  int cross(pt p) { return x * p.y - y * p.x; }
  int cross(pt a, pt b) { return (a - *this).cross(b - *this); }
  int dot(pt p) { return x * p.x + y * p.y; }
};
bool cmp_x(pt a, pt b)
{
  if (a.x != b.x)
    return a.x < b.x;
  return a.y < b.y;
}
vector<pt> convex_hull(vector<pt> pts)
{
  if (pts.size() <= 1)
    return pts;
  sort(pts.begin(), pts.end(), cmp_x);
  vector<pt> h(pts.size() + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(pts.begin(), pts.end()))
  {
    for (auto const &p : pts)
    {
      while (t >= s + 2 && h[t - 2].cross(h[t - 1], p) <= 0)
        t--;
      h[t++] = p;
    }
  }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
int max_suffix(vector<pair<long long, long double>> s, bool mi = false)
{
  s.push_back(*min_element(s.begin(), s.end()));
  s.back().first -= 1;
  s.back().second -= 1;
  int ans = 0;
  for (int i = 1; i < s.size(); i++)
  {
    int j = 0;
    while (ans + j < i and s[i + j] == s[ans + j])
      j++;
    if (s[i + j] > s[ans + j])
    {
      if (!mi or i != s.size() - 2)
        ans = i;
    }
    else if (j)
      i += j - 1;
  }
  return ans;
}
vector<pair<long long, long double>> max_cyclic_shift(vector<pair<long long,
     long double>> s)
{
  int n = s.size();
  for (int i = 0; i < n; i++)
    s.pb(s[i]);
  int id = max_suffix(s);
  vector<pair<long long, long double>> ans;
  for (int i = 0; i < n; i++)
  {
    ans.pb(s[id]);
    id = (id + 1) % n;
  }
  return ans;
}
int sqr(int x)
{
  return x * x;
}
int dd(pt a, pt b)
{
  return sqr(a.x - b.x) + sqr(a.y - b.y);
}
long long dot(pt a, pt b)
{
  return a.x * b.x + a.y * b.y;
}
vector<pair<long long, long double>> get_sides_and_dots(vector<pt> v)
{
  int n = (int)v.size();
  vector<pair<long long, long double>> ans;
  for (int i = 0; i < n; i++)
  {
    pt prv = v[i ? i - 1 : n - 1];
    pt nxt = v[i + 1 < n ? i + 1 : 0];
    long long dist = dd(v[i], v[(i + 1) % n]);
    long double angle = dot(prv - v[i], nxt - v[i]);
    ans.emplace_back(dist, angle);
  }
  return ans;
}
signed main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);
  int n, m;
  cin >> n >> m;
  vector<pt> a(n);
  for (int i = 0; i < n; i++)
  {
    cin >> a[i].x >> a[i].y;
  }
  auto cha = convex_hull(a);
  auto distsA = get_sides_and_dots(cha);
  vector<pt> b(m);
  for (int i = 0; i < m; i++)
  {
    cin >> b[i].x >> b[i].y;
  }
  auto chb = convex_hull(b);
  auto distsB = get_sides_and_dots(chb);
  vector<pair<long long, long double>> aa = max_cyclic_shift(distsA);
  vector<pair<long long, long double>> bb = max_cyclic_shift(distsB);
  (aa == bb) ? cout << "YES\n" : cout << "NO\n";
  return 0;
}
// https://codeforces.com/problemset/problem/1017/E
// dados dois conjuntos de pontos
// achar o convex hull de cada conjunto
// e em seguida ver se os poligonos sao isomorfos

// podemos checar olhando para o comprimento de cada aresta e o dot product
```

3.13 smallest enclosing circle

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
//#define pi pair<double, double>
```

```cpp
#define double long double
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

struct pt
{
  double x, y;
  pt operator+(pt p) { return {x + p.x, y + p.y}; } // soma de pontos
  pt operator-(pt p) { return {x - p.x, y - p.y}; } // subtracao de pontos
  pt operator*(double d) { return {x * d, y * d}; } // multiplicacao por um
       double
  pt operator/(double d) { return {x / d, y / d}; } // divisao por um double
};
struct circle
{
  pt c;
  double r;
};

bool inside(circle c, pt p)
{
  double dist = (c.c.x - p.x) * (c.c.x - p.x) + (c.c.y - p.y) * (c.c.y - p.y);
  return dist <= c.r;
}
circle get_circle(pt a, pt b)
{
  pt c = {(a.x + b.x) / 2.0, (a.y + b.y) / 2.0};
  double dist = sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
  dist /= 2.0;
  dist *= dist;
  return {c, dist};
}
pt get_center(pt b, pt c)
{
  double bb = b.x * b.x + b.y * b.y;
  double cc = c.x * c.x + c.y * c.y;
  double dd = b.x * c.y - b.y * c.x;
  return {(c.y * bb - b.y * cc) / (2 * dd), (b.x * cc - c.x * bb) / (2 * dd)};
}
circle get_circle(pt a, pt b, pt c)
{
  b = b - a;
  c = c - a;
  pt p = get_center(b, c);
  p = p + a;
  double dist = (a.x - p.x) * (a.x - p.x) + (a.y - p.y) * (a.y - p.y);
  return {p, dist};
}
circle solve2(vector<pt> &v)
{
  if (v.empty())
    return {{0, 0}, 0};
  if (v.size() == 1)
    return {v[0], 0};
  if (v.size() == 2)
    return get_circle(v[0], v[1]);
  for (int i = 0; i < 3; i++)
  {
    for (int j = i + 1; j < 3; j++)
    {
      circle c = get_circle(v[i], v[j]);
      bool ok = 1;
      for (auto const &k : v)
        ok &= inside(c, k);
      if (ok)
        return c;
    }
  }
  return get_circle(v[0], v[1], v[2]);
}
circle solve(vector<pt> &v, vector<pt> r, int n)
{
  if (n == 0 || r.size() == 3)
    return solve2(r);
  int idx = rand() % n;
  pt p = v[idx];
  swap(v[idx], v[n - 1]);
  circle c = solve(v, r, n - 1);
  if (inside(c, p))
    return c;
  r.pb(p);
  return solve(v, r, n - 1);
}
circle welzl(vector<pt> v)
{
  random_shuffle(v.begin(), v.end());
  return solve(v, {}, v.size());
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  srand(time(NULL));
  int n;
  cin >> n;
  vector<pt> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i].x >> v[i].y;
  circle ans = welzl(v);
  cout << fixed << setprecision(3) << ans.c.x << " " << ans.c.y << endl;
  cout << fixed << setprecision(3) << sqrt(ans.r) << endl;
  return 0;
}
// acmicpc.net/problem/2626
// achar uma circuferencia
// minimizando o raio
// que cobre todos os pontos dela
// ai oq tem q printar eh o centro dessa circuferencia e o raio
// Minimum enclosing circle
// Welzl's algorithm
// complexidade O(n)
```

4 Graph

4.1 articulation points

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 400005
#define mod 1000000007

int n, m, timer;
vector<int> adj[MAXN];
bool is_cutpoint[MAXN];
int tin[MAXN];
int low[MAXN];
bool vis[MAXN];

void dfs(int v, int p)
{
  vis[v] = true;
  tin[v] = timer, low[v] = timer++;
  int childs = 0;
  for (auto const &u : adj[v])
```

```
  {
    if (u == p)
      continue;
    if (vis[u])
    {
      low[v] = min(low[v], tin[u]);
    }
    else
    {
      dfs(u, v);
      low[v] = min(low[v], low[u]);
      if (low[u] >= tin[v] && p != -1)
        is_cutpoint[v] = true;
      childs++;
    }
  }
  if (p == -1 && childs > 1)
    is_cutpoint[v] = true;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  memset(tin, -1, sizeof(tin));
  memset(low, -1, sizeof(low));
  for (int i = 0; i < n; i++)
  {
    if (!vis[i])
      dfs(i, -1);
  }
  return 0;
}
```

4.2 BFS

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 1
#define MAXN 1001
#define mod 1000000007

int n, m;
vector<int> adj[MAXN];
bool visited[MAXN];

void bfs(int s)
{
  queue<int> q;
  q.push(s);
  while (!q.empty())
  {
    int v = q.front();
    q.pop();
    if (visited[v])
      continue;
    visited[v] = true;
    for (auto const &u : adj[v])
      if (!visited[u])
        q.push(u);
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b, c;
    cin >> a >> b >> c;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  bfs(0);
}
```

4.3 bipartite

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

int n, m;
vector<int> adj[MAXN];

bool is()
{
  vector<int> c(n, -1);
  bool is = 1;
  queue<int> q;
  for (int st = 0; st < n; st++)
  {
    if (c[st] == -1)
    {
      q.push(st);
      c[st] = 0;
      while (!q.empty())
      {
        int v = q.front();
        q.pop();
        for (int u : adj[v])
        {
          if (c[u] == -1)
          {
            c[u] = c[v] ^ 1;
            q.push(u);
          }
          else
          {
            is &= (c[u] != c[v]);
          }
```

```
        }
      }
    }
  }
  return is;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  cout << is() << endl;
  return 0;
}
```

4.4 block-cut-tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

struct dsu
{
  vector<pi> parent;
  vector<int> rank;
  vector<int> bipartite;

  void reset(int v)
  {
    parent[v] = {v, 0};
    rank[v] = 0;
    bipartite[v] = 1;
  }
  dsu(int n)
  {
    parent.resize(n);
    rank.resize(n);
    bipartite.resize(n);
    for (int v = 0; v < n; v++)
      reset(v);
  }
  dsu() {}
  pi find_set(int v)
  {
    if (v != parent[v].fir)
    {
      int parity = parent[v].sec;
      parent[v] = find_set(parent[v].fir);
      parent[v].sec ^= parity;
    }
    return parent[v];
  }
  void add_edge(int a, int b)
  {
    pi pa = find_set(a);
    a = pa.fir;
    int x = pa.sec;
    pi pb = find_set(b);
    b = pb.fir;
    int y = pb.sec;
    if (a == b)
    {
      if (x == y)
        bipartite[a] = 0;
    }
    else
    {
      if (rank[a] < rank[b])
        swap(a, b);
      parent[b] = {a, x ^ y ^ 1};
      bipartite[a] &= bipartite[b];
      if (rank[a] == rank[b])
        rank[a]++;
    }
  }
  bool is_bipartite(int v)
  {
    return bipartite[find_set(v).fir];
  }
};
struct block_cut_tree
{
  // Source: https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/
      Grafos/blockCutTree.cpp
  // Cria a block-cut tree, uma arvore com os blocos
  // e os pontos de articulacao
  // Blocos sao componentes 2-vertice-conexos maximais
  // Uma 2-coloracao da arvore eh tal que uma cor sao
  // os blocos, e a outra cor sao os pontos de art.
  // Funciona para grafo nao conexo
  //
  // art[i] responde o numero de novas componentes conexas
  // criadas apos a remocao de i do grafo g
  // Se art[i] >= 1, i eh ponto de articulacao
  //
  // Para todo i <= blocks.size()
  // blocks[i] eh uma componente 2-vertce-conexa maximal
  // edgblocks[i] sao as arestas do bloco i
  // tree[i] eh um vertice da arvore que corresponde ao bloco i
  // tree - eh a propia block-cut tree
  // pos[i] responde a qual vertice da arvore vertice i pertence
  // Arvore tem no maximo 2n vertices
  //
  // O(n + m)
  vector<vector<int>> g, blocks, tree;
  vector<vector<pi>> edgblocks;
  stack<int> s;
  stack<pi> s2;
  vector<int> id, art, pos;

  block_cut_tree(vector<vector<int>> g_) : g(g_)
  {
    int n = g.size();
    id.resize(n, -1), art.resize(n), pos.resize(n);
    build();
  }
  int dfs(int i, int &t, int p = -1)
  {
    int lo = id[i] = t++;
    s.push(i);
    if (p != -1)
    {
      s2.emplace(i, p);
    }
    for (int j : g[i])
    {
      if (j != p and id[j] != -1)
        s2.emplace(i, j);
    }
    for (int j : g[i])
    {
```

```
      if (j != p)
      {
        if (id[j] == -1)
        {
          int val = dfs(j, t, i);
          lo = min(lo, val);
          if (val >= id[i])
          {
            art[i]++;
            blocks.emplace_back(1, i);
            while (blocks.back().back() != j)
            {
              blocks.back().pb(s.top());
              s.pop();
            }
            edgblocks.emplace_back(1, s2.top());
            s2.pop();
            pi aux = {j, i};
            while (edgblocks.back().back() != aux)
            {
              edgblocks.back().pb(s2.top());
              s2.pop();
            }
          }
          // if (val > id[i]) aresta i-j eh ponte
        }
        else
        {
          lo = min(lo, id[j]);
        }
      }
    }
    if (p == -1 and art[i])
    {
      art[i]--;
    }
    return lo;
  }
  void build()
  {
    int t = 0;
    for (int i = 0; i < g.size(); i++)
    {
      if (id[i] == -1)
        dfs(i, t, -1);
    }
    tree.resize(blocks.size());
    for (int i = 0; i < g.size(); i++)
    {
      if (art[i])
        pos[i] = tree.size(), tree.emplace_back();
    }
    for (int i = 0; i < blocks.size(); i++)
    {
      for (int j : blocks[i])
      {
        if (!art[j])
          pos[j] = i;
        else
          tree[i].pb(pos[j]), tree[pos[j]].pb(i);
      }
    }
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  vector<vector<int>> adj(n);
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  block_cut_tree bt(adj);
  vector<vector<int>> g(n);
  dsu d(n);
  for (auto const &v : bt.edgblocks)
  {
    vector<int> guys;
    for (auto const &j : v)
    {
      guys.pb(j.fir);
      guys.pb(j.sec);
      d.add_edge(j.fir, j.sec);
    }
    bool bip = 1;
    for (auto const &j : guys)
    {
      bip &= d.is_bipartite(j);
    }
    if (bip)
    {
      for (auto const &j : v)
      {
        g[j.fir].pb(j.sec);
        g[j.sec].pb(j.fir);
      }
    }
    for (auto const &j : guys)
    {
      d.reset(j);
    }
  }
  vector<bool> vis(n, 0);
  vector<bool> c(n, 0);
  int a = 0, b = 0;
  for (int i = 0; i < n; i++)
  {
    if (vis[i])
      continue;
    int x = 1, y = 0;
    queue<int> q;
    q.push(i);
    vis[i] = 1;
    while (!q.empty())
    {
      int k = q.front();
      q.pop();
      for (auto const &i : g[k])
      {
        if (!vis[i])
        {
          vis[i] = 1;
          c[i] = c[k] ^ 1;
          (c[i] == 1) ? y++ : x++;
          q.push(i);
        }
      }
    }
    a += (x * (x - 1)) / 2;
    a += (y * (y - 1)) / 2;
    b += (x * y);
  }
  cout << a << " " << b << endl;
  return 0;
}
// https://codeforces.com/gym/103934/problem/M
// pares (a, b) com a < b
// contar pares (a, b) tal que todos os caminhos de a para b possuem distancia
    impar
// contar pares (a, b) tal que todos os caminhos de a para b possuem distancia
    par

// grafo biconexo (ou 2-vertice-conexo) - nao tem ponto de articulacao
// blocos - sao subgrafos biconexos maximais (sem ponto de articulacao)

// block graph
// grafo que tem um vertice para cada bloco do grafo G
// e uma aresta entre dois vertices tal que os blocos correspondentes tem um
    vertice em comum

// block-cut tree
```

```
// um ponto de articulacao eh um vertice que esta em dois ou mais blocos
// a estrutura dos blocos e dos pontos de articulacao de um grafo conectado pode
      ser descrita por uma arvore chamada de arvore de block-cut tree
// essa arvore tem um vertice para cada bloco e para cada ponto de articulacao
      do grafo dado.
// tem uma aresta na block-cut tree para cada par (bloco, ponto de articulacao),
      tal que esse ponto de articulacao ta no bloco

// para o problema:
// para um grafo nao bipartido que e biconexo, tem caminhos de tamanho impar e
      par entre qualquer par de vertices

// um caminho em um grafo G, tem meio que um caminho equivalente na sua block-
      cut tree
// da pra pensar em resolver para cada bloco

// resolvendo pra cada bloco:
// o bloco tem que ser bipartido
// quando o bloco nao eh bipartido, eu nao considero as arestas dele

// considerando o grafo restante sendo bipartido
// da pra resolver pra cada componente conexa
// caminhos entre vertices de mesma cor tem paridade impar
// caminhos entre vertices de cor diferente tem paridade par

// https://codeforces.com/gym/102512/problem/A
// ter queries do tipo
// quantos pontos de articulacao desconectam u e v
// dai monta a block cut tree
// para cada ponto de articulacao, seta a pos[i] dele como 1 na arvore
// e o valor dos demais vertices como 0
// dai responde uma query com hld (ou com lca tbm sai)
```

4.5 bridges

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 400005
#define mod 1000000007

int n, m, timer;
vector<pi> edges;
vector<bool> is_bridge;
vector<pi> adj[MAXN];
int tin[MAXN];
int low[MAXN];
bool vis[MAXN];

void dfs(int v, int p)
{
  vis[v] = true;
  tin[v] = timer, low[v] = timer++;
  for (auto const &u : adj[v])
  {
    if (u.fir == p)
      continue;
    if (vis[u.fir])
    {
      low[v] = min(low[v], tin[u.fir]);
      continue;
    }
    dfs(u.fir, v);
    low[v] = min(low[v], low[u.fir]);
    if (low[u.fir] > tin[v])
      is_bridge[u.sec] = 1;
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  is_bridge.resize(m);
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    edges.pb({a, b});
    adj[a].pb({b, i});
    adj[b].pb({a, i});
  }
  memset(tin, -1, sizeof(tin));
  memset(low, -1, sizeof(low));
  for (int i = 0; i < n; i++)
  {
    if (!vis[i])
      dfs(i, -1);
  }
  return 0;
}
```

4.6 caminhoeuleriano

```
// caminho euleriano em um grafo
// passa por todas as arestas apenas uma unica vez e percorre todas elas
// condicao de existencia:
// todos os vertices possuem grau par (ciclo euleriano) comeca e acaba no mesmo
     vertice
// ou
// apenas 2 vertices possuem grau impar, todos os outros possuem grau par ou ==
     0.
// comeca num vertice de grau impar e termina num vertice de grau impar nesse
     caso.
// solucao:
// rodar um dfs com map de visited para as arestas
// no final por o source no vector path
// ao final teremos o caminho inverso no vector path
// note que o caminho inverso tambem e um caminho valido

#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pd pair<double, int>
#define pib pair<pi, bool>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 10001
#define MAXL 1000001
#define mod 1000000007

int n, m, start;
vector<int> path;
vector<int> adj[MAXN];
map<pi, bool> visited;

void dfs(int s)
{
  for (int i = 0; i < adj[s].size(); i++)
  {
    int v = adj[s][i];
    if (!visited[mp(s, v)])
```

```
    {
      visited[mp(s, v)] = true;
      visited[mp(v, s)] = true;
      dfs(v);
    }
  }
  path.pb(s);
}
bool check()
{
  int odd = 0;
  for (int i = 0; i < n; i++)
    if (adj[i].size() & 1)
      odd++, start = i;
  return (odd == 0 || odd == 2);
}
signed main()
{
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  start = 0;
  bool ok = check();
  (ok) ? cout << "Yes\n" : cout << "No\n";
  if (ok)
  {
    dfs(start);
    for (int i = 0; i < path.size(); i++)
      cout << path[i] << " ";
    cout << "\n";
  }
  return 0;
}
```

4.7 caminhoeuleriano2

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 998244353

int deg[MAXN];
bool vis[MAXN];
vector<int> path;
vector<pi> edges, edges2;
vector<pi> ans;
vector<pi> adj[MAXN];
vector<pi> a[MAXN];

void dfs2(int s)
{
  while (a[s].size() > 0)
  {
    auto v = a[s].back();
    a[s].pop_back();
    if (!vis[v.sec])
    {
      vis[v.sec] = 1;
      dfs2(v.fir);
    }
  }
  path.pb(s);
}
void dfs(int i)
{
  vis[i] = 1;
  for (auto const &j : adj[i])
  {
    if (!vis[j.fir])
    {
      dfs(j.fir);
      if (deg[j.fir])
      {
        ans.pb(edges[j.sec]);
        deg[j.fir] ^= 1;
        deg[i] ^= 1;
      }
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n, m;
    cin >> n >> m;
    edges.clear();
    edges2.clear();
    ans.clear();
    for (int i = 0; i < n; i++)
    {
      adj[i].clear();
      vis[i] = 0;
      deg[i] = 0;
    }
    for (int i = 0; i < m; i++)
    {
      int a, b, c;
      cin >> a >> b >> c;
      a--, b--;
      if (c == 1) // edges obrigatorios
      {
        edges2.pb({a, b});
        deg[a] ^= 1;
        deg[b] ^= 1;
      }
      else // edges nao obrigatorios
      {
        edges.pb({a, b});
        adj[a].pb({b, edges.size() - 1});
        adj[b].pb({a, edges.size() - 1});
      }
    }
    // ajeita pra ver se consegue fazer todo mundo ficar com grau par
    // considerando so os edges nao obrigatorios
    for (int i = 0; i < n; i++)
    {
      if (!vis[i])
        dfs(i);
    }
    bool ok = 1;
    for (int i = 0; i < n; i++)
    {
      if (deg[i])
        ok = 0;
    }
    if (!ok)
    {
      cout << "NO\n";
      continue;
    }
```

```
    for (int i = 0; i < n; i++)
    {
      a[i].clear();
    }
    // se ajeitei, agora dale
    // monta o grafo final e acha o ciclo euleriano
    // funciona para grafos com self loops e multiple edges
    int id = 0;
    for (auto [u, v] : ans)
    {
      a[u].pb({v, id});
      a[v].pb({u, id});
      id++;
    }
    for (auto [u, v] : edges2)
    {
      a[u].pb({v, id});
      a[v].pb({u, id});
      id++;
    }
    for (int i = 0; i < m; i++)
      vis[i] = 0;
    path.clear();
    dfs2(0);
    cout << "YES\n";
    cout << path.size() - 1 << endl;
    for (int i = 0; i < path.size(); i++)
      cout << path[i] + 1 << " ";
    cout << endl;
  }
  return 0;
}
// https://codeforces.com/contest/1994/problem/F
// dado um grafo, tem edges que sao obrigatorios de manter
// e outros q posso remover
// quero fazer com que um grafo tenha um ciclo euleriano
// no qual o grau de cada vertice eh par
// se tiver solucao, eu quero imprimir o ciclo euleriano
```

4.8 centroid decomposition

```
// centroid de uma arvore -> e um no que ao ser removido da arvore, separaria as
// arvores resultantes de modo com que a maior arvore desse conjunto teria no
    maximo
// (n / 2) nos, sendo n o numero de nos da arvore. Para qualquer arvore com n
    nos,
// o centroid sempre existe.

//
    ////////////////////////////////////////////////////////////////////////////////

// centroid decomposition -> muito util para tentar diminuir a complexidade em
    certos
// tipos de consultas a serem feitas, uma maneira melhor de organizar a arvore.

// algoritimo:
// 1) o centroid e a raiz dessa nova arvore
// 2) achar o centroid das arvores menores que surgiram com a remocao do
    centroid "pai"
// 3) por uma aresta entre o centroid "filho" e o centroid "pai"
// 4) repetir isso ate todos os nos serem removidos
// 5) ao final teremos a centroid tree

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 100001
#define mod 1000000007

int n;
vector<int> adj[MAXN];

namespace cd
{
  int sz;
  vector<int> adjl[MAXN];
  vector<int> father, subtree_size;
  vector<bool> visited;

  void dfs(int s, int f)
  {
    sz++;
    subtree_size[s] = 1;
    for (auto const &v : adj[s])
    {
      if (v != f && !visited[v])
      {
        dfs(v, s);
        subtree_size[s] += subtree_size[v];
      }
    }
  }
  int getCentroid(int s, int f)
  {
    bool is_centroid = true;
    int heaviest_child = -1;
    for (auto const &v : adj[s])
    {
      if (v != f && !visited[v])
      {
        if (subtree_size[v] > sz / 2)
          is_centroid = false;
        if (heaviest_child == -1 || subtree_size[v] > subtree_size[
            heaviest_child])
          heaviest_child = v;
      }
    }
    return (is_centroid && sz - subtree_size[s] <= sz / 2) ? s : getCentroid(
        heaviest_child, s);
  }
  int decompose_tree(int s)
  {
    sz = 0;
    dfs(s, s);
    int cend_tree = getCentroid(s, s);
    visited[cend_tree] = true;
    for (auto const &v : adj[cend_tree])
    {
      if (!visited[v])
      {
        int cend_subtree = decompose_tree(v);
        adjl[cend_tree].pb(cend_subtree);
        adjl[cend_subtree].pb(cend_tree);
        father[cend_subtree] = cend_tree;
      }
    }
    return cend_tree;
  }
  void init()
  {
    subtree_size.resize(n);
    visited.resize(n);
    father.assign(n, -1);
    decompose_tree(0);
  }
}
signed main()
```

```
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n;
  for (int i = 0; i < n - 1; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  cd::init();
  return 0;
}
```

4.9 centroid decomposition2

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 998244353

int n, k, resp;
vector<int> adj[MAXN];
vector<int> cnt;

namespace cd
{
  int sz;
  vector<int> subtree_size;
  vector<bool> visited;

  void dfs(int s, int f)
  {
    sz++;
    subtree_size[s] = 1;
    for (auto const &v : adj[s])
    {
      if (v != f && !visited[v])
      {
        dfs(v, s);
        subtree_size[s] += subtree_size[v];
      }
    }
  }
  int get_centroid(int s, int f)
  {
    bool is_centroid = true;
    int heaviest_child = -1;
    for (auto const &v : adj[s])
    {
      if (v != f && !visited[v])
      {
        if (subtree_size[v] > sz / 2)
          is_centroid = false;
        if (heaviest_child == -1 || subtree_size[v] > subtree_size[
            heaviest_child])
          heaviest_child = v;
      }
    }
    return (is_centroid && sz - subtree_size[s] <= sz / 2) ? s : get_centroid(
        heaviest_child, s);
  }
  void dfs2(int s, int f, int d)
  {
    while (d >= cnt.size())
      cnt.pb(0);
    cnt[d]++;
    for (auto const &v : adj[s])
    {
      if (v != f && !visited[v])
        dfs2(v, s, d + 1);
    }
  }
  void solve(int s)
  {
    vector<int> tot;
    for (auto const &v : adj[s])
    {
      if (visited[v])
        continue;
      cnt.clear();
      dfs2(v, s, 1);
      for (int i = 1; i < cnt.size(); i++)
      {
        if (k - i < tot.size() && k - i >= 1)
          resp += (cnt[i] * tot[k - i]);
      }
      for (int i = 1; i < cnt.size(); i++)
      {
        while (i >= tot.size())
          tot.pb(0);
        tot[i] += cnt[i];
      }
    }
    if (k < tot.size())
      resp += tot[k];
  }
  int decompose_tree(int s)
  {
    sz = 0;
    dfs(s, s);
    int cend_tree = get_centroid(s, s);
    visited[cend_tree] = true;
    solve(cend_tree);
    for (auto const &v : adj[cend_tree])
    {
      if (!visited[v])
        decompose_tree(v);
    }
    return cend_tree;
  }
  void init()
  {
    subtree_size.resize(n);
    visited.resize(n);
    decompose_tree(0);
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> k;
  for (int i = 1; i < n; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  cd::init();
  cout << resp << endl;
  return 0;
}
// https://codeforces.com/contest/161/problem/D
// problema: contar quantos pares de vertices (u, v) existem tal que dist(u, v)
    = k
// durante a decomposicao
// pega o centroid atual e resolve o problema pra ele
// isso eh:
// para cada centroid que eu achei, devo contar quantos caminhos
// de tamanho k passam por esse centroid
// somando todas essas respostas, a gente tem a resposta final
```

4.10 cycle detection

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 205
#define MAXP 100001
#define mod 1000000007

int n, m, idx;
vector<int> cycles[MAXN];
vector<int> adj[MAXN];
int color[MAXN];
int parent[MAXN];
int ans[MAXN];

void dfs(int u, int p)
{
  if (color[u] == 2)
    return;
  if (color[u] == 1)
  {
    idx++;
    int curr = p;
    ans[curr] = idx;
    cycles[idx].pb(curr);
    while (curr != u)
    {
      curr = parent[curr];
      cycles[idx].pb(curr);
      ans[curr] = idx;
    }
    return;
  }
  parent[u] = p;
  color[u] = 1;
  for (auto const &v : adj[u])
    if (v != parent[u])
      dfs(v, u);
  color[u] = 2;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  for (int i = 0; i < n; i++)
    if (!color[i])
      dfs(i, -1);
  cout << idx << endl;
  for (int i = 1; i <= idx; i++)
  {
    cout << cycles[i].size() << endl;
    for (auto const &j : cycles[i])
      cout << j + 1 << " ";
    cout << endl;
  }
  return 0;
}
```

4.11 DFS

```cpp
#include <bits/stdc++.h>
using namespace std;

#define MAXN 500000

int n , m ;
int visited [MAXN] ;
vector <int> adj_list [MAXN] ;

void dfs (int x)
{
    for (int i = 0 ; i < adj_list[x].size() ; i++)
    {
        int v = adj_list[x][i] ;

        if(visited[v] == -1)
        {
            visited[v] = visited[x] ;
            dfs(v) ;
        }
    }
}
void initialize ()
{
    for (int i = 1 ; i <= n ; i++)
    {
        visited[i] = -1 ;
    }
}
int main ()
{
    int a , b ;

    cin >> n >> m ;

    initialize();

    for (int i = 1 ; i <= m ; i++)
    {
        cin >> a >> b ;

        adj_list[a].push_back(b) ;
        adj_list[b].push_back(a) ;
    }

    dfs(1) ;

    return 0;
}
```

4.12 Dijkstra

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
```

```
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 1
#define MAXN 1001
#define mod 1000000007

int n, m;
vector<pi> adj[MAXN];
bool visited[MAXN];
int dist[MAXN];

void dijkstra(int s)
{
  for (int i = 0; i < n; i++)
  {
    dist[i] = INT_MAX;
    visited[i] = false;
  }
  priority_queue<pi, vector<pi>, greater<pi>> q;
  dist[s] = 0;
  q.push({dist[s], s});
  while (!q.empty())
  {
    int v = q.top().second;
    q.pop();
    if (visited[v])
      continue;
    visited[v] = true;
    for (auto const &u : adj[v])
    {
      if (dist[u.sec] > dist[v] + u.fir)
      {
        dist[u.sec] = dist[v] + u.fir;
        q.push({dist[u.sec], u.sec});
      }
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b, c;
    cin >> a >> b >> c;
    a--, b--;
    adj[a].pb({c, b});
    adj[b].pb({c, a});
  }
  dijkstra(0);
}
```

4.13 dinic

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 705
#define mod 1000000007
#define INF 1e9

struct edge
{
  int to, from, flow, capacity, id;
};
struct dinic
{
  int n, src, sink;
  vector<vector<edge>> adj;
  vector<int> level;
  vector<int> ptr;

  dinic(int sz)
  {
    n = sz;
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
  }
  void add_edge(int a, int b, int c, int id)
  {
    adj[a].pb({b, (int)adj[b].size(), c, c, id});
    adj[b].pb({a, (int)adj[a].size() - 1, 0, 0, id});
  }
  bool bfs()
  {
    level.assign(n, -1);
    level[src] = 0;
    queue<int> q;
    q.push(src);
    while (!q.empty())
    {
      int u = q.front();
      q.pop();
      for (auto at : adj[u])
      {
        if (at.flow && level[at.to] == -1)
        {
          q.push(at.to);
          level[at.to] = level[u] + 1;
        }
      }
    }
    return level[sink] != -1;
  }
  int dfs(int u, int flow)
  {
    if (u == sink || flow == 0)
      return flow;
    for (int &p = ptr[u]; p < adj[u].size(); p++)
    {
      edge &at = adj[u][p];
      if (at.flow && level[u] == level[at.to] - 1)
      {
        int kappa = dfs(at.to, min(flow, at.flow));
        at.flow -= kappa;
        adj[at.to][at.from].flow += kappa;
        if (kappa != 0)
          return kappa;
      }
    }
    return 0;
  }
  int run()
  {
    int max_flow = 0;
    while (bfs())
    {
      ptr.assign(n, 0);
      while (1)
      {
        int flow = dfs(src, INF);
        if (flow == 0)
          break;
        max_flow += flow;
      }
```

```
    }
    return max_flow;
  }
  vector<pii> cut_edges() // arestas do corte minimo
  {
    bfs();
    vector<pii> ans;
    for (int i = 0; i < n; i++)
    {
      for (auto const &j : adj[i])
      {
        if (level[i] != -1 && level[j.to] == -1 && j.capacity > 0)
          ans.pb({j.capacity, {i, j.to}});
      }
    }
    return ans;
  }
  vector<int> flow_edges(int n, int m) // fluxo em cada aresta, na ordem da
      entrada
  {
    vector<int> ans(m);
    for (int i = 0; i < n; i++)
    {
      for (auto const &j : adj[i])
        if (!j.capacity)
          ans[j.id] = j.flow;
    }
    return ans;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  dinic d(n);
  for (int i = 0; i < m; i++)
  {
    int a, b, c;
    cin >> a >> b >> c;
    a--, b--;
    d.add_edge(a, b, c, i);
  }
  d.src = 0, d.sink = n - 1;
  cout << d.run() << endl;
  vector<int> ans = d.flow_edges(n, m);
  for (auto const &i : ans)
    cout << i << endl;
  return 0;
}
```

4.14 dominator tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

struct dominator_tree
{
  int n, t;
  vector<vector<int>> g, tree, rg, bucket;
  vector<int> dfs_l, dfs_r, idom, sdom, prv, pre, ancestor, label, preorder;

  dominator_tree(vector<vector<int>> &adj, int source)
  {
    int n = adj.size();
    g = adj;
    tree.resize(n);
    rg.resize(n);
    bucket.resize(n);
    dfs_l.resize(n);
    dfs_r.resize(n);
    idom.resize(n);
    sdom.assign(n, -1);
    prv.resize(n);
    pre.assign(n, -1);
    ancestor.assign(n, -1);
    label.resize(n);
    build(source);
  }
  void dfs(int v)
  {
    pre[v] = ++t;
    sdom[v] = label[v] = v;
    preorder.pb(v);
    for (auto const &nxt : g[v])
    {
      if (sdom[nxt] == -1)
      {
        prv[nxt] = v;
        dfs(nxt);
      }
      rg[nxt].pb(v);
    }
  }
  int eval(int v)
  {
    if (ancestor[v] == -1)
      return v;
    if (ancestor[ancestor[v]] == -1)
      return label[v];
    int u = eval(ancestor[v]);
    if (pre[sdom[u]] < pre[sdom[label[v]]])
      label[v] = u;
    ancestor[v] = ancestor[u];
    return label[v];
  }
  void dfs2(int v)
  {
    dfs_l[v] = t++;
    for (auto const &nxt : tree[v])
    {
      dfs2(nxt);
    }
    dfs_r[v] = t++;
  }
  void build(int s)
  {
    t = 0;
    dfs(s);
    if (preorder.size() == 1)
    {
      return;
    }
    int sz = preorder.size();
    for (int i = sz - 1; i >= 1; i--)
    {
      int w = preorder[i];
      for (auto const &v : rg[w])
      {
        int u = eval(v);
        if (pre[sdom[u]] < pre[sdom[w]])
          sdom[w] = sdom[u];
      }
      bucket[sdom[w]].push_back(w);
      ancestor[w] = prv[w];
      for (auto const &v : bucket[prv[w]])
      {
        int u = eval(v);
```

```
        idom[v] = (u == v) ? sdom[v] : u;
      }
      bucket[prv[w]].clear();
    }
    for (int i = 1; i < preorder.size(); i++)
    {
      int w = preorder[i];
      if (idom[w] != sdom[w])
        idom[w] = idom[idom[w]];
      tree[idom[w]].push_back(w);
    }
    idom[s] = sdom[s] = -1;
    t = 0;
    dfs2(s);
  }
  bool dominates(int u, int v)
  {
    if (pre[v] == -1)
      return 1;
    return dfs_l[u] <= dfs_l[v] && dfs_r[v] <= dfs_r[u];
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  vector<vector<int>> adj(n);
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
  }
  dominator_tree d(adj, 0);
  vector<int> ans;
  for (int i = 0; i < n; i++)
  {
    if (d.dominates(i, n - 1))
      ans.pb(i);
  }
  cout << ans.size() << endl;
  for (int i = 0; i < ans.size(); i++)
    cout << ans[i] + 1 << " \n"[i == ans.size() - 1];
}
// https://tanujkhattar.wordpress.com/2016/01/11/dominator-tree-of-a-directed-
    graph/
// https://cses.fi/problemset/task/1703/ (problema desse codigo)
// https://codeforces.com/gym/100513/problem/L
// https://codeforces.com/contest/757/problem/F

// dado um vertice source s
// dizemos que u domina w, se todos os caminhos de
// s ate w passam pelo vertice u

// dizemos que u e um dominador imediato de w se u domina w
// e todos os demais dominadores de w, dominam u

// 1 - todo vertice (tirando o source) tem um dominador
// pois o source domina todos os demais vertices

// 2 - todo vertice (tirando o source) tem exatamente um
// unico dominador imediato

// se eu crio um grafo com todas as arestas do tipo
// (dominador imediato de w) - w
// para todos os vertices w que nao sao a source
// esse grafo eh uma arvore
// e eh a dominator tree
```

4.15 dsu

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 2001
#define mod 1000000007

struct dsu
{
  int tot;
  vector<int> parent;
  vector<int> sz;

  dsu(int n)
  {
    parent.resize(n);
    sz.resize(n);
    tot = n;
    for (int i = 0; i < n; i++)
    {
      parent[i] = i;
      sz[i] = 1;
    }
  }
  int find_set(int i)
  {
    return parent[i] = (parent[i] == i) ? i : find_set(parent[i]);
  }
  void make_set(int x, int y)
  {
    x = find_set(x), y = find_set(y);
    if (x != y)
    {
      if (sz[x] > sz[y])
        swap(x, y);
      parent[x] = y;
      sz[y] += sz[x];
      tot--;
    }
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  dsu d(n);
  int a, b;
  cin >> a >> b;
  d.make_set(a, b);
  d.find_set(a);
}
```

4.16 dsu rollback

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

```cpp
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 600005
#define mod 1000000007

namespace dsu
{
  struct rollback
  {
    int u, v, ranku, rankv;
  };

  int num_sets;
  int parent[MAXN];
  int rank[MAXN];
  stack<rollback> op;

  int Find(int i)
  {
    return (parent[i] == i) ? i : Find(parent[i]);
  }
  bool Union(int x, int y)
  {
    int xx = Find(x);
    int yy = Find(y);
    if (xx != yy)
    {
      num_sets--;
      if (rank[xx] > rank[yy])
        swap(xx, yy);
      op.push({xx, yy, rank[xx], rank[yy]});
      parent[xx] = yy;
      if (rank[xx] == rank[yy])
        rank[yy]++;
      return true;
    }
    return false;
  }
  void do_rollback()
  {
    if (op.empty())
      return;
    rollback x = op.top();
    op.pop();
    num_sets++;
    parent[x.v] = x.v;
    rank[x.v] = x.rankv;
    parent[x.u] = x.u;
    rank[x.u] = x.ranku;
  }
  void init(int n)
  {
    for (int i = 0; i < n; i++)
    {
      parent[i] = i;
      rank[i] = 0;
    }
    num_sets = n;
  }
}
namespace seg
{
  struct query
  {
    int v, u, is_bridge;
  };

  vector<vector<query>> t(4 * MAXN);
  int ans[MAXN];

  void add(int i, int l, int r, int ql, int qr, query q)
  {
    if (l > r || l > qr || r < ql)
      return;
    if (l >= ql && r <= qr)
    {
      t[i].push_back(q);
      return;
    }
    int mid = (l + r) >> 1;
    add((i << 1), l, mid, ql, qr, q);
    add((i << 1) | 1, mid + 1, r, ql, qr, q);
  }
  void dfs(int i, int l, int r)
  {
    for (query &q : t[i])
      if (dsu::Union(q.v, q.u))
        q.is_bridge = 1;
    if (l == r)
      ans[l] = dsu::num_sets;
    else
    {
      int mid = (l + r) >> 1;
      dfs((i << 1), l, mid);
      dfs((i << 1) | 1, mid + 1, r);
    }
    for (query q : t[i])
      if (q.is_bridge)
        dsu::do_rollback();
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  int time = 0;
  map<pi, int> tin;
  vector<int> queries;
  while (q--)
  {
    char t;
    cin >> t;
    if (t == '?')
    {
      queries.pb(++time);
    }
    else if (t == '+')
    {
      int a, b;
      cin >> a >> b;
      a--, b--;
      if (a > b)
        swap(a, b);
      tin[{a, b}] = ++time;
    }
    else
    {
      int a, b;
      cin >> a >> b;
      a--, b--;
      if (a > b)
        swap(a, b);
      seg::query kappa = {a, b, 0};
      seg::add(1, 0, MAXN - 1, tin[{a, b}], ++time, kappa);
      tin[{a, b}] = -1;
    }
  }
  for (auto const &i : tin)
  {
    if (i.sec != -1)
    {
      seg::query kappa = {i.fir.fir, i.fir.sec, 0};
      seg::add(1, 0, MAXN - 1, i.sec, ++time, kappa);
    }
  }
  dsu::init(n);
  seg::dfs(1, 0, MAXN - 1);
  for (auto const &i : queries)
    cout << seg::ans[i] << endl;
```

```
  return 0;
}
// https://codeforces.com/edu/course/2/lesson/7/3/practice/contest/289392/
    problem/C
// conectividade dinamica
// para uma query (u, v)
// podemos descrever em um intervalo [l, r]
// l = quando a aresta (u, v) foi adicionada
// r = quando a aresta (u, v) foi removida
// dai agora que temos um intervalo, podemos adicionar
// a query (u, v) em uma segtree "adaptada"
// no final rodamos um dfs nessa segtree e vamos atualizando as repostas das
    queries
// quando estamos em uma posicao na seg, dou union em todos os caras daquela
    posicao
// e em seguida chamo pros meus filhos, quando chego em uma folha, ela eh
    equivalente
// a uma unidade de "tempo", logo a resposta para aquele tempo eh a resposta
    atual no dsu
// e ao sair recursivamente, vou dando rollbacks no dsu
```

4.17 erdos gallai

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define pci pair<char, int>
#define fir first
#define sec second
#define MAXN 100005
#define mod 998244353

bool erdos_gallai(vector<int> &v)
{
  int sum = 0, n = v.size();
  for (auto const &i : v)
  {
    sum += i;
  }
  if (sum % 2)
  {
    return false;
  }
  sort(v.rbegin(), v.rend());
  vector<int> suf(n + 1, 0);
  int qt = 0, ptr = n;
  sum = 0;
  for (int i = n - 1; i >= 0; i--)
  {
    if (v[i] >= i)
    {
      qt++;
    }
    else
    {
      sum += v[i];
      ptr = i;
    }
    while (ptr < n && v[ptr] >= i)
    {
      qt++;
      sum -= v[ptr];
      ptr++;
    }
    suf[i] = sum + (qt * i);
  }
  sum = 0;
  bool ok = 1;
  for (int i = 0; i < n; i++)
  {
    sum += v[i];
    int curr = i * (i + 1) + suf[i + 1];
    ok &= (sum <= curr);
  }
  return ok;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n;
    cin >> n;
    vector<int> v(n);
    for (int i = 0; i < n; i++)
      cin >> v[i];
    (erdos_gallai(v)) ? cout << "Y\n" : cout << "N\n";
  }
  return 0;
}
// https://codeforces.com/gym/101726/problem/A
// erdos gallai
// dado uma sequencia de n inteiros d[0], d[1], ..., d[n - 1]
// quero saber se existe um grafo simples e undirected com n vertices
// tal que o grau do i-esimo vertice e igual a d[i]
```

4.18 eulertour

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 1
#define MAXN 100001
#define mod 1000000009
#define d 31

int n, idx;
vector<int> adj[MAXN];
int euler[2 * MAXN];
int entrei[MAXN];
int sai[MAXN];

void euler_tour(int s, int f)
{
  euler[idx] = s;
  entrei[s] = idx;
  idx++;
  for (auto const &v : adj[s])
  {
    if (v == f)
      continue;
    euler_tour(v, s);
  }
  euler[idx] = s;
  sai[s] = idx;
  idx++;
}
signed main()
{
  ios_base::sync_with_stdio(false);
```

```
  cin.tie(NULL);
  int n;
  cin >> n;
  for (int i = 0; i < n - 1; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  euler_tour(0, -1);
  for (int i = 0; i < 2 * n; i++)
    cout << euler[i] << " ";
  cout << endl;
  return 0;
}
// euler tour of a tree
// muito util para algumas coisas
// exemplos:
// 1- soma da subarvore de v(com update)
// usando segment trees, podemos fazer uma query(entrei[v], sai[v])
// 2- LCA
// lca(u, v) = query(entrei[u], entrei[v])
// usando uma query de minimo e considerando as profundidade dos vertices
// a resposta sera o vertice de profundidade minima que encontrarmos no
    intervalo
// 3- agilidade para remover arestas/vertices/subtrees da arvore
// basta apenas tratar o segmento equivalente do jeito que for necessario
// 4- reroot a tree
// basta apenas rotacionar o euler path
```

4.19 flow with minimum capacities

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007
#define INF 1e9

struct edge
{
  int dest, back, f, c, id;
};
struct push_relabel
{
  int n;
  vector<vector<edge>> g;
  vector<int> ec;
  vector<edge *> cur;
  vector<vector<int>> hs;
  vector<int> H;
  push_relabel(int sz) : g(sz), ec(sz), cur(sz), hs(2 * sz), H(sz) { n = sz; }
  void add_edge(int s, int t, int cap, int rcap, int id)
  {
    if (s == t)
      return;
    g[s].pb({t, (int)g[t].size(), 0, cap, id});
    g[t].pb({s, (int)g[s].size() - 1, 0, rcap, -1});
  }
  void add_flow(edge &e, int f)
  {
    edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f)
      hs[H[e.dest]].push_back(e.dest);
    e.f += f;
    e.c -= f;
    ec[e.dest] += f;
    back.f -= f;
    back.c += f;
    ec[back.dest] -= f;
  }
  int calc(int s, int t)
  {
    int v = g.size();
    H[s] = v;
    ec[t] = 1;
    vector<int> co(2 * v);
    co[0] = v - 1;
    for (int i = 0; i < v; i++)
      cur[i] = g[i].data();
    for (edge &e : g[s])
      add_flow(e, e.c);
    for (int hi = 0;;)
    {
      while (hs[hi].empty())
        if (!hi--)
          return -ec[s];
      int u = hs[hi].back();
      hs[hi].pop_back();
      while (ec[u] > 0)
      {
        if (cur[u] == g[u].data() + g[u].size())
        {
          H[u] = INF;
          for (edge &e : g[u])
            if (e.c && H[u] > H[e.dest] + 1)
              H[u] = H[e.dest] + 1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            for (int i = 0; i < v; i++)
              if (hi < H[i] && H[i] < v)
                --co[H[i]], H[i] = v + 1;
          hi = H[u];
        }
        else if (cur[u]->c && H[u] == H[cur[u]->dest] + 1)
          add_flow(*cur[u], min(ec[u], cur[u]->c));
        else
          ++cur[u];
      }
    }
  }
  vector<int> flow_edges(int m) // fluxo em cada aresta
  {
    vector<int> ans(m);
    for (int i = 0; i < n; i++)
    {
      for (auto const &j : g[i])
      {
        if (j.id != -1)
          ans[j.id] = j.f;
      }
    }
    return ans;
  }
};
struct flow_with_demands
{
  push_relabel pr;
  vector<int> in, out;
  int n;

  flow_with_demands(int sz) : n(sz), pr(sz + 2), in(sz), out(sz) {}
  void add_edge(int u, int v, int cap, int dem, int id)
  {
    pr.add_edge(u, v, cap - dem, 0, id);
    out[u] += dem, in[v] += dem;
  }
  int run(int s, int t)
  {
```

```
    pr.add_edge(t, s, INF, 0, -1);
    for (int i = 0; i < n; i++)
    {
      pr.add_edge(n, i, in[i], 0, -1);
      pr.add_edge(i, n + 1, out[i], 0, -1);
    }
    return pr.calc(n, n + 1);
  }
  bool check() // todas as constraints foram satisfeitas?
  {
    for (auto const &i : pr.g[n])
    {
      if (i.c > 0)
        return 0;
    }
    return 1;
  }
};

int dx[] = {1, -1, 0, 0};
int dy[] = {0, 0, 1, -1};

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int h, w;
  cin >> h >> w;
  vector<string> v(h);
  for (int i = 0; i < h; i++)
  {
    cin >> v[i];
  }
  vector<pi> s[2];
  for (int i = 0; i < h; i++)
  {
    for (int j = 0; j < w; j++)
    {
      if (v[i][j] != '1')
        s[(i + j) % 2].pb({i, j});
    }
  }
  for (int i = 0; i < 2; i++)
  {
    sort(s[i].begin(), s[i].end());
  }
  flow_with_demands mf(s[0].size() + s[1].size() + 2);
  int src = s[0].size() + s[1].size();
  int sink = s[0].size() + s[1].size() + 1;
  for (int x = 0; x < s[0].size(); x++)
  {
    int i = s[0][x].fir, j = s[0][x].sec;
    if (v[i][j] == '2')
      mf.add_edge(src, x, 1, 1, -1);
    else
      mf.add_edge(src, x, 1, 0, -1);
  }
  for (int x = 0; x < s[1].size(); x++)
  {
    int i = s[1][x].fir, j = s[1][x].sec;
    if (v[i][j] == '2')
      mf.add_edge(s[0].size() + x, sink, 1, 1, -1);
    else
      mf.add_edge(s[0].size() + x, sink, 1, 0, -1);
  }
  for (int x = 0; x < s[0].size(); x++)
  {
    for (int d = 0; d < 4; d++)
    {
      pi curr = {s[0][x].fir + dx[d], s[0][x].sec + dy[d]};
      if (binary_search(s[1].begin(), s[1].end(), curr))
      {
        int y = lower_bound(s[1].begin(), s[1].end(), curr) - s[1].begin();
        mf.add_edge(x, s[0].size() + y, 1, 0, -1);
      }
    }
  }
  mf.run(src, sink);
  // existe um jeito de passar fluxo que satisfaz todas as constraints?
  (mf.check()) ? cout << "Yes\n" : cout << "No\n";
  return 0;
}
// problema exemplo
// https://atcoder.jp/contests/abc285/tasks/abc285_g

// as celulas com 1 eu posso ignorar
// agr pras celulas com 2, eu preciso achar um matching dela com alguem
// so considerando os 2 e as ?

// entao a missao vira achar um matching (nao necessariamente maximo)
// mas que englobe todos os 2
// pode ter 2 de um lado e pode ter 2 do outro

// e se eu pudesse adicionar a seguinte constraint para algumas arestas:
// a quantidade de fluxo passada naquela aresta tem que ser entre [l, r]
// Maximum flow problem with minimum capacities, tambem conhecido como flow with
    demands
// ai da pra dale em resolver

// https://cp-algorithms.com/graph/flow_with_demands.html
```

4.20 Floyd Warshall

```
#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define lli long long int
#define MAXN 10000
#define INF 999999

int n , m , a , b , c ;
int dist [MAXN][MAXN] ;

void floyd_warshall ()
{
    for (int k = 0 ; k < n ; k++)
    {
        for (int i = 0 ; i < n ; i++)
        {
            for (int j = 0 ; j < n ; j++)
            {
                dist[i][j] = min(dist[i][j] , dist[i][k] + dist[k][j]) ;
            }
        }
    }
}
void initialize ()
{
    for (int i = 0 ; i < n ; i++)
    {
        for (int j = 0 ; j < n ; j++)
        {
            if (i == j)
            {
                dist[i][j] = 0 ;
            }
            else
            {
                dist[i][j] = INF ;
            }

        }
    }
}
int main()
{
    cin >> n >> m ;

    initialize () ;

    for (int i = 0 ; i < m ; i++)
    {
        cin >> a >> b >> c ;
```

```
            dist [a][b] = min (dist[a][b] , c) ;
            dist [b][a] = min (dist[b][a] , c) ;
        }

        floyd_warshall () ;

        return 0;
    }
```

4.21 Ford Fulkerson

```
    // ford-fulkerson: obter qual o fluxo maximo de um vertice s ate um vertice d
    // 1 - rodar um bfs para descobrir um novo caminho de s ate d
    // 2 - apos isso pego a aresta de menor custo desse caminho e subtraio o valor
         dela nas outras arestas do caminho
    // 3 - fluxo_maximo += custo da aresta de menor custo desse caminho
    // 4 - rodar isso ate nao existirem mais caminhos disponiveis (com fluxo
         diferente de 0) entre s e d
    // 5 - o fluxo maximo de s ate d sera a soma das arestas de menor custo de cada
         caminho feito

    #include <bits/stdc++.h>
    using namespace std ;

    #define lli long long int
    #define pb push_back
    #define MAXN 10000
    #define INF 999999

    int n , m , a , b , c , s , d , max_flow , flow ;
    vector <int> parent ;
    vector <int> adj [MAXN] ;
    int cost [MAXN][MAXN] ;
    bool visited [MAXN] ;

    void get_menor_custo (int v , int mincost)
    {
        if (v == s)
        {
            flow = mincost ;
            return ;
        }
        else if (parent[v] != -1)
        {
            get_menor_custo(parent[v] , min(mincost , cost[parent[v]][v])) ;
            cost[parent[v]][v] -= flow ;
            cost[v][parent[v]] += flow ;
        }
    }
    void bfs ()
    {
        visited[s] = true ;

        queue <int> q ;
        q.push(s) ;
        parent.assign(MAXN , -1) ;

        while (!q.empty())
        {
            int u = q.front() ;
            q.pop() ;

            if (u == d)
            {
                break ;
            }

            for (int j = 0 ; j < adj[u].size() ; j++)
            {
                int v = adj[u][j] ;

                if (cost[u][v] > 0 && !visited[v])
                {
                    visited[v] = true ;
                    q.push(v) ;
                    parent[v] = u ;
                }
            }
        }
    }
    int ford_fulkerson ()
    {
        max_flow = 0 ;

        while (1)
        {
            flow = 0 ;
            memset(visited , false , sizeof(visited));

            bfs() ;
            get_menor_custo(d , INF) ;

            if (flow == 0)
            {
                break ;
            }

            max_flow += flow ;
        }

        return max_flow ;
    }
    int main ()
    {
        ios_base::sync_with_stdio(false) ;
        cin.tie(NULL) ;

        cin >> n >> m ;

        for (int i = 0 ; i < m ; i++)
        {
            cin >> a >> b >> c ;
            adj[a].pb(b);
            adj[b].pb(a);
            cost[a][b] = c ;
        }

        cin >> s >> d ;

        cout << ford_fulkerson() << endl ;

        return 0 ;
    }
```

4.22 Grafo Bipartido

```
    #include <bits/stdc++.h>
    #include <ext/pb_ds/assoc_container.hpp>
    #include <ext/pb_ds/tree_policy.hpp>
    using namespace std;
    using namespace __gnu_pbds;

    template <class T>
    using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;

    #define int long long int
    #define endl '\n'
    #define pb push_back
    #define pi pair<int, int>
    #define pii pair<int, pi>
    #define fir first
    #define sec second
    #define MAXN 200006
    #define mod 1000000007

    struct dsu
    {
      vector<pi> parent;
      vector<int> rank;
      vector<int> bipartite;
```

```
  dsu(int n)
  {
    parent.resize(n);
    rank.resize(n);
    bipartite.resize(n);
    for (int v = 0; v < n; v++)
    {
      parent[v] = {v, 0};
      rank[v] = 0;
      bipartite[v] = 1;
    }
  }
  dsu() {}
  pi find_set(int v)
  {
    if (v != parent[v].fir)
    {
      int parity = parent[v].sec;
      parent[v] = find_set(parent[v].fir);
      parent[v].sec ^= parity;
    }
    return parent[v];
  }
  void add_edge(int a, int b)
  {
    pi pa = find_set(a);
    a = pa.fir;
    int x = pa.sec;
    pi pb = find_set(b);
    b = pb.fir;
    int y = pb.sec;
    if (a == b)
    {
      if (x == y)
        bipartite[a] = 0;
    }
    else
    {
      if (rank[a] < rank[b])
        swap(a, b);
      parent[b] = {a, x ^ y ^ 1};
      bipartite[a] &= bipartite[b];
      if (rank[a] == rank[b])
        rank[a]++;
    }
  }
  bool is_bipartite(int v)
  {
    return bipartite[find_set(v).fir];
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
```

4.23 hall theorem

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, pi>
#define fir first
#define sec second
#define MAXN 1000006
#define mod 1000000007

int cnt[6];
int cnt_mask[1 << 6];
int tot_mask[1 << 6];

bool halls()
{
  for (int mask = 1; mask < (1 << 6); mask++)
  {
    int x = 0;
    for (int i = 0; i < 6; i++)
    {
      if (mask & (1 << i))
        x += cnt[i];
    }
    if (x < tot_mask[mask])
      return 0;
  }
  return 1;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s;
  cin >> s;
  for (auto const &i : s)
  {
    cnt[i - 'a']++;
  }
  int n = s.size();
  vector<int> mask(n, (1 << 6) - 1); // a mask que diz quais chars nao podem
       aparecer naquela posicao
  int m;
  cin >> m;
  while (m--)
  {
    int i;
    string t;
    cin >> i >> t;
    i--;
    mask[i] = 0;
    for (auto const &j : t)
      mask[i] |= (1 << (j - 'a'));
  }
  for (auto const &i : mask)
  {
    cnt_mask[i]++;
  }
  for (int m = 0; m < (1 << 6); m++)
  {
    for (int s = m; s; s = (s - 1) & m) // soma dos cnt_mask de todas as
         submasks
    {
      tot_mask[m] += cnt_mask[s];
    }
  }
  if (!halls())
  {
    cout << "Impossible\n";
    return 0;
  }
  for (int i = 0; i < n; i++)
  {
    for (int m = 0; m < (1 << 6); m++)
    {
      if ((m & mask[i]) == mask[i])
        tot_mask[m]--;
    }
    for (int j = 0; j < 6; j++)
    {
      if ((mask[i] & (1 << j)) && cnt[j] > 0)
      {
        cnt[j]--;
        if (halls()) // faz s[i] = j e ve se o matching continua a existir
        {
          cout << (char)(j + 'a');
          break;
```

```
        }
        cnt[j]++;
      }
    }
  }
  cout << endl;
  return 0;
}
// https://codeforces.com/contest/1009/problem/G
// problema bem legal, que usa o teorema de hall
// dada uma string s, no qual cada char eh a, b, c, d, e ou f
// eu quero permutar essa string s de alguma forma, tal que a seguinte condicao
    eh satisfeita para todo indice i:
// cada indice tem um set de chars que sao proibidos de estar naquela posicao
// se existir multiplas solucoes, printe a menor string lexicograficamente que
    puder ser a resposta
```

4.24 hld

```
// https://codeforces.com/contest/343/problem/D
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 500001
#define mod 1000000007

struct segtree
{
  int n;
  vector<int> v;
  vector<int> seg;
  vector<int> lazy;

  segtree() {}
  segtree(int sz)
  {
    n = sz;
    seg.assign(4 * n, 0);
    lazy.assign(4 * n, -1);
  }
  int single(int x)
  {
    return x;
  }
  int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a + b;
  }
  void add(int i, int l, int r, int diff)
  {
    seg[i] = (r - l + 1) * diff;
    if (l != r)
    {
      lazy[i << 1] = diff;
      lazy[(i << 1) | 1] = diff;
    }
    lazy[i] = -1;
  }
  void update(int i, int l, int r, int ql, int qr, int diff)
  {
    if (lazy[i] != -1)
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return;
    if (l >= ql && r <= qr)
    {
      add(i, l, r, diff);
      return;
    }
    int mid = (l + r) >> 1;
    update(i << 1, l, mid, ql, qr, diff);
    update((i << 1) | 1, mid + 1, r, ql, qr, diff);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int query(int l, int r, int ql, int qr, int i)
  {
    if (lazy[i] != -1)
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return neutral();
    if (l >= ql && r <= qr)
      return seg[i];
    int mid = (l + r) >> 1;
    return merge(query(l, mid, ql, qr, i << 1), query(mid + 1, r, ql, qr, (i <<
        1) | 1));
  }
};
struct hld
{
  int n, cur_pos;
  segtree seg;
  vector<vector<int>> adj;
  vector<int> parent, depth, heavy, head, pos, sz;

  int dfs(int s)
  {
    int size = 1, max_c_size = 0;
    for (auto const &c : adj[s])
    {
      if (c != parent[s])
      {
        parent[c] = s;
        depth[c] = depth[s] + 1;
        int c_size = dfs(c);
        size += c_size;
        if (c_size > max_c_size)
          max_c_size = c_size, heavy[s] = c;
      }
    }
    return sz[s] = size;
  }
  void decompose(int s, int h)
  {
    head[s] = h;
    pos[s] = cur_pos++;
    if (heavy[s] != -1)
      decompose(heavy[s], h);
    for (int c : adj[s])
    {
      if (c != parent[s] && c != heavy[s])
        decompose(c, c);
    }
  }
  hld(vector<vector<int>> &g)
  {
    n = g.size();
    adj = g;
    seg = segtree(n);
    parent.assign(n, -1);
    depth.assign(n, -1);
    heavy.assign(n, -1);
    head.assign(n, -1);
    pos.assign(n, -1);
    sz.assign(n, 1);
    cur_pos = 0;
    dfs(0);
    decompose(0, 0);
  }
  int query_path(int a, int b)
  {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]])
```

```
    {
      if (depth[head[a]] > depth[head[b]])
        swap(a, b);
      res += seg.query(0, n - 1, pos[head[b]], pos[b], 1);
    }
    if (depth[a] > depth[b])
      swap(a, b);
    res += seg.query(0, n - 1, pos[a], pos[b], 1);
    return res;
  }
  void update_path(int a, int b, int x)
  {
    for (; head[a] != head[b]; b = parent[head[b]])
    {
      if (depth[head[a]] > depth[head[b]])
        swap(a, b);
      seg.update(1, 0, n - 1, pos[head[b]], pos[b], x);
    }
    if (depth[a] > depth[b])
      swap(a, b);
    seg.update(1, 0, n - 1, pos[a], pos[b], x);
  }
  void update_subtree(int a, int x)
  {
    seg.update(1, 0, n - 1, pos[a], pos[a] + sz[a] - 1, x);
  }
  int query_subtree(int a)
  {
    return seg.query(0, n - 1, pos[a], pos[a] + sz[a] - 1, 1);
  }
  int lca(int a, int b)
  {
    if (pos[a] < pos[b])
      swap(a, b);
    return (head[a] == head[b]) ? b : lca(parent[head[a]], b);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<vector<int>> adj(n);
  for (int i = 0; i < n - 1; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  hld h(adj);
  int q;
  cin >> q;
  while (q--)
  {
    int a, b;
    cin >> a >> b;
    b--;
    if (a == 1)
    {
      h.update_subtree(b, 1);
    }
    if (a == 2)
    {
      h.update_path(0, b, 0);
    }
    if (a == 3)
    {
      cout << h.query_path(b, b) << endl;
    }
  }
  return 0;
}
```

4.25 hld edge

```
//https://www.spoj.com/problems/QTREE/
//Don't use cin/cout in this problem (gives TLE)
#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 10001
#define mod 1000000007

int n;
vector<pi> adj[MAXN];
vector<pi> edges;

namespace seg
{
  int seg[4 * MAXN];
  int lazy[4 * MAXN];
  int v[MAXN];

  int single(int x)
  {
    return x;
  }
  int neutral()
  {
    return -1;
  }
  int merge(int a, int b)
  {
    return max(a, b);
  }
  void add(int i, int l, int r, int diff)
  {
    seg[i] = (r - l + 1) * diff;
    if (l != r)
    {
      lazy[i << 1] = diff;
      lazy[(i << 1) | 1] = diff;
    }
    lazy[i] = -1;
  }
  void update(int i, int l, int r, int ql, int qr, int diff)
  {
    if (lazy[i] != -1)
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return;
    if (l >= ql && r <= qr)
    {
      add(i, l, r, diff);
      return;
    }
    int mid = (l + r) >> 1;
    update(i << 1, l, mid, ql, qr, diff);
    update((i << 1) | 1, mid + 1, r, ql, qr, diff);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int query(int l, int r, int ql, int qr, int i)
  {
    if (lazy[i] != -1)
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return neutral();
    if (l >= ql && r <= qr)
      return seg[i];
    int mid = (l + r) >> 1;
    return merge(query(l, mid, ql, qr, i << 1), query(mid + 1, r, ql, qr, (i <<
        1) | 1));
  }
  void build(int l, int r, int i)
```

```
  {
    if (l == r)
    {
      seg[i] = single(v[l]);
      lazy[i] = -1;
      return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, i << 1);
    build(mid + 1, r, (i << 1) | 1);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
    lazy[i] = -1;
  }
} // namespace seg
namespace hld
{
  int cur_pos;
  vector<int> parent, depth, heavy, head, pos, sz, up;

  int dfs(int s)
  {
    int size = 1, max_c_size = 0;
    for (auto const &c : adj[s])
    {
      if (c.fir != parent[s])
      {
        parent[c.fir] = s;
        depth[c.fir] = depth[s] + 1;
        int c_size = dfs(c.fir);
        size += c_size;
        if (c_size > max_c_size)
          max_c_size = c_size, heavy[s] = c.fir;
      }
    }
    return sz[s] = size;
  }
  void decompose(int s, int h)
  {
    head[s] = h;
    pos[s] = cur_pos++;
    seg::v[pos[s]] = up[s];
    for (auto const &c : adj[s])
    {
      if (c.fir != parent[s] && c.fir == heavy[s])
      {
        up[c.fir] = c.sec;
        decompose(heavy[s], h);
      }
    }
    for (auto const &c : adj[s])
    {
      if (c.fir != parent[s] && c.fir != heavy[s])
      {
        up[c.fir] = c.sec;
        decompose(c.fir, c.fir);
      }
    }
  }
  void init()
  {
    parent.assign(MAXN, -1);
    depth.assign(MAXN, -1);
    heavy.assign(MAXN, -1);
    head.assign(MAXN, -1);
    pos.assign(MAXN, -1);
    sz.assign(MAXN, 1);
    up.assign(MAXN, 0);
    cur_pos = 0;
    dfs(0);
    decompose(0, 0);
    seg::build(0, n - 1, 1);
  }
  int query_path(int a, int b)
  {
    int res = -1;
    for (; head[a] != head[b]; b = parent[head[b]])
    {
      if (depth[head[a]] > depth[head[b]])
        swap(a, b);
      res = max(res, seg::query(0, n - 1, pos[head[b]], pos[b], 1));
    }
    if (depth[a] > depth[b])
      swap(a, b);
    res = max(res, seg::query(0, n - 1, pos[a] + 1, pos[b], 1));
    return res;
  }
  void update_path(int a, int b, int x)
  {
    for (; head[a] != head[b]; b = parent[head[b]])
    {
      if (depth[head[a]] > depth[head[b]])
        swap(a, b);
      seg::update(1, 0, n - 1, pos[head[b]], pos[b], x);
    }
    if (depth[a] > depth[b])
      swap(a, b);
    seg::update(1, 0, n - 1, pos[a] + 1, pos[b], x);
  }
  void update_subtree(int a, int x)
  {
    seg::update(1, 0, n - 1, pos[a] + 1, pos[a] + sz[a] - 1, x);
  }
  int query_subtree(int a, int x)
  {
    return seg::query(0, n - 1, pos[a] + 1, pos[a] + sz[a] - 1, 1);
  }
} // namespace hld
signed main()
{
  int q;
  scanf("%d", &q);
  while (q--)
  {
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
      adj[i].clear();
    edges.clear();
    for (int i = 0; i < n - 1; i++)
    {
      int a, b, c;
      scanf("%d %d %d", &a, &b, &c);
      a--, b--;
      adj[a].pb({b, c});
      adj[b].pb({a, c});
      edges.pb({a, b});
    }
    hld::init();
    while (true)
    {
      char k[10];
      scanf("%s", k);
      if (k[0] == 'Q')
      {
        int a, b;
        scanf("%d %d", &a, &b);
        a--, b--;
        printf("%d\n", hld::query_path(a, b));
      }
      else if (k[0] == 'C')
      {
        int a, b;
        scanf("%d %d", &a, &b);
        a--;
        hld::update_path(edges[a].fir, edges[a].sec, b);
      }
      else
      {
        break;
      }
    }
  }
  return 0;
}
```

4.26 hopcroft karp

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// #define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000003
#define mod 998244353
#define INF 1e9

struct hopcroft_karp
{
  vector<int> match;
  vector<int> dist;
  vector<vector<int>> adj;
  int n, m, t;

  hopcroft_karp(int a, int b)
  {
    n = a, m = b;
    t = n + m + 1;
    match.assign(t, n + m);
    dist.assign(t, 0);
    adj.assign(t, vector<int>{});
  }
  void add_edge(int u, int v)
  {
    adj[u].pb(v);
    adj[v].pb(u);
  }
  bool bfs()
  {
    queue<int> q;
    for (int u = 0; u < n; u++)
    {
      if (match[u] == n + m)
        dist[u] = 0, q.push(u);
      else
        dist[u] = INF;
    }
    dist[n + m] = INF;
    while (!q.empty())
    {
      int u = q.front();
      q.pop();
      if (dist[u] < dist[n + m])
      {
        for (auto const &v : adj[u])
        {
          if (dist[match[v]] == INF)
          {
            dist[match[v]] = dist[u] + 1;
            q.push(match[v]);
          }
        }
      }
    }
    return dist[n + m] < INF;
  }
  bool dfs(int u)
  {
    if (u < n + m)
    {
      for (auto const &v : adj[u])
      {
        if (dist[match[v]] == dist[u] + 1 && dfs(match[v]))
        {
          match[v] = u;
          match[u] = v;
          return true;
        }
      }
      dist[u] = INF;
      return false;
    }
    return true;
  }
  vector<pi> run()
  {
    int cnt = 0;
    while (bfs())
      for (int u = 0; u < n; u++)
        if (match[u] == n + m && dfs(u))
          cnt++;
    vector<pi> ans;
    for (int v = n; v < n + m; v++)
      if (match[v] < n + m)
        ans.pb({match[v], v});
    return ans;
  }
  vector<int> mvc() // minimum vertex cover
  {
    vector<pi> ans = run();
    vector<bool> vis(n + m, 0);
    for (int i = 0; i < n; i++)
    {
      if (match[i] == n + m)
      {
        queue<int> q;
        q.push(i);
        while (!q.empty())
        {
          int x = q.front();
          q.pop();
          vis[x] = 1;
          for (auto const &y : adj[x])
          {
            if (!vis[y])
            {
              vis[y] = 1;
              q.push(match[y]);
            }
          }
        }
      }
    }
    vector<int> vc;
    for (int i = 0; i < n; i++)
    {
      if (!vis[i])
        vc.pb(i);
    }
    for (int i = n; i < n + m; i++)
    {
      if (vis[i])
        vc.pb(i);
    }
    return vc;
  }
  vector<pi> mec() // minimum edge cover
  {
    vector<pi> ans = run();
    for (int i = 0; i < n + m; i++)
    {
      if (match[i] == n + m && adj[i].size() > 0)
      {
        if (i < n)
          ans.pb({i, adj[i][0]});
        else
          ans.pb({adj[i][0], i});
      }
    }
    return ans;
  }
```

```
};

// minimum path cover on dag
// minimum set of paths such that each of the vertices belongs to exactly one
    path
vector<vector<int>> mpc(int n, vector<pi> &e)
{
  hopcroft_karp h(n, n);
  for (auto const &i : e)
    h.add_edge(i.fir, n + i.sec);
  vector<pi> mat = h.run();
  vector<int> prv(n, -1);
  vector<int> nxt(n, -1);
  for (int i = 0; i < mat.size(); i++)
  {
    nxt[mat[i].fir] = mat[i].sec - n;
    prv[mat[i].sec - n] = mat[i].fir;
  }
  vector<vector<int>> ans;
  for (int i = 0; i < n; i++)
  {
    if (prv[i] == -1 && nxt[i] == -1)
    {
      ans.pb({i});
    }
    else if (prv[i] == -1)
    {
      vector<int> curr;
      int x = i;
      while (1)
      {
        curr.pb(x);
        if (nxt[x] == -1)
          break;
        x = nxt[x];
      }
      ans.pb(curr);
    }
  }
  return ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  vector<pi> e;
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    e.pb({a, b});
  }
  vector<vector<int>> ans = mpc(n, e);
  cout << ans.size() << endl;
  for (auto const &v : ans)
  {
    for (auto const &i : v)
      cout << i + 1 << " ";
    cout << endl;
  }
  return 0;
}
```

4.27 hungarian

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 505
#define mod 998244353

struct hungarian
{
  int n, inf;
  vector<vector<int>> a;
  vector<int> u, v, p, way;

  hungarian(int n_) : n(n_), u(n + 1), v(n + 1), p(n + 1), way(n + 1)
  {
    a = vector<vector<int>>(n, vector<int>(n));
    inf = numeric_limits<int>::max();
  }
  void add_edge(int x, int y, int c)
  {
    a[x][y] = c;
  }
  pair<int, vector<int>> run()
  {
    for (int i = 1; i <= n; i++)
    {
      p[0] = i;
      int j0 = 0;
      vector<int> minv(n + 1, inf);
      vector<int> used(n + 1, 0);
      do
      {
        used[j0] = true;
        int i0 = p[j0], j1 = -1;
        int delta = inf;
        for (int j = 1; j <= n; j++)
        {
          if (!used[j])
          {
            int cur = a[i0 - 1][j - 1] - u[i0] - v[j];
            if (cur < minv[j])
              minv[j] = cur, way[j] = j0;
            if (minv[j] < delta)
              delta = minv[j], j1 = j;
          }
        }
        for (int j = 0; j <= n; j++)
        {
          if (used[j])
            u[p[j]] += delta, v[j] -= delta;
          else
            minv[j] -= delta;
        }
        j0 = j1;
      } while (p[j0] != 0);
      do
      {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
      } while (j0);
    }
    vector<int> ans(n);
    for (int j = 1; j <= n; j++)
      ans[p[j] - 1] = j - 1;
    return make_pair(-v[0], ans);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
```

```
  vector<vector<int>> c(n, vector<int>(n));
  hungarian h(n);
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      cin >> c[i][j];
      h.add_edge(i, j, c[i][j]);
    }
  }
  cout << h.run().fir << endl;
  return 0;
}
```

4.28 Kruskal

```
// Algoritimo de kruskal - Achar a mst

// 1 - listar todas as arestas em ordem crescente.

// 2 - Cada aresta liga dois vertices x e y, checar se eles ja estao na mesma
    componente conexa
// (aqui, consideramos apenas as arestas ja colocadas na arvore).

// 3 - Se x e y estao na mesma componente, ignoramos a aresta e continuamos o
    procedimento
// (se a usassemos, formariamos um ciclo). Se estiverem em componentes distintas
    , colocamos a aresta
//na arvore e continuamos o procedimento.

// OBS: como a prioridade eh ordenar pelas menores distancias, basta botar o
    custo da aresta como
// first no vector das arestas para poder ordenar

// em suma: ordeno as arestas em ordem crescente com prioridade no custo, depois
     para cada aresta,
// se o find(x) != find(y) sendo x e y os vertices das arestas, eu adiciono eles
     a mst e dou um join
// nos dois, como as arestas tao ordenadas em ordem crescente, o primeiro que eu
     pego
// eh necessariamente a melhor opcao e assim a mst eh formada.

#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 100001

int n, m, a, b, c;
vector<pii> ar;
vector<pii> mst;
int pai[MAXN];
int peso[MAXN];

int find(int x)
{
  if (pai[x] == x)
  {
    return x;
  }
  return pai[x] = find(pai[x]);
}
void join(int a, int b)
{
  a = find(a);
  b = find(b);

  if (peso[a] < peso[b])
  {
    pai[a] = b;
  }
  else if (peso[b] < peso[a])
  {
    pai[b] = a;
  }
  else
  {
    pai[a] = b;
    peso[b]++;
  }
}
void initialize()
{
  for (int i = 1; i <= n; i++)
  {
    pai[i] = i;
  }
}
int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  cin >> n >> m;

  for (int i = 0; i < m; i++)
  {
    cin >> a >> b >> c;
    ar.pb(mp(c, mp(a, b)));
  }

  sort(ar.begin(), ar.end());

  initialize();

  int size = 0;

  for (int i = 0; i < m; i++)
  {
    if (find(ar[i].sec.fir) != find(ar[i].sec.sec))
    {
      join(ar[i].sec.fir, ar[i].sec.sec);
      mst.pb(mp(ar[i].fir, mp(ar[i].sec.fir, ar[i].sec.sec)));
    }
  }

  for (int i = 0; i < mst.size(); i++)
  {
    cout << mst[i].sec.fir << " " << mst[i].sec.sec << " " << mst[i].fir << endl
         ;
  }

  return 0;
}
```

4.29 kuhn

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 2001
#define mod 998244353
```

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

struct kuhn
{
  int n, m, at;
  vector<vector<int>> g;
  vector<int> vis, ma, mb;

  kuhn(int n_, int m_) : n(n_), m(m_), at(0), g(n),
                         vis(n), ma(n, -1), mb(m, -1) {}
  void add_edge(int a, int b) { g[a].push_back(b); }
  bool dfs(int i)
  {
    vis[i] = at;
    for (int j : g[i])
    {
      if (mb[j] == -1)
      {
        ma[i] = j, mb[j] = i;
        return true;
      }
    }
    for (int j : g[i])
    {
      if (vis[mb[j]] != at && dfs(mb[j]))
      {
        ma[i] = j, mb[j] = i;
        return true;
      }
    }
    return false;
  }
  int augment(int i) // pode usar quando tou adicionando o vertice i pela
       primeira vez no grafo
  {
    shuffle(g[i].begin(), g[i].end(), rng);
    if (dfs(i))
      at++;
    return at;
  }
  int matching() // calcula o max matching
  {
    int aum = 1;
    while (aum)
    {
      for (auto &i : g)
        shuffle(i.begin(), i.end(), rng);
      for (int j = 0; j < n; j++)
        vis[j] = 0;
      aum = 0;
      for (int i = 0; i < n; i++)
        if (ma[i] == -1 and dfs(i))
          at++, aum = 1;
    }
    return at;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  vector<int> p(n);
  for (int i = 0; i < n; i++)
  {
    cin >> p[i];
  }
  vector<int> c(n);
  vector<vector<pi>> v(5001);
  for (int i = 0; i < n; i++)
  {
    cin >> c[i];
    c[i]--;
    v[p[i]].pb({i, c[i]});
  }
  int q;
  cin >> q;
  vector<int> qry(q);
  vector<bool> rem(n, 0);
  for (int i = 0; i < q; i++)
  {
    cin >> qry[i];
    qry[i]--;
    rem[qry[i]] = 1;
  }
  int sz = m + 1;
  kuhn h(5001, sz);
  vector<int> ans(q);
  int mex = 0;
  for (auto const &j : v[0])
  {
    if (!rem[j.fir])
      h.add_edge(0, j.sec);
  }
  h.augment(0);
  for (int i = q - 1; i >= 0; i--)
  {
    int x = qry[i];
    while (mex < m && h.at == mex + 1)
    {
      mex++;
      for (auto const &j : v[mex])
      {
        if (!rem[j.fir])
          h.add_edge(mex, j.sec);
      }
      h.augment(mex);
    }
    ans[i] = mex;
    rem[x] = 0;
    if (p[x] <= mex)
    {
      h.add_edge(p[x], c[x]);
      h.matching();
    }
  }
  for (auto const &i : ans)
    cout << i << endl;
  return 0;
}
// problema de matching incremental, ir adicionando arestas e calculando o max
     matching
// https://codeforces.com/contest/1139/problem/E

// selecionar um student de cada club, caso o club possua pelo menos um caba
// pros selecionados, maximizar o mex

// fazer as queries ao contrario talvez seja o caminho
```

4.30 LCA

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 100001
#define mod 1000000007

int n;
```

```
vector<int> adj[MAXN];

namespace lca
{
  int l, timer;
  vector<int> tin, tout, depth;
  vector<vector<int>> up;

  void dfs(int v, int p)
  {
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; i++)
      up[v][i] = up[up[v][i - 1]][i - 1];
    for (auto const &u : adj[v])
    {
      if (p == u)
        continue;
      depth[u] = depth[v] + 1;
      dfs(u, v);
    }
    tout[v] = ++timer;
  }
  bool is_ancestor(int u, int v)
  {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
  }
  int binary_lifting(int u, int v)
  {
    if (is_ancestor(u, v))
      return u;
    if (is_ancestor(v, u))
      return v;
    for (int i = l; i >= 0; --i)
      if (!is_ancestor(up[u][i], v))
        u = up[u][i];
    return up[u][0];
  }
  void init()
  {
    tin.resize(n);
    tout.resize(n);
    depth.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(0, 0);
  }
  int dist(int s, int v)
  {
    int at = binary_lifting(s, v);
    return (depth[s] + depth[v] - 2 * depth[at]);
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n;
  for (int i = 0; i < n - 1; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  lca::init();
  return 0;
}
```

4.31 link cut tree edge

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

struct node
{
  int p, ch[2];
  pii val, sub;
  bool rev;
  int sz, ar;
  int lazy;
  node() {}
  node(pii v, int ar_) : p(-1), val(v), sub(v), rev(0), sz(ar_), ar(ar_), lazy
       (0)
  {
    ch[0] = ch[1] = -1;
  }
};
struct link_cut_tree
{
  vector<node> t;
  map<pair<int, int>, int> aresta;
  int sz, n;

  link_cut_tree(int nn)
  {
    t.clear();
    n = nn;
    sz = 0;
  }
  static pii neutral()
  {
    return {0, {0, 0}};
  }
  pii merge(pii a, pii b)
  {
    return max(a, b);
  }
  void prop(int x)
  {
    if (t[x].lazy)
    {
      if (t[x].ar)
        t[x].val.fir = t[x].lazy;
      t[x].sub = merge(t[x].sub, t[x].val);
      if (t[x].ch[0] + 1)
        t[t[x].ch[0]].lazy = t[x].lazy;
      if (t[x].ch[1] + 1)
        t[t[x].ch[1]].lazy = t[x].lazy;
    }
    if (t[x].rev)
    {
      swap(t[x].ch[0], t[x].ch[1]);
      if (t[x].ch[0] + 1)
        t[t[x].ch[0]].rev ^= 1;
      if (t[x].ch[1] + 1)
        t[t[x].ch[1]].rev ^= 1;
    }
    t[x].lazy = 0, t[x].rev = 0;
  }
  void update(int x)
  {
    t[x].sz = t[x].ar, t[x].sub = t[x].val;
    for (int i = 0; i < 2; i++)
    {
      if (t[x].ch[i] + 1)
      {
        prop(t[x].ch[i]);
        t[x].sz += t[t[x].ch[i]].sz;
        t[x].sub = merge(t[x].sub, t[t[x].ch[i]].sub);
      }
    }
```

```
}
bool is_root(int x)
{
  return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1] != x);
}
void rotate(int x)
{
  int p = t[x].p, pp = t[p].p;
  if (!is_root(p))
    t[pp].ch[t[pp].ch[1] == p] = x;
  bool d = t[p].ch[0] == x;
  t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
  if (t[p].ch[!d] + 1)
    t[t[p].ch[!d]].p = p;
  t[x].p = pp, t[p].p = x;
  update(p), update(x);
}
int splay(int x)
{
  while (!is_root(x))
  {
    int p = t[x].p, pp = t[p].p;
    if (!is_root(p))
      prop(pp);
    prop(p), prop(x);
    if (!is_root(p))
      rotate((t[pp].ch[0] == p) ^ (t[p].ch[0] == x) ? x : p);
    rotate(x);
  }
  return prop(x), x;
}
int access(int v)
{
  int last = -1;
  for (int w = v; w + 1; update(last = w), splay(v), w = t[v].p)
    splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
  return last;
}
void make_tree(int v, pii w = neutral(), int ar = 0)
{
  while (t.size() <= v)
    t.pb(node(neutral(), 0));
  t[v] = node(w, ar);
}
int find_root(int v)
{
  access(v), prop(v);
  while (t[v].ch[0] + 1)
    v = t[v].ch[0], prop(v);
  return splay(v);
}
bool conn(int v, int w)
{
  access(v), access(w);
  return v == w ? true : t[v].p != -1;
}
void rootify(int v)
{
  access(v);
  t[v].rev ^= 1;
}
pii query(int v, int w)
{
  rootify(w), access(v);
  return t[v].sub;
}
void update(int v, int w, int x)
{
  rootify(w), access(v);
  t[v].lazy += x;
}
void link_(int v, int w)
{
  rootify(w);
  t[w].p = v;
}
void link(int v, int w, pii x)
{
  int id = n + sz++;
  aresta[make_pair(v, w)] = id;
  make_tree(id, x, 1);
  link_(v, id), link_(id, w);
}
void cut_(int v, int w)
{
  rootify(w), access(v);
  t[v].ch[0] = t[t[v].ch[0]].p = -1;
}
void cut(int v, int w)
{
  int id = aresta[make_pair(v, w)];
  cut_(v, id), cut_(id, w);
}
int lca(int v, int w)
{
  access(v);
  return access(w);
}
};
struct dsu
{
  int tot;
  vector<int> parent;
  vector<int> sz;

  dsu(int n)
  {
    parent.resize(n);
    sz.resize(n);
    tot = n;
    for (int i = 0; i < n; i++)
    {
      parent[i] = i;
      sz[i] = 1;
    }
  }
  int find_set(int i)
  {
    return parent[i] = (parent[i] == i) ? i : find_set(parent[i]);
  }
  void make_set(int x, int y)
  {
    x = find_set(x), y = find_set(y);
    if (x != y)
    {
      if (sz[x] > sz[y])
        swap(x, y);
      parent[x] = y;
      sz[y] += sz[x];
      tot--;
    }
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n, m, qq;
    cin >> n >> m >> qq;
    vector<pii> e;
    for (int i = 0; i < m; i++)
    {
      int a, b, c;
      cin >> a >> b >> c;
      a--, b--;
      e.pb({c, {a, b}});
    }
    sort(e.begin(), e.end());
    dsu d(n);
    link_cut_tree l(n);
    for (int i = 0; i < n; i++)
    {
      l.make_tree(i);
    }
```

```
    int cost = 0;
    for (auto const &i : e)
    {
      if (d.find_set(i.sec.fir) != d.find_set(i.sec.sec))
      {
        d.make_set(i.sec.fir, i.sec.sec);
        l.link(i.sec.fir, i.sec.sec, i);
        cost += i.fir;
      }
    }
    while (qq--)
    {
      int a, b, c;
      cin >> a >> b >> c;
      a--, b--;
      pii mx = l.query(a, b);
      if (c < mx.fir)
      {
        cost -= mx.fir;
        cost += c;
        l.cut(mx.sec.fir, mx.sec.sec);
        // l.link(a, b, {c, {a, b}}); poderia fazer assim, mas quero testar o
            update
        l.link(a, b, {0, {a, b}});
        l.update(a, b, c);
      }
      cout << cost << endl;
    }
  }
  return 0;
}
// link cut tree com peso nas arestas
// solucao para o: https://codeforces.com/gym/101047/problem/I
// problema onde e dado um grafo inicial e algumas queries
// cada query adiciona uma nova aresta nesse grafo
// e o objetivo e achar a mst apos cada adicao de aresta

// implementacao baseada na: https://github.com/brunomaletta/Biblioteca/blob/
    master/Codigo/Grafos/LCT/lctAresta.cpp

// make_tree(v) cria uma nova arvore com um um unico vertice
// rootify(v) torna v a raiz de sua arvore
// cut(u, v) apaga a aresta u, v
// link(u, v, c) adiciona a aresta de u ate v com peso c
// query(v, w) retorna a aresta de maior peso no caminho de v ate w
// update(v, w, x) faz com que as arestas do caminho de v ate w passem a ter
    peso x
// operacoes tem complexidade O(log(n)) amortizado
```

4.32 link cut tree vertex

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 998244353

struct node
{
  int p, ch[2];
  int val, sub, lazy;
  bool rev;
  int sz;
  node() {}
  node(int v) : p(-1), val(v), sub(v), rev(0), sz(1), lazy(0)
  {
    ch[0] = ch[1] = -1;
  }
};
struct link_cut_tree
{
  vector<node> t;

  link_cut_tree()
  {
    t.clear();
  }
  static int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a ^ b;
  }
  void prop(int x)
  {
    if (t[x].lazy)
    {
      t[x].val = t[x].lazy;
      t[x].sub = t[x].val;
      if (t[x].ch[0] + 1)
        t[t[x].ch[0]].lazy = t[x].lazy;
      if (t[x].ch[1] + 1)
        t[t[x].ch[1]].lazy = t[x].lazy;
    }
    if (t[x].rev)
    {
      swap(t[x].ch[0], t[x].ch[1]);
      if (t[x].ch[0] + 1)
        t[t[x].ch[0]].rev ^= 1;
      if (t[x].ch[1] + 1)
        t[t[x].ch[1]].rev ^= 1;
    }
    t[x].lazy = 0, t[x].rev = 0;
  }
  void update(int x)
  {
    t[x].sz = 1, t[x].sub = t[x].val;
    for (int i = 0; i < 2; i++)
    {
      if (t[x].ch[i] + 1)
      {
        prop(t[x].ch[i]);
        t[x].sz += t[t[x].ch[i]].sz;
        t[x].sub = merge(t[x].sub, t[t[x].ch[i]].sub);
      }
    }
  }
  bool is_root(int x)
  {
    return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1] != x);
  }
  void rotate(int x)
  {
    int p = t[x].p, pp = t[p].p;
    if (!is_root(p))
      t[pp].ch[t[pp].ch[1] == p] = x;
    bool d = t[p].ch[0] == x;
    t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
    if (t[p].ch[!d] + 1)
      t[t[p].ch[!d]].p = p;
    t[x].p = pp, t[p].p = x;
    update(p), update(x);
  }
  int splay(int x)
  {
    while (!is_root(x))
    {
      int p = t[x].p, pp = t[p].p;
      if (!is_root(p))
        prop(pp);
```

```
      prop(p), prop(x);
      if (!is_root(p))
        rotate((t[pp].ch[0] == p) ^ (t[p].ch[0] == x) ? x : p);
      rotate(x);
    }
    return prop(x), x;
  }
  int access(int v)
  {
    int last = -1;
    for (int w = v; w + 1; update(last = w), splay(v), w = t[v].p)
      splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
    return last;
  }
  void make_tree(int v, int w = neutral()) // cria uma arvore com um unico
       vertice, sendo o vertice i
  {
    while (t.size() <= v)
      t.pb(node(neutral()));
    t[v] = node(w);
  }
  int find_root(int v) // acha a raiz da arvore do vertice v
  {
    access(v), prop(v);
    while (t[v].ch[0] + 1)
      v = t[v].ch[0], prop(v);
    return splay(v);
  }
  bool connected(int v, int w) // checa se v e w estao na mesma arvore (aka
       componente conexa)
  {
    access(v), access(w);
    return v == w ? true : t[v].p != -1;
  }
  void rootify(int v) // torna v a raiz de sua arvore
  {
    access(v);
    t[v].rev ^= 1;
  }
  int query(int v, int w) // query no caminho de v ate w
  {
    rootify(w), access(v);
    return t[v].sub;
  }
  void update(int v, int w, int x) // aplica o update em todos os vertices no
       caminho de v ate w
  {
    rootify(w), access(v);
    t[v].lazy = x;
  }
  void link(int v, int w) // adiciona a aresta v - w
  {
    rootify(w);
    t[w].p = v;
  }
  void cut(int v, int w) // remove a aresta v - w
  {
    rootify(w), access(v);
    t[v].ch[0] = t[t[v].ch[0]].p = -1;
  }
  int lca(int v, int w) // acha o lca(v, w)
  {
    access(v);
    return access(w);
  }
};

set<int> adj[MAXN];

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  int x = 0;
  link_cut_tree l;
  for (int i = 0; i < n; i++)
  {
    // l.make_tree(i, i) // poderia fazer isso tbm, para criar o vertice i com
         valor i
    l.make_tree(i);
    l.update(i, i, i); // testa o update
  }
  while (q--)
  {
    int a, b, c;
    cin >> a >> b >> c;
    a = ((a * (1 + x)) % mod) % 2;
    b = ((b * (1 + x)) % mod) % n;
    c = ((c * (1 + x)) % mod) % n;
    if (a == 0)
    {
      adj[b].insert(c);
      adj[c].insert(b);
      l.link(b, c);
    }
    else
    {
      if (!l.connected(b, c))
      {
        x = 0;
        cout << x << endl;
        continue;
      }
      int d = l.query(b, c);
      d ^= b;
      d ^= c;
      x = 0;
      if (d >= 0 && d < n)
      {
        bool ok = 1;
        ok &= (adj[b].find(d) != adj[b].end());
        ok &= (adj[c].find(d) != adj[c].end());
        if (ok)
          x = d + 1;
      }
      cout << x << endl;
    }
  }
}
// problema exemplo: https://atcoder.jp/contests/abc350/tasks/abc350_g
// nesse caso, a link cut tree tem lazy para assign de valor
// e de queries de xor
```

4.33 MatrixDijkstra

```
#include <bits/stdc++.h>
using namespace std ;

#define lli long long int
#define pb push_back
#define MAXN 10000000
typedef pair <int , int> pii ;

int t ;
int dist [MAXN] ;
bool visited [MAXN] ;
vector <pii> adj_list [MAXN] ;

void dijkstra (int s)
{
    dist[s] = 0 ;

    priority_queue <pii , vector<pii> , greater<pii>> q ;

    q.push(pii(dist[s], s)) ;

    while(1)
    {
        int davez = -1 ;
        int menor = INT_MAX ;

        while(!q.empty())
```

```
        {
            int atual = q.top().second ;
            q.pop() ;

            if(!visited[atual])
            {
                davez = atual;
                break;
            }

        }

        if(davez == -1)
        {
            break ;
        }

        visited[davez] = true ;

        for(int i = 0 ; i < adj_list[davez].size() ; i++)
        {
            int distt  = adj_list[davez][i].first ;
            int atual = adj_list[davez][i].second ;

            if(dist[atual] > dist[davez] + distt)
            {
                dist[atual] = dist[davez] + distt ;
                q.push(pii(dist[atual] , atual)) ;
            }
        }
    }

}
void initialize ()
{
    for (int i = 0 ; i < t ; i++)
    {
        visited[i] = false ;
        dist[i] = INT_MAX ;
    }
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n , m ;
    cin >> n >> m ;
    t = n * m ;
    char array [t] ;

    for (int i = 0 ; i < t ; i++)
    {
        cin >> array[i] ;
    }

    for (int i = 0 ; i < t ; i++)
    {
        if (i >= m && array[i] != '#')
        {
            adj_list[i].pb(pii(1 , (i - m))) ;
        }
        if (i < (n * m) - m  && array[i] != '#')
        {
            adj_list[i].pb(pii(1 , (i + m))) ;
        }
        if (i % m != 0 && array[i] != '#')
        {
            adj_list[i].pb(pii(1 , (i - 1))) ;
        }
        if ((i + 1) % m != 0  && array[i] != '#')
        {
            adj_list[i].pb(pii(1 , (i + 1))) ;
        }

    }

    int q ;
    cin >> q ;

    while (q--)
    {
        int a , b , c , d , e ;
        cin >> a >> b >> c >> d >> e ;
        a-- , b-- , c-- , d-- ;

        int index1 = (m * a) + b ;
        int index2 = (m * c) + d ;

        adj_list[index1].pb(pii(e , index2)) ;
        adj_list[index2].pb(pii(e , index1)) ;
    }

    initialize () ;

    dijkstra(0) ;

    cout << dist[t - 1] << endl ;

    return 0 ;
}
```

4.34 max matching without one vertex

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define mod 1000000007
#define MAXN 100005
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define pb push_back
#define INF 1e9

struct hopcroft_karp
{
  vector<int> match;
  vector<int> dist;
  vector<vector<int>> adj;
  int n, m, t;

  hopcroft_karp(int a, int b)
  {
    n = a, m = b;
    t = n + m + 1;
    match.assign(t, n + m);
    dist.assign(t, 0);
    adj.assign(t, vector<int>{});
  }
  void add_edge(int u, int v)
  {
    adj[u].pb(v);
    adj[v].pb(u);
  }
  bool bfs()
  {
    queue<int> q;
    for (int u = 0; u < n; u++)
    {
      if (match[u] == n + m)
        dist[u] = 0, q.push(u);
      else
        dist[u] = INF;
    }
    dist[n + m] = INF;
    while (!q.empty())
    {
      int u = q.front();
      q.pop();
      if (dist[u] < dist[n + m])
      {
```

```
      for (auto const &v : adj[u])
      {
        if (dist[match[v]] == INF)
        {
          dist[match[v]] = dist[u] + 1;
          q.push(match[v]);
        }
      }
    }
  }
  return dist[n + m] < INF;
}
bool dfs(int u)
{
  if (u < n + m)
  {
    for (auto const &v : adj[u])
    {
      if (dist[match[v]] == dist[u] + 1 && dfs(match[v]))
      {
        match[v] = u;
        match[u] = v;
        return true;
      }
    }
    dist[u] = INF;
    return false;
  }
  return true;
}
vector<pi> run()
{
  int cnt = 0;
  while (bfs())
    for (int u = 0; u < n; u++)
      if (match[u] == n + m && dfs(u))
        cnt++;
  vector<pi> ans;
  for (int v = n; v < n + m; v++)
    if (match[v] < n + m)
      ans.pb({match[v], v});
  return ans;
}
vector<int> solve()
{
  vector<pi> ans = run();
  vector<bool> vis(n + m, 0);
  vector<bool> can_remove(n + m, 0);
  for (int i = 0; i < n; i++)
  {
    if (match[i] == n + m)
    {
      queue<int> q;
      q.push(i);
      while (!q.empty())
      {
        int x = q.front();
        q.pop();
        vis[x] = 1;
        can_remove[x] = 1;
        for (auto const &y : adj[x])
        {
          if (!vis[y])
          {
            vis[y] = 1;
            q.push(match[y]);
          }
        }
      }
    }
  }
  vis = vector<bool>(n + m, 0);
  for (int i = n; i < n + m; i++)
  {
    if (match[i] == n + m)
    {
      queue<int> q;
      q.push(i);
      while (!q.empty())
      {
        int x = q.front();
        q.pop();
        vis[x] = 1;
        can_remove[x] = 1;
        for (auto const &y : adj[x])
        {
          if (!vis[y])
          {
            vis[y] = 1;
            q.push(match[y]);
          }
        }
      }
    }
  }
  vector<int> resp;
  for (int i = 0; i < n + m; i++)
  {
    if (can_remove[i])
      resp.pb(i);
  }
  return resp;
 }
};

int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, 1, -1};

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<pi> v(n);
  vector<int> sz(2, 0);
  map<pi, int> mp;
  map<int, pi> rev;
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].fir >> v[i].sec;
    int id = sz[abs(v[i].fir + v[i].sec) % 2]++;
    mp[v[i]] = id;
  }
  for (int i = 0; i < n; i++)
  {
    if (abs(v[i].fir + v[i].sec) % 2)
    {
      mp[v[i]] += sz[0];
    }
    rev[mp[v[i]]] = v[i];
  }
  vector<pi> edges;
  for (auto [i, j] : v)
  {
    if (!(abs(i + j) % 2))
    {
      for (int dir = 0; dir < 4; dir++)
      {
        int x = i + dx[dir];
        int y = j + dy[dir];
        if (mp.find({x, y}) != mp.end())
        {
          edges.pb({mp[{i, j}], mp[{x, y}]});
        }
      }
    }
  }
  hopcroft_karp h(sz[0], sz[1]);
  for (auto [x, y] : edges)
  {
    h.add_edge(x, y);
  }
  vector<int> ans = h.solve();
  cout << ans.size() << endl;
  vector<pi> sorted;
  for (auto const &i : ans)
  {
```

```
    sorted.pb(rev[i]);
  }
  sort(sorted.begin(), sorted.end());
  for (auto [x, y] : sorted)
  {
    cout << x << " " << y << endl;
  }
}
// https://codeforces.com/group/TFrGcBYYxs/contest/583964/problem/E
// https://codeforces.com/gym/105053/problem/C
// https://codeforces.com/group/TFrGcBYYxs/contest/584239/problem/B

// dado um grafo bipartido
// basicamente responde o seguinte problema para cada vertice v:
// apos remover v do grafo, o tamanho do max matching continua o mesmo ou
     diminui em um?
```

4.35 mincostflow

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 301
#define mod 1000000007
#define INF 1e9

namespace mcf
{
  struct edge
  {
    int to, capacity, cost, res;
  };

  int source, destiny;
  vector<edge> adj[MAXN];
  vector<int> dist;
  vector<int> parent;
  vector<int> edge_index;
  vector<bool> in_queue;

  void add_edge(int a, int b, int c, int d)
  {
    adj[a].pb({b, c, d, (int)adj[b].size()});       // aresta normal
    adj[b].pb({a, 0, -d, (int)adj[a].size() - 1}); // aresta do grafo residual
  }
  bool dijkstra(int s) // rodando o dijkstra, terei o caminho de custo minimo
  {                    // que eu consigo passando pelas arestas que possuem
      capacidade > 0
    dist.assign(MAXN, INF);
    parent.assign(MAXN, -1);
    edge_index.assign(MAXN, -1);
    in_queue.assign(MAXN, false);
    dist[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty())
    {
      int u = q.front(), idx = 0;
      q.pop();
      in_queue[u] = false;
      for (auto const &v : adj[u])
      {
        if (v.capacity && dist[v.to] > dist[u] + v.cost)
        {
          dist[v.to] = dist[u] + v.cost;
          parent[v.to] = u;
          edge_index[v.to] = idx;
          if (!in_queue[v.to])
          {
            in_queue[v.to] = true;
            q.push(v.to);
          }
        }
        idx++;
      }
    }
    return dist[destiny] != INF; // se eu cheguei em destiny por esse caminho,
        ainda posso passar fluxo
  }
  int get_cost()
  {
    int flow = 0, cost = 0;
    while (dijkstra(source)) // rodo um dijkstra para saber qual o caminho que
        irei agora
    {
      int curr_flow = INF, curr = destiny;
      while (curr != source) // com isso, vou percorrendo o caminho encontrado
          para achar a aresta "gargalo"
      {
        int p = parent[curr];
        curr_flow = min(curr_flow, adj[p][edge_index[curr]].capacity);
        curr = p;
      }
      flow += curr_flow;                // fluxo que eu posso passar por esse
          caminho = custo da aresta "gargalo"
      cost += curr_flow * dist[destiny]; // quanto eu gasto para passar esse
          fluxo no caminho encontrado
      curr = destiny;
      while (curr != source) // apos achar a aresta gargalo, passamos o fluxo
          pelo caminho encontrado
      {
        int p = parent[curr];
        int res_idx = adj[p][edge_index[curr]].res;
        adj[p][edge_index[curr]].capacity -= curr_flow;
        adj[curr][res_idx].capacity += curr_flow;
        curr = p;
      }
    }
    return cost; // ao final temos a resposta :)
  }
} // namespace mcf
signed main()
{
  int n;
  cin >> n;
  int v[n][n];
  mcf::source = 0, mcf::destiny = (2 * n) + 1;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      cin >> v[i][j];
      mcf::add_edge(i + 1, j + n + 1, 1, v[i][j]);
    }
  }
  for (int i = 1; i <= n; i++)
    mcf::add_edge(mcf::source, i, 1, 0);
  for (int i = n + 1; i <= n + n; i++)
    mcf::add_edge(i, mcf::destiny, 1, 0);
  cout << mcf::get_cost << endl;
}
```

4.36 mo dsu

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
```

```cpp
template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 50001
#define mod 1000000007

const int block = 224;

struct query
{
  int l, r, i;
};

vector<query> queries[block];

namespace dsu
{
  struct rollback
  {
    int u, v, ranku, rankv;
  };

  int num_sets;
  int parent[MAXN];
  int rank[MAXN];
  stack<rollback> op;

  int Find(int i)
  {
    return (parent[i] == i) ? i : Find(parent[i]);
  }
  bool Union(int x, int y, bool can_rollback)
  {
    int xx = Find(x);
    int yy = Find(y);
    if (xx != yy)
    {
      num_sets--;
      if (rank[xx] > rank[yy])
        swap(xx, yy);
      if (can_rollback)
        op.push({xx, yy, rank[xx], rank[yy]});
      parent[xx] = yy;
      if (rank[xx] == rank[yy])
        rank[yy]++;
      return true;
    }
    return false;
  }
  void do_rollback()
  {
    if (op.empty())
      return;
    rollback x = op.top();
    op.pop();
    num_sets++;
    parent[x.v] = x.v;
    rank[x.v] = x.rankv;
    parent[x.u] = x.u;
    rank[x.u] = x.ranku;
  }
  void rollback_all()
  {
    while (!op.empty())
      do_rollback();
  }
  void init(int n)
  {
    for (int i = 0; i < n; i++)
    {
      parent[i] = i;
      rank[i] = 0;
    }
    num_sets = n;
  }
} // namespace dsu
bool cmp(query a, query b)
{
  return a.r < b.r;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  vector<pi> edges(m);
  for (int i = 0; i < m; i++)
  {
    cin >> edges[i].fir >> edges[i].sec;
    edges[i].fir--;
    edges[i].sec--;
  }
  int q;
  cin >> q;
  for (int i = 0; i < q; i++)
  {
    int l, r;
    cin >> l >> r;
    l--, r--;
    queries[l / block].pb({l, r, i});
  }
  for (int i = 0; i < block; i++)
  {
    if (queries[i].size())
      sort(queries[i].begin(), queries[i].end(), cmp);
  }
  vector<int> ans(q);
  for (int i = 0; i < block; i++)
  {
    if (!queries[i].size())
      continue;
    dsu::init(n);
    int limit = (i + 1) * block;
    for (auto const &j : queries[i])
    {
      while (j.r >= limit)
        dsu::Union(edges[limit].fir, edges[limit].sec, false), limit++;
      for (int k = j.l; k <= min(((i + 1) * block) - 1, j.r); k++)
        dsu::Union(edges[k].fir, edges[k].sec, true);
      ans[j.i] = dsu::num_sets;
      dsu::rollback_all();
    }
  }
  for (auto const &i : ans)
    cout << i << endl;
  return 0;
}
// https://codeforces.com/edu/course/2/lesson/7/3/practice/contest/289392/
    problem/B
// temos que fazer algo parecido com um mo algorithm
// sendo que a operacao eh um union/rollback do dsu
// podemos aplicar a seguinte ideia:
// - separamos os queries em blocos (pelo l) de tamanho sqrt(n)
// - para cada bloco, ordenamos esse bloco em ordem crescente do r
// - com isso, em cada query eu posso fazer:
// - de l ate (limite daquele bloco) - 1, adiciono na marra, podendo dar
    rollback
// - como o r eh crescente, para os valores de r que forem maior do que o limit
    daquele block
// - eu ja posso deixar adicionado para sempre sem precisar dar rollback
// fica algo que funciona em coisas que voce pode dar rollback
```

4.37 mo trees

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
```

```cpp
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 998244353

struct qry
{
  int l, r, lca, id;
};

int n, q;
vector<int> adj[MAXN];
int v[MAXN];
int cnt[MAXN];
int freq[MAXN];
int tin[MAXN];
int tout[MAXN];
int depth[MAXN];
int up[MAXN][25];
vector<int> t;
vector<qry> qq;

void dfs(int s, int p)
{
  tin[s] = t.size();
  up[s][0] = p;
  for (int i = 1; i < 25; i++)
    up[s][i] = up[up[s][i - 1]][i - 1];
  t.pb(s);
  for (auto const &i : adj[s])
  {
    if (i == p)
      continue;
    depth[i] = depth[s] + 1;
    dfs(i, s);
  }
  tout[s] = t.size();
  t.pb(s);
}
bool is(int u, int v)
{
  return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v)
{
  if (is(u, v))
    return u;
  if (is(v, u))
    return v;
  for (int i = 24; i >= 0; i--)
  {
    if (!is(up[u][i], v))
      u = up[u][i];
  }
  return up[u][0];
}
void compress()
{
  vector<int> vals;
  for (int i = 0; i < n; i++)
    vals.pb(v[i]);
  sort(vals.begin(), vals.end());
  vals.erase(unique(vals.begin(), vals.end()), vals.end());
  for (int i = 0; i < n; i++)
    v[i] = lower_bound(vals.begin(), vals.end(), v[i]) - vals.begin();
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  while (cin >> n >> q)
  {
    t.clear();
    qq.clear();
    depth[0] = 0;
    memset(cnt, 0, sizeof(cnt));
    memset(freq, 0, sizeof(freq));
    for (int i = 0; i < n; i++)
    {
      adj[i].clear();
      cin >> v[i];
    }
    compress();
    for (int i = 0; i < n - 1; i++)
    {
      int a, b;
      cin >> a >> b;
      a--, b--;
      adj[a].pb(b);
      adj[b].pb(a);
    }
    dfs(0, 0);
    for (int i = 0; i < q; i++)
    {
      int x, y;
      cin >> x >> y;
      x--, y--;
      int l = lca(x, y);
      if (tin[x] > tin[y])
        swap(x, y);
      if (l == x)
        qq.pb({tin[x], tin[y], -1, i});
      else
        qq.pb({tout[x], tin[y], l, i});
    }
    int block = sqrt(n) + 1;
    auto cmp = [&](qry x, qry y)
    {
      if (x.l / block != y.l / block)
        return x.l / block < y.l / block;
      return x.r < y.r;
    };
    sort(qq.begin(), qq.end(), cmp);
    vector<int> ans(q);
    int cl = 0, cr = 0, resp = 0;
    auto add2 = [&](int x)
    {
      freq[v[x]]++;
      if (freq[v[x]] == 1)
        resp++;
    };
    auto rem2 = [&](int x)
    {
      freq[v[x]]--;
      if (freq[v[x]] == 0)
        resp--;
    };
    auto add = [&](int x)
    {
      cnt[x]++;
      if (cnt[x] == 2)
        rem2(x);
      else
        add2(x);
    };
    auto rem = [&](int x)
    {
      cnt[x]--;
      if (cnt[x] == 1)
        add2(x);
      else
        rem2(x);
    };
    for (int i = 0; i < q; i++)
    {
      int idx = qq[i].id;
```

```
      int l = qq[i].l;
      int r = qq[i].r;
      int lc = qq[i].lca;
      while (cl < l)
        rem(t[cl++]);
      while (cl > l)
        add(t[--cl]);
      while (cr <= r)
        add(t[cr++]);
      while (cr > r + 1)
        rem(t[--cr]);
      if (lc != -1)
        add(lc);
      ans[idx] = resp;
      if (lc != -1)
        rem(lc);
    }
    for (auto const &i : ans)
      cout << i << endl;
  }
  return 0;
}
// https://www.spoj.com/problems/COT2/
// quantos caras distintos em um path entre u e v
// mo em arvores
// acha o euler tour da arvore com tin e tout
// desconsidera no mo os indices duplicados no range

// para queries em subtree eh mais simples:
// apenas saber o tamanho da subtree de i
// fazer o euler tour apenas com o tin
// e fzr a query pro range tin[i] ate tin[i] + sz[i] - 1

// pra queries de path com peso nos edges:
// https://codeforces.com/gym/100962/attachments (problema F)
// considera v[i] -> peso do edge que liga ao meu pai na arvore
// dai pra query com o lca == u, nao tenho que considerar v[u] ([tin[u], tin[v
    ]], dps removendo v[u])
// e pra query com o lca != u, so fazer ela normalmente ([tout[u], tin[v]])
```

4.38 mo trees edges

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200001
#define mod 998244353

struct qry
{
  int l, r, ini, id;
};

int n, q;
vector<pi> adj[MAXN];
int v[MAXN];
int cnt[MAXN];
int freq[MAXN];
int in_block[MAXN];
int tin[MAXN];
int tout[MAXN];
int depth[MAXN];
int up[MAXN][25];
vector<int> t;
vector<qry> qq;

void dfs(int s, int p, int par_edge)
{
  v[s] = par_edge;
  tin[s] = t.size();
  up[s][0] = p;
  for (int i = 1; i < 25; i++)
    up[s][i] = up[up[s][i - 1]][i - 1];
  t.pb(s);
  for (auto const &i : adj[s])
  {
    if (i.fir == p)
      continue;
    depth[i.fir] = depth[s] + 1;
    dfs(i.fir, s, i.sec);
  }
  tout[s] = t.size();
  t.pb(s);
}
bool is(int u, int v)
{
  return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v)
{
  if (is(u, v))
    return u;
  if (is(v, u))
    return v;
  for (int i = 24; i >= 0; i--)
  {
    if (!is(up[u][i], v))
      u = up[u][i];
  }
  return up[u][0];
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> q;
  for (int i = 0; i < n - 1; i++)
  {
    int a, b, c;
    cin >> a >> b >> c;
    a--, b--;
    adj[a].pb({b, c});
    adj[b].pb({a, c});
  }
  dfs(0, 0, 0);
  for (int i = 0; i < q; i++)
  {
    int x, y;
    cin >> x >> y;
    x--, y--;
    int l = lca(x, y);
    if (tin[x] > tin[y])
      swap(x, y);
    if (l == x)
      qq.pb({tin[x], tin[y], x, i});
    else
      qq.pb({tout[x], tin[y], -1, i});
  }
  int block = sqrt(n) + 1;
  auto cmp = [&](qry x, qry y)
  {
    if (x.l / block != y.l / block)
      return x.l / block < y.l / block;
    return x.r < y.r;
  };
  sort(qq.begin(), qq.end(), cmp);
  vector<int> ans(q);
  int cl = 0, cr = 0, resp = 0;
  auto add2 = [&](int x)
  {
    if (v[x] >= MAXN)
```

```
      return;
    freq[v[x]]++;
    if (freq[v[x]] == 1)
      in_block[v[x] / block]++;
  };
  auto rem2 = [&](int x)
  {
    if (v[x] >= MAXN)
      return;
    freq[v[x]]--;
    if (freq[v[x]] == 0)
      in_block[v[x] / block]--;
  };
  auto add = [&](int x)
  {
    cnt[x]++;
    if (cnt[x] == 2)
      rem2(x);
    else
      add2(x);
  };
  auto rem = [&](int x)
  {
    cnt[x]--;
    if (cnt[x] == 1)
      add2(x);
    else
      rem2(x);
  };
  for (int i = 0; i < q; i++)
  {
    int idx = qq[i].id;
    int l = qq[i].l;
    int r = qq[i].r;
    int ini = qq[i].ini;
    while (cl < l)
      rem(t[cl++]);
    while (cl > l)
      add(t[--cl]);
    while (cr <= r)
      add(t[cr++]);
    while (cr > r + 1)
      rem(t[--cr]);
    if (ini != -1)
      rem(ini);
    for (int b = 0;; b++)
    {
      if (in_block[b] != block)
      {
        ans[idx] = b * block;
        while (freq[ans[idx]])
          ans[idx]++;
        break;
      }
    }
    if (ini != -1)
      add(ini);
  }
  for (auto const &i : ans)
    cout << i << endl;
  return 0;
}
// https://codeforces.com/gym/100962/attachments (problema F)
// mo em arvore com peso nos edges

// nesse problema em especifico: dado uma arvore, responder queries de mex
// no caminho entre u e v, considerando os pesos de arestas no caminho de u pra
     v

// mo em arvores
// acha o euler tour da arvore com tin e tout
// desconsidera no mo os indices duplicados no range
// e bem parecido com o de peso nos vertices
// considera v[i] -> peso do edge que liga ao meu pai na arvore
// dai pra query com o lca == u, nao tenho que considerar v[u] ([tin[u], tin[v
     ]], dps removendo v[u])
// e pra query com o lca != u, so fazer ela normalmente ([tout[u], tin[v]])
```

4.39 Prim

```
// algoritimo de prim

// 1 - definir a distancia de cada vertice como infinito (similar ao dijkstra).
// 2 - definir a distancia de 0 para o source(0).
// 3 - Em cada passo, encontrar o vertice u, que ainda nao foi processado, que
     possua a menor das distancias.
// 4 - ao termino fazer a soma de todas as distancias e encontrar qual a soma
     das distancias na MST.

#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define pii pair<int, int>
#define mp make_pair
#define MAXN 100001
#define INF 999999
#define sec second
#define fir first

int n, m, a, b, c;
vector<pii> adj[MAXN];
int dist[MAXN];
bool processed[MAXN];

void prim()
{
  for (int i = 0; i < n; i++)
  {
    dist[i] = INF;
  }

  dist[0] = 0;

  priority_queue<pii, vector<pii>, greater<pii>> q;
  q.push(pii(dist[0], 0));

  while (1)
  {
    int davez = -1;

    while (!q.empty())
    {
      int atual = q.top().sec;
      q.pop();

      if (!processed[atual])
      {
        davez = atual;
        break;
      }
    }

    if (davez == -1)
    {
      break;
    }

    processed[davez] = true;

    for (int i = 0; i < adj[davez].size(); i++)
    {
      int distt = adj[davez][i].fir;
      int atual = adj[davez][i].sec;

      if (dist[atual] > distt && !processed[atual])
      {
        dist[atual] = distt;
        q.push(pii(dist[atual], atual));
      }
    }
  }
```

```
  int ans = 0;

  for (int i = 0; i < n; i++)
  {
    ans += dist[i];
  }

  cout << ans << endl;
}
int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  cin >> n >> m;

  for (int i = 0; i < m; i++)
  {
    cin >> a >> b >> c;
    a--;
    b--;
    adj[a].pb(mp(c, b));
    adj[b].pb(mp(c, a));
  }

  prim();

  return 0;
}
```

4.40 push relabel

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 1000000007
#define INF 1e9

struct edge
{
  int dest, back, f, c, id;
};
struct push_relabel
{
  int n;
  vector<vector<edge>> g;
  vector<int> ec;
  vector<edge *> cur;
  vector<vector<int>> hs;
  vector<int> H;
  push_relabel(int sz) : g(sz), ec(sz), cur(sz), hs(2 * sz), H(sz) { n = sz; }
  void add_edge(int s, int t, int cap, int rcap, int id)
  {
    if (s == t)
      return;
    g[s].pb({t, (int)g[t].size(), 0, cap, id});
    g[t].pb({s, (int)g[s].size() - 1, 0, rcap, -1});
  }
  void add_flow(edge &e, int f)
  {
    edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f)
      hs[H[e.dest]].push_back(e.dest);
    e.f += f;
    e.c -= f;
    ec[e.dest] += f;
    back.f -= f;
    back.c += f;
    ec[back.dest] -= f;
  }
  int calc(int s, int t)
  {
    int v = g.size();
    H[s] = v;
    ec[t] = 1;
    vector<int> co(2 * v);
    co[0] = v - 1;
    for (int i = 0; i < v; i++)
      cur[i] = g[i].data();
    for (edge &e : g[s])
      add_flow(e, e.c);
    for (int hi = 0;;)
    {
      while (hs[hi].empty())
        if (!hi--)
          return -ec[s];
      int u = hs[hi].back();
      hs[hi].pop_back();
      while (ec[u] > 0)
      {
        if (cur[u] == g[u].data() + g[u].size())
        {
          H[u] = INF;
          for (edge &e : g[u])
            if (e.c && H[u] > H[e.dest] + 1)
              H[u] = H[e.dest] + 1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            for (int i = 0; i < v; i++)
              if (hi < H[i] && H[i] < v)
                --co[H[i]], H[i] = v + 1;
          hi = H[u];
        }
        else if (cur[u]->c && H[u] == H[cur[u]->dest] + 1)
          add_flow(*cur[u], min(ec[u], cur[u]->c));
        else
          ++cur[u];
      }
    }
  }
  vector<int> flow_edges(int m) // fluxo em cada aresta
  {
    vector<int> ans(m);
    for (int i = 0; i < n; i++)
    {
      for (auto const &j : g[i])
      {
        if (j.id != -1)
          ans[j.id] = j.f;
      }
    }
    return ans;
  }
};
struct flow_with_demands
{
  push_relabel pr;
  vector<int> in, out;
  int n;

  flow_with_demands(int sz) : n(sz), pr(sz + 2), in(sz), out(sz) {}
  void add_edge(int u, int v, int cap, int dem, int id)
  {
    pr.add_edge(u, v, cap - dem, 0, id);
    out[u] += dem, in[v] += dem;
  }
  int run(int s, int t)
  {
    pr.add_edge(t, s, INF, 0, -1);
    for (int i = 0; i < n; i++)
    {
```

```
      pr.add_edge(n, i, in[i], 0, -1);
      pr.add_edge(i, n + 1, out[i], 0, -1);
    }
    return pr.calc(n, n + 1);
  }
  bool check() // todas as constraints foram satisfeitas?
  {
    for (auto const &i : pr.g[n])
    {
      if (i.c > 0)
        return 0;
    }
    return 1;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<pi> v(n);
  vector<int> x_vals, y_vals;
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].fir >> v[i].sec;
    x_vals.pb(v[i].fir);
    y_vals.pb(v[i].sec);
  }
  sort(x_vals.begin(), x_vals.end());
  sort(y_vals.begin(), y_vals.end());
  x_vals.erase(unique(x_vals.begin(), x_vals.end()), x_vals.end());
  y_vals.erase(unique(y_vals.begin(), y_vals.end()), y_vals.end());
  int xx = x_vals.size();
  int yy = y_vals.size();
  vector<int> cntx(xx, 0);
  vector<int> cnty(yy, 0);
  for (int i = 0; i < n; i++)
  {
    v[i].fir = lower_bound(x_vals.begin(), x_vals.end(), v[i].fir) - x_vals.
        begin();
    v[i].sec = lower_bound(y_vals.begin(), y_vals.end(), v[i].sec) - y_vals.
        begin();
    cntx[v[i].fir]++;
    cnty[v[i].sec]++;
  }
  flow_with_demands mf(xx + yy + 2);
  int src = xx + yy;
  int sink = xx + yy + 1;
  int edge_id = 0;
  for (int i = 0; i < xx; i++)
  {
    int half = cntx[i] / 2;
    int can_pass = cntx[i] - half;
    mf.add_edge(src, i, can_pass, half, edge_id++);
  }
  for (int i = 0; i < yy; i++)
  {
    int half = cnty[i] / 2;
    int can_pass = cnty[i] - half;
    mf.add_edge(xx + i, sink, can_pass, half, edge_id++);
  }
  vector<int> middle_edges(n);
  for (int i = 0; i < n; i++)
  {
    middle_edges[i] = edge_id;
    mf.add_edge(v[i].fir, xx + v[i].sec, 1, 0, edge_id++);
  }
  mf.run(src, sink);
  assert(mf.check());
  vector<int> flow_edges = mf.pr.flow_edges(edge_id);
  for (int i = 0; i < n; i++)
  {
    if (flow_edges[middle_edges[i]])
      cout << "L";
    else
      cout << "F";
  }
  cout << endl;
}
// https://codeforces.com/group/TFrGcBYYxs/contest/584448/problem/D
// questao de flow com demands da summer school
// grafo bipartido com n = 10^5
// e esse push relabel do kactl passa, incrivel kkkkkkkkkkk
// https://github.com/kth-competitive-programming/kactl/blob/main/content/graph/
    PushRelabel.h
// obrigado kactl
```

4.41 reroot

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 200001
#define mod 1000000007

int n;
vector<int> adj[MAXN];
int sz[MAXN];
int dp[MAXN];

int dfs(int u, int v)
{
  sz[u] = 1;
  for (auto const &i : adj[u])
    if (i != v)
      sz[u] += dfs(i, u);
  return sz[u];
}
void reroot(int u, int v)
{
  for (auto const &i : adj[u])
  {
    if (i != v)
    {
      int a = sz[u], b = sz[i];
      dp[i] = dp[u];
      dp[i] -= sz[u], dp[i] -= sz[i];
      sz[u] -= sz[i], sz[i] = n;
      dp[i] += sz[u], dp[i] += sz[i];
      reroot(i, u);
      sz[u] = a, sz[i] = b;
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n;
  for (int i = 0; i < n - 1; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  dfs(0, -1);
  for (int i = 0; i < n; i++)
```

```
    dp[0] += sz[i]; // answer when tree is rooted on vertex 0
  reroot(0, -1);
  cout << *max_element(dp, dp + n) << endl;
  return 0;
}
// https://codeforces.com/contest/1187/problem/E
// f(v) = when tree is rooted at vertex v, the current
// answer is the sum of all subtrees sizes
// final answer = max(f(0), f(1), f(2), ..., f(n))
// easy approach: O(N^2)
// with reroot: O(N)
// 1 - run a dfs and calculate f(0)
// 2 - let be dp[i] = f(i)
// 3 - now, lets run a another dfs, and re-calculate the
// answer when tree is rooted at vertex i (dp[i])
// 4 - the final answer is the maximum value of dp[i]
```

4.42 rmq tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 5005
#define mod 998244353

struct lca
{
  int n;
  vector<vector<int>> adj;
  vector<int> vec;
  int l, timer;
  vector<int> tin, tout, depth;
  vector<vector<pi>> up;

  void dfs(int v, int p)
  {
    tin[v] = ++timer;
    up[v][0] = {p, min(vec[v], vec[p])};
    for (int i = 1; i <= l; i++)
    {
      up[v][i].fir = up[up[v][i - 1].fir][i - 1].fir;
      up[v][i].sec = min(up[v][i - 1].sec, up[up[v][i - 1].fir][i - 1].sec);
    }
    for (auto const &u : adj[v])
    {
      if (p == u)
        continue;
      depth[u] = depth[v] + 1;
      dfs(u, v);
    }
    tout[v] = ++timer;
  }
  bool is_ancestor(int u, int v)
  {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
  }
  int find_lca(int u, int v)
  {
    if (is_ancestor(u, v))
      return u;
    if (is_ancestor(v, u))
      return v;
    for (int i = l; i >= 0; i--)
      if (!is_ancestor(up[u][i].fir, v))
        u = up[u][i].fir;
    return up[u][0].fir;
  }
  int dist(int s, int v)
  {
    int at = find_lca(s, v);
    return (depth[s] + depth[v] - 2 * depth[at]);
  }
  int solve(int u, int d)
  {
    int ans = vec[u];
    for (int i = l; i >= 0; i--)
    {
      if (d & (1 << i))
      {
        ans = min(ans, up[u][i].sec);
        u = up[u][i].fir;
      }
    }
    return ans;
  }
  int rmq(int u, int v)
  {
    int l = find_lca(u, v);
    return min(solve(u, dist(u, l)), solve(v, dist(v, l)));
  }
  lca(vector<vector<int>> &_adj, vector<int> &_vec)
  {
    adj = _adj;
    vec = _vec;
    n = adj.size();
    tin.resize(n);
    tout.resize(n);
    depth.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<pi>(l + 1));
    dfs(0, 0);
  }
};
signed main()
{
}
// valores nos vertices
// rmq considerando o caminho entre os vertices u e v
```

4.43 sack

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

vector<int> adj[MAXN];
vector<int> v[MAXN];
int c[MAXN];
int cnt[MAXN];
int sz[MAXN];

void dfs_sz(int x, int p)
{
  sz[x] = 1;
```

```
  for (auto const &i : adj[x])
  {
    if (i != p)
    {
      dfs_sz(i, x);
      sz[x] += sz[i];
    }
  }
}
void modify(int c, int val)
{
  cnt[c] += val;
}
void dfs(int x, int p, bool keep)
{
  int best = -1, big_child = -1;
  for (auto const &i : adj[x])
  {
    if (i != p && sz[i] > best)
    {
      best = sz[i];
      big_child = i;
    }
  }
  for (auto const &i : adj[x])
  {
    if (i != p && i != big_child)
      dfs(i, x, 0);
  }
  if (big_child != -1)
  {
    dfs(big_child, x, 1);
    swap(v[x], v[big_child]); // O(1)
  }
  v[x].pb(x);
  modify(c[x], 1); // adiciona
  for (auto const &i : adj[x])
  {
    if (i != p && i != big_child)
    {
      for (auto const &j : v[i])
      {
        v[x].pb(j);
        modify(c[j], 1); // adiciona
      }
    }
  }
  // a cor c aparece cnt[c] vezes na subtree de x
  // dai vc pode fazer algo tendo essa informacao
  // seja responser queries ou algo do tipo aqui
  if (!keep)
  {
    for (auto const &i : v[x])
      modify(c[i], -1); // remove
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  for (int i = 0; i < n; i++)
    cin >> c[i];
  for (int i = 0; i < n - 1; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  dfs_sz(0, -1);
  dfs(0, -1, 0);
  cout << endl;
}
// https://codeforces.com/blog/entry/44351
// https://codeforces.com/blog/entry/67696
// problema motivacao:
// dada uma arvore
// cada vertice tem uma cor
// tenho consultas do tipo: quantos caras na subtree de v tem cor == x
// com sack da pra resolver isso em O(n * log(n)) (complexidade do dfs do sack)

// para outros problemas, basta mudar a funcao modify
// muito util em problemas em que vc precisa guardar algo da subarvore de v,
    para todo v
// seja pra resolver queries offline ou algo do tipo
```

4.44 scc

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 1000000007

int n, m;

bool vis[MAXN];
int root[MAXN];
vector<int> order;
vector<int> roots;
vector<int> comp;
vector<vector<int>> comps;
vector<int> adj[MAXN];
vector<int> adj_rev[MAXN];
vector<int> adj_scc[MAXN];

void dfs(int v)
{
  vis[v] = true;
  for (auto const &u : adj[v])
    if (!vis[u])
      dfs(u);
  order.pb(v);
}
void dfs2(int v)
{
  comp.pb(v);
  vis[v] = true;
  for (auto const &u : adj_rev[v])
    if (!vis[u])
      dfs2(u);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    adj[a].pb(b);
    adj_rev[b].pb(a);
  }
  for (int i = 0; i < n; i++)
  {
    if (!vis[i])
      dfs(i);
```

```
  }
  reverse(order.begin(), order.end());
  memset(vis, false, sizeof(vis));
  for (auto const &v : order)
  {
    if (!vis[v])
    {
      comp.clear();
      dfs2(v);
      comps.pb(comp);
      // making condensation graph
      /*
      int r = comp.back();
      for (auto const &u : comp)
        root[u] = r;
      roots.push_back(r);
      */
    }
  }
  // making condensation graph
  /*
  for (int v = 0; v < n; v++)
  {
    for (auto const &u : adj[v])
    {
      int root_v = roots[v];
      int root_u = roots[u];
      if (root_u != root_v)
        adj_scc[root_v].pb(root_u);
    }
  }
  */
  // printing scc
  cout << comps.size() << endl;
  for (auto const &comp : comps)
  {
    cout << comp.size() << " ";
    for (auto const &u : comp)
      cout << u << " ";
    cout << endl;
  }
  return 0;
}
// to test: https://judge.yosupo.jp/problem/scc
```

4.45 segtree graph

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 998244353

struct segtree_graph
{
  int n;
  vector<vector<int>> adj;
  vector<int> id_rev, id, deg;

  segtree_graph(int sz)
  {
    n = sz;
    adj.resize(4 * n);
    id.resize(n);
    id_rev.assign(4 * n, -1);
    deg.assign(4 * n, 0);
    build(0, n - 1, 1);
  }
  void build(int l, int r, int i)
  {
    if (l == r)
    {
      id[l] = i;
      id_rev[i] = l;
      return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, i << 1);
    build(mid + 1, r, (i << 1) | 1);
    adj[i << 1].pb(i);
    adj[(i << 1) | 1].pb(i);
    deg[i] += 2;
  }
  void add(int l, int r, int ql, int qr, int i, int x)
  {
    if (l > r || l > qr || r < ql)
    {
      return;
    }
    if (l >= ql && r <= qr)
    {
      adj[i].pb(x);
      deg[x]++;
      return;
    }
    int mid = (l + r) >> 1;
    add(l, mid, ql, qr, i << 1, x);
    add(mid + 1, r, ql, qr, (i << 1) | 1, x);
  }
  void add2(int a, int b)
  {
    adj[a].pb(b);
    deg[b]++;
  }
  void topological_sort()
  {
    vector<bool> vis(4 * n, 0);
    queue<int> q;
    for (int i = 0; i < (4 * n); i++)
    {
      if (!deg[i])
        q.push(i);
    }
    int qt = 0, xx = 1;
    vector<int> ans(n);
    while (!q.empty())
    {
      int x = q.front();
      q.pop();
      qt++;
      if (id_rev[x] != -1)
      {
        ans[id_rev[x]] = xx++;
      }
      for (auto const &i : adj[x])
      {
        deg[i]--;
        if (!deg[i])
          q.push(i);
      }
    }
    if (qt != (4 * n))
    {
      cout << "-1\n";
      return;
    }
    for (auto const &i : ans)
    {
      cout << i << " ";
    }
```

```
    cout << endl;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n;
    cin >> n;
    segtree_graph st(n);
    for (int i = 0; i < n; i++)
    {
      int x;
      cin >> x; // next[i]
      if (x != -1)
      {
        x--;
        if (x > (i + 1))
          st.add(0, n - 1, i + 1, x - 1, 1, st.id[i]);
        if (x < n)
          st.add2(st.id[i], st.id[x]);
      }
    }
    st.topological_sort();
  }
}
// https://codeforces.com/contest/1158/problem/C
// se next[i] = j
// entao p[j] > p[i]
// e para todo k tal que: i < k < j, p[k] < p[i]
// em outras palavras o primeiro caba a direita de i que eh maior do que p[i] eh
     no indice j
// montar o grafo de implicacoes e fazer topsort
// pq se next[i] = j
// entao implica que p[k] < p[i] para i < k < j
// e implica que p[i] < p[j]
// mas tem que ser o grafo de segtree ne

// faz algo similar ao problema legacy: https://codeforces.com/problemset/
    problem/786/B
```

4.46 stable matching

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// #define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define pci pair<char, int>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

bool in[1005][1005];
int tt[1005][1005];
int b[1005];
int ini[1005];

// a.size() <= b.size()
vector<int> stable_marriage(vector<vector<int>> &a, vector<vector<int>> &b)
{
  int n = a.size(), m = b.size();
  assert(a[0].size() == m and b[0].size() == n and n <= m);
  vector<int> match(m, -1), it(n, 0);
  vector<vector<int>> inv_b(m, vector<int>(n));
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      inv_b[i][b[i][j]] = j;
  queue<int> q;
  for (int i = 0; i < n; i++)
    q.push(i);
  while (q.size())
  {
    int i = q.front();
    q.pop();
    int j = a[i][it[i]];
    if (match[j] == -1)
      match[j] = i;
    else if (inv_b[j][i] < inv_b[j][match[j]])
    {
      q.emplace(match[j]);
      it[match[j]]++;
      match[j] = i;
    }
    else
      q.emplace(i), it[i]++;
  }
  vector<int> ret(n);
  for (int i = 0; i < m; i++)
    if (match[i] != -1)
      ret[match[i]] = i;
  return ret;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m, d, e;
  cin >> n >> m >> d >> e;
  if (n > m)
  {
    cout << "impossible\n";
    return 0;
  }
  vector<vector<int>> adj(n);
  vector<vector<int>> adj2(m);
  memset(b, -1, sizeof(b));
  memset(ini, -1, sizeof(ini));
  for (int i = 0; i < e; i++)
  {
    int s, k, t;
    cin >> s >> k >> t;
    k--, t--;
    if (t == -1)
    {
      tt[k][b[k]] += (s - ini[k]);
      b[k] = -1;
      ini[k] = -1;
    }
    else
    {
      if (b[k] != -1)
      {
        tt[k][b[k]] += (s - ini[k]);
      }
      if (!in[k][t])
      {
        in[k][t] = 1;
        adj[k].pb(t);
      }
      b[k] = t;
      ini[k] = s;
    }
  }
  for (int k = 0; k < n; k++)
  {
    if (b[k] != -1)
      tt[k][b[k]] += (d - ini[k]);
  }
```

```
  for (int k = 0; k < n; k++)
  {
    for (int t = 0; t < m; t++)
    {
      if (!in[k][t])
        adj[k].pb(t);
    }
  }
  for (int t = 0; t < m; t++)
  {
    vector<pi> curr;
    for (int k = 0; k < n; k++)
    {
      curr.pb({tt[k][t], k});
    }
    sort(curr.begin(), curr.end());
    for (auto const &i : curr)
      adj2[t].pb(i.sec);
  }
  vector<int> ans = stable_marriage(adj, adj2);
  for (auto const &i : ans)
    cout << i + 1 << " ";
  cout << endl;
}
// solucao pro: https://open.kattis.com/problems/jealousyoungsters

// stable marriage
// voce quer achar um matching em um grafo bipartido
// que todos os caras de um lado dao matching com algum do outro lado

// cada vertice tem um vector
// que diz qual a ordem de preferencia dele
// o que ele mais quer dar matching eh com v[0]
// o segundo com que ele mais quer dar matching eh com v[1]
// e assim vai

// quando a.size() <= b.size(), entao sempre existe um stable matching
```

4.47 strong orientation

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000005
#define mod 1000000007

int n, m, timer, comps, bridges;
vector<pi> edges;
vector<pi> adj[MAXN];
int tin[MAXN];
int low[MAXN];
bool vis[MAXN];
char orient[MAXN];

void find_bridges(int v)
{
  low[v] = timer, tin[v] = timer++;
  for (auto const &p : adj[v])
  {
    if (vis[p.sec])
      continue;
    vis[p.sec] = true;
    orient[p.sec] = (v == edges[p.sec].first) ? '>' : '<';
    if (tin[p.fir] == -1)
    {
      find_bridges(p.fir);
      low[v] = min(low[v], low[p.fir]);
      if (low[p.fir] > tin[v])
        bridges++;
    }
    else
    {
      low[v] = min(low[v], low[p.fir]);
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> m;
  for (int i = 0; i < m; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    edges.pb({a, b});
    adj[a].pb({b, i});
    adj[b].pb({a, i});
  }
  memset(tin, -1, sizeof(tin));
  memset(low, -1, sizeof(low));
  for (int v = 0; v < n; v++)
  {
    if (tin[v] == -1)
    {
      comps++;
      find_bridges(v);
    }
  }
  // numero minimo de scc = numero de componentes + numero de pontes
  cout << comps + bridges << endl;
  // > - a aresta foi orientada da esquerda pra direita
  // < - a aresta foi orientada da direita pra esquerda
  for (int i = 0; i < m; i++)
    cout << orient[i];
  cout << endl;
  return 0;
}
// to_test: https://szkopul.edu.pl/problemset/problem/nldsb4EW1YuZykBlf4lcZL1Y/
    site/?key=statement
// strong orientation:
// encontrar uma orientacao para as arestas tal que o numero
// minimo de scc e o menor possivel
```

4.48 Topological Sort

```
#include <bits/stdc++.h>
using namespace std ;

#define lli long long int
#define pb push_back
#define MAXN 10000

int n , m , a , b ;
vector <int> adj [MAXN] ;
int grau [MAXN];
vector <int> order ;

bool topological_sort ()
{
    int ini = 0 ;

    while (ini < order.size())
    {
        int atual = order[ini] ;
        ini++ ;
```

```
        for (int i = 0 ; i < adj[atual].size() ; i++)
        {
            int v = adj[atual][i] ;
            grau[v]-- ;

            if (grau[v] == 0)
            {
                order.pb(v) ;
            }
        }
    }

    return (order.size() == n) ? true : false ;
}
int main ()
{
    ios_base::sync_with_stdio(false) ;
    cin.tie(NULL) ;

    cin >> n >> m ;

    for (int i = 1 ; i <= m ; i++)
    {
        cin >> a >> b ;
        grau[a]++ ;
        adj[b].pb(a) ;
    }

    for (int i = 1 ; i <= n ; i++)
    {
        if (grau[i] == 0)
        {
            order.pb(i) ;
        }
    }

    if (topological_sort())
    {
        for (int i = 0 ; i < order.size() ; i++)
        {
            cout << order[i] << " " ;
        }

        cout << endl ;
    }
    else
    {
        cout << "Impossible\n" ;
    }

    return 0 ;
}
```

4.49 TreeDiameter

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 100001
#define mod 1000000007

int diameter, best;
vector<int> adj[MAXN];
bool visited[MAXN];

void dfs(int s, int c)
{
  if (c > diameter)
  {
    diameter = c;
    best = s;
  }
  visited[s] = true;
  for (auto const &i : adj[s])
    if (!visited[i])
      dfs2(i, c + 1);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
      adj[i].clear();
    for (int i = 0; i < n - 1; i++)
    {
      int a, b;
      cin >> a >> b;
      a--, b--;
      adj[b].pb(a);
      adj[a].pb(b);
    }
    diameter = 0, best = 0;
    memset(visited, false, sizeof(visited));
    dfs(1, 0);                          // achar o vertice mais distante a partir
        do vertice 0
    memset(visited, false, sizeof(visited));
    dfs(best, 0);                       // achar o mais distante a partir do
        primeiro vertice que achamos
    cout << diameter << endl;
  }
  return 0;
}
```

4.50 tree isomorfism

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 501
#define mod 1000000007

int curr_hash = 1;
map<vector<int>, int> mp;

struct hash_tree
{
  pi h;
  int n;
  vector<int> c, sz, large_comp;
  vector<vector<int>> adj;

  hash_tree(vector<vector<int>> &a)
  {
    n = a.size();
```

```
    adj = a;
  }
  void dfs(int s, int p)
  {
    sz[s] = 1;
    large_comp[s] = 0;
    for (auto const &v : adj[s])
    {
      if (v != p)
      {
        dfs(v, s);
        sz[s] += sz[v];
        large_comp[s] = max(large_comp[s], sz[v]);
      }
    }
    large_comp[s] = max(large_comp[s], n - sz[s]);
  }
  int dfs2(int s, int p)
  {
    if (s == -1)
      return -1;
    vector<int> child;
    for (auto const &v : adj[s])
    {
      if (v != p)
        child.pb(dfs2(v, s));
    }
    sort(child.begin(), child.end());
    if (!mp[child])
      mp[child] = curr_hash++;
    return mp[child];
  }
  pi get_hash()
  {
    sz.assign(n, 0);
    large_comp.assign(n, 0);
    dfs(0, -1);
    int best = 1e18;
    for (int i = 0; i < n; i++)
    {
      if (large_comp[i] < best)
      {
        best = large_comp[i];
        c.clear();
      }
      if (large_comp[i] == best)
        c.pb(i);
    }
    while (c.size() < 2)
      c.pb(-1);
    h.fir = dfs2(c[0], -1);
    h.sec = dfs2(c[1], -1);
    if (h.fir > h.sec)
      swap(h.fir, h.sec);
    return h;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n;
    cin >> n;
    vector<vector<int>> a(n);
    vector<vector<int>> b(n);
    for (int i = 0; i < n - 1; i++)
    {
      int x, y;
      cin >> x >> y;
      x--, y--;
      a[x].pb(y);
      a[y].pb(x);
    }
    for (int i = 0; i < n - 1; i++)
    {
      int x, y;
      cin >> x >> y;
      x--, y--;
      b[x].pb(y);
      b[y].pb(x);
    }
    (hash_tree(a).get_hash() == hash_tree(b).get_hash()) ? cout << "YES\n" :
        cout << "NO\n";
  }
  return 0;
}
// https://www.spoj.com/problems/TREEISO/
// https://www.beecrowd.com.br/judge/en/problems/view/1229
// hash de arvores
// para descobrir se duas arvores sao isomorfas

// 1 - achar todos os centroides da arvore (toda arvore tem no maximo 2
    centroides)
// 2 - achar o hashing com a arvore enraizada em cada centroid
// 3 - dai o hashing da arvore eh um pair ordenado, indicando o hashing de cada
    enraizamento no centroid
```

4.51 two sat

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007

struct two_sat
{
  int n;
  vector<vector<int>> g, gr;  // gr is the reversed graph
  vector<int> comp, ord, ans; // comp[v]: ID of the SCC containing node v
  vector<bool> vis;

  two_sat() {}
  two_sat(int sz)
  {
    n = sz;
    g.assign(2 * n, vector<int>());
    gr.assign(2 * n, vector<int>());
    comp.resize(2 * n);
    vis.resize(2 * n);
    ans.resize(2 * n);
  }
  void add_edge(int u, int v)
  {
    g[u].push_back(v);
    gr[v].push_back(u);
  }
  // int x, bool val: if 'val' is true, we take the variable to be x. Otherwise
      we take it to be x's complement (not x).
  void implies(int i, bool f, int j, bool g) // a -> b
  {
    add_edge(i + (f ? 0 : n), j + (g ? 0 : n));
    add_edge(j + (g ? n : 0), i + (f ? n : 0));
  }
  void add_clause_or(int i, bool f, int j, bool g) // At least one of them is
      true
  {
```

```
    add_edge(i + (f ? n : 0), j + (g ? 0 : n));
    add_edge(j + (g ? n : 0), i + (f ? 0 : n));
  }
  void add_clause_xor(int i, bool f, int j, bool g) // only one of them is true
  {
    add_clause_or(i, f, j, g);
    add_clause_or(i, !f, j, !g);
  }
  void add_clause_and(int i, bool f, int j, bool g) // both of them have the
      same value
  {
    add_clause_xor(i, !f, j, g);
  }
  void set(int i, bool f) // Set a variable
  {
    add_clause_or(i, f, i, f);
  }
  void top_sort(int u)
  {
    vis[u] = 1;
    for (auto const &v : g[u])
    {
      if (!vis[v])
        top_sort(v);
    }
    ord.push_back(u);
  }
  void scc(int u, int id)
  {
    vis[u] = 1;
    comp[u] = id;
    for (auto const &v : gr[u])
    {
      if (!vis[v])
        scc(v, id);
    }
  }
  bool solve()
  {
    fill(vis.begin(), vis.end(), 0);
    for (int i = 0; i < 2 * n; i++)
    {
      if (!vis[i])
        top_sort(i);
    }
    fill(vis.begin(), vis.end(), 0);
    reverse(ord.begin(), ord.end());
    int id = 0;
    for (const auto &v : ord)
    {
      if (!vis[v])
        scc(v, id++);
    }
    for (int i = 0; i < n; i++)
    {
      if (comp[i] == comp[i + n])
        return 0;
      ans[i] = (comp[i] > comp[i + n]) ? 1 : 0;
    }
    return 1;
  }
};
signed main()
{
}
//  https://codeforces.com/blog/entry/92977
//  https://codeforces.com/blog/entry/16205
//  https://cp-algorithms.com/graph/2SAT.html
```

4.52 virtual tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define pci pair<char, int>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

int n, ans;
vector<int> adj[MAXN];
vector<int> with_color[MAXN];
vector<pi> virt[MAXN];
int a[MAXN];
int dp[MAXN];
int dp2[MAXN];
int pot[MAXN];

namespace lca
{
  int l, timer;
  vector<int> tin, tout, depth;
  vector<vector<int>> up;

  void dfs(int v, int p)
  {
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; i++)
      up[v][i] = up[up[v][i - 1]][i - 1];
    for (auto const &u : adj[v])
    {
      if (p == u)
        continue;
      depth[u] = depth[v] + 1;
      dfs(u, v);
    }
    tout[v] = ++timer;
  }
  bool is_ancestor(int u, int v)
  {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
  }
  int find_lca(int u, int v)
  {
    if (is_ancestor(u, v))
      return u;
    if (is_ancestor(v, u))
      return v;
    for (int i = l; i >= 0; i--)
      if (!is_ancestor(up[u][i], v))
        u = up[u][i];
    return up[u][0];
  }
  void init()
  {
    tin.resize(n);
    tout.resize(n);
    depth.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(0, 0);
  }
  int dist(int s, int v)
  {
    int at = find_lca(s, v);
    return (depth[s] + depth[v] - 2 * depth[at]);
  }
}

// dp naive (fazer O(n) para cada cor)
// fixa uma cor c e faz uma dp on tree pra calcular
```

```cpp
// quantas subtrees tem tal que todas as folhas sao da cor c
// e chamando o dfs saindo de qualquer vertice
// void dfs(int s, int p, int c)
// {
//   // dado que eu calculo o dp2[i] para cada filho
//   // para cada possivel subset (i1, i2, ..., ik) nao vazio de filhos
//   // acha o valor de dp2[i1] * dp2[i2] * ... * dp2[ik]
//   // no final eu quero a soma de todos esses valores, isso vai tar nessa
    variavel prod
//   int prod = 1;
//   for (auto const &i : adj[s])
//   {
//     if (i != p)
//     {
//       dfs(i, s, c);
//       prod = (prod * (dp2[i] + 1)) % mod;
//     }
//   }
//   prod = (prod - 1 + mod) % mod;
//   dp[s] = prod;
//   dp2[s] = prod;
//   if (a[s] == c)
//   {
//     dp[s] = (dp[s] + 1) % mod;
//     dp2[s] = (dp2[s] + 1) % mod;
//   }
//   for (auto const &i : adj[s])
//   {
//     if (i == p)
//       continue;
//     if (a[s] != c)
//       dp[s] = (dp[s] - dp2[i] + mod) % mod;
//   }
// }

// virtual tree
// dado um conjunto de vertices v
// montar uma arvore comprimida
// tal que escolhendo dois vertices do conjunto v[i] e v[j]
// lca(v[i], v[j]) tambem ta na arvore
// se o conjunto v tem k vertices
// entao a arvore comprimida tem menos do que 2k vertices
// O(k log(k)), sem considerar a complexidade de achar lca
int build_virt(vector<int> v)
{
  auto cmp = [&](int i, int j)
  {
    return lca::tin[i] < lca::tin[j];
  };
  sort(v.begin(), v.end(), cmp);
  for (int i = v.size() - 1; i > 0; i--)
  {
    v.pb(lca::find_lca(v[i], v[i - 1]));
  }
  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());
  for (int i = 0; i < v.size(); i++)
  {
    virt[v[i]].clear();
  }
  for (int i = 1; i < v.size(); i++)
  {
    virt[lca::find_lca(v[i - 1], v[i])].clear();
  }
  for (int i = 1; i < v.size(); i++)
  {
    int parent = lca::find_lca(v[i - 1], v[i]);
    int d = lca::dist(parent, v[i]);
    virt[parent].pb({v[i], d});
  }
  return v[0];
}
void dfs(int s, int c) // dp naive, so que fazer isso na virtual tree
{
  int prod = 1;
  for (auto const &i : virt[s])
  {
    dfs(i.fir, c);
    prod = (prod * (dp2[i.fir] + 1)) % mod;
  }
  prod = (prod - 1 + mod) % mod;
  dp[s] = prod;
  dp2[s] = prod;
  if (a[s] == c)
  {
    dp[s] = (dp[s] + 1) % mod;
    dp2[s] = (dp2[s] + 1) % mod;
  }
  for (auto const &i : virt[s])
  {
    if (a[s] != c)
      dp[s] = (dp[s] - dp2[i.fir] + mod) % mod;
  }
  ans = (ans + dp[s]) % mod;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  pot[0] = 1;
  for (int i = 1; i < MAXN; i++)
  {
    pot[i] = (pot[i - 1] * 2) % mod;
  }
  cin >> n;
  for (int i = 0; i < n; i++)
  {
    cin >> a[i];
    with_color[a[i]].pb(i);
  }
  for (int i = 1; i < n; i++)
  {
    int a, b;
    cin >> a >> b;
    a--, b--;
    adj[a].pb(b);
    adj[b].pb(a);
  }
  lca::init();
  for (int i = 1; i <= n; i++)
  {
    if (with_color[i].size() > 0)
    {
      int root = build_virt(with_color[i]);
      dfs(root, i);
    }
  }
  cout << ans << endl;
}
// https://atcoder.jp/contests/abc340/tasks/abc340_g
// problema legal
// dado uma arvore com n vertices (n <= 2 * 10^5)
// cada vertice tem uma cor a[i]
// conte quantas subarvores existem tal que:
// todas as folhas nessa subarvore tem a mesma cor

// sei codar em O(n^2), rodando um dfs pra cada cor
// dai montar a virtual tree para cada cor
// e rodar a dp naive na virtual tree

// tambem resolve o https://codeforces.com/gym/103960/problem/L da sub de 2022
```

5 kactl forked

6 kactl forked/combinatorial

6.1 IntPerm

```
/**
 * Author: Simon Lindholm
 * Date: 2018-07-06
```

```
 * License: CC0
 * Description: Permutation -> integer conversion. (Not order preserving.)
 * Integer -> permutation can use a lookup table.
 * Time: O(n)
 */
#pragma once

int permToInt(vi& v) {
	int use = 0, i = 0, r = 0;
	for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
		use |= 1 << x;			// (note: minus, not ~!)
	return r;
}
```

6.2 multinomial

```
/**
 * Author: Mattias de Zalenski, Fredrik Niemela, Per Austrin, Simon Lindholm
 * Date: 2002-09-26
 * Source: Max Bennedich
 * Description: Computes $\displaystyle \binom{k_1 + \dots + k_n}{k_1, k_2, \
     dots, k_n} = \frac{(\sum k_i)!}{k_1!k_2!...k_n!}$.
 * Status: Tested on kattis:lexicography
 */
#pragma once

ll multinomial(vi& v) {
	ll c = 1, m = v.empty() ? 1 : v[0];
	rep(i,1,sz(v)) rep(j,0,v[i]) c = c * ++m / (j+1);
	return c;
}
```

7 kactl forked/data-structures

7.1 FenwickTree

```
/**
 * Author: Lukas Polacek
 * Date: 2009-10-30
 * License: CC0
 * Source: folklore/TopCoder
 * Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and
     updates single elements a[i],
 * taking the difference between the old and new value.
 * Time: Both operations are $O(\log N)$.
 * Status: Stress-tested
 */
#pragma once

struct FT {
	vector<ll> s;
	FT(int n) : s(n) {}
	void update(int pos, ll dif) { // a[pos] += dif
		for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
	}
	ll query(int pos) { // sum of values in [0, pos)
		ll res = 0;
		for (; pos > 0; pos &= pos - 1) res += s[pos-1];
		return res;
	}
	int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
		// Returns n if no sum is >= sum, or -1 if empty sum is.
		if (sum <= 0) return -1;
		int pos = 0;
		for (int pw = 1 << 25; pw; pw >>= 1) {
			if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
				pos += pw, sum -= s[pos-1];
		}
		return pos;
	}
};
```

7.2 FenwickTree2d

```
/**
 * Author: Simon Lindholm
 * Date: 2017-05-11
 * License: CC0
 * Source: folklore
 * Description: Computes sums a[i,j] for all i<I, j<J, and increases single
     elements a[i,j].
 *  Requires that the elements to be updated are known in advance (call
     fakeUpdate() before init()).
 * Time: $O(\log^2 N)$. (Use persistent segment trees for $O(\log N)$.)
 * Status: stress-tested
 */
#pragma once

#include "FenwickTree.h"

struct FT2 {
	vector<vi> ys; vector<FT> ft;
	FT2(int limx) : ys(limx) {}
	void fakeUpdate(int x, int y) {
		for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
	}
	void init() {
		for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
	}
	int ind(int x, int y) {
		return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
	void update(int x, int y, ll dif) {
		for (; x < sz(ys); x |= x + 1)
			ft[x].update(ind(x, y), dif);
	}
	ll query(int x, int y) {
		ll sum = 0;
		for (; x; x &= x - 1)
			sum += ft[x-1].query(ind(x-1, y));
		return sum;
	}
};
```

7.3 HashMap

```
/**
 * Author: Simon Lindholm, chilli
 * Date: 2018-07-23
 * License: CC0
 * Source: http://codeforces.com/blog/entry/60737
 * Description: Hash map with mostly the same API as unordered\_map, but \tilde
 * 3x faster. Uses 1.5x memory.
 * Initial capacity must be a power of 2 (if provided).
 */
#pragma once

#include <bits/extc++.h> /** keep-include */
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
	const uint64_t C = ll(4e18 * acos(0)) | 71;
	ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});

/** For CodeForces, or other places where hacking might be a problem:

const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count
    ();
struct chash { // To use most bits rather than just the lowest ones:
	const uint64_t C = ll(4e18 * acos(0)) | 71; // large odd number
	ll operator()(ll x) const { return __builtin_bswap64((x^RANDOM)*C); }
```

```
};
__gnu_pbds::gp_hash_table<ll, int, chash> h({},{},{},{}, {1 << 16});
*/
```

7.4 LazySegmentTree

```
/**
 * Author: Simon Lindholm
 * Date: 2016-10-08
 * License: CC0
 * Source: me
 * Description: Segment tree with ability to add or set values of large
     intervals, and compute max of intervals.
 * Can be changed to other things.
 * Use with a bump allocator for better performance, and SmallPtr or implicit
     indices to save memory.
 * Time: O(\log N).
 * Usage: Node* tr = new Node(v, 0, sz(v));
 * Status: stress-tested a bit
 */
#pragma once

#include "../various/BumpAllocator.h"

const int inf = 1e9;
struct Node {
	Node *l = 0, *r = 0;
	int lo, hi, mset = inf, madd = 0, val = -inf;
	Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
	Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
		if (lo + 1 < hi) {
			int mid = lo + (hi - lo)/2;
			l = new Node(v, lo, mid); r = new Node(v, mid, hi);
			val = max(l->val, r->val);
		}
		else val = v[lo];
	}
	int query(int L, int R) {
		if (R <= lo || hi <= L) return -inf;
		if (L <= lo && hi <= R) return val;
		push();
		return max(l->query(L, R), r->query(L, R));
	}
	void set(int L, int R, int x) {
		if (R <= lo || hi <= L) return;
		if (L <= lo && hi <= R) mset = val = x, madd = 0;
		else {
			push(), l->set(L, R, x), r->set(L, R, x);
			val = max(l->val, r->val);
		}
	}
	void add(int L, int R, int x) {
		if (R <= lo || hi <= L) return;
		if (L <= lo && hi <= R) {
			if (mset != inf) mset += x;
			else madd += x;
			val += x;
		}
		else {
			push(), l->add(L, R, x), r->add(L, R, x);
			val = max(l->val, r->val);
		}
	}
	void push() {
		if (!l) {
			int mid = lo + (hi - lo)/2;
			l = new Node(lo, mid); r = new Node(mid, hi);
		}
		if (mset != inf)
			l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
		else if (madd)
			l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
	}
};
```

7.5 LineContainer

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+m, and query
     maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

struct Line {
	mutable ll k, m, p;
	bool operator<(const Line& o) const { return k < o.k; }
	bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
	// (for doubles, use inf = 1/.0, div(a,b) = a/b)
	static const ll inf = LLONG_MAX;
	ll div(ll a, ll b) { // floored division
		return a / b - ((a ^ b) < 0 && a % b); }
	bool isect(iterator x, iterator y) {
		if (y == end()) return x->p = inf, 0;
		if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
		else x->p = div(y->m - x->m, x->k - y->k);
		return x->p >= y->p;
	}
	void add(ll k, ll m) {
		auto z = insert({k, m, 0}), y = z++, x = y;
		while (isect(y, z)) z = erase(z);
		if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
		while ((y = x) != begin() && (--x)->p >= y->p)
			isect(x, erase(y));
	}
	ll query(ll x) {
		assert(!empty());
		auto l = *lower_bound(x);
		return l.k * x + l.m;
	}
};
```

7.6 Matrix

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-08-03
 * License: CC0
 * Source: My head
 * Description: Basic operations on square matrices.
 * Usage: Matrix<int, 3> A;
 *  A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};
 *  array<int, 3> vec = {1,2,3};
 *  vec = (A^N) * vec;
 * Status: tested
 */
#pragma once

template<class T, int N> struct Matrix {
	typedef Matrix M;
	array<array<T, N>, N> d{};
	M operator*(const M& m) const {
		M a;
		rep(i,0,N) rep(j,0,N)
			rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
		return a;
	}
	array<T, N> operator*(const array<T, N>& vec) const {
		array<T, N> ret{};
```

```
                rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
                return ret;
        }
        M operator^(ll p) const {
                assert(p >= 0);
                M a, b(*this);
                rep(i,0,N) a.d[i][i] = 1;
                while (p) {
                        if (p&1) a = a*b;
                        b = b*b;
                        p >>= 1;
                }
                return a;
        }
};
```

7.7 MoQueries

```
/**
 * Author: Simon Lindholm
 * Date: 2019-12-28
 * License: CC0
 * Source: https://github.com/hoke-t/tamu-kactl/blob/master/content/data-
     structures/MoQueries.h
 * Description: Answer interval or tree path queries by finding an approximate
     TSP through the queries,
 * and moving from one query to the next by adding/removing points at the ends.
 * If values are on tree edges, change \texttt{step} to add/remove the edge $(a,
     c)$ and remove the initial \texttt{add} call (but keep \texttt{in}).
 * Time: O(N \sqrt Q)
 * Status: stress-tested
 */
#pragma once

void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vi mo(vector<pii> Q) {
        int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
        vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
        iota(all(s), 0);
        sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
        for (int qi : s) {
                pii q = Q[qi];
                while (L > q.first) add(--L, 0);
                while (R < q.second) add(R++, 1);
                while (L < q.first) del(L++, 0);
                while (R > q.second) del(--R, 1);
                res[qi] = calc();
        }
        return res;
}

vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
        int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
        vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
        add(0, 0), in[0] = 1;
        auto dfs = [&](int x, int p, int dep, auto& f) -> void {
                par[x] = p;
                L[x] = N;
                if (dep) I[x] = N++;
                for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
                if (!dep) I[x] = N++;
                R[x] = N;
        };
        dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
        iota(all(s), 0);
        sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
        for (int qi : s) rep(end,0,2) {
                int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                  else { add(c, end); in[c] = 1; } a = c; }
                while (!(L[b] <= L[a] && R[a] <= R[b]))
                        I[i++] = b, b = par[b];
                while (a != b) step(par[a]);
                while (i--) step(I[i]);
                if (end) res[qi] = calc();
        }
        return res;
}
```

7.8 OrderStatisticTree

```
/**
 * Author: Simon Lindholm
 * Date: 2016-03-22
 * License: CC0
 * Source: hacKIT, NWERC 2015
 * Description: A set (not multiset!) with support for finding the n'th
 * element, and finding the index of an element.
 * To get a map, change \texttt{null\_type}.
 * Time: O(\log N)
 */
#pragma once

#include <bits/extc++.h> /** keep-include */
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
        Tree<int> t, t2; t.insert(8);
        auto it = t.insert(10).first;
        assert(it == t.lower_bound(9));
        assert(t.order_of_key(10) == 1);
        assert(t.order_of_key(11) == 2);
        assert(*t.find_by_order(0) == 8);
        t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

7.9 RMQ

```
/**
 * Author: Johan Sannemo, pajenegod
 * Date: 2015-02-06
 * License: CC0
 * Source: Folklore
 * Description: Range Minimum Queries on an array. Returns
 * min(V[a], V[a + 1], ... V[b - 1]) in constant time.
 * Usage:
 *  RMQ rmq(values);
 *  rmq.query(inclusive, exclusive);
 * Time: $O(|V| \log |V| + Q)$
 * Status: stress-tested
 */
#pragma once

template<class T>
struct RMQ {
        vector<vector<T>> jmp;
        RMQ(const vector<T>& V) : jmp(1, V) {
                for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
                        jmp.emplace_back(sz(V) - pw * 2 + 1);
                        rep(j,0,sz(jmp[k]))
                                jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw
                                    ]);
                }
        }
        T query(int a, int b) {
                assert(a < b); // or return inf if a == b
                int dep = 31 - __builtin_clz(b - a);
                return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
```

```
	}
};
```

7.10 SegmentTree

```
/**
 * Author: Lucian Bicsi
 * Date: 2017-10-31
 * License: CC0
 * Source: folklore
 * Description: Zero-indexed max-tree. Bounds are inclusive to the left and
     exclusive to the right.
 * Can be changed by modifying T, f and unit.
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

struct Tree {
	typedef int T;
	static constexpr T unit = INT_MIN;
	T f(T a, T b) { return max(a, b); } // (any associative fn)
	vector<T> s; int n;
	Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
	void update(int pos, T val) {
		for (s[pos += n] = val; pos /= 2;)
			s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
	}
	T query(int b, int e) { // query [b, e)
		T ra = unit, rb = unit;
		for (b += n, e += n; b < e; b /= 2, e /= 2) {
			if (b % 2) ra = f(ra, s[b++]);
			if (e % 2) rb = f(s[--e], rb);
		}
		return f(ra, rb);
	}
};
```

7.11 SubMatrix

```
/**
 * Author: Johan Sannemo
 * Date: 2014-11-28
 * License: CC0
 * Source: Folklore
 * Description: Calculate submatrix sums quickly, given upper-left and lower-
     right corners (half-open).
 * Usage:
 * SubMatrix<int> m(matrix);
 * m.sum(0, 0, 2, 2); // top left 4 elements
 * Time: O(N^2 + Q)
 * Status: Tested on Kattis
 */
#pragma once

template<class T>
struct SubMatrix {
	vector<vector<T>> p;
	SubMatrix(vector<vector<T>>& v) {
		int R = sz(v), C = sz(v[0]);
		p.assign(R+1, vector<T>(C+1));
		rep(r,0,R) rep(c,0,C)
			p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p[r][c];
	}
	T sum(int u, int l, int d, int r) {
		return p[d][r] - p[d][l] - p[u][r] + p[u][l];
	}
};
```

7.12 Treap

```
/**
 * Author: someone on Codeforces
 * Date: 2017-03-14
 * Source: folklore
 * Description: A short self-balancing tree. It acts as a
 *  sequential container with log-time splits/joins, and
 *  is easy to augment with additional data.
 * Time: $O(\log N)$
 * Status: stress-tested
 */
#pragma once

struct Node {
	Node *l = 0, *r = 0;
	int val, y, c = 1;
	Node(int val) : val(val), y(rand()) {}
	void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
	if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
	if (!n) return {};
	if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
		auto [L,R] = split(n->l, k);
		n->l = R;
		n->recalc();
		return {L, n};
	} else {
		auto [L,R] = split(n->r,k - cnt(n->l) - 1); // and just "k"
		n->r = L;
		n->recalc();
		return {n, R};
	}
}

Node* merge(Node* l, Node* r) {
	if (!l) return r;
	if (!r) return l;
	if (l->y > r->y) {
		l->r = merge(l->r, r);
		return l->recalc(), l;
	} else {
		r->l = merge(l, r->l);
		return r->recalc(), r;
	}
}

Node* ins(Node* t, Node* n, int pos) {
	auto [l,r] = split(t, pos);
	return merge(merge(l, n), r);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
	Node *a, *b, *c;
	tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
	if (k <= l) t = merge(ins(a, b, k), c);
	else t = merge(a, ins(c, b, k - r));
}
```

7.13 UnionFind

```
/**
 * Author: Lukas Polacek
 * Date: 2009-10-26
 * License: CC0
 * Source: folklore
```

```
 * Description: Disjoint-set data structure.
 * Time: $O(\alpha(N))$
 */
#pragma once

struct UF {
	vi e;
	UF(int n) : e(n, -1) {}
	bool sameSet(int a, int b) { return find(a) == find(b); }
	int size(int x) { return -e[find(x)]; }
	int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
	bool join(int a, int b) {
		a = find(a), b = find(b);
		if (a == b) return false;
		if (e[a] > e[b]) swap(a, b);
		e[a] += e[b]; e[b] = a;
		return true;
	}
};
```

7.14 UnionFindRollback

```
/**
 * Author: Lukas Polacek, Simon Lindholm
 * Date: 2019-12-26
 * License: CC0
 * Source: folklore
 * Description: Disjoint-set data structure with undo.
 * If undo is not needed, skip st, time() and rollback().
 * Usage: int t = uf.time(); ...; uf.rollback(t);
 * Time: $O(\log(N))$
 * Status: tested as part of DirectedMST.h
 */
#pragma once

struct RollbackUF {
	vi e; vector<pii> st;
	RollbackUF(int n) : e(n, -1) {}
	int size(int x) { return -e[find(x)]; }
	int find(int x) { return e[x] < 0 ? x : find(e[x]); }
	int time() { return sz(st); }
	void rollback(int t) {
		for (int i = time(); i --> t;)
			e[st[i].first] = st[i].second;
		st.resize(t);
	}
	bool join(int a, int b) {
		a = find(a), b = find(b);
		if (a == b) return false;
		if (e[a] > e[b]) swap(a, b);
		st.push_back({a, e[a]});
		st.push_back({b, e[b]});
		e[a] += e[b]; e[b] = a;
		return true;
	}
};
```

8 kactl forked/geometry

8.1 3dHull

```
/**
 * Author: Johan Sannemo
 * Date: 2017-04-18
 * Source: derived from https://gist.github.com/msg555/4963794 by Mark Gordon
 * Description: Computes all faces of the 3-dimension hull of a point set.
 *  *No four points must be coplanar*, or else random results will be returned.
 *  All faces will point outwards.
 * Time: O(n^2)
 * Status: tested on SPOJ CH3D
 */
#pragma once

#include "Point3D.h"

typedef Point3D<double> P3;

struct PR {
	void ins(int x) { (a == -1 ? a : b) = x; }
	void rem(int x) { (a == x ? a : b) = -1; }
	int cnt() { return (a != -1) + (b != -1); }
	int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
	assert(sz(A) >= 4);
	vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
	vector<F> FS;
	auto mf = [&](int i, int j, int k, int l) {
		P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
		if (q.dot(A[l]) > q.dot(A[i]))
			q = q * -1;
		F f{q, i, j, k};
		E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
		FS.push_back(f);
	};
	rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
		mf(i, j, k, 6 - i - j - k);

	rep(i,4,sz(A)) {
		rep(j,0,sz(FS)) {
			F f = FS[j];
			if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
				E(a,b).rem(f.c);
				E(a,c).rem(f.b);
				E(b,c).rem(f.a);
				swap(FS[j--], FS.back());
				FS.pop_back();
			}
		}
		int nw = sz(FS);
		rep(j,0,nw) {
			F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
			C(a, b, c); C(a, c, b); C(b, c, a);
		}
	}
	for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
		A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
	return FS;
};
```

8.2 Angle

```
/**
 * Author: Simon Lindholm
 * Date: 2015-01-31
 * License: CC0
 * Source: me
 * Description: A class for ordering angles (as represented by int points and
 *  a number of rotations around the origin). Useful for rotational sweeping.
 *  Sometimes also represents points or vectors.
 * Usage:
 *  vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
 *  int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
 *  // sweeps j such that (j-i) represents the number of positively oriented
      triangles with vertices at 0 and i
 * Status: Used, works well
 */
#pragma once

struct Angle {
```

```
        int x, y;
        int t;
        Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
        Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
        int half() const {
                assert(x || y);
                return y < 0 || (y == 0 && x < 0);
        }
        Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
        Angle t180() const { return {-x, -y, t + half()}; }
        Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
        // add a.dist2() and b.dist2() to also compare distances
        return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
               make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
        if (b < a) swap(a, b);
        return (b < a.t180() ?
                make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
        Angle r(a.x + b.x, a.y + b.y, a.t);
        if (a.t180() < r) r.t--;
        return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
        int tu = b.t - a.t; a.t = b.t;
        return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

8.3 CircleIntersection

```
/**
 * Author: Simon Lindholm
 * Date: 2015-09-01
 * License: CC0
 * Description: Computes the pair of points at which two circles intersect.
 * Returns false in case of no intersection.
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
        if (a == b) { assert(r1 != r2); return false; }
        P vec = b - a;
        double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
               p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
        if (sum*sum < d2 || dif*dif > d2) return false;
        P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
        *out = {mid + per, mid - per};
        return true;
}
```

8.4 CircleLine

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-10-29
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Finds the intersection between a circle and a line.
 * Returns a vector of either 0, 1, or 2 intersection points.
 * P is intended to be Point<double>.
 * Status: unit tested
 */

#pragma once

#include "Point.h"

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
        P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
        double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
        if (h2 < 0) return {};
        if (h2 == 0) return {p};
        P h = ab.unit() * sqrt(h2);
        return {p - h, p + h};
}
```

8.5 CirclePolygonIntersection

```
/**
 * Author: chilli, Takanori MAEHARA
 * Date: 2019-10-31
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/master/geometry/
     _geom.cc#L744
 * Description: Returns the area of the intersection of a circle with a
 * ccw polygon.
 * Time: O(n)
 * Status: Tested on GNYR 2019 Gerrymandering, stress-tested
 */
#pragma once

#include "../../content/geometry/Point.h"

typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
        auto tri = [&](P p, P q) {
                auto r2 = r * r / 2;
                P d = q - p;
                auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
                auto det = a * a - b;
                if (det <= 0) return arg(p, q) * r2;
                auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
                if (t < 0 || 1 <= s) return arg(p, q) * r2;
                P u = p + d * s, v = q + d * (t-1);
                return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
        };
        auto sum = 0.0;
        rep(i,0,sz(ps))
                sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
        return sum;
}
```

8.6 CircleTangents

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-10-31
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Finds the external tangents of two circles, or internal if r2 is
      negated.
 * Can return 0, 1, or 2 tangents -- 0 if one circle contains the other (or
     overlaps it, in the internal case, or if the circles are the same);
 * 1 if the circles are tangent to each other (in which case .first = .second
     and the tangent line is perpendicular to the line between the centers).
 * .first and .second give the tangency points at circle 1 and 2 respectively.
 * To find the tangents of a circle with a point set r2 to 0.
 * Status: tested
 */
#pragma once
```

```
#include "Point.h"

template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
	P d = c2 - c1;
	double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
	if (d2 == 0 || h2 < 0)  return {};
	vector<pair<P, P>> out;
	for (double sign : {-1, 1}) {
		P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
		out.push_back({c1 + v * r1, c2 + v * r2});
	}
	if (h2 == 0) out.pop_back();
	return out;
}
```

8.7 circumcircle

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-04-11
 * License: CC0
 * Source: http://en.wikipedia.org/wiki/Circumcircle
 * Description:\\
\begin{minipage}{75mm}
The circumcirle of a triangle is the circle intersecting all three vertices.
     ccRadius returns the radius of the circle going through points A, B and C
     and ccCenter returns the center of the same circle.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-2mm}
\includegraphics[width=\textwidth]{content/geometry/circumcircle}
\end{minipage}
 * Status: tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
	return (B-A).dist()*(C-B).dist()*(A-C).dist()/
	                abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
	P b = C-A, c = B-A;
	return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

8.8 ClosestPair

```
/**
 * Author: Simon Lindholm
 * Date: 2019-04-17
 * License: CC0
 * Source: https://codeforces.com/blog/entry/58747
 * Description: Finds the closest pair of points.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
	assert(sz(v) > 1);
	set<P> S;
	sort(all(v), [](P a, P b) { return a.y < b.y; });
	pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
	int j = 0;
	for (P p : v) {
		P d{1 + (ll)sqrt(ret.first), 0};
		while (v[j].y <= p.y - d.x) S.erase(v[j++]);
		auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
		for (; lo != hi; ++lo)
			ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
		S.insert(p);
	}
	return ret.second;
}
```

8.9 ConvexHull

```
/**
 * Author: Stjepan Glavina, chilli
 * Date: 2019-05-05
 * License: Unlicense
 * Source: https://github.com/stjepang/snippets/blob/master/convex_hull.cpp
 * Description:
\\\begin{minipage}{75mm}
Returns a vector of the points of the convex hull in counter-clockwise order.
Points on the edge of the hull between two other points are not considered part
     of the hull.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-6mm}
\includegraphics[width=\textwidth]{content/geometry/ConvexHull}
\vspace{-6mm}
\end{minipage}
 * Time: O(n \log n)
 * Status: stress-tested, tested with kattis:convexhull
*/
#pragma once

#include "Point.h"

typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
	if (sz(pts) <= 1) return pts;
	sort(all(pts));
	vector<P> h(sz(pts)+1);
	int s = 0, t = 0;
	for (int it = 2; it--; s = --t, reverse(all(pts)))
		for (P p : pts) {
			while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
			h[t++] = p;
		}
	return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

8.10 DelaunayTriangulation

```
/**
 * Author: Mattias de Zalenski
 * Date: Unknown
 * Source: Geometry in C
 * Description: Computes the Delaunay triangulation of a set of points.
 *  Each circumcircle contains none of the input points.
 *  If any three points are collinear or any four are on the same circle,
     behavior is undefined.
 * Time: O(n^2)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"
#include "3dHull.h"

template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
	if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
		trifun(0,1+d,2-d); }
	vector<P3> p3;
```

```
        for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
        if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a]).
                        cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
                trifun(t.a, t.c, t.b);
}
```

8.11 FastDelaunay

```
/**
 * Author: Philippe Legault
 * Date: 2016
 * License: MIT
 * Source: https://github.com/Bathlamos/delaunay-triangulation/
 * Description: Fast Delaunay triangulation.
 * Each circumcircle contains none of the input points.
 * There must be no duplicate points.
 * If all points are on a line, no triangles will be returned.
 * Should work for doubles as well, though there may be precision issues in '
     circ'.
 * Returns triangles in order \{t[0][0], t[0][1], t[0][2], t[1][0], \dots\}, all
       counter-clockwise.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
        Q rot, o; P p = arb; bool mark;
        P& F() { return r()->p; }
        Q& r() { return rot->rot; }
        Q prev() { return rot->o->rot; }
        Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
        lll p2 = p.dist2(), A = a.dist2()-p2,
            B = b.dist2()-p2, C = c.dist2()-p2;
        return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
        Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
        H = r->o; r->r()->r() = r;
        rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
        r->p = orig; r->F() = dest;
        return r;
}
void splice(Q a, Q b) {
        swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
        Q q = makeEdge(a->F(), b->p);
        splice(q, a->next());
        splice(q->r(), b);
        return q;
}

pair<Q,Q> rec(const vector<P>& s) {
        if (sz(s) <= 3) {
                Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
                if (sz(s) == 2) return { a, a->r() };
                splice(a->r(), b);
                auto side = s[0].cross(s[1], s[2]);
                Q c = side ? connect(b, a) : 0;
                return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
        }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
        Q A, B, ra, rb;
        int half = sz(s) / 2;
        tie(ra, A) = rec({all(s) - half});
        tie(B, rb) = rec({sz(s) - half + all(s)});
        while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
               (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
        Q base = connect(B->r(), A);
        if (A->p == ra->p) ra = base->r();
        if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
                while (circ(e->dir->F(), H(base), e->F())) { \
                        Q t = e->dir; \
                        splice(e, e->prev()); \
                        splice(e->r(), e->r()->prev()); \
                        e->o = H; H = e; e = t; \
                }
        for (;;) {
                DEL(LC, base->r(), o);  DEL(RC, base, prev());
                if (!valid(LC) && !valid(RC)) break;
                if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
                        base = connect(RC, base->r());
                else
                        base = connect(base->r(), LC->r());
        }
        return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
        sort(all(pts));  assert(unique(all(pts)) == pts.end());
        if (sz(pts) < 2) return {};
        Q e = rec(pts).first;
        vector<Q> q = {e};
        int qi = 0;
        while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
        q.push_back(c->r()); c = c->next(); } while (c != e); }
        ADD; pts.clear();
        while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
        return pts;
}
```

8.12 HullDiameter

```
/**
 * Author: Oleksandr Bacherikov, chilli
 * Date: 2019-05-05
 * License: Boost Software License
 * Source: https://codeforces.com/blog/entry/48868
 * Description: Returns the two points with max distance on a convex hull (ccw,
 * no duplicate/collinear points).
 * Status: stress-tested, tested on kattis:roberthood
 * Time: O(n)
 */
#pragma once
#include "Point.h"

typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
        int n = sz(S), j = n < 2 ? 0 : 1;
        pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
        rep(i,0,j)
                for (;; j = (j + 1) % n) {
                        res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
                        if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
                                break;
                }
        return res.second;
}
```

8.13 InsidePolygon

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Returns true if p lies within the polygon. If strict is true,
 * it returns false for points on the boundary. The algorithm uses
 * products in intermediate steps so watch out for overflow.
 * Time: O(n)
 * Usage:
 * vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
 * bool in = inPolygon(v, P{3, 3}, false);
 * Status: stress-tested and tested on kattis:pointinpolygon
 */
#pragma once

#include "Point.h"
#include "OnSegment.h"
#include "SegmentDistance.h"

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
	int cnt = 0, n = sz(p);
	rep(i,0,n) {
		P q = p[(i + 1) % n];
		if (onSegment(p[i], q, a)) return !strict;
		//or: if (segDist(p[i], q, a) <= eps) return !strict;
		cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
	}
	return cnt;
}
```

8.14 kdTree

```
/**
 * Author: Stanford
 * Date: Unknown
 * Source: Stanford Notebook
 * Description: KD-tree (2d, can be extended to 3d)
 * Status: Tested on excellentengineers
 */
#pragma once

#include "Point.h"

typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
	P pt; // if this is a leaf, the single point in it
	T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
	Node *first = 0, *second = 0;

	T distance(const P& p) { // min squared distance to a point
		T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
		T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
		return (P(x,y) - p).dist2();
	}

	Node(vector<P>&& vp) : pt(vp[0]) {
		for (P p : vp) {
			x0 = min(x0, p.x); x1 = max(x1, p.x);
			y0 = min(y0, p.y); y1 = max(y1, p.y);
		}
		if (vp.size() > 1) {
			// split on x if width >= height (not ideal...)
			sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
			// divide by taking half the array for each child (not
			// best performance with many duplicates in the middle)
			int half = sz(vp)/2;
			first = new Node({vp.begin(), vp.begin() + half});
			second = new Node({vp.begin() + half, vp.end()});
		}
	}
};

struct KDTree {
	Node* root;
	KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

	pair<T, P> search(Node *node, const P& p) {
		if (!node->first) {
			// uncomment if we should not find the point itself:
			// if (p == node->pt) return {INF, P()};
			return make_pair((p - node->pt).dist2(), node->pt);
		}

		Node *f = node->first, *s = node->second;
		T bfirst = f->distance(p), bsec = s->distance(p);
		if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

		// search closest side first, other side if needed
		auto best = search(f, p);
		if (bsec < best.first)
			best = min(best, search(s, p));
		return best;
	}

	// find nearest point to a point, and its squared distance
	// (requires an arbitrary operator< for Point)
	pair<T, P> nearest(const P& p) {
		return search(root, p);
	}
};
```

8.15 linearTransformation

```
/**
 * Author: Per Austrin, Ulf Lundstrom
 * Date: 2009-04-09
 * License: CC0
 * Source:
 * Description:\\
\begin{minipage}{75mm}
 Apply the linear transformation (translation, rotation and scaling) which takes
      line p0-p1 to line q0-q1 to point r.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-8mm}
\includegraphics[width=\textwidth]{content/geometry/linearTransformation}
\vspace{-2mm}
\end{minipage}
 * Status: not tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
		const P& q0, const P& q1, const P& r) {
	P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
	return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

8.16 lineDistance

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source: Basic math
 * Description:\\
\begin{minipage}{75mm}
```

```
Returns the signed distance between point p and the line containing points a and
     b.
Positive value on left side and negative on right as seen from a towards b. a==b
     gives nan.
P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long.
It uses products in intermediate steps so watch out for overflow if using int or
     long long.
Using Point3D will always give a non-negative distance. For Point3D, call .dist
    on the result of the cross product.
\end{minipage}
\begin{minipage}{15mm}
\includegraphics[width=\textwidth]{content/geometry/lineDistance}
\end{minipage}
 * Status: tested
 */
#pragma once

#include "Point.h"

template<class P>
double lineDist(const P& a, const P& b, const P& p) {
        return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

8.17 LineHullIntersection

```
/**
 * Author: Oleksandr Bacherikov, chilli
 * Date: 2019-05-07
 * License: Boost Software License
 * Source: https://github.com/AlCash07/ACTL/blob/master/include/actl/geometry/
     algorithm/intersect/line_convex_polygon.hpp
 * Description: Line-convex polygon intersection. The polygon must be ccw and
     have no collinear points.
 * lineHull(line, poly) returns a pair describing the intersection of a line
     with the polygon:
 *  \begin{itemize*}
 *    \item $(-1, -1)$ if no collision,
 *    \item $(i, -1)$ if touching the corner $i$,
 *    \item $(i, i)$ if along side $(i, i+1)$,
 *    \item $(i, j)$ if crossing sides $(i, i+1)$ and $(j, j+1)$.
 *  \end{itemize*}
 *  In the last case, if a corner $i$ is crossed, this is treated as happening
     on side $(i, i+1)$.
 *  The points are returned in the same order as the line hits the polygon.
 * \texttt{extrVertex} returns the point of a hull with the max projection onto
     a line.
 * Time: O(\log n)
 * Status: stress-tested
 */
#pragma once

#include "Point.h"

#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
        int n = sz(poly), lo = 0, hi = n;
        if (extr(0)) return 0;
        while (lo + 1 < hi) {
                int m = (lo + hi) / 2;
                if (extr(m)) return m;
                int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
                (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
        }
        return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
        int endA = extrVertex(poly, (a - b).perp());
        int endB = extrVertex(poly, (b - a).perp());
        if (cmpL(endA) < 0 || cmpL(endB) > 0)
                return {-1, -1};
        array<int, 2> res;
        rep(i,0,2) {
                int lo = endB, hi = endA, n = sz(poly);
                while ((lo + 1) % n != hi) {
                        int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
                        (cmpL(m) == cmpL(endB) ? lo : hi) = m;
                }
                res[i] = (lo + !cmpL(hi)) % n;
                swap(endA, endB);
        }
        if (res[0] == res[1]) return {res[0], -1};
        if (!cmpL(res[0]) && !cmpL(res[1]))
                switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
                        case 0: return {res[0], res[0]};
                        case 2: return {res[1], res[1]};
                }
        return res;
}
```

8.18 lineIntersection

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-05-05
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description:\\
\begin{minipage}{75mm}
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists
     \{1, point\} is returned.
If no intersection point exists \{0, (0,0)\} is returned and if infinitely many
     exists \{-1, (0,0)\} is returned.
The wrong position will be returned if P is Point<ll> and the intersection point
     does not have integer coordinates.
Products of three coordinates are used in intermediate steps so watch out for
     overflow if using int or ll.
\end{minipage}
\begin{minipage}{15mm}
\includegraphics[width=\textwidth]{content/geometry/lineIntersection}
\end{minipage}
 * Usage:
 *      auto res = lineInter(s1,e1,s2,e2);
 *      if (res.first == 1)
 *              cout << "intersection point at " << res.second << endl;
 * Status: stress-tested, and tested through half-plane tests
 */
#pragma once

#include "Point.h"

template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
        auto d = (e1 - s1).cross(e2 - s2);
        if (d == 0) // if parallel
                return {-(s1.cross(e1, s2) == 0), P(0, 0)};
        auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
        return {1, (s1 * p + e1 * q) / d};
}
```

8.19 LineProjectionReflection

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-10-29
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Projects point p onto line ab. Set refl=true to get reflection
 * of point p across line ab instead. The wrong point will be returned if P is
 * an integer point and the desired point doesn't have integer coordinates.
 * Products of three coordinates are used in intermediate steps so watch out
 * for overflow.
 * Status: stress-tested
```

```
 */
#pragma once

#include "Point.h"

template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
	P v = b - a;
	return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

8.20 ManhattanMST

```
/**
 * Author: chilli, Takanori MAEHARA
 * Date: 2019-11-02
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/master/geometry/
     rectilinear_mst.cc
 * Description: Given N points, returns up to 4*N edges, which are guaranteed
 * to contain a minimum spanning tree for the graph with edge weights w(p, q) =
 * |p.x - q.x| + |p.y - q.y|. Edges are in the form (distance, src, dst). Use a
 * standard MST algorithm on the result to find the final MST.
 * Time: O(N \log N)
 * Status: Stress-tested
 */
#pragma once
#include "Point.h"

typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
	vi id(sz(ps));
	iota(all(id), 0);
	vector<array<int, 3>> edges;
	rep(k,0,4) {
		sort(all(id), [&](int i, int j) {
		     return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
		map<int, int> sweep;
		for (int i : id) {
			for (auto it = sweep.lower_bound(-ps[i].y);
			                it != sweep.end(); sweep.erase(it++)) {
				int j = it->second;
				P d = ps[i] - ps[j];
				if (d.y > d.x) break;
				edges.push_back({d.y + d.x, i, j});
			}
			sweep[-ps[i].y] = i;
		}
		for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
	}
	return edges;
}
```

8.21 MinimumEnclosingCircle

```
/**
 * Author: Andrew He, chilli
 * Date: 2019-05-07
 * License: CC0
 * Source: folklore
 * Description: Computes the minimum circle that encloses a set of points.
 * Time: expected O(n)
 * Status: stress-tested
 */
#pragma once

#include "circumcircle.h"

pair<P, double> mec(vector<P> ps) {
	shuffle(all(ps), mt19937(time(0)));
	P o = ps[0];
	double r = 0, EPS = 1 + 1e-8;
	rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
		o = ps[i], r = 0;
		rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
			o = (ps[i] + ps[j]) / 2;
			r = (o - ps[i]).dist();
			rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
				o = ccCenter(ps[i], ps[j], ps[k]);
				r = (o - ps[i]).dist();
			}
		}
	}
	return {o, r};
}
```

8.22 OnSegment

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Returns true iff p lies on the line segment from s to e.
 * Use \texttt{(segDist(s,e,p)<=epsilon)} instead when using Point<double>.
 * Status:
 */
#pragma once

#include "Point.h"

template<class P> bool onSegment(P s, P e, P p) {
	return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

8.23 Point

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-02-26
 * License: CC0
 * Source: My head with inspiration from tinyKACTL
 * Description: Class to handle points in the plane.
 *      T can be e.g. double or long long. (Avoid int.)
 * Status: Works fine, used a lot
 */
#pragma once

template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
	typedef Point P;
	T x, y;
	explicit Point(T x=0, T y=0) : x(x), y(y) {}
	bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
	bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
	P operator+(P p) const { return P(x+p.x, y+p.y); }
	P operator-(P p) const { return P(x-p.x, y-p.y); }
	P operator*(T d) const { return P(x*d, y*d); }
	P operator/(T d) const { return P(x/d, y/d); }
	T dot(P p) const { return x*p.x + y*p.y; }
	T cross(P p) const { return x*p.y - y*p.x; }
	T cross(P a, P b) const { return (a-*this).cross(b-*this); }
	T dist2() const { return x*x + y*y; }
	double dist() const { return sqrt((double)dist2()); }
	// angle to x-axis in interval [-pi, pi]
	double angle() const { return atan2(y, x); }
	P unit() const { return *this/dist(); } // makes dist()=1
	P perp() const { return P(-y, x); } // rotates +90 degrees
	P normal() const { return perp().unit(); }
	// returns point rotated 'a' radians ccw around the origin
	P rotate(double a) const {
		return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
	friend ostream& operator<<(ostream& os, P p) {
```

```
		return os << "(" << p.x << "," << p.y << ")"; }
};
```

8.24 Point3D

```
/**
 * Author: Ulf Lundstrom with inspiration from tinyKACTL
 * Date: 2009-04-14
 * License: CC0
 * Source:
 * Description: Class to handle points in 3D space.
 *      T can be e.g. double or long long.
 * Usage:
 * Status: tested, except for phi and theta
 */
#pragma once

template<class T> struct Point3D {
	typedef Point3D P;
	typedef const P& R;
	T x, y, z;
	explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
	bool operator<(R p) const {
		return tie(x, y, z) < tie(p.x, p.y, p.z); }
	bool operator==(R p) const {
		return tie(x, y, z) == tie(p.x, p.y, p.z); }
	P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
	P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
	P operator*(T d) const { return P(x*d, y*d, z*d); }
	P operator/(T d) const { return P(x/d, y/d, z/d); }
	T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
	P cross(R p) const {
		return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
	}
	T dist2() const { return x*x + y*y + z*z; }
	double dist() const { return sqrt((double)dist2()); }
	//Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
	double phi() const { return atan2(y, x); }
	//Zenith angle (latitude) to the z-axis in interval [0, pi]
	double theta() const { return atan2(sqrt(x*x+y*y),z); }
	P unit() const { return *this/(T)dist(); } //makes dist()=1
	//returns unit vector normal to *this and p
	P normal(P p) const { return cross(p).unit(); }
	//returns point rotated 'angle' radians ccw around axis
	P rotate(double angle, P axis) const {
		double s = sin(angle), c = cos(angle); P u = axis.unit();
		return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
	}
};
```

8.25 PointInsideHull

```
/**
 * Author: chilli
 * Date: 2019-05-17
 * License: CC0
 * Source: https://github.com/ngthanhtrung23/ACM_Notebook_new
 * Description: Determine whether a point t lies inside a convex hull (CCW
 * order, with no collinear points). Returns true if point lies within
 * the hull. If strict is true, points on the boundary aren't included.
 * Usage:
 * Status: stress-tested
 * Time: O(\log N)
 */
#pragma once

#include "Point.h"
#include "sideOf.h"
#include "OnSegment.h"

typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
	int a = 1, b = sz(l) - 1, r = !strict;
	if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
	if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
	if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
		return false;
	while (abs(a - b) > 1) {
		int c = (a + b) / 2;
		(sideOf(l[0], l[c], p) > 0 ? b : a) = c;
	}
	return sgn(l[a].cross(l[b], p)) < r;
}
```

8.26 PolygonArea

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source: tinyKACTL
 * Description: Returns twice the signed area of a polygon.
 *  Clockwise enumeration gives negative area. Watch out for overflow if using
     int as T!
 * Status: Stress-tested and tested on kattis:polygonarea
 */
#pragma once

#include "Point.h"

template<class T>
T polygonArea2(vector<Point<T>>& v) {
	T a = v.back().cross(v[0]);
	rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
	return a;
}
```

8.27 PolygonCenter

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-04-08
 * License: CC0
 * Source:
 * Description: Returns the center of mass for a polygon.
 * Time: O(n)
 * Status: Tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
	P res(0, 0); double A = 0;
	for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
		res = res + (v[i] + v[j]) * v[j].cross(v[i]);
		A += v[j].cross(v[i]);
	}
	return res / A / 3;
}
```

8.28 PolygonCut

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source:
```

```
 * Description:\\
\begin{minipage}{75mm}
 Returns a vector with the vertices of a polygon with everything to the left of
      the line going from s to e cut away.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-6mm}
\includegraphics[width=\textwidth]{content/geometry/PolygonCut}
\vspace{-6mm}
\end{minipage}
 * Usage:
 *      vector<P> p = ...;
 *      p = polygonCut(p, P(0,0), P(1,0));
 * Status: tested but not extensively
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
        vector<P> res;
        rep(i,0,sz(poly)) {
                P cur = poly[i], prev = i ? poly[i-1] : poly.back();
                auto a = s.cross(e, cur), b = s.cross(e, prev);
                if ((a < 0) != (b < 0))
                        res.push_back(cur + (prev - cur) * (a / (a - b)));
                if (a < 0)
                        res.push_back(cur);
        }
        return res;
}
```

8.29 PolygonUnion

```
/**
 * Author: black_horse2014, chilli
 * Date: 2019-10-29
 * License: Unknown
 * Source: https://codeforces.com/gym/101673/submission/50481926
 * Description: Calculates the area of the union of $n$ polygons (not
     necessarily
 * convex). The points within each polygon must be given in CCW order.
 * (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be
     needed.)
 * Time: $O(N^2)$, where $N$ is the total number of points
 * Status: stress-tested, Submitted on ECNA 2017 Problem A
 */
#pragma once

#include "Point.h"
#include "sideOf.h"

typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
        double ret = 0;
        rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
                P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
                vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
                rep(j,0,sz(poly)) if (i != j) {
                        rep(u,0,sz(poly[j])) {
                                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[
                                    j])];
                                int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
                                if (sc != sd) {
                                        double sa = C.cross(D, A), sb = C.cross(
                                            D, B);
                                        if (min(sc, sd) < 0)
                                                segs.emplace_back(sa / (sa - sb)
                                                    , sgn(sc - sd));
                                } else if (!sc && !sd && j<i && sgn((B-A).dot(D-
                                    C))>0){
                                        segs.emplace_back(rat(C - A, B - A), 1);
                                        segs.emplace_back(rat(D - A, B - A), -1)
                                            ;
                                }
                        }
                }
                sort(all(segs));
                for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
                double sum = 0;
                int cnt = segs[0].second;
                rep(j,1,sz(segs)) {
                        if (!cnt) sum += segs[j].first - segs[j - 1].first;
                        cnt += segs[j].second;
                }
                ret += A.cross(B) * sum;
        }
        return ret / 2;
}
```

8.30 PolyhedronVolume

```
/**
 * Author: Mattias de Zalenski
 * Date: 2002-11-04
 * Description: Magic formula for the volume of a polyhedron. Faces should point
      outwards.
 * Status: tested
 */
#pragma once

template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
        double v = 0;
        for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
        return v / 6;
}
```

8.31 SegmentDistance

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source:
 * Description:\\
\begin{minipage}{75mm}
Returns the shortest distance between point p and the line segment from point s
     to e.
\end{minipage}
\begin{minipage}{15mm}
\vspace{-10mm}
\includegraphics[width=\textwidth]{content/geometry/SegmentDistance}
\end{minipage}
 * Usage:
 *      Point<double> a, b(2,2), p(1,1);
 *      bool onSegment = segDist(a,b,p) < 1e-10;
 * Status: tested
 */
#pragma once

#include "Point.h"

typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
        if (s==e) return (p-s).dist();
        auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
        return ((p-s)*d-(e-s)*t).dist()/d;
}
```

8.32 SegmentIntersection

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-27
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description:\\
\begin{minipage}{75mm}
If a unique intersection point between the line segments going from s1 to e1 and
      from s2 to e2 exists then it is returned.
If no intersection point exists an empty vector is returned.
If infinitely many exist a vector with 2 elements is returned, containing the
     endpoints of the common line segment.
The wrong position will be returned if P is Point<ll> and the intersection point
      does not have integer coordinates.
Products of three coordinates are used in intermediate steps so watch out for
     overflow if using int or long long.
\end{minipage}
\begin{minipage}{15mm}
\includegraphics[width=\textwidth]{content/geometry/SegmentIntersection}
\end{minipage}
 * Usage:
 * vector<P> inter = segInter(s1,e1,s2,e2);
 * if (sz(inter)==1)
 *   cout << "segments intersect at " << inter[0] << endl;
 * Status: stress-tested, tested on kattis:intersection
 */
#pragma once

#include "Point.h"
#include "OnSegment.h"

template<class P> vector<P> segInter(P a, P b, P c, P d) {
        auto oa = c.cross(d, a), ob = c.cross(d, b),
             oc = a.cross(b, c), od = a.cross(b, d);
        // Checks if intersection is single non-endpoint point.
        if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
                return {(a * ob - b * oa) / (ob - oa)};
        set<P> s;
        if (onSegment(c, d, a)) s.insert(a);
        if (onSegment(c, d, b)) s.insert(b);
        if (onSegment(a, b, c)) s.insert(c);
        if (onSegment(a, b, d)) s.insert(d);
        return {all(s)};
}
```

8.33 sideOf

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-03-21
 * License: CC0
 * Source:
 * Description: Returns where $p$ is as seen from $s$ towards $e$. 1/0/-1 $\
     Leftrightarrow$ left/on line/right.
 * If the optional argument $eps$ is given 0 is returned if $p$ is within
     distance $eps$ from the line.
 * P is supposed to be Point<T> where T is e.g. double or long long.
 * It uses products in intermediate steps so watch out for overflow if using int
      or long long.
 * Usage:
 *      bool left = sideOf(p1,p2,q)==1;
 * Status: tested
 */
#pragma once

#include "Point.h"

template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
        auto a = (e-s).cross(p-s);
        double l = (e-s).dist()*eps;
        return (a > l) - (a < -l);
}
```

8.34 sphericalDistance

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-04-07
 * License: CC0
 * Source: My geometric reasoning
 * Description: Returns the shortest distance on the sphere with radius radius
     between the points
 * with azimuthal angles (longitude) f1 ($\phi_1$) and f2 ($\phi_2$) from x axis
      and zenith angles
 * (latitude) t1 ($\theta_1$) and t2 ($\theta_2$) from z axis (0 = north pole).
     All angles measured
 * in radians. The algorithm starts by converting the spherical coordinates to
     cartesian coordinates
 * so if that is what you have you can use only the two last rows. dx*radius is
     then the difference
 * between the two points in the x direction and d*radius is the total distance
     between the points.
 * Status: tested on kattis:airlinehub
 */
#pragma once

double sphericalDistance(double f1, double t1,
                         double f2, double t2, double radius) {
        double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
        double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
        double dz = cos(t2) - cos(t1);
        double d = sqrt(dx*dx + dy*dy + dz*dz);
        return radius*2*asin(d/2);
}
```

9 kactl forked/graph

9.1 2sat

```
/**
 * Author: Emil Lenngren, Simon Lindholm
 * Date: 2011-11-29
 * License: CC0
 * Source: folklore
 * Description: Calculates a valid assignment to boolean variables a, b, c,...
     to a 2-SAT problem,
 * so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$
 * becomes true, or reports that it is unsatisfiable.
 * Negated variables are represented by bit-inversions (\texttt{\tilde{}x}).
 * Usage:
 *  TwoSat ts(number of boolean variables);
 *  ts.either(0, \tilde3); // Var 0 is true or var 3 is false
 *  ts.setValue(2); // Var 2 is true
 *  ts.atMostOne({0,\tilde1,2}); // <= 1 of vars 0, \tilde1 and 2 are true
 *  ts.solve(); // Returns true iff it is solvable
 *  ts.values[0..N-1] holds the assigned values to the vars
 * Time: O(N+E), where N is the number of boolean variables, and E is the number
      of clauses.
 * Status: stress-tested
 */
#pragma once

struct TwoSat {
        int N;
        vector<vi> gr;
        vi values; // 0 = false, 1 = true

        TwoSat(int n = 0) : N(n), gr(2*n) {}

        int addVar() { // (optional)
                gr.emplace_back();
                gr.emplace_back();
                return N++;
        }
```

```
        void either(int f, int j) {
                f = max(2*f, -1-2*f);
                j = max(2*j, -1-2*j);
                gr[f].push_back(j^1);
                gr[j].push_back(f^1);
        }
        void setValue(int x) { either(x, x); }

        void atMostOne(const vi& li) { // (optional)
                if (sz(li) <= 1) return;
                int cur = ~li[0];
                rep(i,2,sz(li)) {
                        int next = addVar();
                        either(cur, ~li[i]);
                        either(cur, next);
                        either(~li[i], next);
                        cur = ~next;
                }
                either(cur, ~li[1]);
        }

        vi val, comp, z; int time = 0;
        int dfs(int i) {
                int low = val[i] = ++time, x; z.push_back(i);
                for(int e : gr[i]) if (!comp[e])
                        low = min(low, val[e] ?: dfs(e));
                if (low == val[i]) do {
                        x = z.back(); z.pop_back();
                        comp[x] = low;
                        if (values[x>>1] == -1)
                                values[x>>1] = x&1;
                } while (x != i);
                return val[i] = low;
        }

        bool solve() {
                values.assign(N, -1);
                val.assign(2*N, 0); comp = val;
                rep(i,0,2*N) if (!comp[i]) dfs(i);
                rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
                return 1;
        }
};
```

9.2 BellmanFord

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-23
 * License: CC0
 * Source: http://en.wikipedia.org/wiki/Bellman-Ford_algorithm
 * Description: Calculates shortest paths from $s$ in a graph that might have
     negative edge weights.
 * Unreachable nodes get dist = inf; nodes reachable through negative-weight
     cycles get dist = -inf.
 * Assumes $V^2 \max |w_i| < \tilde{} 2^{63}$.
 * Time: O(VE)
 * Status: Tested on kattis:shortestpath3
 */
#pragma once

const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
        nodes[s].dist = 0;
        sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

        int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
        rep(i,0,lim) for (Ed ed : eds) {
                Node cur = nodes[ed.a], &dest = nodes[ed.b];
                if (abs(cur.dist) == inf) continue;
                ll d = cur.dist + ed.w;
                if (d < dest.dist) {
                        dest.prev = ed.a;
                        dest.dist = (i < lim-1 ? d : -inf);
                }
        }
        rep(i,0,lim) for (Ed e : eds) {
                if (nodes[e.a].dist == -inf)
                        nodes[e.b].dist = -inf;
        }
}
```

9.3 BiconnectedComponents

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-17
 * License: CC0
 * Source: folklore
 * Description: Finds all biconnected components in an undirected graph, and
 *  runs a callback for the edges in each. In a biconnected component there
 *  are at least two distinct paths between any two nodes. Note that a node can
 *  be in several components. An edge which is not in a component is a bridge,
 *  i.e., not part of any cycle.
 * Usage:
 *  int eid = 0; ed.resize(N);
 *  for each edge (a,b) {
 *    ed[a].emplace_back(b, eid);
 *    ed[b].emplace_back(a, eid++); }
 *  bicomps([\&](const vi\& edgelist) {...});
 * Time: O(E + V)
 * Status: tested during MIPT ICPC Workshop 2017
 */
#pragma once

vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
        int me = num[at] = ++Time, top = me;
        for (auto [y, e] : ed[at]) if (e != par) {
                if (num[y]) {
                        top = min(top, num[y]);
                        if (num[y] < me)
                                st.push_back(e);
                } else {
                        int si = sz(st);
                        int up = dfs(y, e, f);
                        top = min(top, up);
                        if (up == me) {
                                st.push_back(e);
                                f(vi(st.begin() + si, st.end()));
                                st.resize(si);
                        }
                        else if (up < me) st.push_back(e);
                        else { /* e is a bridge */ }
                }
        }
        return top;
}

template<class F>
void bicomps(F f) {
        num.assign(sz(ed), 0);
        rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

9.4 BinaryLifting

```
/**
 * Author: Johan Sannemo
 * Date: 2015-02-06
```

```
 * License: CC0
 * Source: Folklore
 * Description: Calculate power of two jumps in a tree,
 * to support fast upward jumps and LCAs.
 * Assumes the root node points to itself.
 * Time: construction $O(N \log N)$, queries $O(\log N)$
 * Status: Tested at Petrozavodsk, also stress-tested via LCA.cpp
 */
#pragma once

vector<vi> treeJump(vi& P){
	int on = 1, d = 1;
	while(on < sz(P)) on *= 2, d++;
	vector<vi> jmp(d, P);
	rep(i,1,d) rep(j,0,sz(P))
		jmp[i][j] = jmp[i-1][jmp[i-1][j]];
	return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
	rep(i,0,sz(tbl))
		if(steps&(1<<i)) nod = tbl[i][nod];
	return nod;
}

int lca(vector<vi>& tbl, vi& depth, int a, int b) {
	if (depth[a] < depth[b]) swap(a, b);
	a = jmp(tbl, a, depth[a] - depth[b]);
	if (a == b) return a;
	for (int i = sz(tbl); i--;) {
		int c = tbl[i][a], d = tbl[i][b];
		if (c != d) a = c, b = d;
	}
	return tbl[0][a];
}
```

9.5 CompressTree

```
/**
 * Author: Simon Lindholm
 * Date: 2016-01-14
 * License: CC0
 * Description: Given a rooted tree and a subset S of nodes, compute the minimal
 * subtree that contains all the nodes by adding all (at most $|S|-1$)
 * pairwise LCA's and compressing edges.
 * Returns a list of (par, orig\_index) representing a tree rooted at 0.
 * The root points to itself.
 * Time: $O(|S| \log |S|)$
 * Status: Tested at CodeForces
 */
#pragma once

#include "LCA.h"

typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
	static vi rev; rev.resize(sz(lca.time));
	vi li = subset, &T = lca.time;
	auto cmp = [&](int a, int b) { return T[a] < T[b]; };
	sort(all(li), cmp);
	int m = sz(li)-1;
	rep(i,0,m) {
		int a = li[i], b = li[i+1];
		li.push_back(lca.lca(a, b));
	}
	sort(all(li), cmp);
	li.erase(unique(all(li)), li.end());
	rep(i,0,sz(li)) rev[li[i]] = i;
	vpi ret = {pii(0, li[0])};
	rep(i,0,sz(li)-1) {
		int a = li[i], b = li[i+1];
		ret.emplace_back(rev[lca.lca(a, b)], b);
	}
	return ret;
}
```

9.6 DFSMatching

```
/**
 * Author: Lukas Polacek
 * Date: 2009-10-28
 * License: CC0
 * Source:
 * Description: Simple bipartite matching algorithm. Graph $g$ should be a list
 * of neighbors of the left partition, and $btoa$ should be a vector full of
 * -1's of the same size as the right partition. Returns the size of
 * the matching. $btoa[i]$ will be the match for vertex $i$ on the right side,
 * or $-1$ if it's not matched.
 * Time: O(VE)
 * Usage: vi btoa(m, -1); dfsMatching(g, btoa);
 * Status: works
 */
#pragma once

bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
	if (btoa[j] == -1) return 1;
	vis[j] = 1; int di = btoa[j];
	for (int e : g[di])
		if (!vis[e] && find(e, g, btoa, vis)) {
			btoa[e] = di;
			return 1;
		}
	return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
	vi vis;
	rep(i,0,sz(g)) {
		vis.assign(sz(btoa), 0);
		for (int j : g[i])
			if (find(j, g, btoa, vis)) {
				btoa[j] = i;
				break;
			}
	}
	return sz(btoa) - (int)count(all(btoa), -1);
}
```

9.7 Dinic

```
/**
 * Author: chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://cp-algorithms.com/graph/dinic.html
 * Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max |\
    text{cap}|$.
 * $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite
    matching.
 * Status: Tested on SPOJ FASTFLOW and SPOJ MATCHING, stress-tested
 */
#pragma once

struct Dinic {
	struct Edge {
		int to, rev;
		ll c, oc;
		ll flow() { return max(oc - c, 0LL); } // if you need flows
	};
	vi lvl, ptr, q;
	vector<vector<Edge>> adj;
	Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
	void addEdge(int a, int b, ll c, ll rcap = 0) {
		adj[a].push_back({b, sz(adj[b]), c, c});
		adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
	}
	ll dfs(int v, int t, ll f) {
		if (v == t || !f) return f;
		for (int& i = ptr[v]; i < sz(adj[v]); i++) {
			Edge& e = adj[v][i];
```

```
                        if (lvl[e.to] == lvl[v] + 1)
                                if (ll p = dfs(e.to, t, min(f, e.c))) {
                                        e.c -= p, adj[e.to][e.rev].c += p;
                                        return p;
                                }
                        }
                        return 0;
                }
                ll calc(int s, int t) {
                        ll flow = 0; q[0] = s;
                        rep(L,0,31) do { // 'int L=30' maybe faster for random data
                                lvl = ptr = vi(sz(q));
                                int qi = 0, qe = lvl[s] = 1;
                                while (qi < qe && !lvl[t]) {
                                        int v = q[qi++];
                                        for (Edge e : adj[v])
                                                if (!lvl[e.to] && e.c >> (30 - L))
                                                        q[qe++] = e.to, lvl[e.to] = lvl[
                                                            v] + 1;
                                }
                                while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
                        } while (lvl[t]);
                        return flow;
                }
                bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

9.8 DirectedMST

```
/**
 * Author: chilli, Takanori MAEHARA, Benq, Simon Lindholm
 * Date: 2019-05-10
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/master/graph/
     arborescence.cc
 * and https://github.com/bqi343/USACO/blob/42
     d177dfb9d6ce350389583cfa71484eb8ae614c/Implementations/content/graphs
     %20(12)/Advanced/DirectedMST.h for the reconstruction
 * Description: Finds a minimum spanning
 * tree/arborescence of a directed graph, given a root node. If no MST exists,
     returns -1.
 * Time: O(E \log V)
 * Status: Stress-tested, also tested on NWERC 2018 fastestspeedrun
 */
#pragma once

#include "../data-structures/UnionFindRollback.h"

struct Edge { int a, b; ll w; };
struct Node { /// lazy skew heap node
        Edge key;
        Node *l, *r;
        ll delta;
        void prop() {
                key.w += delta;
                if (l) l->delta += delta;
                if (r) r->delta += delta;
                delta = 0;
        }
        Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
        if (!a || !b) return a ?: b;
        a->prop(), b->prop();
        if (a->key.w > b->key.w) swap(a, b);
        swap(a->l, (a->r = merge(b, a->r)));
        return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
        RollbackUF uf(n);
        vector<Node*> heap(n);
        for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
        ll res = 0;
        vi seen(n, -1), path(n), par(n);
        seen[r] = r;
        vector<Edge> Q(n), in(n, {-1,-1}), comp;
        deque<tuple<int, int, vector<Edge>>> cycs;
        rep(s,0,n) {
                int u = s, qi = 0, w;
                while (seen[u] < 0) {
                        if (!heap[u]) return {-1,{}};
                        Edge e = heap[u]->top();
                        heap[u]->delta -= e.w, pop(heap[u]);
                        Q[qi] = e, path[qi++] = u, seen[u] = s;
                        res += e.w, u = uf.find(e.a);
                        if (seen[u] == s) { /// found cycle, contract
                                Node* cyc = 0;
                                int end = qi, time = uf.time();
                                do cyc = merge(cyc, heap[w = path[--qi]]);
                                while (uf.join(u, w));
                                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                                cycs.push_front({u, time, {&Q[qi], &Q[end]}});
                        }
                }
                rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
        }

        for (auto& [u,t,comp] : cycs) { // restore sol (optional)
                uf.rollback(t);
                Edge inEdge = in[u];
                for (auto& e : comp) in[uf.find(e.b)] = e;
                in[uf.find(inEdge.b)] = inEdge;
        }
        rep(i,0,n) par[i] = in[i].a;
        return {res, par};
}
```

9.9 EdgeColoring

```
/**
 * Author: Simon Lindholm
 * Date: 2020-10-12
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Misra_%26_Gries_edge_coloring_algorithm
 * https://codeforces.com/blog/entry/75431 for the note about bipartite graphs.
 * Description: Given a simple, undirected graph with max degree $D$, computes a
 * $(D + 1)$-coloring of the edges such that no neighboring edges share a color.
 * ($D$-coloring is NP-hard, but can be done for bipartite graphs by repeated
     matchings of
 * max-degree nodes.)
 * Time: O(NM)
 * Status: stress-tested, tested on kattis:gamescheduling
 */
#pragma once

vi edgeColoring(int N, vector<pii> eds) {
        vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
        for (pii e : eds) ++cc[e.first], ++cc[e.second];
        int u, v, ncols = *max_element(all(cc)) + 1;
        vector<vi> adj(N, vi(ncols, -1));
        for (pii e : eds) {
                tie(u, v) = e;
                fan[0] = v;
                loc.assign(ncols, 0);
                int at = u, end = u, d, c = free[u], ind = 0, i = 0;
                while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
                        loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
                cc[loc[d]] = c;
                for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
                        swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
                while (adj[fan[i]][d] != -1) {
                        int left = fan[i], right = fan[++i], e = cc[i];
                        adj[u][e] = left;
                        adj[left][e] = u;
                        adj[right][e] = -1;
                        free[right] = e;
                }
                adj[u][d] = fan[i];
```

```
                adj[fan[i]][d] = u;
                for (int y : {fan[0], u, end})
                        for (int& z = free[y] = 0; adj[y][z] != -1; z++);
        }
        rep(i,0,sz(eds))
                for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
        return ret;
}
```

9.10 EdmondsKarp

```
/**
 * Author: Chen Xing
 * Date: 2009-10-13
 * License: CC0
 * Source: N/A
 * Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge
      flow values, compare
 * capacities before and after, and take the positive values only.
 * Status: stress-tested
 */
#pragma once

template<class T> T edmondsKarp(vector<unordered_map<int, T>>&
                graph, int source, int sink) {
        assert(source != sink);
        T flow = 0;
        vi par(sz(graph)), q = par;

        for (;;) {
                fill(all(par), -1);
                par[source] = 0;
                int ptr = 1;
                q[0] = source;

                rep(i,0,ptr) {
                        int x = q[i];
                        for (auto e : graph[x]) {
                                if (par[e.first] == -1 && e.second > 0) {
                                        par[e.first] = x;
                                        q[ptr++] = e.first;
                                        if (e.first == sink) goto out;
                                }
                        }
                }
                return flow;
out:
                T inc = numeric_limits<T>::max();
                for (int y = sink; y != source; y = par[y])
                        inc = min(inc, graph[par[y]][y]);

                flow += inc;
                for (int y = sink; y != source; y = par[y]) {
                        int p = par[y];
                        if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
                        graph[y][p] += inc;
                }
        }
}
```

9.11 EulerWalk

```
/**
 * Author: Simon Lindholm
 * Date: 2019-12-31
 * License: CC0
 * Source: folklore
 * Description: Eulerian undirected/directed path/cycle algorithm.
 * Input should be a vector of (dest, global edge index), where
 * for undirected graphs, forward/backward edges have the same index.
 * Returns a list of nodes in the Eulerian path/cycle with src at both start and
      end, or
 * empty list if no cycle/path exists.
 * To get edge indices back, add .second to s and ret.
 * Time: O(V + E)
 * Status: stress-tested
 */
#pragma once

vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
        int n = sz(gr);
        vi D(n), its(n), eu(nedges), ret, s = {src};
        D[src]++; // to allow Euler paths, not just cycles
        while (!s.empty()) {
                int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
                if (it == end){ ret.push_back(x); s.pop_back(); continue; }
                tie(y, e) = gr[x][it++];
                if (!eu[e]) {
                        D[x]--, D[y]++;
                        eu[e] = 1; s.push_back(y);
                }}
        for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
        return {ret.rbegin(), ret.rend()};
}
```

9.12 FloydWarshall

```
/**
 * Author: Simon Lindholm
 * Date: 2016-12-15
 * License: CC0
 * Source: http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm
 * Description: Calculates all-pairs shortest path in a directed graph that
      might have negative edge weights.
 * Input is an distance matrix $m$, where $m[i][j] = \texttt{inf}$ if $i$ and
      $j$ are not adjacent.
 * As output, $m[i][j]$ is set to the shortest distance between $i$ and $j$, \
      texttt{inf} if no path,
 * or \texttt{-inf} if the path goes through a negative-weight cycle.
 * Time: O(N^3)
 * Status: slightly tested
 */
#pragma once

const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
        int n = sz(m);
        rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
        rep(k,0,n) rep(i,0,n) rep(j,0,n)
                if (m[i][k] != inf && m[k][j] != inf) {
                        auto newDist = max(m[i][k] + m[k][j], -inf);
                        m[i][j] = min(m[i][j], newDist);
                }
        rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
                if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

9.13 GeneralMatching

```
/**
 * Author: Simon Lindholm
 * Date: 2016-12-09
 * License: CC0
 * Source: http://www.mimuw.edu.pl/~mucha/pub/mucha_sankowski_focs04.pdf
 * Description: Matching for general graphs.
 * Fails with probability $N / mod$.
 * Time: O(N^3)
 * Status: not very well tested
 */
#pragma once

#include "../numerical/MatrixInverse-mod.h"

vector<pii> generalMatching(int N, vector<pii>& ed) {
```

```
	vector<vector<ll>> mat(N, vector<ll>(N)), A;
	for (pii pa : ed) {
		int a = pa.first, b = pa.second, r = rand() % mod;
		mat[a][b] = r, mat[b][a] = (mod - r) % mod;
	}

	int r = matInv(A = mat), M = 2*N - r, fi, fj;
	assert(r % 2 == 0);

	if (M != N) do {
		mat.resize(M, vector<ll>(M));
		rep(i,0,N) {
			mat[i].resize(M);
			rep(j,N,M) {
				int r = rand() % mod;
				mat[i][j] = r, mat[j][i] = (mod - r) % mod;
			}
		}
	} while (matInv(A = mat) != M);

	vi has(M, 1); vector<pii> ret;
	rep(it,0,M/2) {
		rep(i,0,M) if (has[i])
			rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
				fi = i; fj = j; goto done;
			} assert(0); done:
		if (fj < N) ret.emplace_back(fi, fj);
		has[fi] = has[fj] = 0;
		rep(sw,0,2) {
			ll a = modpow(A[fi][fj], mod-2);
			rep(i,0,M) if (has[i] && A[i][fj]) {
				ll b = A[i][fj] * a % mod;
				rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) %
				    mod;
			}
			swap(fi,fj);
		}
	}
	return ret;
}
```

9.14 GlobalMinCut

```
/**
 * Author: Simon Lindholm
 * Date: 2021-01-09
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Stoer%E2%80%93Wagner_algorithm
 * Description: Find a global minimum cut in an undirected graph, as represented
       by an adjacency matrix.
 * Time: O(V^3)
 * Status: Stress-tested together with GomoryHu
 */
#pragma once

pair<int, vi> globalMinCut(vector<vi> mat) {
	pair<int, vi> best = {INT_MAX, {}};
	int n = sz(mat);
	vector<vi> co(n);
	rep(i,0,n) co[i] = {i};
	rep(ph,1,n) {
		vi w = mat[0];
		size_t s = 0, t = 0;
		rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio. queue
			w[t] = INT_MIN;
			s = t, t = max_element(all(w)) - w.begin();
			rep(i,0,n) w[i] += mat[t][i];
		}
		best = min(best, {w[t] - mat[t][t], co[t]});
		co[s].insert(co[s].end(), all(co[t]));
		rep(i,0,n) mat[s][i] += mat[t][i];
		rep(i,0,n) mat[i][s] = mat[s][i];
		mat[0][t] = INT_MIN;
	}
	return best;
}
```

9.15 GomoryHu

```
/**
 * Author: chilli, Takanori MAEHARA
 * Date: 2020-04-03
 * License: CC0
 * Source: https://github.com/spaghetti-source/algorithm/blob/master/graph/
     gomory_hu_tree.cc#L102
 * Description: Given a list of edges representing an undirected flow graph,
 * returns edges of the Gomory-Hu tree. The max flow between any pair of
 * vertices is given by minimum edge weight along the Gomory-Hu tree path.
 * Time: $O(V)$ Flow Computations
 * Status: Tested on CERC 2015 J, stress-tested
 *
 * Details: The implementation used here is not actually the original
 * Gomory-Hu, but Gusfield's simplified version: "Very simple methods for all
 * pairs network flow analysis". PushRelabel is used here, but any flow
 * implementation that supports `leftOfMinCut` also works.
 */
#pragma once

#include "PushRelabel.h"

typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
	vector<Edge> tree;
	vi par(N);
	rep(i,1,N) {
		PushRelabel D(N); // Dinic also works
		for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
		tree.push_back({i, par[i], D.calc(i, par[i])});
		rep(j,i+1,N)
			if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
	}
	return tree;
}
```

9.16 HLD

```
/**
 * Author: Benjamin Qi, Oleksandr Kulkov, chilli
 * Date: 2020-01-12
 * License: CC0
 * Source: https://codeforces.com/blog/entry/53170, https://github.com/bqi343/
     USACO/blob/master/Implementations/content/graphs%20(12)/Trees%20(10)/HLD
     %20(10.3).h
 * Description: Decomposes a tree into vertex disjoint heavy paths and light
 * edges such that the path from any leaf to the root contains at most log(n)
 * light edges. Code does additive modifications and max queries, but can
 * support commutative segtree modifications/queries on paths and subtrees.
 * Takes as input the full adjacency list. VALS\_EDGES being true means that
 * values are stored in the edges, as opposed to the nodes. All values
 * initialized to the segtree default. Root must be 0.
 * Time: O((\log N)^2)
 * Status: stress-tested against old HLD
 */
#pragma once

#include "../data-structures/LazySegmentTree.h"

template <bool VALS_EDGES> struct HLD {
	int N, tim = 0;
	vector<vi> adj;
	vi par, siz, rt, pos;
	Node *tree;
	HLD(vector<vi> adj_)
		: N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
		  rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
	void dfsSz(int v) {
		for (int& u : adj[v]) {
			adj[u].erase(find(all(adj[u]), v));
			par[u] = v;
			dfsSz(u);
```

```
                siz[v] += siz[u];
                if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
        }
    }
    void dfsHld(int v) {
        pos[v] = tim++;
        for (int u : adj[v]) {
            rt[u] = (u == adj[v][0] ? rt[v] : u);
            dfsHld(u);
        }
    }
    template <class B> void process(int u, int v, B op) {
        for (;; v = par[rt[v]]) {
            if (pos[u] > pos[v]) swap(u, v);
            if (rt[u] == rt[v]) break;
            op(pos[rt[v]], pos[v] + 1);
        }
        op(pos[u] + VALS_EDGES, pos[v] + 1);
    }
    void modifyPath(int u, int v, int val) {
        process(u, v, [&](int l, int r) { tree->add(l, r, val); });
    }
    int queryPath(int u, int v) { // Modify depending on problem
        int res = -1e9;
        process(u, v, [&](int l, int r) {
            res = max(res, tree->query(l, r));
        });
        return res;
    }
    int querySubtree(int v) { // modifySubtree is similar
        return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);
    }
};
```

9.17 hopcroftKarp

```
/**
 * Author: Chen Xing
 * Date: 2009-10-13
 * License: CC0
 * Source: N/A
 * Description: Fast bipartite matching algorithm. Graph $g$ should be a list
 * of neighbors of the left partition, and $btoa$ should be a vector full of
 * -1's of the same size as the right partition. Returns the size of
 * the matching. $btoa[i]$ will be the match for vertex $i$ on the right side,
 * or $-1$ if it's not matched.
 * Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
 * Time: O(\sqrt{V}E)
 * Status: stress-tested by MinimumVertexCover, and tested on oldkattis.
     adkbipmatch and SPOJ:MATCHING
 */
#pragma once

bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0);
        fill(all(B), 0);
        /// Find the starting nodes for BFS (i.e. layer 0).
        cur.clear();
        for (int a : btoa) if(a != -1) A[a] = -1;
        rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
        /// Find all layers using bfs.
        for (int lay = 1;; lay++) {
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
                else if (btoa[b] != a && !B[b]) {
                    B[b] = lay;
                    next.push_back(btoa[b]);
                }
            }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay;
            cur.swap(next);
        }
        /// Use DFS to scan for augmenting paths.
        rep(a,0,sz(g))
            res += dfs(a, 0, g, btoa, A, B);
    }
}
```

9.18 LCA

```
/**
 * Author: chilli, pajenegod
 * Date: 2020-02-20
 * License: CC0
 * Source: Folklore
 * Description: Data structure for computing lowest common ancestors in a tree
 * (with 0 as root). C should be an adjacency list of the tree, either directed
 * or undirected.
 * Time: $O(N \log N + Q)$
 * Status: stress-tested
 */
#pragma once

#include "../data-structures/RMQ.h"

struct LCA {
    int T = 0;
    vi time, path, ret;
    RMQ<int> rmq;

    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
    void dfs(vector<vi>& C, int v, int par) {
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.push_back(v), ret.push_back(time[v]);
            dfs(C, y, v);
        }
    }

    int lca(int a, int b) {
        if (a == b) return a;
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)];
    }
    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};
```

9.19 LinkCutTree

```
/**
 * Author: Simon Lindholm
 * Date: 2016-07-25
 * Source: https://github.com/ngthanhtrung23/ACM_Notebook_new/blob/master/
     DataStructure/LinkCutTree.h
 * Description: Represents a forest of unrooted trees. You can add and remove
 * edges (as long as the result is still a forest), and check whether
```

```
 * two nodes are in the same tree.
 * Time: All operations take amortized O(\log N).
 * Status: Stress-tested a bit for N <= 20
 */
#pragma once

struct Node { // Splay tree. Root's pp contains tree's parent.
	Node *p = 0, *pp = 0, *c[2];
	bool flip = 0;
	Node() { c[0] = c[1] = 0; fix(); }
	void fix() {
		if (c[0]) c[0]->p = this;
		if (c[1]) c[1]->p = this;
		// (+ update sum of subtree elements etc. if wanted)
	}
	void pushFlip() {
		if (!flip) return;
		flip = 0; swap(c[0], c[1]);
		if (c[0]) c[0]->flip ^= 1;
		if (c[1]) c[1]->flip ^= 1;
	}
	int up() { return p ? p->c[1] == this : -1; }
	void rot(int i, int b) {
		int h = i ^ b;
		Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
		if ((y->p = p)) p->c[up()] = y;
		c[i] = z->c[i ^ 1];
		if (b < 2) {
			x->c[h] = y->c[h ^ 1];
			y->c[h ^ 1] = x;
		}
		z->c[i ^ 1] = this;
		fix(); x->fix(); y->fix();
		if (p) p->fix();
		swap(pp, y->pp);
	}
	void splay() { /// Splay this up to the root. Always finishes without
	    flip set.
		for (pushFlip(); p; ) {
			if (p->p) p->p->pushFlip();
			p->pushFlip(); pushFlip();
			int c1 = up(), c2 = p->up();
			if (c2 == -1) p->rot(c1, 2);
			else p->p->rot(c2, c1 != c2);
		}
	}
	Node* first() { /// Return the min element of the subtree rooted at this
	    , splayed to the top.
		pushFlip();
		return c[0] ? c[0]->first() : (splay(), this);
	}
};

struct LinkCut {
	vector<Node> node;
	LinkCut(int N) : node(N) {}

	void link(int u, int v) { // add an edge (u, v)
		assert(!connected(u, v));
		makeRoot(&node[u]);
		node[u].pp = &node[v];
	}
	void cut(int u, int v) { // remove an edge (u, v)
		Node *x = &node[u], *top = &node[v];
		makeRoot(top); x->splay();
		assert(top == (x->pp ?: x->c[0]));
		if (x->pp) x->pp = 0;
		else {
			x->c[0] = top->p = 0;
			x->fix();
		}
	}
	bool connected(int u, int v) { // are u, v in the same tree?
		Node* nu = access(&node[u])->first();
		return nu == access(&node[v])->first();
	}
	void makeRoot(Node* u) { /// Move u to root of represented tree.
		access(u);
		u->splay();
		if(u->c[0]) {
			u->c[0]->p = 0;
			u->c[0]->flip ^= 1;
			u->c[0]->pp = u;
			u->c[0] = 0;
			u->fix();
		}
	}
	Node* access(Node* u) { /// Move u to root aux tree. Return the root of
	    the root aux tree.
		u->splay();
		while (Node* pp = u->pp) {
			pp->splay(); u->pp = 0;
			if (pp->c[1]) {
				pp->c[1]->p = 0; pp->c[1]->pp = pp; }
			pp->c[1] = u; pp->fix(); u = pp;
		}
		return u;
	}
};
```

9.20 MaximalCliques

```
/**
 * Author: Simon Lindholm
 * Date: 2018-07-18
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm
 * Description: Runs a callback for all maximal cliques in a graph (given as a
 * symmetric bitset matrix; self-edges not allowed). Callback is given a bitset
 * representing the maximal clique.
 * Time: O(3^{n/3}), much faster for sparse graphs
 * Status: stress-tested
 */
#pragma once
/// Possible optimization: on the top-most
/// recursion level, ignore 'cands', and go through nodes in order of increasing
/// degree, where degrees go down as nodes are removed.
/// (mostly irrelevant given MaximumClique)

typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
	if (!P.any()) { if (!X.any()) f(R); return; }
	auto q = (P | X)._Find_first();
	auto cands = P & ~eds[q];
	rep(i,0,sz(eds)) if (cands[i]) {
		R[i] = 1;
		cliques(eds, f, P & eds[i], X & eds[i], R);
		R[i] = P[i] = 0; X[i] = 1;
	}
}
```

9.21 MaximumClique

```
/**
 * Author: chilli, SJTU, Janez Konc
 * Date: 2019-05-10
 * License: GPL3+
 * Source: https://en.wikipedia.org/wiki/MaxCliqueDyn_maximum_clique_algorithm,
     https://gitlab.com/janezkonc/mcqd/blob/master/mcqd.h
 * Description: Quickly finds a maximum clique of a graph (given as symmetric
     bitset
 * matrix; self-edges not allowed). Can be used to find a maximum independent
 * set by finding a clique of the complement graph.
 * Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs
 * faster for sparse graphs.
 * Status: stress-tested
 */
typedef vector<bitset<200>> vb;
struct Maxclique {
	double limit=0.025, pk=0;
```

```
	struct Vertex { int i, d=0; };
	typedef vector<Vertex> vv;
	vb e;
	vv V;
	vector<vi> C;
	vi qmax, q, S, old;
	void init(vv& r) {
		for (auto& v : r) v.d = 0;
		for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
		sort(all(r), [](auto a, auto b) { return a.d > b.d; });
		int mxD = r[0].d;
		rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
	}
	void expand(vv& R, int lev = 1) {
		S[lev] += S[lev - 1] - old[lev];
		old[lev] = S[lev - 1];
		while (sz(R)) {
			if (sz(q) + R.back().d <= sz(qmax)) return;
			q.push_back(R.back().i);
			vv T;
			for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i})
				;
			if (sz(T)) {
				if (S[lev]++ / ++pk < limit) init(T);
				int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) +
					1, 1);
				C[1].clear(), C[2].clear();
				for (auto v : T) {
					int k = 1;
					auto f = [&](int i) { return e[v.i][i];
						};
					while (any_of(all(C[k]), f)) k++;
					if (k > mxk) mxk = k, C[mxk + 1].clear()
						;
					if (k < mnk) T[j++].i = v.i;
					C[k].push_back(v.i);
				}
				if (j > 0) T[j - 1].d = 0;
				rep(k,mnk,mxk + 1) for (int i : C[k])
					T[j].i = i, T[j++].d = k;
				expand(T, lev + 1);
			} else if (sz(q) > sz(qmax)) qmax = q;
			q.pop_back(), R.pop_back();
		}
	}
	vi maxClique() { init(V), expand(V); return qmax; }
	Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
		rep(i,0,sz(e)) V.push_back({i});
	}
};
```

9.22 MaximumIndependentSet

```
/**
 * Author: chilli
 * Date: 2019-05-17
 * Source: Wikipedia
 * Description: To obtain a maximum independent set of a graph, find a max
 * clique of the complement. If the graph is bipartite, see MinimumVertexCover.
 */
```

9.23 MinCostMaxFlow

```
/**
 * Author: Stanford
 * Date: Unknown
 * Source: Stanford Notebook
 * Description: Min-cost max-flow.
 *  If costs can be negative, call setpi before maxflow, but note that negative
     cost cycles are not supported.
 *  To obtain the actual flow, look at positive values only.
 * Status: Tested on kattis:mincostmaxflow, stress-tested against another
     implementation
 * Time: $O(F E \log(V))$ where F is max flow. $O(VE)$ for setpi.
 */
#pragma once

// #include <bits/extc++.h> /// include-line, keep-include

const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
	struct edge {
		int from, to, rev;
		ll cap, cost, flow;
	};
	int N;
	vector<vector<edge>> ed;
	vi seen;
	vector<ll> dist, pi;
	vector<edge*> par;

	MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

	void addEdge(int from, int to, ll cap, ll cost) {
		if (from == to) return;
		ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
		ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
	}

	void path(int s) {
		fill(all(seen), 0);
		fill(all(dist), INF);
		dist[s] = 0; ll di;

		__gnu_pbds::priority_queue<pair<ll, int>> q;
		vector<decltype(q)::point_iterator> its(N);
		q.push({ 0, s });

		while (!q.empty()) {
			s = q.top().second; q.pop();
			seen[s] = 1; di = dist[s] + pi[s];
			for (edge& e : ed[s]) if (!seen[e.to]) {
				ll val = di - pi[e.to] + e.cost;
				if (e.cap - e.flow > 0 && val < dist[e.to]) {
					dist[e.to] = val;
					par[e.to] = &e;
					if (its[e.to] == q.end())
						its[e.to] = q.push({ -dist[e.to
							], e.to });
					else
						q.modify(its[e.to], { -dist[e.to
							], e.to });
				}
			}
		}
		rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
	}

	pair<ll, ll> maxflow(int s, int t) {
		ll totflow = 0, totcost = 0;
		while (path(s), seen[t]) {
			ll fl = INF;
			for (edge* x = par[t]; x; x = par[x->from])
				fl = min(fl, x->cap - x->flow);

			totflow += fl;
			for (edge* x = par[t]; x; x = par[x->from]) {
				x->flow += fl;
				ed[x->to][x->rev].flow -= fl;
			}
		}
		rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
		return {totflow, totcost/2};
	}

	// If some costs can be negative, call this before maxflow:
	void setpi(int s) { // (otherwise, leave this out)
		fill(all(pi), INF); pi[s] = 0;
		int it = N, ch = 1; ll v;
```

```
                while (ch-- && it--)
                        rep(i,0,N) if (pi[i] != INF)
                          for (edge& e : ed[i]) if (e.cap)
                                  if ((v = pi[i] + e.cost) < pi[e.to])
                                          pi[e.to] = v, ch = 1;
                assert(it >= 0); // negative cost cycle
        }
};
```

9.24 MinCut

```
/**
 * Author: Simon Lindholm
 * Date: 2015-05-13
 * Source: Wikipedia
 * Description: After running max-flow, the left side of a min-cut from $s$ to
     $t$ is given
 * by all vertices reachable from $s$, only traversing edges with positive
     residual capacity.
 * Status: works
 */
```

9.25 MinimumVertexCover

```
/**
 * Author: Johan Sannemo, Simon Lindholm
 * Date: 2016-12-15
 * License: CC0
 * Description: Finds a minimum vertex cover in a bipartite graph.
 *  The size is the same as the size of a maximum matching, and
 *  the complement is a maximum independent set.
 * Status: stress-tested
 */
#pragma once

#include "DFSMatching.h"

vi cover(vector<vi>& g, int n, int m) {
        vi match(m, -1);
        int res = dfsMatching(g, match);
        vector<bool> lfound(n, true), seen(m);
        for (int it : match) if (it != -1) lfound[it] = false;
        vi q, cover;
        rep(i,0,n) if (lfound[i]) q.push_back(i);
        while (!q.empty()) {
                int i = q.back(); q.pop_back();
                lfound[i] = 1;
                for (int e : g[i]) if (!seen[e] && match[e] != -1) {
                        seen[e] = true;
                        q.push_back(match[e]);
                }
        }
        rep(i,0,n) if (!lfound[i]) cover.push_back(i);
        rep(i,0,m) if (seen[i]) cover.push_back(n+i);
        assert(sz(cover) == res);
        return cover;
}
```

9.26 PushRelabel

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-24
 * License: CC0
 * Source: Wikipedia, tinyKACTL
 * Description: Push-relabel using the highest label selection rule and the gap
     heuristic. Quite fast in practice.
 *  To obtain the actual flow, look at positive values only.
 * Time: $O(V^2\sqrt E)$
 * Status: Tested on Kattis and SPOJ, and stress-tested
 */
#pragma once


struct PushRelabel {
        struct Edge {
                int dest, back;
                ll f, c;
        };
        vector<vector<Edge>> g;
        vector<ll> ec;
        vector<Edge*> cur;
        vector<vi> hs; vi H;
        PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

        void addEdge(int s, int t, ll cap, ll rcap=0) {
                if (s == t) return;
                g[s].push_back({t, sz(g[t]), 0, cap});
                g[t].push_back({s, sz(g[s])-1, 0, rcap});
        }

        void addFlow(Edge& e, ll f) {
                Edge &back = g[e.dest][e.back];
                if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
                e.f += f; e.c -= f; ec[e.dest] += f;
                back.f -= f; back.c += f; ec[back.dest] -= f;
        }
        ll calc(int s, int t) {
                int v = sz(g); H[s] = v; ec[t] = 1;
                vi co(2*v); co[0] = v-1;
                rep(i,0,v) cur[i] = g[i].data();
                for (Edge& e : g[s]) addFlow(e, e.c);

                for (int hi = 0;;) {
                        while (hs[hi].empty()) if (!hi--) return -ec[s];
                        int u = hs[hi].back(); hs[hi].pop_back();
                        while (ec[u] > 0)  // discharge u
                                if (cur[u] == g[u].data() + sz(g[u])) {
                                        H[u] = 1e9;
                                        for (Edge& e : g[u]) if (e.c && H[u] > H
                                             [e.dest]+1)
                                                H[u] = H[e.dest]+1, cur[u] = &e;
                                        if (++co[H[u]], !--co[hi] && hi < v)
                                                rep(i,0,v) if (hi < H[i] && H[i]
                                                     < v)
                                                        --co[H[i]], H[i] = v +
                                                             1;
                                        hi = H[u];
                                } else if (cur[u]->c && H[u] == H[cur[u]->dest
                                     ]+1)
                                        addFlow(*cur[u], min(ec[u], cur[u]->c));
                                else ++cur[u];
                }
        }
        bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

9.27 SCC

```
/**
 * Author: Lukas Polacek
 * Date: 2009-10-28
 * License: CC0
 * Source: Czech graph algorithms book, by Demel. (Tarjan's algorithm)
 * Description: Finds strongly connected components in a
 * directed graph. If vertices $u, v$ belong to the same component,
 * we can reach $u$ from $v$ and vice versa.
 * Usage: scc(graph, [\&](vi\& v) { ... }) visits all components
 * in reverse topological order. comp[i] holds the component
 * index of a node (a component only has edges to components with
 * lower index). ncomps will contain the number of components.
 * Time: O(E + V)
 * Status: Bruteforce-tested for N <= 5
 */
#pragma once
```

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
        int low = val[j] = ++Time, x; z.push_back(j);
        for (auto e : g[j]) if (comp[e] < 0)
                low = min(low, val[e] ?: dfs(e,g,f));

        if (low == val[j]) {
                do {
                        x = z.back(); z.pop_back();
                        comp[x] = ncomps;
                        cont.push_back(x);
                } while (x != j);
                f(cont); cont.clear();
                ncomps++;
        }
        return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
        int n = sz(g);
        val.assign(n, 0); comp.assign(n, -1);
        Time = ncomps = 0;
        rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

9.28 TopoSort

```
/**
 * Author: Unknown
 * Date: 2002-09-13
 * Source: predates tinyKACTL
 * Description: Topological sorting. Given is an oriented graph.
 * Output is an ordering of vertices, such that there are edges only from left
     to right.
 * If there are cycles, the returned list will have size smaller than $n$ --
     nodes reachable
 * from cycles will not be returned.
 * Time: $O(|V|+|E|)$
 * Status: stress-tested
 */
#pragma once

vi topoSort(const vector<vi>& gr) {
        vi indeg(sz(gr)), q;
        for (auto& li : gr) for (int x : li) indeg[x]++;
        rep(i,0,sz(gr)) if (indeg[i] == 0) q.push_back(i);
        rep(j,0,sz(q)) for (int x : gr[q[j]])
                if (--indeg[x] == 0) q.push_back(x);
        return q;
}
```

9.29 WeightedMatching

```
/**
 * Author: Benjamin Qi, chilli
 * Date: 2020-04-04
 * License: CC0
 * Source: https://github.com/bqi343/USACO/blob/master/Implementations/content/
     graphs%20(12)/Matching/Hungarian.h
 * Description: Given a weighted bipartite graph, matches every node on
 * the left with a node on the right such that no
 * nodes are in two matchings and the sum of the edge weights is minimal. Takes
 * cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and
 * returns (min cost, match), where L[i] is matched with
 * R[match[i]]. Negate costs for max cost. Requires $N \le M$.
 * Time: O(N^2M)
 * Status: Tested on kattis:cordonbleu, stress-tested
 */
#pragma once

pair<int, vi> hungarian(const vector<vi> &a) {
        if (a.empty()) return {0, {}};
        int n = sz(a) + 1, m = sz(a[0]) + 1;
        vi u(n), v(m), p(m), ans(n - 1);
        rep(i,1,n) {
                p[0] = i;
                int j0 = 0; // add "dummy" worker 0
                vi dist(m, INT_MAX), pre(m, -1);
                vector<bool> done(m + 1);
                do { // dijkstra
                        done[j0] = true;
                        int i0 = p[j0], j1, delta = INT_MAX;
                        rep(j,1,m) if (!done[j]) {
                                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                                if (dist[j] < delta) delta = dist[j], j1 = j;
                        }
                        rep(j,0,m) {
                                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                                else dist[j] -= delta;
                        }
                        j0 = j1;
                } while (p[j0]);
                while (j0) { // update alternating path
                        int j1 = pre[j0];
                        p[j0] = p[j1], j0 = j1;
                }
        }
        rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
        return {-v[0], ans}; // min cost
}
```

10 kactl forked/number-theory

10.1 ContinuedFractions

```
/**
 * Author: Simon Lindholm
 * Date: 2018-07-15
 * License: CC0
 * Source: Wikipedia
 * Description: Given $N$ and a real number $x \ge 0$, finds the closest
     rational approximation $p/q$ with $p, q \le N$.
 * It will obey $|p/q - x| \le 1/qN$.
 *
 * For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$.
 * ($p_k/q_k$ alternates between $>x$ and $<x$.)
 * If $x$ is rational, $y$ eventually becomes $\infty$;
 * if $x$ is the root of a degree $2$ polynomial the $a$'s eventually become
     cyclic.
 * Time: O(\log N)
 * Status: stress-tested for n <= 300
 */

typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
        ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
        for (;;) {
                ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
                   a = (ll)floor(y), b = min(a, lim),
                   NP = b*P + LP, NQ = b*Q + LQ;
                if (a > b) {
                        // If b > a/2, we have a semi-convergent that gives us a
                        // better approximation; if b = a/2, we *may* have one.
                        // Return {P, Q} here for a more canonical approximation
                            .
                        return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
                                make_pair(NP, NQ) : make_pair(P, Q);
                }
                if (abs(y = 1/(y - (d)a)) > 3*N) {
                        return {NP, NQ};
                }
                LP = P; P = NP;
                LQ = Q; Q = NQ;
```

```
	}
}
```

10.2 CRT

```
/**
 * Author: Simon Lindholm
 * Date: 2019-05-22
 * License: CC0
 * Description: Chinese Remainder Theorem.
 *
 * \texttt{crt(a, m, b, n)} computes $x$ such that $x\equiv a \pmod m$, $x\equiv
      b \pmod n$.
 * If $|a| < m$ and $|b| < n$, $x$ will obey $0 \le x < \text{lcm}(m, n)$.
 * Assumes $mn < 2^{62}$.
 * Time: $\log(n)$
 * Status: Works
 */
#pragma once

#include "euclid.h"

ll crt(ll a, ll m, ll b, ll n) {
	if (n > m) swap(a, b), swap(m, n);
	ll x, y, g = euclid(m, n, x, y);
	assert((a - b) % g == 0); // else no solution
	x = (b - a) % n * x % n / g * m + a;
	return x < 0 ? x + m*n/g : x;
}
```

10.3 Eratosthenes

```
/**
 * Author: Hakan Terelius
 * Date: 2009-08-26
 * License: CC0
 * Source: http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes
 * Description: Prime sieve for generating all primes up to a certain limit.
     isprime$[i]$ is true iff $i$ is a prime.
 * Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30\% faster if only odd indices
     are stored.
 * Status: Tested
 */
#pragma once

const int MAX_PR = 5'000'000;
bitset<MAX_PR> isprime;
vi eratosthenesSieve(int lim) {
	isprime.set(); isprime[0] = isprime[1] = 0;
	for (int i = 4; i < lim; i += 2) isprime[i] = 0;
	for (int i = 3; i*i < lim; i += 2) if (isprime[i])
		for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
	vi pr;
	rep(i,2,lim) if (isprime[i]) pr.push_back(i);
	return pr;
}
```

10.4 euclid

```
/**
 * Author: Unknown
 * Date: 2002-09-15
 * Source: predates tinyKACTL
 * Description: Finds two integers $x$ and $y$, such that $ax+by=\gcd(a,b)$. If
 * you just need gcd, use the built in \texttt{\_\_gcd} instead.
 * If $a$ and $b$ are coprime, then $x$ is the inverse of $a \pmod{b}$.
 */
#pragma once

ll euclid(ll a, ll b, ll &x, ll &y) {
	if (!b) return x = 1, y = 0, a;
	ll d = euclid(b, a % b, y, x);
	return y -= a/b * x, d;
}
```

10.5 Factor

```
/**
 * Author: chilli, SJTU, pajenegod
 * Date: 2020-03-04
 * License: CC0
 * Source: own
 * Description: Pollard-rho randomized factorization algorithm. Returns prime
 * factors of a number, in arbitrary order (e.g. 2299 -> \{11, 19, 11\}).
 * Time: $O(n^{1/4})$, less for numbers with small factors.
 * Status: stress-tested
 *
 * Details: This implementation uses the improvement described here
 * (https://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm#Variants), where
 * one can accumulate gcd calls by some factor (40 chosen here through
 * exhaustive testing). This improves performance by approximately 6-10x
 * depending on the inputs and speed of gcd. Benchmark found here:
 * (https://ideone.com/nGGD9T)
 *
 * GCD can be improved by a factor of 1.75x using Binary GCD
 * (https://lemire.me/blog/2013/12/26/fastest-way-to-compute-the-greatest-common
     -divisor/).
 * However, with the gcd accumulation the bottleneck moves from the gcd calls
 * to the modmul. As GCD only constitutes ~12% of runtime, speeding it up
 * doesn't matter so much.
 *
 * This code can probably be sped up by using a faster mod mul - potentially
 * montgomery reduction on 128 bit integers.
 * Alternatively, one can use a quadratic sieve for an asymptotic improvement,
 * which starts being faster in practice around 1e13.
 *
 * Brent's cycle finding algorithm was tested, but doesn't reduce modmul calls
 * significantly.
 *
 * Subtle implementation notes:
 * - prd starts off as 2 to handle the case n = 4; it's harmless for other n
 *   since we're guaranteed that n > 2. (Pollard rho has problems with prime
 *   powers in general, but all larger ones happen to work.)
 * - t starts off as 30 to make the first gcd check come earlier, as an
 *   optimization for small numbers.
 * - we vary f between restarts because the cycle finding algorithm does not
 *   find the first element in the cycle but rather one at distance k*|cycle|
 *   from the start, and that can result in continual failures if all cycles
 *   have the same size for all prime factors. E.g. fixing f(x) = x^2 + 1 would
 *   loop infinitely for n = 352523 * 352817, where all cycles have size 821.
 * - we operate on residues in [i, n + i) which modmul is not designed to
 *   handle, but specifically modmul(x, x) still turns out to work for small
 *   enough i. (With reference to the proof in modmul-proof.tex, the argument
 *   for "S is in [-c, 2c)" goes through unchanged, while S < 2^63 now follows
 *   from S < 2c and S = x^2 (mod c) together implying S < c + i^2.)
 */
#pragma once

#include "ModMulLL.h"
#include "MillerRabin.h"

ull pollard(ull n) {
	ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
	auto f = [&](ull x) { return modmul(x, x, n) + i; };
	while (t++ % 40 || __gcd(prd, n) == 1) {
		if (x == y) x = ++i, y = f(x);
		if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
		x = f(x), y = f(f(y));
	}
	return __gcd(prd, n);
}
vector<ull> factor(ull n) {
	if (n == 1) return {};
	if (isPrime(n)) return {n};
```

```
        ull x = pollard(n);
        auto l = factor(x), r = factor(n / x);
        l.insert(l.end(), all(r));
        return l;
}
```

10.6 FastEratosthenes

```
/**
 * Author: Jakob Kogler, chilli, pajenegod
 * Date: 2020-04-12
 * License: CC0
 * Description: Prime sieve for generating all primes smaller than LIM.
 * Time: LIM=1e9 $\approx$ 1.5s
 * Status: Stress-tested
 * Details: Despite its n log log n complexity, segmented sieve is still faster
 * than other options, including bitset sieves and linear sieves. This is
 * primarily due to its low memory usage, which reduces cache misses. This
 * implementation skips even numbers.
 *
 * Benchmark can be found here: https://ideone.com/e7TbX4
 *
 * The line `for (int i=idx; i<S+L; idx = (i += p))` is done on purpose for
     performance reasons.
 * Se https://github.com/kth-competitive-programming/kactl/pull/166#
     discussion_r408354338
 */
#pragma once

const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
        const int S = (int)round(sqrt(LIM)), R = LIM / 2;
        vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1));
        vector<pii> cp;
        for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
                cp.push_back({i, i * i / 2});
                for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
        }
        for (int L = 1; L <= R; L += S) {
                array<bool, S> block{};
                for (auto &[p, idx] : cp)
                        for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
                rep(i,0,min(S, R - L))
                        if (!block[i]) pr.push_back((L + i) * 2 + 1);
        }
        for (int i : pr) isPrime[i] = 1;
        return pr;
}
```

10.7 FracBinarySearch

```
/**
 * Author: Lucian Bicsi, Simon Lindholm
 * Date: 2017-10-31
 * License: CC0
 * Description: Given $f$ and $N$, finds the smallest fraction $p/q \in [0, 1]$
 * such that $f(p/q)$ is true, and $p, q \le N$.
 * You may want to throw an exception from $f$ if it finds an exact solution,
 * in which case $N$ can be removed.
 * Usage: fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}
 * Time: O(\log(N))
 * Status: stress-tested for n <= 300
 */

struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
        bool dir = 1, A = 1, B = 1;
        Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
        if (f(lo)) return lo;
        assert(f(hi));
        while (A || B) {
                ll adv = 0, step = 1; // move hi if dir, else lo
                for (int si = 0; step; (step *= 2) >>= si) {
                        adv += step;
                        Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
                        if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                                adv -= step; si = 2;
                        }
                }
                hi.p += lo.p * adv;
                hi.q += lo.q * adv;
                dir = !dir;
                swap(lo, hi);
                A = B; B = !!adv;
        }
        return dir ? hi : lo;
}
```

10.8 MillerRabin

```
/**
 * Author: chilli, c1729, Simon Lindholm
 * Date: 2019-03-28
 * License: CC0
 * Source: Wikipedia, https://miller-rabin.appspot.com/
 * Description: Deterministic Miller-Rabin primality test.
 * Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers,
     use Python and extend A randomly.
 * Time: 7 times the complexity of $a^b \mod c$.
 * Status: Stress-tested
 */
#pragma once

#include "ModMulLL.h"

bool isPrime(ull n) {
        if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
        ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
            s = __builtin_ctzll(n-1), d = n >> s;
        for (ull a : A) {   // ^ count trailing zeroes
                ull p = modpow(a%n, d, n), i = s;
                while (p != 1 && p != n - 1 && a % n && i--)
                        p = modmul(p, p, n);
                if (p != n-1 && i != s) return 0;
        }
        return 1;
}
```

10.9 ModInverse

```
/**
 * Author: Simon Lindholm
 * Date: 2016-07-24
 * License: CC0
 * Source: Russian page
 * Description: Pre-computation of modular inverses. Assumes LIM $\le$ mod and
     that mod is a prime.
 * Status: Works
 */
#pragma once

// const ll mod = 1000000007, LIM = 200000; ///include-line
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

10.10 ModLog

```
/**
 * Author: Bjorn Martinsson
 * Date: 2020-06-03
 * License: CC0
 * Source: own work
 * Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or
 * $-1$ if no such $x$ exists. modLog(a,1,m) can be used to
 * calculate the order of $a$.
 * Time: $O(\sqrt m)$
 * Status: tested for all 0 <= a,x < 500 and 0 < m < 500.
 *
 * Details: This algorithm uses the baby-step giant-step method to
 * find (i,j) such that a^(n * i) = b * a^j (mod m), where n > sqrt(m)
 * and 0 < i, j <= n. If a and m are coprime then a^j has a modular
 * inverse, which means that a^(i * n - j) = b (mod m$).
 *
 * However this particular implementation of baby-step giant-step works even
 * without assuming a and m are coprime, using the following idea:
 *
 * Assume p^x is a prime divisor of m. Then we have 3 cases
 *       1. b is divisible by p^x
 *       2. b is divisible only by some p^y, 0<y<x
 *       3. b is not divisible by p
 * The important thing to note is that in case 2, modLog(a,b,m) (if
 * it exists) cannot be > sqrt(m), (technically it cannot be >= log2(m)).
 * So once all exponenents of a that are <= sqrt(m) has been checked, you
 * cannot have case 2. Case 2 is the only tricky case.
 *
 * So the modification allowing for non-coprime input involves checking all
 * exponents of a that are <= n, and then handling the non-tricky cases by
 * a simple gcd(a^n,m) == gcd(b,m) check.
 */
#pragma once

ll modLog(ll a, ll b, ll m) {
        ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
        unordered_map<ll, ll> A;
        while (j <= n && (e = f = e * a % m) != b % m)
                A[e * b % m] = j++;
        if (e == b % m) return j;
        if (__gcd(m, e) == __gcd(m, b))
                rep(i,2,n+2) if (A.count(e = e * f % m))
                        return n * i - A[e];
        return -1;
}
```

10.11 ModMulLL

```
/**
 * Author: chilli, Ramchandra Apte, Noam527, Simon Lindholm
 * Date: 2019-04-24
 * License: CC0
 * Source: https://github.com/RamchandraApte/OmniTemplate/blob/master/src/
     number_theory/modulo.hpp
 * Description: Calculate $a\cdot b\bmod c$ (or $a^b \bmod c$) for $0 \le a, b \
     le c \le 7.2\cdot 10^{18}$.
 * Time: O(1) for \texttt{modmul}, O(\log b) for \texttt{modpow}
 * Status: stress-tested, proven correct
 * Details:
 * This runs ~2x faster than the naive (__int128_t)a * b % M.
 * A proof of correctness is in doc/modmul-proof.tex. An earlier version of the
     proof,
 * from when the code used a * b / (long double)M, is in doc/modmul-proof.md.
 * The proof assumes that long doubles are implemented as x87 80-bit floats; if
     they
 * are 64-bit, as on e.g. MSVC, the implementation is only valid for
 * $0 \le a, b \le c < 2^{52} \approx 4.5 \cdot 10^{15}$.
 */
#pragma once

typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
        ll ret = a * b - M * ull(1.L / M * a * b);
        return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
        ull ans = 1;
        for (; e; b = modmul(b, b, mod), e /= 2)
                if (e & 1) ans = modmul(ans, b, mod);
        return ans;
}
```

10.12 ModPow

```
/**
 * Author: Noam527
 * Date: 2019-04-24
 * License: CC0
 * Source: folklore
 * Description:
 * Status: tested
 */
#pragma once

const ll mod = 1000000007; // faster if const

ll modpow(ll b, ll e) {
        ll ans = 1;
        for (; e; b = b * b % mod, e /= 2)
                if (e & 1) ans = ans * b % mod;
        return ans;
}
```

10.13 ModSqrt

```
/**
 * Author: Simon Lindholm
 * Date: 2016-08-31
 * License: CC0
 * Source: http://eli.thegreenplace.net/2009/03/07/computing-modular-square-
     roots-in-python/
 * Description: Tonelli-Shanks algorithm for modular square roots. Finds $x$ s.t
     . $x^2 = a \pmod p$ ($-x$ gives the other solution).
 * Time: O(\log^2 p) worst case, O(\log p) for most $p$
 * Status: Tested for all a,p <= 10000
 */
#pragma once

#include "ModPow.h"

ll sqrt(ll a, ll p) {
        a %= p; if (a < 0) a += p;
        if (a == 0) return 0;
        assert(modpow(a, (p-1)/2, p) == 1); // else no solution
        if (p % 4 == 3) return modpow(a, (p+1)/4, p);
        // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
        ll s = p - 1, n = 2;
        int r = 0, m;
        while (s % 2 == 0)
                ++r, s /= 2;
        /// find a non-square mod p
        while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
        ll x = modpow(a, (s + 1) / 2, p);
        ll b = modpow(a, s, p), g = modpow(n, s, p);
        for (;; r = m) {
                ll t = b;
                for (m = 0; m < r && t != 1; ++m)
                        t = t * t % p;
                if (m == 0) return x;
                ll gs = modpow(g, 1LL << (r - m - 1), p);
                g = gs * gs % p;
                x = x * gs % p;
                b = b * g % p;
        }
}
```

10.14 ModSum

```
/**
 * Author: Simon Lindholm
 * Date: 2015-06-23
 * License: CC0
 * Source: own work
 * Description: Sums of mod'ed arithmetic progressions.
 *
 * \texttt{modsum(to, c, k, m)} = $\sum_{i=0}^{\mathrm{to}-1}{(ki+c) \% m}$.
 * \texttt{divsum} is similar but for floored division.
 * Time: $\log(m)$, with a large constant.
 * Status: Tested for all |k|,|c|,to,m <= 50, and on kattis:aladin
 */
#pragma once

typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
/// ^ written in a weird way to deal with overflows correctly

ull divsum(ull to, ull c, ull k, ull m) {
	ull res = k / m * sumsq(to) + c / m * to;
	k %= m; c %= m;
	if (!k) return res;
	ull to2 = (to * k + c) / m;
	return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
	c = ((c % m) + m) % m;
	k = ((k % m) + m) % m;
	return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

10.15 ModularArithmetic

```
/**
 * Author: Lukas Polacek
 * Date: 2009-09-28
 * License: CC0
 * Source: folklore
 * Description: Operators for modular arithmetic. You need to set {\tt mod} to
 * some number first and then you can use the structure.
 */
#pragma once

#include "euclid.h"

const ll mod = 17; // change to something else
struct Mod {
	ll x;
	Mod(ll xx) : x(xx) {}
	Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
	Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
	Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
	Mod operator/(Mod b) { return *this * invert(b); }
	Mod invert(Mod a) {
		ll x, y, g = euclid(a.x, mod, x, y);
		assert(g == 1); return Mod((x + mod) % mod);
	}
	Mod operator^(ll e) {
		if (!e) return Mod(1);
		Mod r = *this ^ (e / 2); r = r * r;
		return e&1 ? *this * r : r;
	}
};
```

10.16 phiFunction

```
/**
 * Author: Hakan Terelius
 * Date: 2009-09-25
 * License: CC0
 * Source: http://en.wikipedia.org/wiki/Euler's_totient_function
 * Description: \emph{Euler's $\phi$} function is defined as $\phi(n):=\#$ of
     positive integers $\leq n$ that are coprime with $n$.
 * $\phi(1)=1$, $p$ prime $\Rightarrow \phi(p^k)=(p-1)p^{k-1}$, $m,n$ coprime $\
     Rightarrow \phi(mn)=\phi(m)\phi(n)$.
 * If $n=p_1^{k_1}p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1-1)p_1^{k_1
     -1}...(p_r-1)p_r^{k_r-1}$.
 * $\phi(n)=n \cdot \prod_{p|n}(1-1/p)$.
 *
 * $\sum_{d|n} \phi(d) = n$, $\sum_{1\leq k \leq n, \gcd(k,n)=1} k = n \phi(n)
     /2, n>1$
 *
 * \textbf{Euler's thm}: $a,n$ coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n
     }$.
 *
 * \textbf{Fermat's little thm}: $p$ prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p
     }$ $\forall a$.
 * Status: Tested
 */
#pragma once

const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
	rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
	for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
		for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

11 kactl forked/numerical

11.1 BerlekampMassey

```
/**
 * Author: Lucian Bicsi
 * Date: 2017-10-31
 * License: CC0
 * Source: Wikipedia
 * Description: Recovers any $n$-order linear recurrence relation from the first
 * $2n$ terms of the recurrence.
 * Useful for guessing linear recurrences after brute-forcing the first terms.
 * Should work on any field, but numerical stability for floats is not
     guaranteed.
 * Output will have size $\le n$.
 * Usage: berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}
 * Time: O(N^2)
 * Status: bruteforce-tested mod 5 for n <= 5 and all s
 */
#pragma once

#include "../number-theory/ModPow.h"

vector<ll> berlekampMassey(vector<ll> s) {
	int n = sz(s), L = 0, m = 0;
	vector<ll> C(n), B(n), T;
	C[0] = B[0] = 1;

	ll b = 1;
	rep(i,0,n) { ++m;
		ll d = s[i] % mod;
		rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
		if (!d) continue;
		T = C; ll coef = d * modpow(b, mod-2) % mod;
		rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
		if (2 * L > i) continue;
		L = i + 1 - L; B = T; b = d; m = 0;
	}

	C.resize(L + 1); C.erase(C.begin());
	for (ll& x : C) x = (mod - x) % mod;
```

```
        return C;
}
```

11.2 Determinant

```
/**
 * Author: Simon Lindholm
 * Date: 2016-09-06
 * License: CC0
 * Source: folklore
 * Description: Calculates determinant of a matrix. Destroys the matrix.
 * Time: $O(N^3)$
 * Status: somewhat tested
 */
#pragma once

double det(vector<vector<double>>& a) {
        int n = sz(a); double res = 1;
        rep(i,0,n) {
                int b = i;
                rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
                if (i != b) swap(a[i], a[b]), res *= -1;
                res *= a[i][i];
                if (res == 0) return 0;
                rep(j,i+1,n) {
                        double v = a[j][i] / a[i][i];
                        if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
                }
        }
        return res;
}
```

11.3 FastFourierTransform

```
/**
 * Author: Ludo Pulles, chilli, Simon Lindholm
 * Date: 2019-01-09
 * License: CC0
 * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf (do read, it's
     excellent)
   Accuracy bound from http://www.daemonology.net/papers/fft.pdf
 * Description: fft(a) computes $\hat f(k) = \sum_x a[x] \exp(2\pi i \cdot k x /
     N)$ for all $k$. N must be a power of 2.
   Useful for convolution:
   \texttt{conv(a, b) = c}, where $c[x] = \sum a[i]b[x-i]$.
   For convolution of complex numbers or more than two vectors: FFT, multiply
   pointwise, divide by n, reverse(start+1, end), FFT back.
   Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2{N} < 9\cdot10^{14}$
   (in practice $10^{16}$; higher for random inputs).
   Otherwise, use NTT/FFTMod.
 * Time: O(N \log N) with $N = |A|+|B|$ ($\tilde 1s$ for $N=2^{22}$)
 * Status: somewhat tested
 * Details: An in-depth examination of precision for both FFT and FFTMod can be
     found
 * here (https://github.com/simonlindholm/fft-precision/blob/master/fft-
     precision.md)
 */
#pragma once

typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
        int n = sz(a), L = 31 - __builtin_clz(n);
        static vector<complex<long double>> R(2, 1);
        static vector<C> rt(2, 1);  // (^ 10% faster if double)
        for (static int k = 2; k < n; k *= 2) {
                R.resize(n); rt.resize(n);
                auto x = polar(1.0L, acos(-1.0L) / k);
                rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
        }
        vi rev(n);
        rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
        rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int k = 1; k < n; k *= 2)
                for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
                        // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
                            rolled)  /// include-line
                        auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];
                                /// exclude-line
                        C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
                                /// exclude-line
                        a[i + j + k] = a[i + j] - z;
                        a[i + j] += z;
                }
}
vd conv(const vd& a, const vd& b) {
        if (a.empty() || b.empty()) return {};
        vd res(sz(a) + sz(b) - 1);
        int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
        vector<C> in(n), out(n);
        copy(all(a), begin(in));
        rep(i,0,sz(b)) in[i].imag(b[i]);
        fft(in);
        for (C& x : in) x *= x;
        rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
        fft(out);
        rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
        return res;
}
```

11.4 FastFourierTransformMod

```
/**
 * Author: chilli
 * Date: 2019-04-25
 * License: CC0
 * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf
 * Description: Higher precision FFT, can be used for convolutions modulo
     arbitrary integers
 * as long as $N\log_2N\cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice $10
     ^{16}$ or higher).
 * Inputs must be in $[0, \text{mod})$.
 * Time: O(N \log N), where $N = |A|+|B|$ (twice as slow as NTT or FFT)
 * Status: stress-tested
 * Details: An in-depth examination of precision for both FFT and FFTMod can be
     found
 * here (https://github.com/simonlindholm/fft-precision/blob/master/fft-
     precision.md)
 */
#pragma once

#include "FastFourierTransform.h"

typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
        if (a.empty() || b.empty()) return {};
        vl res(sz(a) + sz(b) - 1);
        int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
        vector<C> L(n), R(n), outs(n), outl(n);
        rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
        rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
        fft(L), fft(R);
        rep(i,0,n) {
                int j = -i & (n - 1);
                outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
                outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
        }
        fft(outl), fft(outs);
        rep(i,0,sz(res)) {
                ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
                ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
                res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
        }
        return res;
}
```

11.5 FastSubsetTransform

```
/**
 * Author: Lucian Bicsi
 * Date: 2015-06-25
 * License: GNU Free Documentation License 1.2
 * Source: csacademy
 * Description: Transform to a basis with fast convolutions of the form
 * $\displaystyle c[z] = \sum\nolimits_{z = x \oplus y} a[x] \cdot b[y]$,
 * where $\oplus$ is one of AND, OR, XOR. The size of $a$ must be a power of two
     .
 * Time: O(N \log N)
 * Status: stress-tested
 */
#pragma once

void FST(vi& a, bool inv) {
        for (int n = sz(a), step = 1; step < n; step *= 2) {
                for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
                        int &u = a[j], &v = a[j + step]; tie(u, v) =
                                inv ? pii(v - u, u) : pii(v, u + v); // AND
                                // inv ? pii(v, u - v) : pii(u + v, u); // OR
                                     /// include-line
                                // pii(u + v, u - v);                   // XOR
                                     /// include-line
                }
        }
        // if (inv) for (int& x : a) x /= sz(a); // XOR only /// include-line
}
vi conv(vi a, vi b) {
        FST(a, 0); FST(b, 0);
        rep(i,0,sz(a)) a[i] *= b[i];
        FST(a, 1); return a;
}
```

11.6 GoldenSectionSearch

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-04-17
 * License: CC0
 * Source: Numeriska algoritmer med matlab, Gerd Eriksson, NADA, KTH
 * Description: Finds the argument minimizing the function $f$ in the interval $
     [a,b]$
 * assuming $f$ is unimodal on the interval, i.e. has only one local minimum and
      no local
 * maximum. The maximum error in the result is $eps$. Works equally well for
     maximization
 * with a small change in the code. See TernarySearch.h in the Various chapter
     for a
 * discrete version.
 * Usage:
        double func(double x) { return 4+x+.3*x*x; }
        double xmin = gss(-1000,1000,func);
 * Time: O(\log((b-a) / \epsilon))
 * Status: tested
 */
#pragma once

/// It is important for r to be precise, otherwise we don't necessarily maintain
      the inequality a < x1 < x2 < b.
double gss(double a, double b, double (*f)(double)) {
        double r = (sqrt(5)-1)/2, eps = 1e-7;
        double x1 = b - r*(b-a), x2 = a + r*(b-a);
        double f1 = f(x1), f2 = f(x2);
        while (b-a > eps)
                if (f1 < f2) { //change to > to find maximum
                        b = x2; x2 = x1; f2 = f1;
                        x1 = b - r*(b-a); f1 = f(x1);
                } else {
                        a = x1; x1 = x2; f1 = f2;
                        x2 = a + r*(b-a); f2 = f(x2);
                }
        return a;
}
```

11.7 HillClimbing

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-04
 * License: CC0
 * Source: Johan Sannemo
 * Description: Poor man's optimization for unimodal functions.
 * Status: used with great success
 */
#pragma once

typedef array<double, 2> P;

template<class F> pair<double, P> hillClimb(P start, F f) {
        pair<double, P> cur(f(start), start);
        for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
                rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
                        P p = cur.second;
                        p[0] += dx*jmp;
                        p[1] += dy*jmp;
                        cur = min(cur, make_pair(f(p), p));
                }
        }
        return cur;
}
```

11.8 IntDeterminant

```
/**
 * Author: Unknown
 * Date: 2014-11-27
 * Source: somewhere on github
 * Description: Calculates determinant using modular arithmetics.
 * Modulos can also be removed to get a pure-integer version.
 * Time: $O(N^3)$
 * Status: bruteforce-tested for N <= 3, mod <= 7
 */
#pragma once

const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
        int n = sz(a); ll ans = 1;
        rep(i,0,n) {
                rep(j,i+1,n) {
                        while (a[j][i] != 0) { // gcd step
                                ll t = a[i][i] / a[j][i];
                                if (t) rep(k,i,n)
                                        a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                                swap(a[i], a[j]);
                                ans *= -1;
                        }
                }
                ans = ans * a[i][i] % mod;
                if (!ans) return 0;
        }
        return (ans + mod) % mod;
}
```

11.9 Integrate

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-11
 * License: CC0
 * Source: Wikipedia
 * Description: Simple integration of a function over an interval using
 *  Simpson's rule. The error should be proportional to $h^4$, although in
 *  practice you will want to verify that the result is stable to desired
 *  precision when epsilon changes.
```

```
 * Status: mostly untested
 */
#pragma once

template<class F>
double quad(double a, double b, F f, const int n = 1000) {
	double h = (b - a) / 2 / n, v = f(a) + f(b);
	rep(i,1,n*2)
		v += f(a + i*h) * (i&1 ? 4 : 2);
	return v * h / 3;
}
```

11.10 IntegrateAdaptive

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-11
 * License: CC0
 * Source: Wikipedia
 * Description: Fast integration using an adaptive Simpson's rule.
 * Usage:
	double sphereVolume = quad(-1, 1, [](double x) {
	return quad(-1, 1, [\&](double y) {
	return quad(-1, 1, [\&](double z) {
	return x*x + y*y + z*z < 1; });});});
 * Status: mostly untested
 */
#pragma once

typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
	d c = (a + b) / 2;
	d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
	if (abs(T - S) <= 15 * eps || b - a < 1e-10)
		return T + (T - S) / 15;
	return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
	return rec(f, a, b, eps, S(a, b));
}
```

11.11 LinearRecurrence

```
/**
 * Author: Lucian Bicsi
 * Date: 2018-02-14
 * License: CC0
 * Source: Chinese material
 * Description: Generates the $k$'th term of an $n$-order
 * linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$,
 * given $S[0 \ldots \ge n-1]$ and $tr[0 \ldots n-1]$.
 * Faster than matrix multiplication.
 * Useful together with Berlekamp--Massey.
 * Usage: linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number
 * Time: O(n^2 \log k)
 * Status: bruteforce-tested mod 5 for n <= 5
 */
#pragma once

const ll mod = 5; /** exclude-line */

typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
	int n = sz(tr);

	auto combine = [&](Poly a, Poly b) {
		Poly res(n * 2 + 1);
		rep(i,0,n+1) rep(j,0,n+1)
			res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
		for (int i = 2 * n; i > n; --i) rep(j,0,n)
			res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod
				;
		res.resize(n + 1);
		return res;
	};

	Poly pol(n + 1), e(pol);
	pol[0] = e[1] = 1;

	for (++k; k; k /= 2) {
		if (k % 2) pol = combine(pol, e);
		e = combine(e, e);
	}

	ll res = 0;
	rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
	return res;
}
```

11.12 MatrixInverse-mod

```
/**
 * Author: Simon Lindholm
 * Date: 2016-12-08
 * Source: The regular matrix inverse code
 * Description: Invert matrix $A$ modulo a prime.
 * Returns rank; result is stored in $A$ unless singular (rank < n).
 * For prime powers, repeatedly set $A^{-1} = A^{-1} (2I - AA^{-1})\  (\text{mod }p^k)$ where $A^{-1}$ starts as
 * the inverse of A mod p, and k is doubled in each step.
 * Time: O(n^3)
 * Status: Slightly tested
 */
#pragma once

#include "../number-theory/ModPow.h"

int matInv(vector<vector<ll>>& A) {
	int n = sz(A); vi col(n);
	vector<vector<ll>> tmp(n, vector<ll>(n));
	rep(i,0,n) tmp[i][i] = 1, col[i] = i;

	rep(i,0,n) {
		int r = i, c = i;
		rep(j,i,n) rep(k,i,n) if (A[j][k]) {
			r = j; c = k; goto found;
		}
		return i;
found:
		A[i].swap(A[r]); tmp[i].swap(tmp[r]);
		rep(j,0,n)
			swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
		swap(col[i], col[c]);
		ll v = modpow(A[i][i], mod - 2);
		rep(j,i+1,n) {
			ll f = A[j][i] * v % mod;
			A[j][i] = 0;
			rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
			rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
		}
		rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
		rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
		A[i][i] = 1;
	}

	for (int i = n-1; i > 0; --i) rep(j,0,i) {
		ll v = A[j][i];
		rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod;
	}

	rep(i,0,n) rep(j,0,n)
		A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0)*mod;
	return n;
}
```

11.13 MatrixInverse

```
/**
 * Author: Max Bennedich
 * Date: 2004-02-08
 * Description: Invert matrix $A$. Returns rank; result is stored in $A$ unless
     singular (rank < n).
 * Can easily be extended to prime moduli; for prime powers, repeatedly
 * set $A^{-1} = A^{-1} (2I - AA^{-1})\  (\text{mod }p^k)$ where $A^{-1}$ starts
      as
 * the inverse of A mod p, and k is doubled in each step.
 * Time: O(n^3)
 * Status: Slightly tested
 */
#pragma once

int matInv(vector<vector<double>>& A) {
        int n = sz(A); vi col(n);
        vector<vector<double>> tmp(n, vector<double>(n));
        rep(i,0,n) tmp[i][i] = 1, col[i] = i;

        rep(i,0,n) {
                int r = i, c = i;
                rep(j,i,n) rep(k,i,n)
                        if (fabs(A[j][k]) > fabs(A[r][c]))
                                r = j, c = k;
                if (fabs(A[r][c]) < 1e-12) return i;
                A[i].swap(A[r]); tmp[i].swap(tmp[r]);
                rep(j,0,n)
                        swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
                swap(col[i], col[c]);
                double v = A[i][i];
                rep(j,i+1,n) {
                        double f = A[j][i] / v;
                        A[j][i] = 0;
                        rep(k,i+1,n) A[j][k] -= f*A[i][k];
                        rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
                }
                rep(j,i+1,n) A[i][j] /= v;
                rep(j,0,n) tmp[i][j] /= v;
                A[i][i] = 1;
        }

        /// forget A at this point, just eliminate tmp backward
        for (int i = n-1; i > 0; --i) rep(j,0,i) {
                double v = A[j][i];
                rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
        }

        rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
        return n;
}
```

11.14 NumberTheoreticTransform

```
/**
 * Author: chilli
 * Date: 2019-04-16
 * License: CC0
 * Source: based on KACTL's FFT
 * Description: ntt(a) computes $\hat f(k) = \sum_x a[x] g^{xk}$ for all $k$,
     where $g=\text{root}^{(mod-1)/N}$.
 * N must be a power of 2.
 * Useful for convolution modulo specific nice primes of the form $2^a b+1$,
 * where the convolution result has size at most $2^a$. For arbitrary modulo,
     see FFTMod.
   \texttt{conv(a, b) = c}, where $c[x] = \sum a[i]b[x-i]$.
   For manual convolution: NTT the inputs, multiply
   pointwise, divide by n, reverse(start+1, end), NTT back.
 * Inputs must be in [0, mod).
 * Time: O(N \log N)
 * Status: stress-tested
 */
#pragma once

#include "../number-theory/ModPow.h"

const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
        int n = sz(a), L = 31 - __builtin_clz(n);
        static vl rt(2, 1);
        for (static int k = 2, s = 2; k < n; k *= 2, s++) {
                rt.resize(n);
                ll z[] = {1, modpow(root, mod >> s)};
                rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
        }
        vi rev(n);
        rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
        rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int k = 1; k < n; k *= 2)
                for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
                        ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
                        a[i + j + k] = ai - z + (z > ai ? mod : 0);
                        ai += (ai + z >= mod ? z - mod : z);
                }
}
vl conv(const vl &a, const vl &b) {
        if (a.empty() || b.empty()) return {};
        int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
            n = 1 << B;
        int inv = modpow(n, mod - 2);
        vl L(a), R(b), out(n);
        L.resize(n), R.resize(n);
        ntt(L), ntt(R);
        rep(i,0,n)
                out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
        ntt(out);
        return {out.begin(), out.begin() + s};
}
```

11.15 PolyInterpolate

```
/**
 * Author: Simon Lindholm
 * Date: 2017-05-10
 * License: CC0
 * Source: Wikipedia
 * Description: Given $n$ points (x[i], y[i]), computes an n-1-degree polynomial
      $p$ that
 *  passes through them: $p(x) = a[0]*x^0 + ... + a[n-1]*x^{n-1}$.
 *  For numerical precision, pick $x[k] = c*\cos(k/(n-1)*\pi), k=0 \dots n-1$.
 * Time: O(n^2)
 */
#pragma once

typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
        vd res(n), temp(n);
        rep(k,0,n-1) rep(i,k+1,n)
                y[i] = (y[i] - y[k]) / (x[i] - x[k]);
        double last = 0; temp[0] = 1;
        rep(k,0,n) rep(i,0,n) {
                res[i] += y[k] * temp[i];
                swap(last, temp[i]);
                temp[i] -= last * x[k];
        }
        return res;
}
```

11.16 Polynomial

```
/**
 * Author: David Rydh, Per Austrin
 * Date: 2003-03-16
 * Description:
 */
#pragma once

struct Poly {
	vector<double> a;
	double operator()(double x) const {
		double val = 0;
		for (int i = sz(a); i--;) (val *= x) += a[i];
		return val;
	}
	void diff() {
		rep(i,1,sz(a)) a[i-1] = i*a[i];
		a.pop_back();
	}
	void divroot(double x0) {
		double b = a.back(), c; a.back() = 0;
		for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
		a.pop_back();
	}
};
```

11.17 PolyRoots

```
/**
 * Author: Per Austrin
 * Date: 2004-02-08
 * License: CC0
 * Description: Finds the real roots to a polynomial.
 * Usage: polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0
 * Time: O(n^2 \log(1/\epsilon))
 */
#pragma once

#include "Polynomial.h"

vector<double> polyRoots(Poly p, double xmin, double xmax) {
	if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
	vector<double> ret;
	Poly der = p;
	der.diff();
	auto dr = polyRoots(der, xmin, xmax);
	dr.push_back(xmin-1);
	dr.push_back(xmax+1);
	sort(all(dr));
	rep(i,0,sz(dr)-1) {
		double l = dr[i], h = dr[i+1];
		bool sign = p(l) > 0;
		if (sign ^ (p(h) > 0)) {
			rep(it,0,60) { // while (h - l > 1e-8)
				double m = (l + h) / 2, f = p(m);
				if ((f <= 0) ^ sign) l = m;
				else h = m;
			}
			ret.push_back((l + h) / 2);
		}
	}
	return ret;
}
```

11.18 Simplex

```
/**
 * Author: Stanford
 * Source: Stanford Notebook
 * License: MIT
 * Description: Solves a general linear maximization problem: maximize $c^T x$
     subject to $Ax \le b$, $x \ge 0$.
 * Returns -inf if there is no solution, inf if there are arbitrarily good
     solutions, or the maximum value of $c^T x$ otherwise.
 * The input vector is set to an optimal $x$ (or in the unbounded case, an
     arbitrary solution fulfilling the constraints).
 * Numerical stability is not guaranteed. For better performance, define
     variables such that $x = 0$ is viable.
 * Usage:
 * vvd A = {{1,-1}, {-1,1}, {-1,-2}};
 * vd b = {1,1,-4}, c = {-1,-1}, x;
 * T val = LPSolver(A, b, c).solve(x);
 * Time: O(NM * \#pivots), where a pivot may be e.g. an edge relaxation. O(2^n)
     in the general case.
 * Status: seems to work?
 */
#pragma once

typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
	int m, n;
	vi N, B;
	vvd D;

	LPSolver(const vvd& A, const vd& b, const vd& c) :
		m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
			rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
			rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i
				];}
			rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
			N[n] = -1; D[m+1][n] = 1;
		}

	void pivot(int r, int s) {
		T *a = D[r].data(), inv = 1 / a[s];
		rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
			T *b = D[i].data(), inv2 = b[s] * inv;
			rep(j,0,n+2) b[j] -= a[j] * inv2;
			b[s] = a[s] * inv2;
		}
		rep(j,0,n+2) if (j != s) D[r][j] *= inv;
		rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
		D[r][s] = inv;
		swap(B[r], N[s]);
	}

	bool simplex(int phase) {
		int x = m + phase - 1;
		for (;;) {
			int s = -1;
			rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
			if (D[x][s] >= -eps) return true;
			int r = -1;
			rep(i,0,m) {
				if (D[i][s] <= eps) continue;
				if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
					< MP(D[r][n+1] / D[r][s], B[r])) r
					= i;
			}
			if (r == -1) return false;
			pivot(r, s);
		}
	}

	T solve(vd &x) {
		int r = 0;
		rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
		if (D[r][n+1] < -eps) {
			pivot(r, n);
			if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
			rep(i,0,m) if (B[i] == -1) {
				int s = 0;
				rep(j,1,n+1) ltj(D[i]);
```

```
                        pivot(i, s);
                }
        }
        bool ok = simplex(1); x = vd(n);
        rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
        return ok ? D[m][n+1] : inf;
    }
};
```

11.19 SolveLinear

```
/**
 * Author: Per Austrin, Simon Lindholm
 * Date: 2004-02-08
 * License: CC0
 * Description: Solves $A * x = b$. If there are multiple solutions, an
     arbitrary one is returned.
 *  Returns rank, or -1 if no solutions. Data in $A$ and $b$ is lost.
 * Time: O(n^2 m)
 * Status: tested on kattis:equationsolver, and bruteforce-tested mod 3 and 5
     for n,m <= 3
 */
#pragma once

typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
        int n = sz(A), m = sz(x), rank = 0, br, bc;
        if (n) assert(sz(A[0]) == m);
        vi col(m); iota(all(col), 0);

        rep(i,0,n) {
                double v, bv = 0;
                rep(r,i,n) rep(c,i,m)
                        if ((v = fabs(A[r][c])) > bv)
                                br = r, bc = c, bv = v;
                if (bv <= eps) {
                        rep(j,i,n) if (fabs(b[j]) > eps) return -1;
                        break;
                }
                swap(A[i], A[br]);
                swap(b[i], b[br]);
                swap(col[i], col[bc]);
                rep(j,0,n) swap(A[j][i], A[j][bc]);
                bv = 1/A[i][i];
                rep(j,i+1,n) {
                        double fac = A[j][i] * bv;
                        b[j] -= fac * b[i];
                        rep(k,i+1,m) A[j][k] -= fac*A[i][k];
                }
                rank++;
        }

        x.assign(m, 0);
        for (int i = rank; i--;) {
                b[i] /= A[i][i];
                x[col[i]] = b[i];
                rep(j,0,i) b[j] -= A[j][i] * b[i];
        }
        return rank; // (multiple solutions if rank < m)
}
```

11.20 SolveLinear2

```
/**
 * Author: Simon Lindholm
 * Date: 2016-09-06
 * License: CC0
 * Source: me
 * Description: To get all uniquely determined values of $x$ back from
     SolveLinear, make the following changes:
 * Status: tested on kattis:equationsolverplus, stress-tested
 */
#pragma once

#include "SolveLinear.h"

rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
        rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
        x[col[i]] = b[i] / A[i][i];
fail:; }
```

11.21 SolveLinearBinary

```
/**
 * Author: Simon Lindholm
 * Date: 2016-08-27
 * License: CC0
 * Source: own work
 * Description: Solves $Ax = b$ over $\mathbb F_2$. If there are multiple
     solutions, one is returned arbitrarily.
 *  Returns rank, or -1 if no solutions. Destroys $A$ and $b$.
 * Time: O(n^2 m)
 * Status: bruteforce-tested for n, m <= 4
 */
#pragma once

typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
        int n = sz(A), rank = 0, br;
        assert(m <= sz(x));
        vi col(m); iota(all(col), 0);
        rep(i,0,n) {
                for (br=i; br<n; ++br) if (A[br].any()) break;
                if (br == n) {
                        rep(j,i,n) if(b[j]) return -1;
                        break;
                }
                int bc = (int)A[br]._Find_next(i-1);
                swap(A[i], A[br]);
                swap(b[i], b[br]);
                swap(col[i], col[bc]);
                rep(j,0,n) if (A[j][i] != A[j][bc]) {
                        A[j].flip(i); A[j].flip(bc);
                }
                rep(j,i+1,n) if (A[j][i]) {
                        b[j] ^= b[i];
                        A[j] ^= A[i];
                }
                rank++;
        }

        x = bs();
        for (int i = rank; i--;) {
                if (!b[i]) continue;
                x[col[i]] = 1;
                rep(j,0,i) b[j] ^= A[j][i];
        }
        return rank; // (multiple solutions if rank < m)
}
```

11.22 Tridiagonal

```
/**
 * Author: Ulf Lundstrom, Simon Lindholm
 * Date: 2009-08-15
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm
 * Description: $x=\textrm{tridiagonal}(d,p,q,b)$ solves the equation system
```

```
\[
\left(\begin{array}{c}b_0\\b_1\\b_2\\b_3\\\vdots\\b_{n-1}\end{array}\right) =
\left(\begin{array}{cccccc}
d_0 & p_0 & 0 & 0 & \cdots & 0\\
q_0 & d_1 & p_1 & 0 & \cdots & 0\\
0 & q_1 & d_2 & p_2 & \cdots & 0\\
\vdots & \vdots & \ddots & \ddots & \ddots & \vdots\\
0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2}\\
0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1}\\
\end{array}\right)
\left(\begin{array}{c}x_0\\x_1\\x_2\\x_3\\\vdots\\x_{n-1}\end{array}\right).
\]

This is useful for solving problems on the type
\[ a_i=b_ia_{i-1}+c_ia_{i+1}+d_i,\,1\leq i\leq n, \]
where $a_0$, $a_{n+1}$, $b_i$, $c_i$ and $d_i$ are known. $a$ can then be
    obtained from
\begin{align*}
\{a_i\}=\textrm{tridiagonal}(&\{1,-1,-1,...,-1,1\}, \{0,c_1,c_2,\dots,c_n\},\\
&\{b_1,b_2,\dots,b_n,0\}, \{a_0,d_1,d_2,\dots,d_n,a_{n+1}\}).
\end{align*}
Fails if the solution is not unique.

If $|d_i| > |p_i| + |q_{i-1}|$ for all $i$, or $|d_i| > |p_{i-1}| + |q_i|$, or
    the matrix is positive definite,
the algorithm is numerically stable and neither \texttt{tr} nor the check for \
    texttt{diag[i] == 0} is needed.
 * Time: O(N)
 * Status: Brute-force tested mod 5 and 7 and stress-tested for real matrices
     obeying the criteria above.
 */
#pragma once

typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
                const vector<T>& sub, vector<T> b) {
        int n = sz(b); vi tr(n);
        rep(i,0,n-1) {
                if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
                        b[i+1] -= b[i] * diag[i+1] / super[i];
                        if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
                        diag[i+1] = sub[i]; tr[++i] = 1;
                } else {
                        diag[i+1] -= super[i]*sub[i]/diag[i];
                        b[i+1] -= b[i]*sub[i]/diag[i];
                }
        }
        for (int i = n; i--;) {
                if (tr[i]) {
                        swap(b[i], b[i-1]);
                        diag[i-1] = diag[i];
                        b[i] /= super[i-1];
                } else {
                        b[i] /= diag[i];
                        if (i) b[i-1] -= b[i]*super[i-1];
                }
        }
        return b;
}
```

12 kactl forked/strings

12.1 AhoCorasick

```
/**
 * Author: Simon Lindholm
 * Date: 2015-02-18
 * License: CC0
 * Source: marian's (TC) code
 * Description: Aho-Corasick automaton, used for multiple pattern matching.
 * Initialize with AhoCorasick ac(patterns); the automaton start node will be at
     index 0.
 * find(word) returns for each position the index of the longest word that ends
     there, or -1 if none.
 * findAll($-$, word) finds all words (up to $N \sqrt N$ many if no duplicate
     patterns)
 * that start at each position (shortest first).
 * Duplicate patterns are allowed; empty patterns are not.
 * To find the longest words that start at each position, reverse all input.
 * For large alphabets, split each symbol into chunks, with sentinel bits for
     symbol boundaries.
 * Time: construction takes $O(26N)$, where $N =$ sum of length of patterns.
 * find(x) is $O(N)$, where N = length of x. findAll is $O(NM)$.
 * Status: stress-tested
 */
#pragma once

struct AhoCorasick {
        enum {alpha = 26, first = 'A'}; // change this!
        struct Node {
                // (nmatches is optional)
                int back, next[alpha], start = -1, end = -1, nmatches = 0;
                Node(int v) { memset(next, v, sizeof(next)); }
        };
        vector<Node> N;
        vi backp;
        void insert(string& s, int j) {
                assert(!s.empty());
                int n = 0;
                for (char c : s) {
                        int& m = N[n].next[c - first];
                        if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
                        else n = m;
                }
                if (N[n].end == -1) N[n].start = j;
                backp.push_back(N[n].end);
                N[n].end = j;
                N[n].nmatches++;
        }
        AhoCorasick(vector<string>& pat) : N(1, -1) {
                rep(i,0,sz(pat)) insert(pat[i], i);
                N[0].back = sz(N);
                N.emplace_back(0);

                queue<int> q;
                for (q.push(0); !q.empty(); q.pop()) {
                        int n = q.front(), prev = N[n].back;
                        rep(i,0,alpha) {
                                int &ed = N[n].next[i], y = N[prev].next[i];
                                if (ed == -1) ed = y;
                                else {
                                        N[ed].back = y;
                                        (N[ed].end == -1 ? N[ed].end : backp[N[
                                             ed].start])
                                                = N[y].end;
                                        N[ed].nmatches += N[y].nmatches;
                                        q.push(ed);
                                }
                        }
                }
        }
        vi find(string word) {
                int n = 0;
                vi res; // ll count = 0;
                for (char c : word) {
                        n = N[n].next[c - first];
                        res.push_back(N[n].end);
                        // count += N[n].nmatches;
                }
                return res;
        }
        vector<vi> findAll(vector<string>& pat, string word) {
                vi r = find(word);
                vector<vi> res(sz(word));
                rep(i,0,sz(word)) {
                        int ind = r[i];
                        while (ind != -1) {
                                res[i - sz(pat[ind]) + 1].push_back(ind);
                                ind = backp[ind];
                        }
                }
                return res;
        }
```

```
};
```

12.2 Hashing-codeforces

```
/**
 * Author: Simon Lindholm
 * Date: 2015-03-15
 * License: CC0
 * Source: own work
 * Description: Various self-explanatory methods for string hashing.
 * Use on Codeforces, which lacks 64-bit support and where solutions can be
       hacked.
 * Status: stress-tested
 */
#pragma once

typedef uint64_t ull;
static int C; // initialized below

// Arithmetic mod two primes and 2^32 simultaneously.
// "typedef uint64_t H;" instead if Thue-Morse does not apply.
template<int M, class B>
struct A {
        int x; B b; A(int x=0) : x(x), b(x) {}
        A(int x, B b) : x(x), b(b) {}
        A operator+(A o){int y = x+o.x; return{y - (y>=M)*M, b+o.b};}
        A operator-(A o){int y = x-o.x; return{y + (y< 0)*M, b-o.b};}
        A operator*(A o) { return {(int)(1LL*x*o.x % M), b*o.b}; }
        explicit operator ull() { return x ^ (ull) b << 21; }
        bool operator==(A o) const { return (ull)*this == (ull)o; }
        bool operator<(A o) const { return (ull)*this < (ull)o; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;

struct HashInterval {
        vector<H> ha, pw;
        HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
                pw[0] = 1;
                rep(i,0,sz(str))
                        ha[i+1] = ha[i] * C + str[i],
                        pw[i+1] = pw[i] * C;
        }
        H hashInterval(int a, int b) { // hash [a, b)
                return ha[b] - ha[a] * pw[b - a];
        }
};

vector<H> getHashes(string& str, int length) {
        if (sz(str) < length) return {};
        H h = 0, pw = 1;
        rep(i,0,length)
                h = h * C + str[i], pw = pw * C;
        vector<H> ret = {h};
        rep(i,length,sz(str)) {
                ret.push_back(h = h * C + str[i] - pw * str[i-length]);
        }
        return ret;
}

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}

#include <sys/time.h>
int main() {
        timeval tp;
        gettimeofday(&tp, 0);
        C = (int)tp.tv_usec; // (less than modulo)
        assert((ull)(H(1)*2+1-3) == 0);
        // ...
}
```

12.3 Hashing

```
/**
 * Author: Simon Lindholm
 * Date: 2015-03-15
 * License: CC0
 * Source: own work
 * Description: Self-explanatory methods for string hashing.
 * Status: stress-tested
 */
#pragma once

// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
typedef uint64_t ull;
struct H {
        ull x; H(ull x=0) : x(x) {}
        H operator+(H o) { return x + o.x + (x + o.x < x); }
        H operator-(H o) { return *this + ~o.x; }
        H operator*(H o) { auto m = (__uint128_t)x * o.x;
                return H((ull)m) + (ull)(m >> 64); }
        ull get() const { return x + !~x; }
        bool operator==(H o) const { return get() == o.get(); }
        bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)

struct HashInterval {
        vector<H> ha, pw;
        HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
                pw[0] = 1;
                rep(i,0,sz(str))
                        ha[i+1] = ha[i] * C + str[i],
                        pw[i+1] = pw[i] * C;
        }
        H hashInterval(int a, int b) { // hash [a, b)
                return ha[b] - ha[a] * pw[b - a];
        }
};

vector<H> getHashes(string& str, int length) {
        if (sz(str) < length) return {};
        H h = 0, pw = 1;
        rep(i,0,length)
                h = h * C + str[i], pw = pw * C;
        vector<H> ret = {h};
        rep(i,length,sz(str)) {
                ret.push_back(h = h * C + str[i] - pw * str[i-length]);
        }
        return ret;
}

H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

12.4 KMP

```
/**
 * Author: Johan Sannemo
 * Date: 2016-12-15
 * License: CC0
 * Description: pi[x] computes the length of the longest prefix of s that ends
       at x,
 * other than s[0...x] itself (abacaba -> 0010123).
 * Can be used to find all occurrences of a string.
 * Time: O(n)
 * Status: Tested on kattis:stringmatching
 */
#pragma once

vi pi(const string& s) {
        vi p(sz(s));
        rep(i,1,sz(s)) {
                int g = p[i-1];
                while (g && s[i] != s[g]) g = p[g-1];
                p[i] = g + (s[i] == s[g]);
```

```
        }
        return p;
}

vi match(const string& s, const string& pat) {
        vi p = pi(pat + '\0' + s), res;
        rep(i,sz(p)-sz(s),sz(p))
                if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
        return res;
}
```

12.5 Manacher

```
/**
 * Author: User adamant on CodeForces
 * Source: http://codeforces.com/blog/entry/12143
 * Description: For each position in a string, computes p[0][i] = half length of
 *  longest even palindrome around pos i, p[1][i] = longest odd (half rounded
     down).
 * Time: O(N)
 * Status: Stress-tested
 */
#pragma once

array<vi, 2> manacher(const string& s) {
        int n = sz(s);
        array<vi,2> p = {vi(n+1), vi(n)};
        rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
                int t = r-i+!z;
                if (i<r) p[z][i] = min(t, p[z][l+t]);
                int L = i-p[z][i], R = i+p[z][i]-!z;
                while (L>=1 && R+1<n && s[L-1] == s[R+1])
                        p[z][i]++, L--, R++;
                if (R>r) l=L, r=R;
        }
        return p;
}
```

12.6 MinRotation

```
/**
 * Author: Stjepan Glavina
 * License: Unlicense
 * Source: https://github.com/stjepang/snippets/blob/master/min_rotation.cpp
 * Description: Finds the lexicographically smallest rotation of a string.
 * Time: O(N)
 * Usage:
 *  rotate(v.begin(), v.begin()+minRotation(v), v.end());
 * Status: Stress-tested
 */
#pragma once

int minRotation(string s) {
        int a=0, N=sz(s); s += s;
        rep(b,0,N) rep(k,0,N) {
                if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
                if (s[a+k] > s[b+k]) { a = b; break; }
        }
        return a;
}
```

12.7 SuffixArray

```
/**
 * Author: Luo Sui Qian , chilli
 * Date: 2019-04-11
 * License: Unknown
 * Source: Suffix array - a powerful tool for dealing with strings
 * (Chinese IOI National team training paper, 2009)
 * Description: Builds suffix array for a string.
 * \texttt{sa[i]} is the starting index of the suffix which
 * is $i$'th in the sorted suffix array.
 * The returned vector is of size $n+1$, and \texttt{sa[0] = n}.
 * The \texttt{lcp} array contains longest common prefixes for
 * neighbouring strings in the suffix array:
 * \texttt{lcp[i] = lcp(sa[i], sa[i-1])}, \texttt{lcp[0] = 0}.
 * The input string must not contain any nul chars.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once

struct SuffixArray {
        vi sa, lcp;
        SuffixArray(string s, int lim=256) { // or vector<int>
                s.push_back(0); int n = sz(s), k = 0, a, b;
                vi x(all(s)), y(n), ws(max(n, lim));
                sa = lcp = y, iota(all(sa), 0);
                for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
                        p = j, iota(all(y), n - j);
                        rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
                        fill(all(ws), 0);
                        rep(i,0,n) ws[x[i]]++;
                        rep(i,1,lim) ws[i] += ws[i - 1];
                        for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
                        swap(x, y), p = 1, x[sa[0]] = 0;
                        rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 :
                                        p++;
                }
                for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
                        for (k && k--, j = sa[x[i] - 1];
                                        s[i + k] == s[j + k]; k++);
        }
};
```

12.8 SuffixTree

```
/**
 * Author: Unknown
 * Date: 2017-05-15
 * Source: https://e-maxx.ru/algo/ukkonen
 * Description: Ukkonen's algorithm for online suffix tree construction.
 *  Each node contains indices [l, r) into the string, and a list of child nodes
     .
 *  Suffixes are given by traversals of this tree, joining [l, r) substrings.
 *  The root is 0 (has l = -1, r = 0), non-existent children are -1.
 *  To get a complete tree, append a dummy symbol -- otherwise it may contain
 *  an incomplete path (still useful for substring matching, though).
 * Time: $O(26N)$
 * Status: stress-tested a bit
 */
#pragma once

struct SuffixTree {
        enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
        int toi(char c) { return c - 'a'; }
        string a; // v = cur node, q = cur position
        int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;

        void ukkadd(int i, int c) { suff:
                if (r[v]<=q) {
                        if (t[v][c]==-1) { t[v][c]=m;  l[m]=i;
                                p[m++]=v; v=s[v]; q=r[v];  goto suff; }
                        v=t[v][c]; q=l[v];
                }
                if (q==-1 || c==toi(a[q])) q++; else {
                        l[m+1]=i;  p[m+1]=m;  l[m]=l[v];  r[m]=q;
                        p[m]=p[v];  t[m][c]=m+1;  t[m][toi(a[q])]=v;
                        l[v]=q;  p[v]=m;  t[p[m]][toi(a[l[m]])]=m;
                        v=s[p[m]];  q=l[m];
                        while (q<r[m]) { v=t[v][toi(a[q])];  q+=r[v]-l[v]; }
                        if (q==r[m])  s[m]=v;  else s[m]=m+2;
                        q=r[v]-(q-r[m]);  m+=2;  goto suff;
```

```
                }
        }

        SuffixTree(string a) : a(a) {
                fill(r,r+N,sz(a));
                memset(s, 0, sizeof s);
                memset(t, -1, sizeof t);
                fill(t[1],t[1]+ALPHA,0);
                s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
                rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
        }

        // example: find longest common substring (uses ALPHA = 28)
        pii best;
        int lcs(int node, int i1, int i2, int olen) {
                if (l[node] <= i1 && i1 < r[node]) return 1;
                if (l[node] <= i2 && i2 < r[node]) return 2;
                int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
                rep(c,0,ALPHA) if (t[node][c] != -1)
                        mask |= lcs(t[node][c], i1, i2, len);
                if (mask == 3)
                        best = max(best, {len, r[node] - len});
                return mask;
        }
        static pii LCS(string s, string t) {
                SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
                st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
                return st.best;
        }
};
```

12.9 Zfunc

```
/**
 * Author: chilli
 * License: CC0
 * Description: z[i] computes the length of the longest common prefix of s[i:]
     and s,
 * except z[0] = 0. (abacaba -> 0010301)
 * Time: O(n)
 * Status: stress-tested
 */
#pragma once

vi Z(const string& S) {
        vi z(sz(S));
        int l = -1, r = -1;
        rep(i,1,sz(S)) {
                z[i] = i >= r ? 0 : min(r - i, z[i - l]);
                while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
                        z[i]++;
                if (i + z[i] > r)
                        l = i, r = i + z[i];
        }
        return z;
}
```

13 kactl forked/various

13.1 BumpAllocator

```
/**
 * Author: Simon Lindholm
 * Date: 2015-09-12
 * License: CC0
 * Source: me
 * Description: When you need to dynamically allocate many objects and don't
     care about freeing them.
 * "new X" otherwise has an overhead of something like 0.05us + 16 bytes per
     allocation.
 * Status: tested
 */
#pragma once

// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
        static size_t i = sizeof buf;
        assert(s < i);
        return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

13.2 BumpAllocatorSTL

```
/**
 * Author: Simon Lindholm
 * Date: 2016-07-23
 * License: CC0
 * Source: me
 * Description: BumpAllocator for STL containers.
 * Usage: vector<vector<int, small<int>>> ed(N);
 * Status: tested
 */
#pragma once

char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;

template<class T> struct small {
        typedef T value_type;
        small() {}
        template<class U> small(const U&) {}
        T* allocate(size_t n) {
                buf_ind -= n * sizeof(T);
                buf_ind &= 0 - alignof(T);
                return (T*)(buf + buf_ind);
        }
        void deallocate(T*, size_t) {}
};
```

13.3 ConstantIntervals

```
/**
 * Author: Simon Lindholm
 * Date: 2015-03-20
 * License: CC0
 * Source: me
 * Description: Split a monotone function on [from, to) into a minimal set of
     half-open intervals on which it has the same value.
 *  Runs a callback g for each such interval.
 * Usage: constantIntervals(0, sz(v), [\&](int x){return v[x];}, [\&](int lo,
     int hi, T val){...});
 * Time: O(k\log\frac{n}{k})
 * Status: tested
 */
#pragma once

template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
        if (p == q) return;
        if (from == to) {
                g(i, to, p);
                i = to; p = q;
        } else {
                int mid = (from + to) >> 1;
                rec(from, mid, f, g, i, p, f(mid));
                rec(mid+1, to, f, g, i, p, q);
        }
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
```

```
        if (to <= from) return;
        int i = from; auto p = f(i), q = f(to-1);
        rec(from, to-1, f, g, i, p, q);
        g(i, to, q);
}
```

13.4 DivideAndConquerDP

```
/**
 * Author: Simon Lindholm
 * License: CC0
 * Source: Codeforces
 * Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i, k))$ where the (
     minimal)
 * optimal $k$ increases with $i$, computes $a[i]$ for $i = L..R-1$.
 * Time: O((N + (hi-lo)) \log N)
 * Status: tested on http://codeforces.com/contest/321/problem/E
 */
#pragma once

struct DP { // Modify at will:
        int lo(int ind) { return 0; }
        int hi(int ind) { return ind; }
        ll f(int ind, int k) { return dp[ind][k]; }
        void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

        void rec(int L, int R, int LO, int HI) {
                if (L >= R) return;
                int mid = (L + R) >> 1;
                pair<ll, int> best(LLONG_MAX, LO);
                rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
                        best = min(best, make_pair(f(mid, k), k));
                store(mid, best.second, best.first);
                rec(L, mid, LO, best.second+1);
                rec(mid+1, R, best.second, HI);
        }
        void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

13.5 FastInput

```
/**
 * Author: chilli
 * License: CC0
 * Source: Own work
 * Description: Read an integer from stdin. Usage requires your program to pipe
     in
 * input from file.
 * Usage: ./a.out < input.txt
 * Time: About 5x as fast as cin/scanf.
 * Status: tested on SPOJ INTEST, unit tested
 */
#pragma once

inline char gc() { // like getchar()
        static char buf[1 << 16];
        static size_t bc, be;
        if (bc >= be) {
                buf[0] = 0, bc = 0;
                be = fread(buf, 1, sizeof(buf), stdin);
        }
        return buf[bc++]; // returns 0 on EOF
}

int readInt() {
        int a, c;
        while ((a = gc()) < 40);
        if (a == '-') return -readInt();
        while ((c = gc()) >= 48) a = a * 10 + c - 480;
        return a - 48;
}
```

13.6 FastKnapsack

```
/**
 * Author: Marten Wiman
 * License: CC0
 * Source: Pisinger 1999, "Linear Time Algorithms for Knapsack Problems with
     Bounded Weights"
 * Description: Given N non-negative integer weights w and a non-negative target
     t,
 * computes the maximum S <= t such that S is the sum of some subset of the
     weights.
 * Time: O(N \max(w_i))
 * Status: Tested on kattis:eavesdropperevasion, stress-tested
 */
#pragma once

int knapsack(vi w, int t) {
        int a = 0, b = 0, x;
        while (b < sz(w) && a + w[b] <= t) a += w[b++];
        if (b == sz(w)) return a;
        int m = *max_element(all(w));
        vi u, v(2*m, -1);
        v[a+m-t] = b;
        rep(i,b,sz(w)) {
                u = v;
                rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
                for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
                        v[x-w[j]] = max(v[x-w[j]], j);
        }
        for (a = t; v[a+m-t] < 0; a--) ;
        return a;
}
```

13.7 FastMod

```
/**
 * Author: Simon Lindholm
 * Date: 2020-05-30
 * License: CC0
 * Source: https://en.wikipedia.org/wiki/Barrett_reduction
 * Description: Compute $a \% b$ about 5 times faster than usual, where $b$ is
     constant but not known at compile time.
 * Returns a value congruent to $a \pmod b$ in the range $[0, 2b)$.
 * Status: proven correct, stress-tested
 * Measured as having 4 times lower latency, and 8 times higher throughput, see
     stress-test.
 * Details:
 * More precisely, it can be proven that the result equals 0 only if $a = 0$,
 * and otherwise lies in $[1, (1 + a/2^64) * b)$.
 */
#pragma once

typedef unsigned long long ull;
struct FastMod {
        ull b, m;
        FastMod(ull b) : b(b), m(-1ULL / b) {}
        ull reduce(ull a) { // a % b + (0 or b)
                return a - (ull)((__uint128_t(m) * a) >> 64) * b;
        }
};
```

13.8 IntervalContainer

```
/**
 * Author: Simon Lindholm
 * License: CC0
 * Description: Add and remove intervals from a set of disjoint intervals.
 * Will merge the added interval with any overlapping intervals in the set when
     adding.
```

```
 * Intervals are [inclusive, exclusive).
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
	if (L == R) return is.end();
	auto it = is.lower_bound({L, R}), before = it;
	while (it != is.end() && it->first <= R) {
		R = max(R, it->second);
		before = it = is.erase(it);
	}
	if (it != is.begin() && (--it)->second >= L) {
		L = min(L, it->first);
		R = max(R, it->second);
		is.erase(it);
	}
	return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L, int R) {
	if (L == R) return;
	auto it = addInterval(is, L, R);
	auto r2 = it->second;
	if (it->first == L) is.erase(it);
	else (int&)it->second = L;
	if (R != r2) is.emplace(R, r2);
}
```

13.9 IntervalCover

```
/**
 * Author: Johan Sannemo
 * License: CC0
 * Description: Compute indices of smallest set of intervals covering another
     interval.
 * Intervals should be [inclusive, exclusive). To support [inclusive, inclusive
     ],
 * change (A) to add \texttt{|| R.empty()}. Returns empty set on failure (or if
     G is empty).
 * Time: O(N \log N)
 * Status: Tested on kattis:intervalcover
 */
#pragma once

template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
	vi S(sz(I)), R;
	iota(all(S), 0);
	sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
	T cur = G.first;
	int at = 0;
	while (cur < G.second) { // (A)
		pair<T, int> mx = make_pair(cur, -1);
		while (at < sz(I) && I[S[at]].first <= cur) {
			mx = max(mx, make_pair(I[S[at]].second, S[at]));
			at++;
		}
		if (mx.second == -1) return {};
		cur = mx.first;
		R.push_back(mx.second);
	}
	return R;
}
```

13.10 KnuthDP

```
/**
 * Author: Simon Lindholm
 * License: CC0
 * Source: http://codeforces.com/blog/entry/8219
 * Description: When doing DP on intervals: $a[i][j] = \min_{i < k < j}(a[i][k]
     + a[k][j]) + f(i, j)$, where the (minimal) optimal $k$ increases with both
      $i$ and $j$,
 *  one can solve intervals in increasing order of length, and search $k = p[i][
    j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$.
 *  This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \le f
    (a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \
    le d$.
 *  Consider also: LineContainer (ch. Data structures), monotone queues, ternary
     search.
 * Time: O(N^2)
 */
```

13.11 LIS

```
/**
 * Author: Johan Sannemo
 * License: CC0
 * Description: Compute indices for the longest increasing subsequence.
 * Time: $O(N \log N)$
 * Status: Tested on kattis:longincsubseq, stress-tested
 */
#pragma once

template<class I> vi lis(const vector<I>& S) {
	if (S.empty()) return {};
	vi prev(sz(S));
	typedef pair<I, int> p;
	vector<p> res;
	rep(i,0,sz(S)) {
		// change 0 -> i for longest non-decreasing subsequence
		auto it = lower_bound(all(res), p{S[i], 0});
		if (it == res.end()) res.emplace_back(), it = res.end()-1;
		*it = {S[i], i};
		prev[i] = it == res.begin() ? 0 : (it-1)->second;
	}
	int L = sz(res), cur = res.back().second;
	vi ans(L);
	while (L--) ans[L] = cur, cur = prev[cur];
	return ans;
}
```

13.12 SIMD

```
/**
 * Author: Simon Lindholm
 * Date: 2015-03-18
 * License: CC0
 * Source: https://software.intel.com/sites/landingpage/IntrinsicsGuide/
 * Description: Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on
     several numbers at once.
 * Can provide a constant factor improvement of about 4, orthogonal to loop
     unrolling.
 * Operations follow the pattern \texttt{"\_mm(256)?\_name\_(si(128|256)|epi
     (8|16|32|64)|pd|ps)"}. Not all are described here;
 * grep for \texttt{\_mm\_} in \texttt{/usr/lib/gcc/{*}/4.9/include/} for more.
 * If AVX is unsupported, try 128-bit operations, "emmintrin.h" and \#define \
     texttt{\_\_SSE\_\_} and \texttt{\_\_MMX\_\_} before including it.
 * For aligned memory use \texttt{\_mm\_malloc(size, 32)} or \texttt{int buf[N]
     alignas(32)}, but prefer loadu/storeu.
 */
#pragma once

#pragma GCC target ("avx2") // or sse4.1
#include "immintrin.h" /** keep-include */

typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))

// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd) (z?y:x), movemask_epi8 (hibits of bytes)
```

```
// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_epi8(x, y) takes a vector instead of an imm

// Methods that work with most data types (append e.g. _epi32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)

int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
        int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }

ll example_filteredDotProduct(int n, short* a, short* b) {
        int i = 0; ll r = 0;
        mi zero = _mm256_setzero_si256(), acc = zero;
        while (i + 16 <= n) {
                mi va = L(a[i]), vb = L(b[i]); i += 16;
                va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
                mi vp = _mm256_madd_epi16(va, vb);
                acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
                        _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
        }
        union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
        for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <- equiv
        return r;
}
```

13.13 SmallPtr

```
/**
 * Author: Simon Lindholm
 * Date: 2016-08-23
 * License: CC0
 * Source: me
 * Description: A 32-bit pointer that points into BumpAllocator memory.
 * Status: tested
 */
#pragma once

#include "BumpAllocator.h"

template<class T> struct ptr {
        unsigned ind;
        ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0) {
                assert(ind < sizeof buf);
        }
        T& operator*() const { return *(T*)(buf + ind); }
        T* operator->() const { return &**this; }
        T& operator[](int a) const { return (&**this)[a]; }
        explicit operator bool() const { return ind; }
};
```

13.14 TernarySearch

```
/**
 * Author: Simon Lindholm
 * Date: 2015-05-12
 * License: CC0
 * Source: own work
 * Description:
 * Find the smallest i in $[a,b]$ that maximizes $f(i)$, assuming that $f(a) < \
     dots < f(i) \ge \dots \ge f(b)$.
 * To reverse which of the sides allows non-strict inequalities, change the <
     marked with (A) to <=, and reverse the loop at (B).
 * To minimize $f$, change it to >, also at (B).
 * Usage:
       int ind = ternSearch(0,n-1,[\&](int i){return a[i];});
 * Time: O(\log(b-a))
 * Status: tested
 */
#pragma once

template<class F>
int ternSearch(int a, int b, F f) {
        assert(a <= b);
        while (b - a >= 5) {
                int mid = (a + b) / 2;
                if (f(mid) < f(mid+1)) a = mid; // (A)
                else b = mid+1;
        }
        rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
        return a;
}
```

13.15 Unrolling

```
/**
 * Author: Simon Lindholm
 * Date: 2015-03-19
 * License: CC0
 * Source: me
 * Description:
 */
#pragma once

#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if needed
while (i + 4 <= to) { F F F F }
while (i < to) F
```

14 Math

14.1 baby step gigant step

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class string>
using ordered_set = tree<string, null_type, less<string>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 2001
#define mod 1000000007

int bsgs(int a, int b, int m)
{
  if (a == 0 && b == 0)
    return 1;
  a %= m, b %= m;
  int k = 1, add = 0, g;
  while ((g = __gcd(a, m)) > 1) // fazer a e m serem coprimos
  {
    if (b == k)
```

```
      return add;
    if (b % g)
      return -1;
    b /= g, m /= g, ++add;
    k = (k * 1ll * a / g) % m;
  }
  int n = sqrt(m) + 1;
  int an = 1;
  for (int i = 0; i < n; i++)
    an = (an * 1ll * a) % m;
  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q <= n; q++)
  {
    vals[cur] = q;
    cur = (cur * 1ll * a) % m;
  }
  for (int p = 1, cur = k; p <= n; p++)
  {
    cur = (cur * 1ll * an) % m;
    if (vals.count(cur))
    {
      int ans = n * p - vals[cur] + add;
      return ans;
    }
  }
  return -1;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int a, b, m;
    cin >> a >> b >> m;
    cout << bsgs(a, b, m) << endl;
  }
  return 0;
}
// menor x tal que: (a^x) % m = b % m
// a e m sao coprimos
// se nao forem coprimos tem como tratar
// complexidade: sqrt(m)
```

14.2 berlekamp massey

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

struct modint
{
  int val;
  modint(int v = 0) { val = ((v % mod) + mod) % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};

// berlekamp massey (muito roubado)
// mas precisa que o mod seja primo (para poder achar inverso)
// dado os n primeiros termos de uma recorrencia linear
// a[0], a[1], a[2], ..., a[n - 1]
// ele acha a reccorrencia linear mais curta que da matching com os n primeiros
    valores
vector<modint> berlekamp_massey(vector<modint> x)
{
  vector<modint> ls, cur;
  int lf, ld;
  for (int i = 0; i < x.size(); i++)
  {
    modint t = 0;
    for (int j = 0; j < cur.size(); j++)
    {
      t += (x[i - j - 1] * cur[j]);
    }
    if (modint(t - x[i]).val == 0)
      continue;
    if (cur.empty())
    {
      cur.resize(i + 1);
      lf = i;
      ld = (t - x[i]) % mod;
      continue;
    }
    modint k = -(x[i] - t);
    k *= modint(ld).inv();
    vector<modint> c(i - lf - 1);
    c.pb(k);
    for (auto const &j : ls)
    {
      modint curr = modint(j.val * -1) * k;
      c.pb(curr);
    }
    if (c.size() < cur.size())
      c.resize(cur.size());
    for (int j = 0; j < cur.size(); j++)
    {
      c[j] = c[j] + cur[j];
    }
    if (i - lf + ls.size() >= cur.size())
    {
      tie(ls, lf, ld) = make_tuple(cur, i, t - x[i]);
    }
    cur = c;
  }
  return cur;
}
modint get_nth(vector<modint> rec, vector<modint> dp, int n)
{
  int m = rec.size();
  vector<modint> s(m), t(m);
  s[0] = 1;
```

```
  if (m != 1)
    t[1] = 1;
  else
    t[0] = rec[0];
  auto mul = [&rec](vector<modint> v, vector<modint> w)
  {
    vector<modint> ans(2 * v.size());
    for (int j = 0; j < v.size(); j++)
    {
      for (int k = 0; k < v.size(); k++)
        ans[j + k] += v[j] * w[k];
    }
    for (int j = 2 * v.size() - 1; j >= v.size(); j--)
    {
      for (int k = 1; k <= v.size(); k++)
        ans[j - k] += ans[j] * rec[k - 1];
    }
    ans.resize(v.size());
    return ans;
  };
  while (n)
  {
    if (n & 1)
      s = mul(s, t);
    t = mul(t, t);
    n >>= 1;
  }
  modint ret = 0;
  for (int i = 0; i < m; i++)
    ret += s[i] * dp[i];
  return ret;
}
modint guess_nth_term(vector<modint> x, int n)
{
  if (n < x.size())
    return x[n];
  vector<modint> coef = berlekamp_massey(x); // coeficientes da recorrencia
  /*for (auto const &i : coef)
    cout << i.val << " ";
  cout << endl;*/
  if (coef.empty())
    return 0;
  return get_nth(coef, x, n);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  vector<modint> vals;
  vals.pb(0);
  vals.pb(1);
  for (int i = 2; i <= 200; i++)
    vals.pb(vals[vals.size() - 1] + vals[vals.size() - 2]);
  int n;
  cin >> n;
  cout << guess_nth_term(vals, n).val << endl;
  return 0;
}
// exemplo fibonacci
```

14.3 binomial theorem

```
#include <bits/stdc++.h>
using namespace std;

#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
#pragma gcc optimize("Ofast")

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second

#define MAXN 100005
#define mod 1000000007

struct modint
{
  int val;
  modint(int v = 0) { val = (v < 0) ? v + mod : v; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};

modint f[MAXN];
modint invfat[MAXN];

void calc()
{
  f[0] = 1;
  for (int i = 1; i < MAXN; i++)
    f[i] = f[i - 1] * i;
  for (int i = 0; i < MAXN; i++)
    invfat[i] = f[i].inv();
}
modint ncr(int n, int k)
{
  modint ans = f[n] * invfat[k];
  ans *= invfat[n - k];
  return ans;
}

struct strange_modint // escrever um numero na forma: a + (b * sqrt(5))
{
  modint a, b;
  strange_modint() { a = 0, b = 0; }
  strange_modint(modint v) { a = v, b = 0; }
  strange_modint(modint v, modint v2) { a = v, b = v2; }
  strange_modint operator*(strange_modint o)
  {
    return strange_modint((a * o.a) + (b * o.b * 5), (a * o.b) + (b * o.a));
  }
  strange_modint operator+(strange_modint o)
  {
    return strange_modint(a + o.a, b + o.b);
  }
  strange_modint operator-(strange_modint o)
  {
    return strange_modint(a - o.a, b - o.b);
  }
  strange_modint pow(int y)
  {
    strange_modint x(a, b);
    strange_modint z(1, 0);
    while (y)
    {
      if (y & 1)
        z = z * x;
```

```
      x = x * x;
      y >>= 1;
    }
    return z;
  }
};

strange_modint matrix_ans[3][3];

void matrix_multiply(strange_modint a[3][3], strange_modint b[3][3])
{
  strange_modint res[3][3];
  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < 3; j++)
    {
      for (int k = 0; k < 3; k++)
        res[i][j] = res[i][j] + (a[i][k] * b[k][j]);
    }
  }
  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < 3; j++)
      a[i][j] = res[i][j];
  }
}
void matrix_pow(strange_modint mat[3][3], int m)
{
  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < 3; j++)
      matrix_ans[i][j] = strange_modint((i == j));
  }
  while (m > 0)
  {
    if (m & 1)
      matrix_multiply(matrix_ans, mat);
    m = m / 2;
    matrix_multiply(mat, mat);
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  calc();
  int n, k;
  cin >> n >> k;
  strange_modint a(modint(1) / 2, modint(1) / 2);
  strange_modint b(modint(1) / 2, -(modint(1) / 2));
  strange_modint c(0, modint(1) / 5);
  strange_modint ans(0, 0);
  for (int j = 0; j <= k; j++)
  {
    strange_modint curr(modint(1000000006).pow(j), 0); // (-1)^j
    curr = curr * strange_modint(ncr(k, j), 0);
    strange_modint prod = a.pow(k - j) * b.pow(j);
    strange_modint s0(0, 0);
    strange_modint s1(1, 0);
    strange_modint mat[3][3] = {{prod, s0, s1}, {s1, s0, s0}, {s0, s0, s1}};
    matrix_pow(mat, n);
    curr = curr * (matrix_ans[0][0] + matrix_ans[0][2]);
    ans = ans + curr;
  }
  ans = ans * c.pow(k);
  cout << ans.a.val << endl;
  return 0;
}
// https://codeforces.com/gym/104412/problem/F
```

14.4 catalan

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 400005
#define mod 998244353

struct modint
{
  int val;
  modint(int v = 0) { val = v % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};

modint f[MAXN];
modint inv[MAXN];
modint invfat[MAXN];

void calc()
{
  f[0] = 1;
  for (int i = 1; i < MAXN; i++)
  {
    f[i] = f[i - 1] * i;
  }
  invfat[0] = 1;
  for (int i = MAXN - 1; i >= 1; i--)
  {
    invfat[i] = modint(f[i]).inv();
  }
}
modint ncr(int n, int k)
{
  modint ans = f[n] * invfat[k];
  ans *= invfat[n - k];
  return ans;
}
modint catalan(int x)
{
  modint ans = modint(1) / modint(x + 1);
  ans *= ncr(2 * x, x);
  return ans;
}
signed main()
{
```

```
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  calc();
  cout << catalan(10).val << endl;
  return 0;
}
// catalan numbers:
// 1, 1, 2, 5, 14, 42, 132, 429, 1430, ...
// c[i] = (1 / (n + 1)) * ncr(2n, n)

// existem c[i] regular bracket sequences de tamanho 2 * n
// o numero de binary trees completas com n + 1 folhas
// o numero de possibilidades de conectar 2n numeros em um circulo em n arestas
// e uma serie de coisas

// https://cp-algorithms.com/combinatorics/catalan-numbers.html
```

14.5 crivo

```
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pd pair<double, int>
#define pii pair<int, pi>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 100001
#define mod 1000000007

bitset <MAXN> prime;

void crivo ()
{
  prime.set();
  prime[0] = false;
  prime[1] = false;
  for (int i = 2 ; i < MAXN ; i++)
    if(prime[i])
      for(int j = 2 ; j * i < MAXN ; j++)
        prime[j * i] = false;
}
signed main()
{
  crivo();
  int q;
  cin >> q;
  while(q--)
  {
    int n;
    cin >> n;
    (prime[n]) ? cout << "YES\n" : cout << "NO\n" ;
  }
  return 0;
}
```

14.6 crt

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 2000006
//#define mod 1000000007

namespace crt
{
  vector<pi> eq;

  int gcd(int a, int b, int &x, int &y)
  {
    if (b == 0)
    {
      x = 1, y = 0;
      return a;
    }
    int x1, y1, d = gcd(b, a % b, x1, y1);
    x = y1, y = x1 - y1 * (a / b);
    return d;
  }
  pi crt()
  {
    int a1 = eq[0].fir, m1 = eq[0].sec;
    a1 %= m1;
    for (int i = 1; i < eq.size(); i++)
    {
      int a2 = eq[i].fir, m2 = eq[i].sec;
      int g = __gcd(m1, m2);
      if (a1 % g != a2 % g)
        return {-1, -1};
      int p, q;
      gcd(m1 / g, m2 / g, p, q);
      int mod = m1 / g * m2;
      int x = (a1 * (m2 / g) % mod * q % mod + a2 * (m1 / g) % mod * p % mod) %
           mod;
      a1 = x;
      if (a1 < 0)
        a1 += mod;
      m1 = mod;
    }
    return {a1, m1};
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  for (int i = 0; i < n; i++)
  {
    int a, b;
    cin >> a >> b;
    crt::eq.pb({a, b});
  }
  pi ans = crt::crt();
  if (ans.fir == -1)
    cout << "No solution\n";
  else
    cout << ans.fir << " " << ans.sec << endl;
  return 0;
}
// references:
// https://forthright48.com/chinese-remainder-theorem-part-2-non-coprime-moduli/
// https://cp-algorithms.com/algebra/chinese-remainder-theorem.html
// https://www.geeksforgeeks.org/chinese-remainder-theorem-set-1-introduction/

// teorema chines do resto(crt)
// para resolver sistemas de congruencias modulares
// o menor inteiro a que satisfaz:
// a mod p1 = x1
// a mod p2 = x2
// ...
// a mod pn = xn
// a funcao crt retorna um pair {a, mod}
```

```
// dai a solucao pode ser descrita como
// x = a % mod
// entao os valores possiveis sao:
// a, (a + mod), a + (2 * mod), a + (3 * mod), ...
```

14.7 crt trick

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 200005

vector<pi> eq;
map<int, int> by_mod;

bool check(pi curr)
{
  if (by_mod.find(curr.sec) != by_mod.end())
  {
    return by_mod[curr.sec] == curr.fir;
  }
  // no maximo O(sqrt(n)) mods distintos
  // ja que tem O(sqrt(n)) tamanhos de ciclos distintos na permutacao
  // ja que a soma dos tamanhos dos ciclos da permutacao eh igual a n
  for (auto [x, mod] : eq)
  {
    if ((curr.fir - x) % __gcd(curr.sec, mod))
      return 0;
  }
  return 1;
}
void add(pi curr)
{
  eq.pb(curr);
  by_mod[curr.sec] = curr.fir;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<int> p(n);
  for (int i = 0; i < n; i++)
  {
    cin >> p[i];
    p[i]--;
  }
  vector<int> a(n);
  for (int i = 0; i < n; i++)
  {
    cin >> a[i];
    a[i]--;
  }
  vector<bool> vis(n, 0);
  vector<int> comp(n);
  vector<int> pos(n);
  vector<vector<int>> vals(n);
  vector<vector<int>> vals2(n);
  for (int i = 0; i < n; i++)
  {
    if (!vis[i])
    {
      int x = i;
      while (!vis[x])
      {
        vis[x] = 1;
        comp[x] = i;
        pos[x] = vals[i].size();
        vals[i].pb(a[x]);
        vals2[i].pb(x);
        x = p[x];
      }
    }
  }
  set<int> st;
  vector<int> resp(n);
  for (int i = 0; i < n; i++)
  {
    if (st.find(comp[i]) == st.end())
    {
      st.insert(comp[i]);
      int x = comp[i];
      int sz = vals[x].size();
      int mn = 1e18, op = -1;
      for (int i = 0; i < sz; i++)
      {
        // numeros x tal que x % sz = i
        if (check({i, sz}) && vals[x][i] < mn)
        {
          mn = vals[x][i];
          op = i;
        }
      }
      for (int i = 0; i < sz; i++)
      {
        resp[vals2[x][i]] = vals[x][op];
        op = (op + 1) % sz;
      }
      add({op, sz});
    }
  }
  for (auto const &i : resp)
  {
    cout << i + 1 << " ";
  }
  cout << endl;
}
// a[i] -> a[p[i]]
// i -> p[i]
// tenho varios ciclos
// e quero minimizar lexocograficamente a permutacao
// o que eu quero e:
// para cada ciclo da permutacao, ver se em um dado momento do ciclo
// eh possivel satisfazer o sistema de equacoes modulares

// dado um sistema com duas equacoes modulares
// x == n1 (mod m1)
// x == n2 (mod m2)
// existe solucao se (n1 - n2) % gcd(m1, m2) == 0

// para n equacoes podemos fazer esse check
// iterando por todos os pares de equacoes
// e checando se todo par satisfaz

// se eu ja tenho n equacoes no conjunto
// e eu sei que esse sistema de n equacoes tem solucao
// entao pra adicionar mais uma
// so preciso checar iterando pelas as que ja existem e dale
```

14.8 diophantine

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
```

```
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200001
#define mod 998244353

namespace dio
{
  vector<pi> sols;

  int gcd(int a, int b, int &x, int &y)
  {
    if (b == 0)
    {
      x = 1, y = 0;
      return a;
    }
    int x1, y1, d = gcd(b, a % b, x1, y1);
    x = y1, y = x1 - y1 * (a / b);
    return d;
  }
  void one_sol(int a, int b, int c)
  {
    int x0, y0, g;
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g)
      return;
    x0 *= (c / g);
    y0 *= (c / g);
    if (a < 0)
      x0 *= -1;
    if (b < 0)
      y0 *= -1;
    sols.pb({x0, y0});
  }
  void more_sols(int a, int b, int c)
  {
    int g = __gcd(a, b);
    int x0 = sols[0].fir, y0 = sols[0].sec;
    for (int k = -200000; k <= 200000; k++)
    {
      int x = x0 + k * (b / g);
      int y = y0 - k * (a / g);
      sols.pb({x, y});
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int a, b, c;
  cin >> a >> b >> c;
  dio::one_sol(a, b, c);
  if (!dio::sols.size())
  {
    cout << "No\n";
    return 0;
  }
  dio::more_sols(a, b, c);
  bool can = false;
  for (auto const &i : dio::sols)
    can |= (i.fir >= 0 && i.sec >= 0);
  (can) ? cout << "Yes\n" : cout << "No\n";
  return 0;
}
// equacoes do tipo:
// ax + by = c
// o caso a = 0 e b = 0, nao eh tratado nesse codigo
// nesse caso quero checar se equacao diofantina tem uma solucao
// com x >= 0 e y >= 0
```

14.9 division trick

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 1000000007

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  auto soma_pa = [&](int l, int r)
  {
    int x = (l + r) % mod;
    x = (x * ((r - l + 1) % mod)) % mod;
    x = (x * 500000004) % mod; // divisao por 2
    return x;
  };
  int ans = 0;
  for (int l = 1, r; l <= n; l = r + 1)
  {
    // todos os numeros i no intervalo [l, r] possuem (n / i) == x
    r = n / (n / l);
    int x = (n / l);
    // for (int i = l; i <= r; i++)
    //   assert((n / i) == x);
    ans += (x * soma_pa(l, r));
    ans %= mod;
  }
  cout << ans << endl;
}
// https://cses.fi/problemset/task/1082
// dado um n, ache a soma dos divisores de todos os numeros de 1 ate n, (n <=
    10^12)
// se a gnt computar a contribuicao de cada divisor na resposta
// isso fica: achar a soma de i * floor(n / i) para todos os numeros de 1 ate n
// a quatidade de valores distintos de floor(n / i) eh O(sqrt(n))
// entao o que queremos eh iterar por todos os valores possiveis, e ver o
    intervalo de numeros
// que tem o valor de floor(n / i) sendo um determinado x
```

14.10 divisors

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define fir first
#define sec second
#define MAXN 5001
#define mod 1000000007

signed main()
{
```

```
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  int ans = 0;
  for (int i = 1; i <= sqrt(n); i++)
  {
    if (!(n % i))
    {
      ans++;
      if (n / i != i)
        ans++;
    }
  }
  cout << ans << endl;
}
```

14.11 extended euclidean

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200001
#define mod 998244353

int gcd(int a, int b, int &x, int &y)
{
  if (b == 0)
  {
    x = 1;
    y = 0;
    return a;
  }
  int x1, y1;
  int d = gcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
signed main()
{
  int n;
  cin >> n;
  int k = 2;
  while (1)
  {
    int x, y;
    if (gcd(k, n, x, y) == 1)
    {
      x = ((x % n) + n) % n;
      cout << x << endl;
      return 0;
    }
    k++;
  }
  return 0;
}
// achar os numeros x e y tal que:
// a * x + b * y = gcd(a, b)

// problema exemplo:
// https://codeforces.com/group/btcK4I5D5f/contest/451372/problem/J
// dado um numero k
// quero achar um numero x, se possivel, tal que:
// (k * x) % n = 1

// k * x + n * y = 1
// se gcd(k, n) = 1, entao:
// k * x + n * y = gcd(k, n)
// note que, se gcd(k, n) > 1, logo nao existe solucao
```

14.12 fft

```
#include <bits/stdc++.h>

using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 125001
#define mod 1000000007
#define cd complex<double>

namespace fft
{
  int n;

  void fft(vector<cd> &a, bool invert)
  {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++)
    {
      int bit = n >> 1;
      for (; j & bit; bit >>= 1)
        j ^= bit;
      j ^= bit;
      if (i < j)
        swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1)
    {
      double ang = 2 * PI / len * (invert ? -1 : 1);
      cd wlen(cos(ang), sin(ang));
      for (int i = 0; i < n; i += len)
      {
        cd w(1);
        for (int j = 0; j < len / 2; j++)
        {
          cd u = a[i + j], v = a[i + j + len / 2] * w;
          a[i + j] = u + v;
          a[i + j + len / 2] = u - v;
          w *= wlen;
        }
      }
    }
    if (invert)
      for (cd &x : a)
        x /= n;
  }
  vector<int> mul(vector<int> a, vector<int> b)
  {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    n = 1;
    while (n < a.size() + b.size())
      n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
      fa[i] *= fb[i];
    fft(fa, true);
    vector<int> ans(n);
    for (int i = 0; i < n; i++)
      ans[i] = round(fa[i].real());
    return ans;
  }
}
signed main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);
```

```
  int n;
  cin >> n;
  int m = n + n;
  vector<int> a(m, 0);
  vector<int> aa(m, 0);
  for (int i = 0; i < n; i++)
  {
    cin >> a[i];
    aa[i] = (a[i] == 47) ? 1 : 0;
    a[i + n] = a[i];
    aa[i + n] = aa[i];
  }
  vector<int> b(m, 0);
  vector<int> bb(m, 0);
  for (int i = n - 1; i >= 0; i--)
  {
    cin >> b[i + n];
    bb[i + n] = (b[i + n] == 47) ? 1 : 0;
  }
  vector<int> ans1 = fft::mul(a, b);
  vector<int> ans2 = fft::mul(aa, bb);
  int ans = 0;
  for (int i = (m - 1); i < (m - 1) + n; i++)
  {
    if (ans2[i] > 0)
      continue;
    ans = max(ans1[i], ans); // produto escalar de algum cyclic shift
  }
  cout << ans << endl;
  return 0;
}
// https://algo.sk/br24/problem.php?problem=d3-badsquare
// exemplo do all possible scalar products
// dados dois arrays a e b de tamanho n
// quero computar o scalar product de todos os cyclics shifts de a com b
// duplicar o array a
// dar reverse no array b e adicionar n zeros no inicio
```

14.13 fraction

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200006
#define mod 1000000007

struct fraction
{
  int x, y; // x / y

  fraction() {}
  fraction(int x, int y) : x(x), y(y) {}
  bool operator==(fraction o) { return (x * o.y == o.x * y); }
  bool operator!=(fraction o) { return (x * o.y != o.x * y); }
  bool operator>(fraction o) { return (x * o.y > o.x * y); }
  bool operator>=(fraction o) { return (x * o.y >= o.x * y); }
  bool operator<(fraction o) { return (x * o.y < o.x * y); }
  bool operator<=(fraction o) { return (x * o.y <= o.x * y); }
  fraction operator+(fraction o)
  {
    fraction ans;
    ans.y = (y == o.y) ? y : y * o.y;
    ans.x = (x) * (ans.y / y) + (o.x) * (ans.y / o.y);
    // ans.simplify();
    return ans;
  }
  fraction operator*(fraction o)
  {
    fraction ans;
    ans.x = x * o.x;
    ans.y = y * o.y;
    // ans.simplify();
    return ans;
  }
  fraction inv()
  {
    fraction ans = fraction(x, y);
    swap(ans.x, ans.y);
    return ans;
  }
  fraction neg()
  {
    fraction ans = fraction(x, y);
    ans.x *= -1;
    return ans;
  }
  void simplify()
  {
    if (abs(x) > 1e9 || abs(y) > 1e9) // slow simplification
    {
      int g = __gcd(y, x);
      x /= g;
      y /= g;
    }
  }
  // substraction and division can be easily done
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
```

14.14 fwht

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1025
#define mod 998244353

struct modint
{
  int val;
  modint(int v = 0) { val = ((v % mod) + mod) % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
```

```
      z *= x;
    x *= x;
    y >>= 1;
  }
  return z.val;
}
int inv() { return pow(mod - 2); }
void operator=(int o) { val = o % mod; }
void operator=(modint o) { val = o.val % mod; }
void operator+=(modint o) { *this = *this + o; }
void operator-=(modint o) { *this = *this - o; }
void operator*=(modint o) { *this = *this * o; }
void operator/=(modint o) { *this = *this / o; }
bool operator==(modint o) { return val == o.val; }
bool operator!=(modint o) { return val != o.val; }
int operator*(modint o) { return ((val * o.val) % mod); }
int operator/(modint o) { return (val * o.inv()) % mod; }
int operator+(modint o) { return (val + o.val) % mod; }
int operator-(modint o) { return (val - o.val + mod) % mod; }
};

vector<modint> fwht(char op, vector<modint> f, bool inv = 0)
{
  int n = f.size();
  for (int k = 0; (n - 1) >> k; k++)
  {
    for (int i = 0; i < n; i++)
    {
      if (i >> k & 1)
      {
        int j = i ^ (1 << k);
        if (op == '^')
          f[j] += f[i], f[i] = f[j] - modint(2) * f[i];
        if (op == '|')
          f[i] += modint(inv ? -1 : 1) * f[j];
        if (op == '&')
          f[j] += modint(inv ? -1 : 1) * f[i];
      }
    }
  }
  if (op == '^' and inv)
  {
    for (auto &i : f)
      i /= n;
  }
  return f;
}
vector<modint> conv(char op, vector<modint> a, vector<modint> b)
{
  a = fwht(op, a, 0);
  b = fwht(op, b, 0);
  for (int i = 0; i < a.size(); i++)
  {
    a[i] *= b[i];
  }
  return fwht(op, a, 1);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  n = 1 << n;
  vector<modint> a(n);
  for (int i = 0; i < n; i++)
  {
    int x;
    cin >> x;
    a[i] = x;
  }
  vector<modint> b(n);
  for (int i = 0; i < n; i++)
  {
    int x;
    cin >> x;
    b[i] = x;
  }
  vector<modint> c = conv('^', a, b); // convolucao de xor
  for (auto const &i : c)
    cout << i.val << " ";
  cout << endl;
  vector<modint> d = conv('&', a, b); // convolucao de and
  for (auto const &i : d)
    cout << i.val << " ";
  cout << endl;
  return 0;
}
// o tipo ta como modint, mas tem como mudar para qualquer um
// usar preferencialmente tamanho como potencia de 2

// faz a convolucao de a com b
// c[k] = (a[i] * b[j]), com (i op j) = k
// op pode ser xor, and ou or

// para testar
// https://judge.yosupo.jp/problem/bitwise_xor_convolution
// https://judge.yosupo.jp/problem/bitwise_and_convolution
```

14.15 gaussian elimination

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define double long double
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 1
#define MAXN 2001
#define mod 1000000007
#define EPS 1e-9

vector<double> ans;

int gauss(vector<vector<double>> a)
{
  int n = a.size(), m = a[0].size() - 1, ret = 1;
  ans.assign(m, 0);
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; col++, row++)
  {
    int sel = row;
    for (int i = row; i < n; i++)
      if (abs(a[i][col]) > abs(a[sel][col]))
        sel = i;
    if (abs(a[sel][col]) < EPS)
      continue;
    for (int i = col; i <= m; i++)
      swap(a[sel][i], a[row][i]);
    where[col] = row;
    for (int i = 0; i < n; i++)
    {
      if (i != row)
      {
        double c = a[i][col] / a[row][col];
        for (int j = col; j <= m; j++)
          a[i][j] -= a[row][j] * c;
      }
    }
  }
  for (int i = 0; i < m; i++)
  {
```

```
    if (where[i] != -1)
      ans[i] = (a[where[i]][m] / a[where[i]][i]);
    else
      ret = 2;
  }
  for (int i = 0; i < n; i++)
  {
    double sum = 0;
    for (int j = 0; j < m; j++)
      sum += (ans[j] * a[i][j]);
    if (abs(sum - a[i][m]) > EPS)
      ret = 0;
  }
  return ret; // 0 = nao existe solucao, 1 = existe uma solucao, 2 = existem
       multiplas solucoes
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  vector<vector<double>> a = {{1.0, 1.0, 20.0},  // 1x + 1y = 20
                              {3.0, 4.0, 72.0}}; // 3x + 4y = 72
  cout << gauss(a) << endl;
  for (auto const &i : ans) // x = 8 e y = 12
    cout << i << " ";
  cout << endl;
}
// eliminacao gaussiana
// para resolver sistemas com n equacoes e m incognitas

// para isso iremos utilizar uma representacao usando
// matrizes, no qual uma coluna extra e adicionada,
// representando os resultados de cada equacao.

// algoritimo:
// ideia: qualquer equacao pode ser reescrita como uma combinacao linear dela
     mesma
// 1- dividir a primeira linha(primeira equacao) por a[0][0]
// 2- adicionar a primeira linha as linhas restantes, de modo que, os
//    coeficientes da primeira coluna se tornem todos zeros, para que
//    isso aconteca, na i-esima linha devemos adicionar a primeira linha
//    multiplicada por (a[i][0] * -1)
// 3- com isso, o elemento a[0][0] = 1 e os demais elementos da primeira coluna
//    serao iguais a zero
// 4- continuamos o algoritimo a partir da etapa 1 novamente, dessa vez
//    com a segunda coluna e a segunda linha, dividindo a linha por a[1][1]
//    e assim sucessivamente
// 5- ao final, teremos a resposta

// complexidade O(min(n, m) * n * m);
// se n == m, logo a complexidade sera O(n^3)
```

14.16 gaussian elimination2

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 4007
#define mod 998244353
#define EPS 1e-9

bitset<MAXN> ans;

int gauss(vector<bitset<MAXN>> &a)
{
  ans.reset();
  int n = a.size(), m = a[0].size() - 1, ret = 1;
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; col++)
  {
    for (int i = row; i < n; i++)
    {
      if (a[i][col])
      {
        swap(a[i], a[row]);
        break;
      }
    }
    if (!a[row][col])
      continue;
    where[col] = row;
    for (int i = 0; i < n; i++)
      if (i != row && a[i][col])
        a[i] ^= a[row];
    ++row;
  }
  for (int i = 0; i < m; i++)
  {
    if (where[i] != -1)
      ans[i] = (a[where[i]][m] / a[where[i]][i]);
    else
      ret = 2;
  }
  for (int i = 0; i < n; i++)
  {
    double sum = 0;
    for (int j = 0; j < m; j++)
      sum += (ans[j] * a[i][j]);
    if (abs(sum - a[i][m]) > EPS)
      ret = 0;
  }
  return ret;
}
signed main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);
  int m, n;
  cin >> m >> n;
  string s;
  getline(cin, s);
  auto get_id = [&](string st)
  {
    return n - stoi(st.substr(1));
  };
  vector<bitset<MAXN>> v(n + m);
  for (int i = 0; i < m; i++)
  {
    getline(cin, s);
    s = s.substr(1); // (
    if (i == m - 1)
      s = s.substr(0, s.size() - 1); // )
    else
      s = s.substr(0, s.size() - 5); // ) and
    istringstream input_stream(s);
    string t;
    while (input_stream >> t)
    {
      if (t == "not") // not var
      {
        input_stream >> t;
        v[i][n + get_id(t)] = v[i][n + get_id(t)] ^ 1;
      }
      else if (t != "or") // var
      {
        v[i][get_id(t)] = v[i][get_id(t)] ^ 1;
      }
    }
    v[i][MAXN - 1] = v[i][MAXN - 1] ^ 1;
  }
  for (int i = 0; i < n; i++)
  {
    v[i + m][i] = v[i + m][i] ^ 1;
    v[i + m][n + i] = v[i + m][n + i] ^ 1;
    v[i + m][MAXN - 1] = v[i + m][MAXN - 1] ^ 1;
  }
  if (gauss(v) == 0)
  {
```

```
    cout << "impossible\n";
    return 0;
  }
  string resp(n, 'F');
  int id = n - 1;
  for (int i = 0; i < (n + n); i++)
  {
    if (ans[i])
    {
      (i < n) ? resp[id] = 'T' : resp[id] = 'F';
    }
    id--;
    if (id < 0)
      id = n - 1;
  }
  cout << resp << endl;
  return 0;
}
// exemplo de solucao para o https://codeforces.com/gym/101908/problem/M
// esse codigo ja acha a menor solucao lexicograficamente (caso exista)
// caso a gente queira a maior lexicograficamente (que e o caso desse problema
    exemplo)
// basta considerar as variaveis na ordem contraria
```

14.17 lagrange

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 600005
#define mod 1000000007

struct modint
{
  int val;
  modint(int v = 0) { val = v % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};
struct lagrange
{
  int n;
  vector<modint> den;
  vector<modint> y;
  vector<modint> fat;
  vector<modint> inv_fat;

  lagrange(vector<modint> &v) // f(i) = v[i], gera um polinomio de grau n - 1
  {
    n = v.size();
    calc(n);
    calc_den(n);
    y = v;
  }
  void calc_den(int n)
  {
    den.resize(n);
    for (int i = 0; i < n; i++)
    {
      den[i] = inv_fat[n - i - 1] * inv_fat[i];
      if ((n - i - 1) % 2 == 1)
      {
        int x = (mod - den[i].val) % mod;
        den[i] = x;
      }
    }
  }
  void calc(int n)
  {
    fat.resize(n + 1);
    inv_fat.resize(n + 1);
    fat[0] = 1;
    inv_fat[0] = 1;
    for (int i = 1; i <= n; i++)
    {
      fat[i] = fat[i - 1] * i;
      inv_fat[i] = fat[i].inv();
    }
  }
  modint get_val(int x) // complexidade: O(n)
  {
    x %= mod;
    vector<modint> l(n);
    vector<modint> r(n);
    l[0] = 1, r[n - 1] = 1;
    for (int i = 1; i < n; i++)
    {
      modint cof = (x - (i - 1) + mod);
      l[i] = l[i - 1] * cof;
    }
    for (int i = n - 2; i >= 0; i--)
    {
      modint cof = (x - (i + 1) + mod);
      r[i] = r[i + 1] * cof;
    }
    modint ans = 0;
    for (int i = 0; i < n; i++)
    {
      modint cof = l[i] * r[i];
      ans += modint(cof * y[i]) * den[i];
    }
    return ans;
  }
  vector<modint> find_coefs() // encontra os coeficientes do polinomio
  {
    int nn = n;
    int d = nn - 1;
    vector<modint> c(nn, 0);
    for (int i = 0; i < y.size(); i++)
    {
      c[d] += (y[i] * den[i]);
    }
    for (int p = nn - 2; p >= 0; p--)
    {
      nn--;
      calc_den(nn);
```

```
      for (int i = 0; i <= p; i++)
      {
        y[i] -= (c[p + 1] * modint(i).pow(d));
        c[p] += (y[i] * den[i]);
      }
      d--;
    }
    return c;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, k;
  cin >> n >> k;
  vector<modint> v;
  v.pb(0);
  int lim = k + 1;
  for (int i = 1; i <= lim; i++)
    v.pb(v.back() + modint(i).pow(k));
  lagrange l(v);
  cout << l.get_val(n).val << endl;
  return 0;
}
// https://codeforces.com/contest/622/problem/F
// https://codeforces.com/contest/1817/problem/C
// https://codeforces.com/gym/103388/problem/A
```

14.18 lucas theorem

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000006
#define mod 2

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n, k;
    cin >> n >> k;
    for (int i = 0; i < n; i++)
    {
      if (((n - 1) & i) == i)
        cout << k << " ";
      else
        cout << 0 << " ";
    }
    cout << endl;
  }
}
// https://codeforces.com/contest/2072/problem/F
// problema interessante de um div3, calcular ncr(n, k) mod 2
// ncr(n - 1, i) mod 2 = (((n - 1) & i) == i)

// sejam m e n numeros inteiros nao negativos e p um numero primo
// desenvolver n e m na base p
// ou seja:
// m = m[k]*p^(k) + m[k - 1]*p^(k - 1) + ... + m[0]*p^(0)
// n = n[k]*p^(k) + n[k - 1]*p^(k - 1) + ... + n[0]*p^(0)

// entao:
// ncr(m, n) mod p = produtorio de (ncr(m[i], n[i]) mod p)

// dai pra generalizar pro mod 2 eh deboas, pq se tiver um bit setado em n[i]
    que nao ta setado em m[i], entao miou, vai dar zero
```

14.19 markov

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

struct modint
{
  int val;
  modint(int v = 0) { val = ((v % mod) + mod) % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};

vector<modint> ans;

int gauss(vector<vector<modint>> a)
{
  int n = a.size(), m = a[0].size() - 1, ret = 1;
  ans.assign(m, 0);
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; col++, row++)
  {
    int sel = row;
    for (int i = row; i < n; i++)
    {
      if (a[i][col].val > a[sel][col].val)
```

```
      sel = i;
    }
    if (!a[sel][col].val)
    {
      continue;
    }
    for (int i = col; i <= m; i++)
    {
      swap(a[sel][i], a[row][i]);
    }
    where[col] = row;
    for (int i = 0; i < n; i++)
    {
      if (i != row)
      {
        modint c = a[i][col] / a[row][col];
        for (int j = col; j <= m; j++)
          a[i][j] -= modint(a[row][j] * c);
      }
    }
  }
  for (int i = 0; i < m; i++)
  {
    if (where[i] != -1)
      ans[i] = (a[where[i]][m] / a[where[i]][i]);
    else
      ret = 2;
  }
  for (int i = 0; i < n; i++)
  {
    modint sum = 0;
    for (int j = 0; j < m; j++)
      sum += (ans[j] * a[i][j]);
    if ((sum - a[i][m]) > 0)
      ret = 0;
  }
  assert(ret == 1); // so uma solucao existe
  return ret;
}

int dx[] = {-1, 0, 1, 0};
int dy[] = {0, -1, 0, 1};
modint a[4];

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int r;
  cin >> r;
  modint sum = 0;
  for (int i = 0; i < 4; i++)
  {
    cin >> a[i].val;
    sum += a[i];
  }
  for (int i = 0; i < 4; i++)
  {
    a[i] /= sum;
  }
  vector<pi> in;
  queue<pi> q;
  set<pi> vis;
  q.push({0, 0});
  vis.insert({0, 0});
  // quais posicoes tao a distancia <= R da origem
  while (!q.empty())
  {
    auto [x, y] = q.front();
    q.pop();
    if ((x * x) + (y * y) > (r * r))
      continue;
    in.pb({x, y});
    for (int dir = 0; dir < 4; dir++)
    {
      int i = x + dx[dir];
      int j = y + dy[dir];
      if (vis.find({i, j}) == vis.end())
      {
        q.push({i, j});
        vis.insert({i, j});
      }
    }
  }
  sort(in.begin(), in.end());
  int sz = in.size() + 1;
  auto get_id = [&](int x, int y)
  {
    if ((x * x) + (y * y) > (r * r))
      return sz - 1;
    return (int)(lower_bound(in.begin(), in.end(), pi(x, y)) - in.begin());
  };
  vector<vector<modint>> prob(sz, vector<modint>(sz, 0));
  // primeiro passo:
  // para cada nao terminal ii computamos prob[ii][j], probabilidade de ir de ii
       pra j em um move
  // todas as celulas com distancia > r, eu considero por um unico indice nessa
       representacao
  for (int ii = 0; ii < (sz - 1); ii++)
  {
    auto [x, y] = in[ii];
    for (int dir = 0; dir < 4; dir++)
    {
      int i = x + dx[dir];
      int j = y + dy[dir];
      int id = get_id(i, j);
      prob[ii][id] += a[dir];
    }
  }
  // agr quero calcular EV(i) para todo i
  // valor esperado de moves saindo da posicao i
  // monta o sisteminha de equacoes e dale
  vector<vector<modint>> eqs;
  for (int i = 0; i < sz; i++)
  {
    vector<modint> curr(sz + 1, 0);
    if (i == sz - 1) // estado terminal, EV(i) = 0
    {
      curr[sz - 1] = 1;
      eqs.pb(curr);
    }
    else // nao terminal, EV(i) = 1 + soma dos (prob[i][j] * EV(j))
    {
      for (int j = 0; j < sz; j++)
      {
        if (i == j)
          curr[j] = modint(1) - prob[i][j];
        else
          curr[j] = prob[i][j] * -1;
      }
      curr[sz] = 1;
      eqs.pb(curr);
    }
  }
  gauss(eqs);
  for (int i = 0; i < in.size(); i++)
  {
    if (in[i].fir == 0 && in[i].sec == 0) // ja que eu quero o valor esperado
         saindo da posicao (0,0)
      cout << ans[i].val << endl;
  }
  return 0;
}
// solucao tle (porem correta) de um problema legal
// https://codeforces.com/contest/963/problem/E

// tem um chip inicialmente na posicao (0, 0)
// e ele vai comecar a se mover aleatoriamente
// tem a[i] / (a[1] + a[2] + a[3] + a[4]) para se mover para a direcao i
// ele pode ir pra (x + 1, y) (x, y + 1), (x - 1, y), (x, y - 1)

// quero saber o valor esperado ate ele checar em uma celula
// com distancia maior do que R do (0, 0)
```

14.20 matrix exponentiation

```cpp
// https://codeforces.com/gym/102644/problem/C
// achar o n-esimo termo da sequencia de fibonacci mod (10^9 + 7) em O(log(n))
// n <= 10^18
// podemos escrever a recorrencia de fibonnaci como uma exponenciacao de matriz
/*
  (  fib(n)  )       (1 1) ^ (n - 1)       (fib(1) = 1)
  (fib(n - 1))   =   (1 0)              *   (fib(0) = 1)
*/
// e possivel fazer essa exponenciacao em O(log(n)) com um algoritimo muito
    similar ao de exponenciacao rapida
// dai calculamos o n-esimo termo da sequencia de fibonacci mod (10^9 + 7) em O(
    log(n))

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 201
#define mod 1000000007

namespace matrix
{
  vector<vector<int>> ans;

  int multi(int x, int y)
  {
    return (x * y) % mod;
  }
  int sum(int a, int b)
  {
    return (a + b >= mod) ? a + b - mod : a + b;
  }
  vector<vector<int>> multiply(vector<vector<int>> a, vector<vector<int>> b)
  {
    vector<vector<int>> res(a[0].size(), vector<int>(b[0].size()));
    for (int i = 0; i < a.size(); i++)
    {
      for (int j = 0; j < b[0].size(); j++)
      {
        res[i][j] = 0;
        for (int k = 0; k < a[0].size(); k++)
          res[i][j] = sum(res[i][j], multi(a[i][k], b[k][j]));
      }
    }
    return res;
  }
  vector<vector<int>> expo(vector<vector<int>> mat, int m)
  {
    ans = vector<vector<int>>(mat.size(), vector<int>(mat[0].size()));
    for (int i = 0; i < mat.size(); i++)
      for (int j = 0; j < mat[0].size(); j++)
        ans[i][j] = (i == j);
    while (m > 0)
    {
      if (m & 1)
        ans = multiply(ans, mat);
      m = m / 2;
      mat = multiply(mat, mat);
    }
    return ans;
  }
}

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<vector<int>> mat = {{1, 1}, {1, 0}};
  vector<vector<int>> ans = matrix::expo(mat, n);
  cout << ans[0][1] << endl;
  return 0;
}
```

14.21 matrix exponentiation2

```cpp
// https://www.spoj.com/problems/ITRIX12E/
// count some {f(0) + f(1) + ... + f(n)} with just one matrix exponentiation
// creates an extra dimension in the matrix and initializes that column with 1s

#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define pb push_back
#define mp make_pair
#define int long long int
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 100001
#define MAXL 20
#define INF 200001
#define mod 1000000007

const int n = 11;
vector<vector<int>> ans(n, vector<int>(n));

vector<vector<int>> multiply(vector<vector<int>> a, vector<vector<int>> b)
{
  vector<vector<int>> res(n, vector<int>(n));
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      res[i][j] = 0;
      for (int k = 0; k < n; k++)
        res[i][j] = (res[i][j] + (((a[i][k] % mod) * (b[k][j] % mod)) % mod)) %
            mod;
    }
  }
  return res;
}
vector<vector<int>> expo(vector<vector<int>> mat, int m)
{
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      ans[i][j] = (i == j);
  while (m > 0)
  {
    if (m & 1)
      ans = multiply(ans, mat);
    m = m / 2;
    mat = multiply(mat, mat);
  }
  return ans;
}
bool is_prime(int n)
{
  for (int i = 2; i < n; i++)
    if (!(n % i))
      return false;
  return true;
}
signed main()
{
```

```
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int k;
    cin >> k;
    int resp = 0;
    vector<vector<int>> mat(n, vector<int>(n, 0));
    for (int i = 1; i <= 9; i++)
      for (int j = 1; j <= 9; j++)
        if (is_prime(i + j))
          mat[i][j] = 1;
    for (int i = 0; i <= 10; i++)
      mat[i][10] = 1;
    vector<vector<int>> ans = expo(mat, k - 1);
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        resp = (resp + ans[i][j]) % mod;
    cout << resp - 7 << endl;
  }
  return 0;
}
```

14.22 matrix inverse and determinant

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 998244353

struct modint
{
  int val;
  modint(int v = 0) { val = (v + mod) % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};
pair<int, modint> gauss(vector<vector<modint>> &a, int pivot_end)
{
  int n = a.size(), m = a[0].size();
  int rank = 0, curr = pivot_end;
  if (curr == -1)
    curr = m;
  modint det = 1;
  for (int j = 0; j < curr; j++)
  {
    int idx = -1;
    for (int i = rank; i < n; i++)
    {
      if (a[i][j].val != 0)
      {
        idx = i;
        break;
      }
    }
    if (idx == -1)
    {
      det = 0;
      continue;
    }
    if (rank != idx)
    {
      det *= -1;
      swap(a[rank], a[idx]);
    }
    det *= a[rank][j];
    if (a[rank][j].val != 1)
    {
      modint coef = a[rank][j].inv();
      for (int k = j; k < m; k++)
        a[rank][k] *= coef;
    }
    int is = 0;
    for (int i = is; i < n; i++)
    {
      if (i == rank)
        continue;
      if (a[i][j].val != 0)
      {
        modint coef = a[i][j] / a[rank][j];
        for (int k = j; k < m; k++)
          a[i][k] -= a[rank][k] * coef;
      }
    }
    rank++;
  }
  return {rank, det};
}
vector<vector<modint>> inverse_matrix(vector<vector<modint>> a)
{
  int n = a.size();
  vector<vector<modint>> m(n, vector<modint>(2 * n));
  for (int i = 0; i < n; i++)
  {
    copy(begin(a[i]), end(a[i]), begin(m[i]));
    m[i][n + i] = 1;
  }
  auto [rank, det] = gauss(m, n);
  if (rank != n)
    return {};
  vector<vector<modint>> b(n);
  for (int i = 0; i < n; i++)
    copy(begin(m[i]) + n, end(m[i]), back_inserter(b[i]));
  return b;
}
modint determinant(vector<vector<modint>> a)
{
  return gauss(a, -1).sec;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<vector<modint>> v(n, vector<modint>(n));
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
      cin >> v[i][j].val;
  }
```

```
  if (determinant(v).val == 0)
  {
    cout << "-1\n";
    return 0;
  }
  vector<vector<modint>> ans = inverse_matrix(v);
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
      cout << ans[i][j].val << " ";
    cout << endl;
  }
  return 0;
}
// https://judge.yosupo.jp/problem/inverse_matrix
// https://judge.yosupo.jp/problem/matrix_det
// como precisa de divisao, entao o mod tem que ser primo
```

14.23 max xor subsequence

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 300005
#define mod 1000000007

int modpow(int a, int b)
{
  int res = 1;
  while (b > 0)
  {
    if (b & 1)
      res = (res * a) % mod;
    a = (a * a) % mod;
    b >>= 1;
  }
  return res;
}

int all, qt;
int dp[33];

void add(int x)
{
  all++;
  for (int i = 32; i >= 0; i--)
  {
    if (x & (1ll << i))
    {
      if (dp[i] == 0)
      {
        dp[i] = x;
        qt++;
        return;
      }
      x ^= dp[i];
    }
  }
}
int get(int x) // qual o x-esimo menor valor de xor de uma subsequencia
{
  int tot = (1ll << qt), ans = 0;
  for (int i = 32; i >= 0; i--)
  {
    if (dp[i] > 0)
    {
      int d = tot / 2;
      if (d < x && !(ans & (1ll << i)))
        ans ^= dp[i];
      else if (d >= x && (ans & (1ll << i)))
        ans ^= dp[i];
      if (d < x)
        x -= d;
      tot /= 2;
    }
  }
  return ans;
}
bool check(int x) // se existe pelo menos uma subsequencia com xor x
{
  for (int i = 32; i >= 0; i--)
  {
    if (x & (1ll << i))
    {
      if (!dp[i])
        return 0;
      x ^= dp[i];
    }
  }
  return 1;
}
int count(int x) // quantas subsequencias tem xor x
{
  if (!check(x))
    return 0;
  return modpow(2, all - qt);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
  {
    cin >> v[i];
    add(v[i]);
  }
  int x = get(1ll << qt); // maior xor possivel de uma subsequencia
  int y = get(1);         // maior xor possivel != 0 (o 0 sempre eh possivel -
       subsequencia vazia)
  return 0;
}
// referencia:
// https://codeforces.com/blog/entry/68953

// problemas:
// https://codeforces.com/gym/103708/problem/A
// https://codeforces.com/contest/959/problem/F
// https://codeforces.com/contest/1101/problem/G
// https://atcoder.jp/contests/abc283/tasks/abc283_g
```

14.24 mobius

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
```

```cpp
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 5000005
#define mod 1000000001

int lpf[MAXN];
int mobius[MAXN];
int g[MAXN];

void calc_lpf()
{
  for (int i = 2; i < MAXN; i++)
  {
    if (!lpf[i])
    {
      for (int j = i; j < MAXN; j += i)
      {
        if (!lpf[j])
          lpf[j] = i;
      }
    }
  }
}
void calc_mobius()
{
  calc_lpf();
  mobius[1] = 1;
  for (int i = 2; i < MAXN; i++)
  {
    if (lpf[i / lpf[i]] == lpf[i])
      mobius[i] = 0;
    else
      mobius[i] = -1 * mobius[i / lpf[i]];
  }
}
int count_pairs(int n)
{
  // f(n) -> contar pares (i, j) com __gcd(i, j) == 1 e 1 <= i, j <= n
  int ans = 0;
  for (int d = 1; d <= n; d++)
  {
    // quadrado pq sao pares (2 caras)
    // mas se fossem x caras seria (n / d)^x
    int sq = (n / d) * (n / d);
    int x = mobius[d] * sq;
    ans += x;
  }
  return ans;
}
int gcd_sum(int n)
{
  // soma de todos os gcd(i, j) com 1 <= i, j <= n
  int ans = 0;
  for (int k = 1; k <= n; k++) // fixa o valor do gcd(i, j) e conta quantos
       pares com gcd(i, j) == k
  {
    int lim = n / k;
    int curr = k * count_pairs(lim);
    ans += curr;
  }
  return ans;
}
int lcm_sum(int n)
{
  // soma de todos os lcm(i, j) com 1 <= i, j <= n
  for (int i = 1; i <= n; i++)
    g[i] = 0;
  for (int i = 1; i <= n; i++)
  {
    for (int j = i; j <= n; j += i)
      g[j] += (mobius[i] * j * i);
  }
  int ans = 0;
  for (int l = 1; l <= n; l++)
  {
    int cima = (1 + n / l) * (n / l);
    int f = (cima / 2) * (cima / 2);
    f *= g[l];
    ans += f;
  }
  return ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  calc_mobius();
  for (int i = 1; i <= q; i++)
  {
    int n;
    cin >> n;
    int ans = lcm_sum(n);
    for (int i = 1; i <= n; i++)
      ans -= i;
    ans /= 2;
    cout << "Case " << i << ": " << ans << endl;
  }
  return 0;
}
// https://codeforces.com/blog/entry/53925
// mobius inversion
// sejam f(x) e g(x) funcoes
// e g(x) e definida da seguinte maneira
// g(x) = soma dos f(d), no qual d eh um divisor de x

// temos que:
// f(n) = soma dos (g(d) * u(n / d)), no qual d eh um divisor de x
// u(x) -> mobius function

// propiedade legal:
// seja 1(x) -> soma de u(d), para cada divisor d de x
// 1(1) = 1
// 1(x) = 0, x > 1

// problemas iniciais:
// https://vjudge.net/problem/AtCoder-abc162_e
// https://vjudge.net/problem/CodeChef-SMPLSUM
```

14.25 mobius2

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200001
#define mod 998244353

int lpf[MAXN];
int mobius[MAXN];
int mp[MAXN];
vector<int> d[MAXN];

void calc_lpf()
{
  for (int i = 2; i < MAXN; i++)
  {
    if (!lpf[i])
    {
      for (int j = i; j < MAXN; j += i)
      {
        if (!lpf[j])
          lpf[j] = i;
      }
    }
  }
```

```
}
void calc()
{
  for (int i = 2; i < MAXN; i++) // divisores
  {
    for (int j = i; j < MAXN; j += i)
      d[j].pb(i);
  }
  calc_lpf();
  mobius[1] = 1;
  for (int i = 2; i < MAXN; i++)
  {
    if (lpf[i / lpf[i]] == lpf[i])
      mobius[i] = 0;
    else
      mobius[i] = -1 * mobius[i / lpf[i]];
  }
}
void add(int x, int dd) // adiciona dd em todos os val[i] que gcd(x, i) > 1
{
  for (auto const &i : d[x])
    mp[i] += dd;
}
int sum(int x) // valor de val[x]
{
  int ans = 0;
  for (auto const &i : d[x])
    ans += (mobius[i] * -1 * mp[i]);
  return ans;
}
signed main()
{
  return 0;
}
// mobius/inclusao-exclusao com os fatores primos
// a funcao de mobius eh definida como:
// mi(n) = 1, se n e um square-free com um numero par de fatores primos
// mi(n) = -1, se n e um square-free com um numero impar de fatores primos
// mi(n) = 0, caso nenhum dos dois
// square-free = nenhum fator primo aparece duas vezes ou mais

// dai pra problemas que da pra se fazer com inclusao-exclusao nos fatores
    primos
// tambem sai com mobius
```

14.26 modular arithmetic

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

struct modint
{
  int val;
  modint(int v = 0) { val = v % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};

modint f[MAXN];
modint inv[MAXN];
modint invfat[MAXN];

void calc()
{
  f[0] = 1;
  for (int i = 1; i < MAXN; i++)
  {
    f[i] = f[i - 1] * i;
  }
  inv[1] = 1;
  for (int i = 2; i < MAXN; ++i)
  {
    int val = mod / i;
    val = (inv[mod % i] * val) % mod;
    val = mod - val;
    inv[i] = val;
  }
  invfat[0] = 1;
  invfat[MAXN - 1] = modint(f[MAXN - 1]).inv();
  for (int i = MAXN - 2; i >= 1; i--)
  {
    invfat[i] = invfat[i + 1] * (i + 1);
  }
}
modint ncr(int n, int k) // combinacao
{
  modint ans = f[n] * invfat[k];
  ans *= invfat[n - k];
  return ans;
}
modint arr(int n, int k) // arranjo
{
  modint ans = f[n] * invfat[n - k];
  return ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
```

14.27 ntt

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
```

```
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 250005
#define mod 998244353

struct modint
{
  int val;
  modint(int v = 0) { val = ((v % mod) + mod) % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};
namespace fft
{
  // para o modulo ser valido
  // precisa ser primo
  // precisa possuir a forma c * 2^k + 1
  // 998244353 - possui a forma - c * 2^k + 1 e eh primo
  int n;
  int root = -1;
  int root_1 = -1;
  int pw = __builtin_ctz(mod - 1);
  int root_pw = (1 << pw);

  void find_root()
  {
    if (root != -1)
      return;
    int r = 2;
    while (!(modint(r).pow((1 << pw)) == 1 && modint(r).pow((1 << (pw - 1))) !=
         1))
      r++;
    root = r;
    root_1 = modint(root).inv();
  }
  void ntt(vector<modint> &a, bool invert)
  {
    find_root();
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++)
    {
      int bit = n >> 1;
      for (; j & bit; bit >>= 1)
        j ^= bit;
      j ^= bit;
      if (i < j)
        swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1)
    {
      modint wlen = (invert) ? root_1 : root;
      for (int i = len; i < root_pw; i <<= 1)
        wlen *= wlen;
      for (int i = 0; i < n; i += len)
      {
        modint w = 1;
        for (int j = 0; j < len / 2; j++)
        {
          modint u = a[i + j];
          modint v = a[i + j + len / 2] * w;
          a[i + j] = u + v;
          a[i + j + len / 2] = u - v;
          w *= wlen;
        }
      }
    }
    if (invert)
    {
      modint n_1 = modint(n).inv();
      for (int i = 0; i < a.size(); i++)
        a[i] *= n_1;
    }
  }
  vector<modint> mul(vector<modint> a, vector<modint> b)
  {
    n = 1;
    while (n < 2 * max(a.size(), b.size()))
      n <<= 1;
    a.resize(n);
    b.resize(n);
    ntt(a, false);
    ntt(b, false);
    for (int i = 0; i < n; i++)
      a[i] *= b[i];
    ntt(a, true);
    return a;
  }
} // namespace fft
  // https://codeforces.com/contest/1613/problem/F
```

14.28 operadores binarios

```
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pd pair<double, int>
#define pii pair<int, pi>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 200001
#define mod 1000000007

void shifts ()
{
  bitset <4> bs;
  bs.reset();
  bs[2] = true;
  bs[3] = true;
  cout << bs << endl ; // 1100
  bs >>= 1; // 0110
  bs <<= 1; // 1100
  bs >>= 2; // 0011
  bs <<= 2; // 1100
  bs >>= 3; // 0001
  bs <<= 3; // 1000
  cout << bs << endl ;
}
```

```
void op_xor ()
{
  // 0 ^ 0 = 0
  // 0 ^ 1 = 1
  // 1 ^ 0 = 1
  // 1 ^ 1 = 0
  bitset <4> bs , bs2;
  bs.reset();
  bs2.reset();
  bs[2] = true;
  bs[3] = true;
  bs2[1] = true;
  bs2[3] = true;
  bs ^= bs2; // bs = bs ^ bs2
  cout << bs.count() << endl ;
}
void op_and ()
{
  // 0 & 0 = 0
  // 0 & 1 = 0
  // 1 & 0 = 0
  // 1 & 1 = 1
  bitset <4> bs , bs2;
  bs.reset();
  bs2.reset();
  bs[2] = true;
  bs[3] = true;
  bs2[1] = true;
  bs2[3] = true;
  bs &= bs2; // bs = bs & bs2
  cout << bs.count() << endl ;
}
void op_or ()
{
  // 0 | 0 = 0
  // 0 | 1 = 1
  // 1 | 0 = 1
  // 1 | 1 = 1
  bitset <4> bs , bs2;
  bs.reset(); // poe tudo 0
  bs2.reset();
  bs[2] = true;
  bs[3] = true;
  bs2[1] = true;
  bs2[3] = true;
  bs |= bs2; // bs = bs | bs2
  cout << bs.count() << endl ; // quantidade de 1
}
signed main()
{
  op_or();
  op_and();
  op_xor();
  shifts();
  return 0;
}
```

14.29 pollard rho

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int __int128
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000001
#define mod 998244353

int read() // __int128 functions
{
  int x = 0, f = 1;
  char ch = getchar();
  while (ch < '0' || ch > '9')
  {
    if (ch == '-')
      f = -1;
    ch = getchar();
  }
  while (ch >= '0' && ch <= '9')
  {
    x = x * 10 + ch - '0';
    ch = getchar();
  }
  return x * f;
}
void print(__int128 x) // __int128 functions
{
  if (x < 0)
  {
    cout << "-";
    x = -x;
  }
  stack<char> s;
  while (x)
  {
    s.push((x % 10) + '0');
    x = x / 10;
  }
  while (!s.empty())
  {
    cout << s.top();
    s.pop();
  }
}
namespace pollard_rho
{
  int multiplicate(int x, int y, int m)
  {
    return (x * y) % m;
  }
  int modpow(int x, int y, int m)
  {
    int z = 1;
    while (y)
    {
      if (y & 1)
        z = (z * x) % m;
      x = (x * x) % m;
      y >>= 1;
    }
    return z;
  }
  bool is_composite(int n, int a, int d, int s)
  {
    int x = modpow(a, d, n);
    if (x == 1 or x == n - 1)
      return false;
    for (int r = 1; r < s; r++)
    {
      x = multiplicate(x, x, n);
      if (x == n - 1LL)
        return false;
    }
    return true;
  };
  int miller_rabin(int n)
  {
    if (n < 2)
      return false;
    int r = 0, d = n - 1LL;
    while ((d & 1LL) == 0)
    {
```

```
      d >>= 1;
      r++;
    }
    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37})
    {
      if (n == a)
        return true;
      if (is_composite(n, a, d, r))
        return false;
    }
    return true;
  }
  int f(int x, int m)
  {
    return multiplicate(x, x, m) + 1;
  }
  int rho(int n)
  {
    int x0 = 1, t = 0, prd = 2;
    int x = 0, y = 0, q;
    while (t % 40 || __gcd(prd, n) == 1)
    {
      if (x == y)
      {
        x0++;
        x = x0;
        y = f(x, n);
      }
      q = multiplicate(prd, max(x, y) - min(x, y), n);
      if (q != 0)
        prd = q;
      x = f(x, n);
      y = f(y, n);
      y = f(y, n);
      t++;
    }
    return __gcd(prd, n);
  }
  vector<int> fact(int n)
  {
    if (n == 1)
      return {};
    if (miller_rabin(n))
      return {n};
    int x = rho(n);
    auto l = fact(x), r = fact(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
  }
}
signed main()
{
  //ios_base::sync_with_stdio(false);
  //cin.tie(NULL);
  while (1)
  {
    int n = read();
    if (n == 0)
      break;
    vector<int> factors = pollard_rho::fact(n);
    sort(factors.begin(), factors.end());
    int prev = -1, cnt = 0;
    for (auto const &i : factors)
    {
      if (prev != i)
      {
        if (prev != -1)
        {
          print(prev);
          printf("^");
          print(cnt);
          printf(" ");
        }
        prev = i;
        cnt = 0;
      }
      cnt++;
    }
    if (prev != -1)
    {
      print(prev);
      printf("^");
      print(cnt);
      printf(" ");
    }
    printf("\n");
  }
  return 0;
}
// sources:
// https://github.com/PauloMiranda98/Competitive-Programming-Notebook/blob/
    master/code/math/prime.h
// https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/Matematica/
    pollardrho.cpp
// fast integer factorization with pollard-rho
// https://www.spoj.com/problems/FACT0/ - ok
// https://www.spoj.com/problems/FACT1/ - ok
// https://www.spoj.com/problems/FACT2/ - sigkill
// since the limit is at most 29 digits(in FACT2), we need to use __int128
```

14.30 primefactors

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define mp make_pair
#define pi pair<int, int>
#define fir first
#define sec second
#define MAXN 501
#define MAXL 20
#define mod 1000000007

vector<int> facts;
void primefactors(int n)
{
  while (n % 2 == 0)
  {
    facts.pb(2);
    n = n / 2;
  }
  for (int i = 3; i <= sqrt(n); i += 2)
  {
    while (n % i == 0)
    {
      facts.pb(i);
      n = n / i;
    }
  }
  if (n > 2)
    facts.pb(n);
}
signed main()
{
  int n;
  cin >> n;
  primefactors(n);
  sort(facts.begin(), facts.end());
  for (auto const &i : facts)
    cout << i << endl;
  return 0;
}
```

14.31 primefactors2

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
```

```
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 1000001
#define mod 1000000007

namespace primefactors
{
  bitset<MAXN> prime;
  vector<int> nxt(MAXN);
  vector<int> factors;

  void crivo()
  {
    prime.set();
    prime[0] = false, prime[1] = false;
    for (int i = 2; i < MAXN; i++)
    {
      if (prime[i])
      {
        nxt[i] = i;
        for (int j = 2; j * i < MAXN; j++)
        {
          prime[j * i] = false;
          nxt[j * i] = i;
        }
      }
    }
  }
  void fact(int n)
  {
    factors.clear();
    while (n > 1)
    {
      factors.pb(nxt[n]);
      n = n / nxt[n];
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
```

14.32 segmentedsieve

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 1000003
#define mod 1000000007

vector<int> prime;

void segmentedsieve(int l, int r)
{
  int lim = sqrt(r);
  vector<bool> mark(lim + 1, false);
  vector<int> primes;
  for (int i = 2; i <= lim; ++i)
  {
    if (!mark[i])
    {
      primes.pb(i);
      for (int j = i * i; j <= lim; j += i)
        mark[j] = true;
    }
  }
  vector<bool> isprime(r - l + 1, true);
  for (int i : primes)
    for (int j = max(i * i, (l + i - 1) / i * i); j <= r; j += i)
      isprime[j - l] = false;
  if (l == 1)
    isprime[0] = false;
  for (int i = 0; i < isprime.size(); i++)
    if (isprime[i])
      prime.pb(i + l);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int l, r;
  cin >> l >> r;
  segmentedsieve(l, r);
  for (auto const &i : prime)
    cout << i << " ";
  return 0;
}
```

14.33 simplex

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define int long long int
#define pi pair<int, int>
#define fir first
#define sec second
#define mod 2147483647
#define pb push_back
#define double long double

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

const double eps = 1e-8;
const double inf = 1e18;
#define MP make_pair
#define ltj(X)                                          \
  if (s == -1 || MP(X[j], N[j]) < MP(X[s], N[s])) \
  s = j

// resolve um problema de programacao linear para maximizar uma funcao  c[0] * x
    [0] + c[1] * x[1] + ... <= b (c[i] eh o coeficiente do i-esimo cara na
    funcao objetiva)
```

```cpp
// sujeito a restricoes que tem a seguinte forma:
// a[0] * x[0] + a[1] * x[1] + ... <= b, (a[i] eh o coeficiente)
// ai todas as restricoes sao passadas nos vectors a e b
// complexidade: 2^n, mas na pratica pode ser melhor do que isso
struct lp_solver
{
  int m, n;
  vector<int> N, B;
  vector<vector<double>> D;

  lp_solver(const vector<vector<double>> &A, const vector<double> &b, const
       vector<double> &c) : m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2,
       vector<double>(n + 2))
  {
    for (int i = 0; i < m; i++)
    {
      for (int j = 0; j < n; j++)
        D[i][j] = A[i][j];
    }
    for (int i = 0; i < m; i++)
    {
      B[i] = n + i;
      D[i][n] = -1;
      D[i][n + 1] = b[i];
    }
    for (int j = 0; j < n; j++)
    {
      N[j] = j;
      D[m][j] = -c[j];
    }
    N[n] = -1;
    D[m + 1][n] = 1;
  }
  void pivot(int r, int s)
  {
    double *a = D[r].data(), inv = 1 / a[s];
    for (int i = 0; i < m + 2; i++)
    {
      if (i != r && abs(D[i][s]) > eps)
      {
        double *b = D[i].data(), inv2 = b[s] * inv;
        for (int j = 0; j < n + 2; j++)
          b[j] -= a[j] * inv2;
        b[s] = a[s] * inv2;
      }
    }
    for (int j = 0; j < n + 2; j++)
      if (j != s)
        D[r][j] *= inv;
    for (int i = 0; i < m + 2; i++)
      if (i != r)
        D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }
  bool simplex(int phase)
  {
    int x = m + phase - 1;
    for (;;)
    {
      int s = -1;
      for (int j = 0; j < n + 1; j++)
      {
        if (N[j] != -phase)
          ltj(D[x]);
      }
      if (D[x][s] >= -eps)
        return true;
      int r = -1;
      for (int i = 0; i < m; i++)
      {
        if (D[i][s] <= eps)
          continue;
        if (r == -1 || MP(D[i][n + 1] / D[i][s], B[i]) < MP(D[r][n + 1] / D[r][s
             ], B[r]))
          r = i;
      }
      if (r == -1)
        return false;
      pivot(r, s);
    }
  }
  double solve()
  {
    int r = 0;
    for (int i = 1; i < m; i++)
    {
      if (D[i][n + 1] < D[r][n + 1])
        r = i;
    }
    if (D[r][n + 1] < -eps)
    {
      pivot(r, n);
      if (!simplex(2) || D[m + 1][n + 1] < -eps)
        return -inf;
      for (int i = 0; i < m; i++)
      {
        if (B[i] == -1)
        {
          int s = 0;
          for (int j = 1; j < n + 1; j++)
            ltj(D[i]);
          pivot(i, s);
        }
      }
    }
    bool ok = simplex(1);
    vector<double> x = vector<double>(n); // os valores escolhidos pra cada x[i]
         (se quiser eles tbm, so retornar)
    for (int i = 0; i < m; i++)
    {
      if (B[i] < n)
        x[B[i]] = D[i][n + 1];
    }
    return ok ? D[m][n + 1] : inf;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int k;
  cin >> k;
  vector<double> t(k), l(k), r(k);
  for (int i = 0; i < k; i++)
    cin >> t[i] >> l[i] >> r[i];
  int q;
  cin >> q;
  while (q--)
  {
    int aa, bb;
    cin >> aa >> bb;
    int p = aa * bb;
    vector<vector<double>> a;
    vector<double> b;
    vector<double> curr(k, 1);
    a.pb(curr);
    b.pb(bb);
    curr = vector<double>(k, -1);
    a.pb(curr);
    b.pb(-bb);
    a.pb(t);
    b.pb(p);
    curr = vector<double>(k, 0);
    for (int i = 0; i < k; i++)
    {
      curr[i] = 1;
      a.pb(curr);
      b.pb(r[i]);
      curr[i] = 0;
    }
    for (int i = 0; i < k; i++)
    {
      curr[i] = -1;
      a.pb(curr);
      b.pb(-l[i]);
```

```
      curr[i] = 0;
    }
    int x = a.size();
    lp_solver l(a, b, t);
    int ans = round(l.solve());
    if (ans == p)
      cout << "yes\n";
    else
      cout << "no\n";
  }
}
// solucao pro: https://open.kattis.com/problems/joiningflows
// source: https://github.com/kth-competitive-programming/kactl/blob/main/
    content/numerical/Simplex.h

// lembrete: quando eu quero adicionar algo com <= ao inves de >=, basta
    multiplicar os dois lados por -1 :)
// TODO: escrever melhor isso tudo depois
```

14.34 stars and bars

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

struct modint
{
  int val;
  modint(int v = 0) { val = v % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z = z * x;
      x = x * x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};

modint ncr(int n, int k)
{
  // calcular combinacao para n grande
  // nesse problema n <= 10^12
  // em O(k)
  modint num = 1;
  modint den = 1;
  for (int i = 0; i < k; i++)
  {
    num = num * modint(n - i);
    den = den * modint(i + 1);
  }
  modint ans = num / den;
  return ans;
}
modint stars_and_bars(int n, int k)
{
  // para pares de inteiros n e k
  // enconte a quantidade de k-tuplas com soma == n
  // x1 + x2 + ... + xk = n
  return ncr(n + k - 1, k - 1);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, s;
  cin >> n >> s;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  modint all = stars_and_bars(s, n);
  modint to_sub = 0;
  for (int mask = 1; mask < (1 << n); mask++)
  {
    int sum = 0;
    for (int j = 0; j < n; j++)
    {
      if (mask & (1 << j))
        sum += (v[j] + 1);
    }
    if (sum <= s)
    {
      modint curr = stars_and_bars(s - sum, n);
      to_sub = (__builtin_popcount(mask) % 2) ? to_sub + curr : to_sub - curr;
    }
  }
  all = all - to_sub;
  cout << all.val << endl;
  return 0;
}
// stars and bars
// dado dois inteiros positivos n e k
// conte o numero de k-tuplas (x1, x2, .., xk) tal que
// x1 + x2 + ... + xk = n
// com x1, x2, ..., xk >= 0
// resposta = ncr(n + k - 1, k - 1)

// para k-tuplas com x1, x2, ..., xk > 0:
// resposta = ncr(n - 1, k - 1)

// problema exemplo:
// https://codeforces.com/contest/451/problem/E
// contar quantas k-tuplas com soma == n
// tal que: x[i] >= 0 e x[i] <= f[i]
// k <= 20

// solucao:
// conta tudo com stars and bars
// dai preciso subtrair todas as possibilidades invalidas (com pelo menos um i
    tal que x[i] > f[i])
// seja n(i) as possibilidades com x[i] > f[i]
// dai eu quero calcular a quantidade de elementos na uniao entre todos os n(i)
// dai da pra fzr usando a formulinha de uniao de conjuntos:
// n(A uniao B uniao C) = n(A) + n(B) + n(C) - n(A intersecao B) ... + n(A
    intersecao B intersecao C)
// itera por todos os 2^n subsets e calcula o que deve subtrair/somar com
    aqueles caras
```

14.35 totient

```
#define MAXN 100000

int phi[MAXN];

void calc()
{
```

```
  for (int i = 0; i < MAXN; i++)
    phi[i] = i;
  for (int i = 2; i < MAXN; i++)
  {
    if (phi[i] == i)
    {
      for (int j = i; j < MAXN; j += i)
        phi[j] -= phi[j] / i;
    }
  }
}
int calc_phi(int n)
{
  int ans = n;
  for (int i = 2; i * i <= n; i++)
  {
    if (n % i == 0)
    {
      while (n % i == 0)
        n /= i;
      ans -= ans / i;
    }
  }
  if (n > 1)
    ans -= ans / n;
  return ans;
}
```

14.36 xor trie

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

struct node
{
  int me, cnt, id;
  int down[2];
  node(int c = 0) : me(c)
  {
    cnt = 0;
    id = -1;
    fill(begin(down), end(down), -1);
  }
};
struct trie_xor
{
  vector<node> t;

  trie_xor()
  {
    t.resize(1);
  }
  void add(int n, int id)
  {
    int v = 0;
    t[v].cnt++;
    for (int i = 30; i >= 0; i--)
    {
      int bit = (n & (1 << i)) ? 1 : 0;
      if (t[v].down[bit] == -1)
      {
        t[v].down[bit] = t.size();
        t.emplace_back(bit);
      }
      v = t[v].down[bit];
      t[v].cnt++;
    }
    t[v].id = id;
  }
  void rem(int n, int id)
  {
    int v = 0;
    t[v].cnt--;
    for (int i = 30; i >= 0; i--)
    {
      int bit = (n & (1 << i)) ? 1 : 0;
      v = t[v].down[bit];
      t[v].cnt--;
    }
  }
  int qry(int n) // maximum xor with n
  {
    if (t[0].cnt == 0) // no element
      return -1;
    int v = 0;
    for (int i = 30; i >= 0; i--)
    {
      int bit = (n & (1 << i)) ? 0 : 1;
      int nxt = t[v].down[bit];
      if (nxt != -1 && t[nxt].cnt > 0)
        v = nxt;
      else
        v = t[v].down[bit ^ 1];
    }
    return t[v].id;
  }
};
signed main()
{
}
// alguns problemas:
// https://codeforces.com/problemset/problem/706/D
// https://codeforces.com/contest/1625/problem/D
// https://codeforces.com/contest/888/problem/G
```

15 Miscellaneous

15.1 bitmasks

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, mask;
```

```
  vector<int> masks;

  // quantidade de bits setados na mask
  cout << __builtin_popcount(mask) << endl;

  // para printar o valor do bit i
  for (int i = 0; i < n; i++)
    cout << ((mask >> i) & 1) << " ";
  cout << endl;

  // quando eh necessario percorrer todas as submasks ate (1 << n)
  // e fazer algo com todas as submasks dessa mask
  // util em problemas de dp com mask por exemplo
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < (1 << n); j++)
    {
      if ((j >> i & 1) == 0)
      {
        //alguma coisa aqui sabendo que a mask(j) eh uma submask de(j ^ 1 << i)
      }
    }
  }

  // para percorrer por todas as submasks de uma mask
  for (int s = mask; s; s = (s - 1) & mask)
  {
    // alguma coisa aqui sabendo que s eh uma submask de mask
  }

  // quando eh necessario percorrer todas as submasks ate (1 << n)
  // e fazer algo com todas as submasks dessa mask O(3^n)
  // util em problemas de dp com mask por exemplo
  for (int m = 0; m < (1 << n); m++)
  {
    for (int s = m; s; s = (s - 1) & m)
    {
      // alguma coisa aqui sabendo que mask s eh uma submask de m
    }
  }

  // comprimindo as masks de um vector baseada em uma mask qualquer
  for (int i = 0; i < masks.size(); i++)
  {
    int compressed = 0, curr_bit = 0;
    for (int j = 0; j < n; j++)
    {
      if (!(mask & (1LL << j)))
        continue;
      if (masks[i] & (1LL << j))
        compressed |= (1LL << curr_bit);
      curr_bit++;
    }
    // alguma coisa sabendo que a mask compressed eh a mask comprimida da mask
        atual
  }
  return 0;
}
```

15.2 coordinate compression

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 1000000007

void compress(vector<int> &v)
{
  vector<int> val;
  for (auto const &i : v)
    val.pb(i);
  sort(val.begin(), val.end());
  val.erase(unique(val.begin(), val.end()), val.end());
  for (auto &i : v)
    i = lower_bound(val.begin(), val.end(), i) - val.begin();
}
```

15.3 inversion count

```
// seja S = a1, a2 , ... , an
// uma inversao S e um par (i,j) com i < j e ai > aj

// Solucao O(n2) nao ideal:
//for(int i=0;i<n;i++)
//      for(int j=i+1;j<n;j++)
//              if(v[i]>v[j]) ans++;

// Em vez de trabalharmos com o vetor inteiro(n2), vamos dividir o vetor ao meio
     e trabalhar com suas metades,
// que chamaremos de u1 e u2.

// Queremos saber o valor de inv, o numero de inversoes em v. Ha tres tipos de
     inversoes (i,j)(i,j) em v:
// aquelas em que i e j estao ambos em u1, aquelas em que i e j estao ambos em
     u2 e aquelas
// em que i esta em u1 e j esta em u2.
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define pb push_back
#define in insert
#define pi pair<int, int>
#define pii pair<int, pi>
#define mp make_pair
#define fir first
#define sec second
#define MAXN 100001
#define INF 1000000000

int merge_sort(vector<int> &v)
{
  int ans = 0;

  if (v.size() == 1)
  {
    return 0;
  }

  vector<int> u1, u2;

  for (int i = 0; i < v.size() / 2; i++)
  {
    u1.pb(v[i]);
  }
  for (int i = v.size() / 2; i < v.size(); i++)
  {
    u2.pb(v[i]);
  }

  ans += merge_sort(u1);
  ans += merge_sort(u2);

  u1.pb(INF);
  u2.pb(INF);

  int ini1 = 0, ini2 = 0;
```

```
  for (int i = 0; i < v.size(); i++)
  {
    if (u1[ini1] <= u2[ini2])
    {
      v[i] = u1[ini1];
      ini1++;
    }
    else
    {
      v[i] = u2[ini2];
      ini2++;
      ans += u1.size() - ini1 - 1;
    }
  }

  return ans;
}
signed main()
{
  int n;
  cin >> n;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  cout << merge_sort(v) << endl;
  return 0;
}
```

15.4 max plus convolution

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007

bool needs[MAXN];
int dp0[MAXN];
multiset<int> dp[MAXN];

struct dsu
{
  int tot;
  vector<int> parent;
  vector<int> sz;
  vector<int> sum;

  dsu(int n)
  {
    parent.resize(n);
    sz.resize(n);
    sum.assign(n, 0);
    tot = n;
    for (int i = 0; i < n; i++)
    {
      parent[i] = i;
      sz[i] = 1;
      sum[i] = needs[i];
    }
  }
  int find_set(int i)
  {
    return parent[i] = (parent[i] == i) ? i : find_set(parent[i]);
  }
  int make_set(int x, int y)
  {
    x = find_set(x), y = find_set(y);
    if (x != y)
    {
      if (sz[x] > sz[y])
        swap(x, y);
      parent[x] = y;
      sz[y] += sz[x];
      sum[y] += sum[x];
      tot--;
    }
    return y;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n, m, p;
    cin >> n >> m >> p;
    for (int i = 0; i < n; i++)
    {
      needs[i] = 0;
      dp[i].clear();
    }
    for (int i = 0; i < p; i++)
    {
      int x;
      cin >> x;
      x--;
      needs[x] = 1;
    }
    vector<pii> edges;
    for (int i = 0; i < m; i++)
    {
      int u, v, w;
      cin >> u >> v >> w;
      u--, v--;
      edges.pb({w, {u, v}});
    }
    for (int i = 0; i < n; i++)
    {
      dp[i].insert(0);
      dp0[i] = 0;
    }
    sort(edges.begin(), edges.end());
    dsu d(n);
    for (auto const &ii : edges)
    {
      int w = ii.fir;
      auto [u, v] = ii.sec;
      if (d.find_set(u) != d.find_set(v))
      {
        u = d.find_set(u);
        v = d.find_set(v);
        if (d.sz[u] > d.sz[v])
        {
          swap(u, v);
        }
        // problema brabo e educational
        // o que funcionava ate a hard version era algo tipo isso
        //  for (int i = 0; i <= szu; i++)
        // {
        //   for (int j = 0; j <= szv; j++)
        //     new_dp[i + j] = min(new_dp[i + j], dp[u][i] + dp[v][j]);
        // }
        // isso eh uma (min, +) convolution
        // e ambas as sequencias satisfazem:
        // dp[i + 1] - dp[i] <= dp[i] - dp[i - 1]
        // dai pra fazer a ideia desse blog, descrita em (max, +) convolution
        // https://codeforces.com/blog/entry/98663
```

```
        // que eh basicamente ir "mergindo" as slopes de forma gulosa
        // entretanto a unica coisa que muda eh que em cada iteracao desse
            kruskal
        // temos que setar o valor de dp[0] para algo diferente antes de fzr a (
            min, +) convolution
        // ai tem que refletir isso no multiset das diferencas
        {
          int ini = *dp[u].begin();
          dp[u].erase(dp[u].find(ini));
          int new_dp0 = d.sum[u] * w;
          // se antes diferenca era dp[1] - dp[0]
          // eu aumentei o dp[0] por x
          // agr vai ser (dp[1] - dp[0]) - x
          int x = new_dp0 - dp0[u];
          dp[u].insert(ini - x);
          dp0[u] = new_dp0;
        }
        {
          int ini = *dp[v].begin();
          dp[v].erase(dp[v].find(ini));
          int new_dp0 = d.sum[v] * w;
          // se antes diferenca era dp[1] - dp[0]
          // eu aumentei o dp[0] por x
          // agr vai ser (dp[1] - dp[0]) - x
          int x = new_dp0 - dp0[v];
          dp[v].insert(ini - x);
          dp0[v] = new_dp0;
        }
        for (auto const &x : dp[u])
        {
          dp[v].insert(x);
        }
        dp0[v] += dp0[u];
        dp[u].clear();
        d.make_set(u, v);
      }
    }
    int par = d.find_set(0);
    int x = dp0[par];
    // como eh uma min+ convolution, vai da menor slope pra maior slope
    for (auto const &val : dp[par])
    {
      x += val;
      cout << x << " ";
    }
    cout << endl;
  }
  return 0;
}
// https://codeforces.com/contest/2021/problem/E3
```

15.5 meetinthemiddle

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 1000001

int n, t;
vector<int> v;
vector<int> a;
vector<int> b;

void solve2(int i, int j, int k)
{
  if (i == j)
  {
    b.pb(k);
    return;
  }
  solve2(i + 1, j, k);
  solve2(i + 1, j, k + v[i]);
}
void solve(int i, int j, int k)
{
  if (i == j)
  {
    a.pb(k);
    return;
  }
  solve(i + 1, j, k);
  solve(i + 1, j, k + v[i]);
}
int upper(int l, int r, int x)
{
  while (l < r)
  {
    int mid = (l + r + 1) >> 1;
    (b[mid] <= x) ? l = mid : r = mid - 1;
  }
  return b[l];
}
int meetinthemiddle()
{
  solve(0, (n >> 1) + 1, 0);
  solve2((n >> 1) + 1, n, 0);
  sort(b.begin(), b.end());
  int ans = 0;
  for (auto const &i : a)
  {
    if (i > t)
      continue;
    ans = max(ans, i);
    int kappa = i + upper(0, b.size() - 1, t - i);
    if (kappa <= t)
      ans = max(ans, kappa);
  }
  return ans;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> t;
  v.resize(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  cout << meetinthemiddle() << endl;
  return 0;
}
```

15.6 prefix sum 2d

```
// https://cses.fi/problemset/task/1652
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007
```

```
int v[1001][1001];
int p[1001][1001];

int qry(int x1, int y1, int x2, int y2)
{
  return p[x2 + 1][y2 + 1] - p[x2 + 1][y1] - p[x1][y2 + 1] + p[x1][y1];
}
signed main()
{
  int n, q;
  cin >> n >> q;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      char c;
      cin >> c;
      v[i][j] = (c == '*');
    }
  }
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      p[i + 1][j + 1] = p[i][j + 1] + p[i + 1][j] - p[i][j];
      p[i + 1][j + 1] += v[i][j];
    }
  }
  while (q--)
  {
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    a--, b--, c--, d--;
    cout << qry(a, b, c, d) << endl;
  }
  return 0;
}
// prefix sum 2d
// me enrolo pra codar toda vez, e bom deixar na lib
```

15.7 rectangle union

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200007

vector<int> x_vals;

struct segtree
{
  vector<int> seg, tag;
  segtree()
  {
    seg.assign(8 * x_vals.size(), 0);
    tag.assign(8 * x_vals.size(), 0);
  }
  void add(int ql, int qr, int x, int v, int l, int r)
  {
    if (qr <= l || r <= ql)
    {
      return;
    }
    if (ql <= l && r <= qr)
    {
      tag[v] += x;
      if (tag[v] == 0)
      {
        if (l != r)
          seg[v] = seg[v << 1] + seg[(v << 1) | 1];
        else
          seg[v] = 0;
      }
      else
      {
        seg[v] = x_vals[r] - x_vals[l];
      }
    }
    else
    {
      int mid = (l + r) >> 1;
      add(ql, qr, x, (v << 1), l, mid);
      add(ql, qr, x, ((v << 1) | 1), mid, r);
      if (tag[v] == 0 && l != r)
        seg[v] = seg[v << 1] + seg[(v << 1) | 1];
    }
  }
  int qry()
  {
    return seg[1];
  }
  void upd(int l, int r, int x)
  {
    add(l, r, x, 1, 0, x_vals.size());
  }
};
struct rect
{
  int x1, y1, x2, y2;
};
struct event
{
  int time, l, r, type;
  bool operator<(const event &b)
  {
    if (time != b.time)
      return time < b.time;
    return type > b.type;
  }
};

const int inf = 1e9;

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<rect> v(n);
  for (int i = 0; i < n; i++)
  {
    cin >> v[i].x1 >> v[i].y1 >> v[i].x2 >> v[i].y2;
    x_vals.pb(v[i].x1);
    x_vals.pb(v[i].x2);
  }
  // comprime o x
  sort(x_vals.begin(), x_vals.end());
  x_vals.erase(unique(x_vals.begin(), x_vals.end()), x_vals.end());
  vector<event> ev;
  for (int i = 0; i < n; i++)
  {
    v[i].x1 = lower_bound(x_vals.begin(), x_vals.end(), v[i].x1) - x_vals.begin
        ();
    v[i].x2 = lower_bound(x_vals.begin(), x_vals.end(), v[i].x2) - x_vals.begin
        ();
    ev.pb({v[i].y1, v[i].x1, v[i].x2, 0}); // adicao
    ev.pb({v[i].y2, v[i].x1, v[i].x2, 1}); // remocao
  }
  segtree s;
```

```
  sort(ev.begin(), ev.end());
  int area = 0, l = -inf;
  for (auto const &i : ev)
  {
    if (l == -inf)
    {
      l = i.time;
      s.upd(i.l, i.r, 1);
    }
    else if (i.type == 1)
    {
      int curr = s.qry();
      s.upd(i.l, i.r, -1);
      if (s.qry() != curr)
      {
        int new_t = (s.qry() == 0) ? -inf : i.time;
        int lo = l, hi = i.time - 1;
        area += ((hi - lo + 1) * curr);
        l = new_t;
      }
    }
    else
    {
      int curr = s.qry();
      s.upd(i.l, i.r, 1);
      if (s.qry() != curr)
      {
        int lo = l, hi = i.time - 1;
        area += ((hi - lo + 1) * curr);
        l = i.time;
      }
    }
  }
  cout << area << endl;
  return 0;
}
// area da uniao de retangulos
// comprime coordenada no x pra montar a segtree dos valores de x
// faz o line sweep pelo y

// testado em dois judges:
// https://cses.fi/problemset/task/1741/
// n <= 10^5
// -10^6 <= x, y <= 10^6

// https://judge.yosupo.jp/problem/area_of_union_of_rectangles
// n <= 5 * 10^5
// 0 <= x, y <= 10^9
```

15.8 segment covering

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 1000000007

int st[MAXN + 1][21];

void naive(vector<pi> &v) // guloso do point covering
{
  // os segmentos precisam estar ordenados pelo .second
  int n = v.size(), last = -1;
  vector<int> ans;
  for (int i = 0; i < n; i++)
  {
    if (v[i].fir > last)
    {
      ans.pb(v[i].sec);
      last = v[i].sec;
    }
  }
}
bool can(int l, int r, int x)
{
  for (int i = 20; i >= 0; i--)
  {
    if (x & (1 << i))
      l = st[l][i];
  }
  return l > r;
}
void solve(vector<pi> &v, int a, int b) // segment covering com binary lifting (
    da pra fazer point covering de forma bem similar)
{
  for (int i = 0; i <= MAXN; i++)
  {
    st[i][0] = i;
  }
  for (auto const &i : v)
  {
    st[i.fir][0] = max(st[i.fir][0], i.sec + 1);
  }
  for (int i = 1; i <= MAXN; i++) // se um segmento com l menor tem um r maior
  {
    st[i][0] = max(st[i][0], st[i - 1][0]);
  }
  for (int i = 1; i < 21; i++)
  {
    for (int v = 0; v <= MAXN; v++)
      st[v][i] = st[st[v][i - 1]][i - 1];
  }
  int lo = 1, hi = v.size();
  while (lo < hi) // busca binaria na resposta
  {
    int mid = (lo + hi) >> 1;
    (can(a, b, mid)) ? hi = mid : lo = mid + 1;
  }
  if (can(a, b, lo))
    cout << lo << endl;
  else
    cout << "-1\n";
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// um tipo de problema que eu achei bem legal
// https://codeforces.com/problemset/problem/1175/E
// https://codeforces.com/gym/101221 (Problem K)
// https://vjudge.net/contest/512192#problem/E

// problema exemplo:
// dado um conjunto de n segmentos [l[i], r[i]]
// qual o numero minimo de pontos que vc pode escolher, tal que:
// para cada segmento [l[i], r[i]], pelo menos um ponto escolhido ta nesse
    segmento
// tem a solucao gulosa em O(N)
// mas que da pra ser otimizada com binary lifting/sparse table (em caso de ter
    varias queries sobre o conjunto de segmentos)
// depois de adicionar um ponto a resposta, acho o nxt dele: o proximo ponto que
    irei colocar na resposta depois de adicionar ele

// outro problema exemplo:
// dado um conjunto de n segmentos [l[i], r[i]]
// voce quer selecionar o numero minimo de segmentos do conjunto
// para cobrir todo o segmento [a, b]
// bem parecido, tem uma solucao gulosa normal
```

```
// e se tu quer fazer varias queries, otimiza com binary lifting/sparse table
```

15.9 sprague grundy

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500009
#define mod 1000000001

vector<int> v = {2, 3, 4, 5, 6};
unordered_map<int, bool> vis;
unordered_map<int, int> dp;

int g(int x) // achar o grundy number na marra
{
    if (x == 0)
        return 0;
    vector<bool> ok(4, 0);
    int mex = 0;
    for (auto const &i : v)
    {
        int curr = g(x / i);
        if (curr < 4)
            ok[curr] = 1;
        while (ok[mex])
            mex++;
    }
    vis[x] = 1;
    return dp[x] = mex;
}
int solve(int x) // padraozin
{
    vector<int> ini = {0, 1, 2, 2, 3, 3, 0, 0, 0, 0, 0, 0};
    while (x >= 12)
        x /= 12;
    return ini[x];
}
signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int q;
    cin >> q;
    while (q--)
    {
        int n;
        cin >> n;
        int x = 0;
        for (int i = 0; i < n; i++)
        {
            int k;
            cin >> k;
            x ^= solve(k);
        }
        (x > 0) ? cout << "Henry\n" : cout << "Derek\n";
    }
    return 0;
}
/*
game theory (um exemplo simples de problema pra ficar no repo)
- pro nim classico
- existem n pilhas cada uma possui x[i] blocos
- em uma play posso escolher uma pilha e tirar uma quantidade qualquer de blocos
    dela
- quem ganha?
- o jogador que comeca ganha se o xor dos tamanhos das pilhas for != 0

- teorema sprague-grundy (transformar um jogo qualquer em nim)
- seja v um estado que eu tou do jogo, podemos calcular o grundy number desse
    estado
- seja o conjuntos de estados adjacentes a v {u1, u2, ..., un}
- g(v) = mex(g(u1), g(u2), ..., g(un))
- se v nao tem nenhum extado adjacente, entao g(v) = 0
- g(v) -> grundy number do estado v
- com isso se tivemos varios estados iniciais (varias pilhas)
- podemos simplesmente achar o grundy number de cada um deles e depois saber
    quem ganha
- pelo valor do xor dos grundy numbers

- exemplo: floor division game
- existem n numeros e em uma play posso escolher um deles e dividir por 2, 3, 4,
    5 ou 6
- quem ganha?
- achar o grundy number de cada um dos n numeros
- se o xor for != 0, ganha quem comeca jogando
- caso contrario, o outro jogador ganha

- as vzs e util tbm ver se existe um padrao (em caso de altas constantes)
- notando o padrao, da pra achar o grundy number de forma mais eficiente e
    resolver o problema
*/
```

15.10 stack trick

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define fir first
#define sec second
#define MAXN 300001
#define mod 1000000007

int n;
vector<int> v;
vector<int> ans;

void solve()
{
  stack<pi> s;
  for (int i = n - 1; i >= 0; i--)
  {
    while (!s.empty() && s.top().fir <= v[i])
      s.pop();
    (!s.empty()) ? ans[i] = s.top().sec : ans[i] = -1;
    s.push({v[i], i});
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n;
  v.resize(n);
  ans.resize(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  solve();
  for (auto const &i : ans)
    cout << i << " ";
  cout << endl;
}
```

```
// WITHOUT SEGMENT TREE
// for each index (0 <= i < n), find another index (0 <= j < n)
// which v[j] > v[i] and j > i and j is as close as possible to i.
// if this index does not exist, print -1

/*
5
1 3 3 4 5
*/
/*
1 3 3 4 -1
*/
```

15.11 sum hash

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 300005
#define mod 998244353

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
  int n, k;
  cin >> n >> k;
  vector<int> h(k + 1, 0);
  for (int i = 1; i < k; i++)
  {
    h[i] = rng();
    h[k] -= h[i];
  }
  vector<int> v(n);
  int sum = 0, ans = 0;
  map<int, int> mp;
  mp[0] = 0;
  for (int i = 0; i < n; i++)
  {
    cin >> v[i];
    sum += h[v[i]];
    if (mp.find(sum) != mp.end())
      ans = max(ans, i - mp[sum] + 1);
    else
      mp[sum] = i + 1;
  }
  cout << ans << endl;
}
// solucao pra C da final brasileira da maratona de 2023
// dado um array com n inteiros, cada a[i] ta entre 1 e k
// qual o maior tamanho de um subarray no qual todos os numeros de 1 ate k
// tem a mesma frequencia nesse subarray
```

15.12 tower of hanoi

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

// #define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 998244353

vector<pair<char, char>> ans;

void solve(int n, char a, char b, char c)
{
  if (n == 0)
    return;
  solve(n - 1, a, c, b);
  ans.pb({a, b});
  solve(n - 1, c, b, a);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, k;
  cin >> n >> k;
  solve(n, 'A', 'C', 'B');
  if (ans.size() > k)
  {
    cout << "N\n";
    return 0;
  }
  cout << "Y\n";
  k -= ans.size();
  if (k % 2 == 0)
  {
    for (int i = 0; i < (k / 2); i++)
    {
      cout << "A B\n";
      cout << "B A\n";
    }
  }
  else
  {
    for (int i = 0; i < (k / 2) - 1; i++)
    {
      cout << "A B\n";
      cout << "B A\n";
    }
    cout << "A B\n";
    cout << "B C\n";
    cout << "C A\n";
  }
  for (auto const &i : ans)
    cout << i.fir << " " << i.sec << endl;
  return 0;
}
// torre de hanoi
// 3 pilhas, sendo uma pilha com n discos e as outras duas pilhas vazias
// em cada movimento, vc tira o disco do topo de uma pilha e poe no topo de
     outra pilha
// desde que o raio do disco seja menor do que o raio do disco que ta no topo da
     outra pilha
// os n discos tem raios distintos aos pares
// fazer com que todos os discos vao parar em outra pilha

// https://codeforces.com/gym/101879/problem/I
// resolver a torre de hanoi com k movimentos
// se for possivel resolver, printar os movimentos feitos

// numero minimo pra resolver pros primeiros n
// 1, 3, 7, 15, 31, 63, 127, 255
```

```
// f(n) = 2^n - 1
```

15.13 two pointers

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 3005
#define mod 1000000007

const int inf = LLONG_MAX;
stack<pii> s[2];

void add(int x, int i)
{
  int mn = inf, mx = -inf;
  if (!s[i].empty())
  {
    mn = min(mn, s[i].top().sec.fir);
    mx = max(mx, s[i].top().sec.sec);
  }
  mn = min(mn, x);
  mx = max(mx, x);
  s[i].push({x, {mn, mx}});
}
void change()
{
  while (!s[1].empty())
  {
    int x = s[1].top().fir;
    s[1].pop();
    add(x, 0);
  }
}
void rem()
{
  if (!s[0].size())
    change();
  s[0].pop();
}
int q()
{
  int mn = inf, mx = -inf;
  for (int i = 0; i < 2; i++)
  {
    if (!s[i].empty())
    {
      mn = min(mn, s[i].top().sec.fir);
      mx = max(mx, s[i].top().sec.sec);
    }
  }
  if (mn == inf)
    return 0;
  return mx - mn;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, k;
  cin >> n >> k;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  int ans = 0, i = 0;
  for (int j = 0; j < n; j++)
  {
    add(v[j], 1);
    while (q() > k)
    {
      rem();
      i++;
    }
    ans += (j - i + 1);
  }
  cout << ans << endl;
  return 0;
}
// https://codeforces.com/edu/course/2/lesson/9/2/practice/contest/307093/
     problem/F
// Given an array of n integers, Let's say that a segment of this array is good
// if the difference between the maximum and minimum elements on this segment is
     at most k
// Your task is to find the number of different good segments
// amazing trick using stack
```

16 STL

16.1 ordered set

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG false
#define MAXN 200002

template <class T> // template do ordered set
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  ordered_set<int> s; // ordered_set
  s.insert(1);
  s.insert(1);
  s.insert(2);
  s.insert(4);
  s.insert(3);
  for (auto const &i : s) // nao adiciona elementos repetidos, que nem o set
       normal
    cout << i << " ";
  cout << endl;
  cout << *(s.find_by_order(0)) << endl; // iterator do elemento 0
  cout << *(s.find_by_order(1)) << endl; // iterator do elemento 1
  cout << s.order_of_key(4) << endl;     // quantidade de elementos que sao
       menores do que 4
  cout << s.order_of_key(6) << endl;     // quantidade de elementos que sao
       menores do que 4
}
// find_by_order : O(log n), retorna (um iterator) qual o k-esimo elemento do
     set
// order_of_key: O(log n), retorna qual a quantidade de elementos menores do que
     x no set
```

16.2 STL

```
1)Vector

vector <int> v; //Criacao o vector

v.push_back(10); //Adiciono o elemento 10 no final do vector v
v.size() // retorna o tamanho do vector
v.resize(10); //Muda o tamanho do vector v para 10.
v.pop_back(); //Apaga o ultimo elemento do vector V.
v.clear(); // apaga todos os elementos do vector v .
sort(v.begin(), v.end()); //Ordena todo o vector v


2)Pair

pair <string, int> p; // criando a pair relacionando um first com um second

p.first = "Joao"; // adicionando elementos
p.second = 8 ; //adicionando elementos

// utilidade: vector de pair
vector< pair <int, string> > v; // criando o vector v de pair
v.push_back(make_pair(a,b)); // dando push back em uma pair no vector usando
     make_pair
sort(v.begin(), v.end()); // tambem e possivel ordenar o vector de pair

3)Queue / FIla

queue <int> f; // criando a queque

f.push(10); // adiciona alguem na fila
f.pop(); // remove o elemento que esta na frente da fila
f.front(); // olha qual o elemento esta na frete da fila
f.empty() // retorna true se a fila estiver vazia e false se nao estiver vazia


4)Stack / Pilha

stack <int> p ; // criando a stack

pilha.push(x); //Adiciona o elemento x no topo da pilha
pilha.pop(); //Remove elemento do topo da pilha
pilha.top(); // retorna o elemento do topo da pilha
pilha.empty(); // verifica se a pilha esta vazia ou nao

5) Set

set <int> s ; // criando a set
// obs: a set nao adiciona elementos repetidos

s.insert(10); //Adiciona o elemento 10 no set
s.find(10) // Para realizar uma busca no set utilizamos o comando find,
o find retorna um ponteiro que aponta para o elemento procurado caso o elemento
     esteja no set ou para o final do set, caso o elemento procurado nao esteja
     no set , em complexidade O(log n)

if(s.find(10) != s.end()) // procurando pelo 10, se ele estiver no set
s.erase(10); //Apaga o elemento 10 do set em O(log n)
s.clear(); // Apaga todos os elementos
s.size(); //  Retorna a quantidade de elementos
s.begin(); // Retorna um ponteiro para o inicio do set
s.end(); // Retorna um ponteiro para o final do set


6)Map
map <string, int> m; //Cria uma variavel do tipo map que mapeia strings em int
// Em um map cada elemento esta diretamente ligado a um valor, ou seja, cada
     elemento armazenado no map possui um valor correspondente
// Se tivermos um map de strings em inteiros e inserimos os pair ("Joao", 1), ("
     Alana", 10), ("Rodrigo", 9)
// Caso facamos uma busca pela chave "Alana" receberemos o numero 10 como
     retorno.

m.insert(make_pair("Alana", 10)); //Inserimos uma variavel do tipo pair
     diretamente no map, O(log n)
M["Alana"] = 10; //Relacionando o valor 10 a chave "Alana"
if(m.find("Alana") != m.end()){ //Se a chave "Alana" foi inserida no map
cout << m["Alana"] << endl; //Imprime o valor correspondente a chave "Alana", no
      caso, o valor 10.
m.erase("Alana"); //Apaga o elemento que possui a chave "Alana" do map
m.clear(); // Apaga todos os elementos
m.size(); //  Retorna a quantidade de elementos
m.begin(); // Retorna um ponteiro para o inicio do map
m.end(); // Retorna um ponteiro para o final do map


7)Priority Queue
priority_queue <int> q; // declarando a priority queue
// Para utilizar a priority_queue do C++ e importante apenas saber que o maior
     elemento sempre estara na primeiro posicao.
// Com execao disso, todos os outros metodos sao semelhantes ao uso de uma queue
      comum, porem para manter a estrutura organizada, a complexidade da
     operacao de insercao e O(logn).
p.push(i) // adiciono o elemento i na priority_queue
p.pop(); // apago o primeiro da fila
p.top(); // vejo quem esta no topo
```

17 Strings

17.1 aho corasick

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 5001
#define mod 1000000007

namespace aho
{
  int go(int v, char ch);
  const int K = 26; // tamanho do alfabeto
  struct trie
  {
    char me;              // char correspondente ao no atual
    int go[K];            // proximo vertice que eu devo ir estando em um estado (
         v, c)
    int down[K];          // proximo vertice da trie
    int is_leaf = 0;      // se o vertice atual da trie eh uma folha (fim de uma
         ou mais strings)
    int parent = -1;      // no ancestral do no atual
    int link = -1;        // link de sufixo do no atual (outro no com o maior
         matching de sufixo)
    int exit_link = -1; // folha mais proxima que pode ser alcancada a partir de
          v usando links de sufixo
    trie(int p = -1, char ch = '$') : parent(p), me(ch)
    {
      fill(begin(go), end(go), -1);
      fill(begin(down), end(down), -1);
    }
  };
  vector<trie> ac;
  void init() // criar a raiz da trie
  {
    ac.resize(1);
  }
```

```
  void add_string(string s) // adicionar string na trie
  {
    int v = 0;
    for (auto const &ch : s)
    {
      int c = ch - 'a';
      if (ac[v].down[c] == -1)
      {
        ac[v].down[c] = ac.size();
        ac.emplace_back(v, ch);
      }
      v = ac[v].down[c];
    }
    ac[v].is_leaf++;
  }
  int get_link(int v) // pegar o suffix link saindo de v
  {
    if (ac[v].link == -1)
      ac[v].link = (!v || !ac[v].parent) ? 0 : go(get_link(ac[v].parent), ac[v].
          me);
    return ac[v].link;
  }
  int go(int v, char ch) // proximo estado saindo do estado(v, ch)
  {
    int c = ch - 'a';
    if (ac[v].go[c] == -1)
    {
      if (ac[v].down[c] != -1)
        ac[v].go[c] = ac[v].down[c];
      else
        ac[v].go[c] = (!v) ? 0 : go(get_link(v), ch);
    }
    return ac[v].go[c];
  }
  int get_exit_link(int v) // suffix link mais proximo de v que seja uma folha
  {
    if (ac[v].exit_link == -1)
    {
      int curr = get_link(v);
      if (!v || !curr)
        ac[v].exit_link = 0;
      else if (ac[curr].is_leaf)
        ac[v].exit_link = curr;
      else
        ac[v].exit_link = get_exit_link(curr);
    }
    return ac[v].exit_link;
  }
  int query(string s) // query O(n + ans)
  {
    int ans = 0, curr = 0, at;
    for (auto const &i : s)
    {
      curr = go(curr, i);
      ans += ac[curr].is_leaf;
      at = get_exit_link(curr);
      while (at)
      {
        ans += ac[at].is_leaf;
        at = get_exit_link(at);
      }
    }
    return ans;
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  aho::init();
  for (int i = 0; i < n; i++)
  {
    string s;
    cin >> s;
    aho::add_string(s);
  }
  while (q--)
  {
    string t;
    cin >> t;
    cout << aho::query(t) << endl;
  }
  return 0;
}
// automato de aho-corasick
// imagine o seguinte problema:
// temos um conjunto de n strings
// e q queries para processar
// em cada uma das q queries, voce recebe uma string s
// e quer saber, o numero de ocorrencias de
// alguma string do conjunto como
// substring de s e em tempo linear
```

17.2 de bruijn

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 1000000009

int n, m, k, sz;
string ans, ss, path;
vector<int> d;
set<string> st;

void dfs(string s)
{
  if (ans.size() + path.size() == sz) // a sagacidade aqui
  {
    ans += path;
    cout << ans << endl;
    exit(0);
  }
  for (auto const &i : d)
  {
    string t = s;
    t.pb('0' + i);
    if (!st.count(t))
    {
      st.insert(t);
      string nxt = t.substr(1);
      path.pb('0' + i);
      dfs(nxt);
      path.pop_back();
      ans.pb('0' + i);
      if (ans.size() == sz)
      {
        cout << ans << endl;
        exit(0);
      }
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
```

```
  cin.tie(NULL);
  srand(time(NULL));
  cin >> n >> m >> k;
  d.resize(m);
  for (int i = 0; i < m; i++)
  {
    cin >> d[i];
  }
  sz = n + k - 1;
  if (n >= 40) // n grande -> a probabilidade de colisao eh muito baixa
  {
    string s;
    for (int i = 0; i < sz; i++)
    {
      char c = '0' + d[rand() % m];
      s.pb(c);
    }
    cout << s << endl; // vai uma string gerada no random e gg
    return 0;
  }
  // n pequeno -> vamo achar um caminho euleriano
  for (int i = 1; i < n; i++)
  {
    ss.pb('0' + d[0]);
  }
  dfs(ss);
  ans += ss;
  while (ans.size() > sz)
    ans.pop_back();
  cout << ans << endl;
  return 0;
}
// vou escrever pq achei mto dahora esse problema
// https://codeforces.com/gym/102001/problem/C

// o problema basicamente eh:
// ache uma string s, minimizando o tamanho dessa string
// tal que ela tem k substrings distintas de tamanho n

// ai vai ser tipo:
// achar uma string na qual todas as substrings de tamanho n sao distintas

// alem disso, o alfabeto contem m letras
// essa string vai ter comprimento n + k - 1

// essa string eh chamada de de Bruijn sequence pro caso de k = m^n
// dai o que queremos eh basicamente achar um prefixo de uma de Bruijn sequence,
     pro k < m^n

// dai da pra transformar em um problema de achar um caminho euleriano num grafo
     direcionado
// montar um grafo no qual os vertices sao strings de tamanho n - 1
// e existe uma aresta direcionada u -> v se:
// v pode ser obtida adicionando um char no final de u, e tirando o primeiro
     char de u
```

17.3 kmp

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 100005
#define mod 998244353

string s;
int n, m;
string a, b;
int c[MAXN][26];

vector<int> kmp(string &s)
{
  int n = s.size();
  vector<int> p(n);
  for (int i = 1; i < n; i++)
  {
    int j = p[i - 1];
    while (j > 0 && s[i] != s[j])
      j = p[j - 1];
    if (s[i] == s[j])
      j++;
    p[i] = j;
  }
  return p;
}
void compute(string s)
{
  s.pb('*');
  vector<int> p = kmp(s);
  for (int i = 0; i < s.size(); i++)
  {
    for (int cc = 0; cc < 26; cc++)
    {
      int j = i;
      while (j > 0 && 'a' + cc != s[j])
        j = p[j - 1];
      if ('a' + cc == s[j])
        j++;
      c[i][cc] = j;
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s;
  cin >> s;
  compute(s);
  return 0;
}
// kmp
// algoritmo eh online, vai coonstruindo da esquerda pra direita
// calcula pi[i], a seguinte funcao:
// seja a substring s.substr(0, i + 1)
// pi[i] = tamanho do maior prefixo que tbm eh um sufixo dessa substring

// dai por exemplo
// da pra contar a quantidade de matchings de s em t
// so concatenar as strings fazendo: t = s + "*" + t
// dai contar as posicoes com pi[i] = s.size()

// tambem eh possivel construir um automato do kmp
// do tipo
// se meu pi[i] == x, e leio a letra c
// dai devo ir pro estado p[i] == y
// as transicoes podem ser computadas e isso pode ser muito util
```

17.4 manacher

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define fir first
```

```cpp
#define sec second
#define MAXN 100001
#define mod 1000000007

vector<int> d1;
vector<int> d2;

void manacher(string s)
{
  d1.resize(s.size());
  d2.resize(s.size());
  int l = 0, r = -1;
  for (int i = 0; i < s.size(); i++)
  {
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
    while (0 <= i - k && i + k < s.size() && s[i - k] == s[i + k])
      k++;
    d1[i] = k;
    k = k - 1;
    if (i + k > r)
      l = i - k, r = i + k;
  }
  l = 0, r = -1;
  for (int i = 0; i < s.size(); i++)
  {
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < s.size() && s[i - k - 1] == s[i + k])
      k++;
    d2[i] = k;
    k = k - 1;
    if (i + k > r)
      l = i - k - 1, r = i + k;
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s;
  cin >> s;
  manacher(s);
  return 0;
}

// algoritimo de manacher

// motivacao: dada uma string s, encontre todos os pares (l, r) tal que, a
     substring s[l,r]
// e palindroma.

// para cada posicao (0 <= i < s.size()), vamos encontrar os valores de d1[i] e
     d2[i],
// sendo estes o numero de palindromos com comprimentos impares e com
     comprimentos pares
// e com i sendo a posicao central desses palindromos

// algoritimo mais facil:
// para cada posicao (0 <= i < s.size()), ele tenta aumentar a resposta em 1
// ate q nao seja mais possivel
// while(s[i - curr] == s[i + curr])
// complexidade O(N^2)

// algoritimo de manacher:
// para cada posicao (0 <= i < s.size()):
// seja o par (l, r) os extremos da substring palindroma que possui o maior r
     entre todas as encontradas ate entao
// se i > r, o fim do ultimo palindromo foi antes de i: iremos rodar o
     algoritimo mais facil mais facil e ir ate o limite.
// caso contrario, so precisamos rodar o algoritimo a partir de onde nao foi
     percorrido previamente.
// ao final se o r atual e maior do que o nosso antigo r, atualizamos o par (l,
     r)
// por incrivel que pareca, a complexidade e O(N)

// voltando para a motivacao:
// se temos os valores de d1[i] e d2[i]:
// a substring s[i - k, i + k] e palindroma, para todo (0 <= k < d1[i])
// a substring s[i - k - 1, i + k] e palindroma, para todo (0 <= k < d2[i])
// dai temos todos os intervalos
// note que a complexidade do algoritimo de manacher e O(N),
// mas como a quantidade maxima de palindromos em uma string e n^2,
// imprimir todos os intervalos consequentemente teria complexidade O(N^2) no
     pior caso
```

17.5 min suffix

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class string>
using ordered_set = tree<string, null_type, less<string>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 2001
#define mod 1000000007

int max_suffix(string s, bool mi = false)
{
  s.push_back(*min_element(s.begin(), s.end()) - 1);
  int ans = 0;
  for (int i = 1; i < s.size(); i++)
  {
    int j = 0;
    while (ans + j < i and s[i + j] == s[ans + j])
      j++;
    if (s[i + j] > s[ans + j])
    {
      if (!mi or i != s.size() - 2)
        ans = i;
    }
    else if (j)
      i += j - 1;
  }
  return ans;
}
int min_suffix(string s)
{
  for (auto &i : s)
    i *= -1;
  s.push_back(*max_element(s.begin(), s.end()) + 1);
  return max_suffix(s, true);
}
int max_cyclic_shift(string s)
{
  int n = s.size();
  for (int i = 0; i < n; i++)
    s.pb(s[i]);
  return max_suffix(s);
}
int min_cyclic_shift(string s)
{
  for (auto &i : s)
    i *= -1;
  return max_cyclic_shift(s);
}
// retorna a posicao de inicio menor/maior sufixo/shift de uma string
```

17.6 rabin-karp

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
```

```cpp
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 100001

const int p = 31;
const int mod = 1e9 + 9;

int multiplicate(int x, int y)
{
  return (x * y) % mod;
}
int subtract(int a, int b)
{
  return (a - b < 0) ? a - b + mod : a - b;
}
int sum(int a, int b)
{
  return (a + b >= mod) ? a + b - mod : a + b;
}
vector<int> rabin_karp(string s, string t)
{
  int n = s.size(), m = t.size();
  vector<int> pot(n);
  pot[0] = 1;
  for (int i = 1; i < n; i++)
    pot[i] = multiplicate(pot[i - 1], p);
  vector<int> pref(n + 1, 0);
  for (int i = 0; i < n; i++)
  {
    int val = multiplicate(pref[i], p);
    pref[i + 1] = sum(s[i], val);
  }
  int hs = 0;
  for (int i = 0; i < m; i++)
  {
    int val = multiplicate(hs, p);
    hs = sum(t[i], val);
  }
  vector<int> ans;
  for (int i = 0; i + m - 1 < n; i++)
  {
    int cur_h = subtract(pref[i + m], multiplicate(pref[i], pot[m]));
    if (cur_h == hs)
      ans.pb(i);
  }
  return ans;
}
signed main()
{
  string s, t;
  cin >> s >> t;
  vector<int> ans = rabin_karp(s, t);
  for (auto const &i : ans)
    cout << i << " " << i + t.size() - 1 << endl;
  return 0;
}
// rabin-karp for pattern matching
// given two string s and t, determine all occurrences of t in s
// 1- calcule the hash of string t
// 2- calcule the prefix hash of string s
// 3- compare every substring of s with length |t|
// 4- store all occurrences in a vector and return this vector
// complexity: O(|t| + |s|)
```

17.7 stringhashing

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000001

// https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/Strings/
    hashingLargeMod.cpp
const int MOD = (1ll << 61) - 1;
int P;
int mulmod(int a, int b)
{
  const static int LOWER = (1ll << 30) - 1, GET31 = (1ll << 31) - 1;
  int l1 = a & LOWER, h1 = a >> 30, l2 = b & LOWER, h2 = b >> 30;
  int m = l1 * h2 + l2 * h1, h = h1 * h2;
  int ans = l1 * l2 + (h >> 1) + ((h & 1) << 60) + (m >> 31) + ((m & GET31) <<
      30) + 1;
  ans = (ans & MOD) + (ans >> 61), ans = (ans & MOD) + (ans >> 61);
  return ans - 1;
}
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
int uniform(int l, int r)
{
  uniform_int_distribution<int> uid(l, r);
  return uid(rng);
}
struct string_hashing
{
  vector<int> h, p;
  string_hashing() {}
  string_hashing(string s) : h(s.size()), p(s.size())
  {
    p[0] = 1, h[0] = s[0];
    for (int i = 1; i < s.size(); i++)
      p[i] = mulmod(p[i - 1], P), h[i] = (mulmod(h[i - 1], P) + s[i]) % MOD;
  }
  int get(int l, int r)
  {
    int hash = h[r] - (l ? mulmod(h[l - 1], p[r - l + 1]) : 0);
    return hash < 0 ? hash + MOD : hash;
  }
  int append(int h, int hb, int blen)
  {
    return (hb + mulmod(h, p[blen])) % MOD;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  P = uniform(256, MOD - 1);
  vector<string_hashing> v(n);
  vector<string_hashing> v_rev(n);
  vector<int> sz(n);
  int ans = 0;
  for (int i = 0; i < n; i++)
  {
    string s;
    cin >> s;
    v[i] = string_hashing(s);
    sz[i] = s.size();
    ans += (s.size() * n);
```

```
    ans += (s.size() * n);
    reverse(s.begin(), s.end());
    v_rev[i] = string_hashing(s);
  }
  unordered_map<int, int> mp;
  for (int i = 0; i < n; i++)
  {
    for (int j = 1; j <= sz[i]; j++)
      mp[v[i].get(0, j - 1)]++;
  }
  for (int i = 0; i < n; i++)
  {
    int acc = 0;
    for (int j = sz[i]; j >= 1; j--)
    {
      int curr = mp[v_rev[i].get(0, j - 1)];
      ans -= ((curr - acc) * j * 2);
      acc = curr;
    }
  }
  cout << ans << endl;
}
// https://codeforces.com/contest/1902/problem/E
// solucao usando hash mod 2^61 - 1
```

17.8 stringhashing2

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 2001
#define mod 1000000009

struct modint
{
  int val;
  modint(int v = 0) { val = v % mod; }
  int pow(int y)
  {
    modint x = val;
    modint z = 1;
    while (y)
    {
      if (y & 1)
        z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};
struct string_hashing
{
  modint d;
  modint h;
  vector<modint> pref;
  vector<modint> pot;

  string_hashing() {}
  string_hashing(int base, string &s)
  {
    d = base;
    pref.resize(s.size() + 1);
    pref[0] = 0;
    for (int i = 0; i < s.size(); i++)
    {
      modint val = pref[i] * d;
      pref[i + 1] = val + s[i];
    }
    h = pref[s.size()];
    pot.resize(s.size() + 1);
    pot[0] = 1;
    for (int i = 1; i <= s.size(); i++)
      pot[i] = pot[i - 1] * d;
  }
  modint get(int l, int r)
  {
    return pref[r + 1] - (pref[l] * pot[r - l + 1]);
  }
  modint append(modint hb, int blen)
  {
    h = hb + (h * pot[blen]);
    return h;
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s;
  cin >> s;
  string_hashing h(256, s); // (base, string)
  // string_hashing h(227, s); // (base, string)
  return 0;
}
```

17.9 substring fft

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007
#define cd complex<double>

const double eps = 1e-12;
const int alphabet_size = 26;

namespace fft
```

```
{
  void dft(vector<cd> &a)
  {
    int n = a.size();
    if (n == 1)
      return;
    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++)
    {
      a0[i] = a[2 * i];
      a1[i] = a[2 * i + 1];
    }
    dft(a0);
    dft(a1);
    double ang = 2 * PI / n;
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++)
    {
      a[i] = a0[i] + w * a1[i];
      a[i + n / 2] = a0[i] - w * a1[i];
      w *= wn;
    }
  }
  void inverse_dft(vector<cd> &a)
  {
    int n = a.size();
    if (n == 1)
      return;
    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++)
    {
      a0[i] = a[2 * i];
      a1[i] = a[2 * i + 1];
    }
    inverse_dft(a0);
    inverse_dft(a1);
    double ang = 2 * PI / n * -1;
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++)
    {
      a[i] = a0[i] + w * a1[i];
      a[i + n / 2] = a0[i] - w * a1[i];
      a[i] /= 2;
      a[i + n / 2] /= 2;
      w *= wn;
    }
  }
  vector<double> mul(vector<cd> a, vector<cd> b)
  {
    int n = 1;
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    while (n < a.size() + b.size())
      n <<= 1;
    fa.resize(n);
    fb.resize(n);
    dft(fa);
    dft(fb);
    for (int i = 0; i < n; i++)
      fa[i] *= fb[i];
    inverse_dft(fa);
    vector<double> ans(n);
    for (int i = 0; i < n; i++)
      ans[i] = fa[i].real();
    return ans;
  }
} // namespace fft
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s, t;
  cin >> s >> t;
  int n = s.size(), m = t.size();
  reverse(t.begin(), t.end());
  vector<cd> a(n);
  vector<cd> b(m);
  for (int i = 0; i < n; i++)
  {
    int ch = s[i] - 'a';
    double ang = (2 * PI * ch) / alphabet_size;
    a[i] = cd(cos(ang), sin(ang));
  }
  for (int i = 0; i < m; i++)
  {
    int ch = t[i] - 'a';
    double ang = (2 * PI * ch) / alphabet_size;
    b[i] = cd(cos(ang), -sin(ang));
  }
  vector<double> ans = fft::mul(a, b);
  int matches = 0;
  for (int i = m - 1; i < n; i++)
    matches += (abs(ans[i] - m) <= eps);
  cout << matches << endl;
  return 0;
}
// number of matches of a pattern in string
// using fft
```

17.10 suffix array

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define pci pair<char, int>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

vector<int> suffix_array(string s)
{
  s += "$"; // menor do que todos os chars da string st
  int n = s.size(), N = max(n + 1, 260ll);
  vector<int> sa(n), ra(n);
  for (int i = 0; i < n; i++)
  {
    sa[i] = i, ra[i] = s[i];
  }
  for (int k = 0; k < n; k ? k *= 2 : k++)
  {
    vector<int> nsa(sa), nra(n), cnt(N);
    for (int i = 0; i < n; i++)
    {
      nsa[i] = (nsa[i] - k + n) % n;
      cnt[ra[i]]++;
    }
    for (int i = 1; i < N; i++)
    {
      cnt[i] += cnt[i - 1];
    }
    for (int i = n - 1; i + 1; i--)
    {
      sa[--cnt[ra[nsa[i]]]] = nsa[i];
    }
    for (int i = 1, r = 0; i < n; i++)
    {
      nra[sa[i]] = r += (ra[sa[i]] != ra[sa[i - 1]] || ra[(sa[i] + k) % n] != ra
          [(sa[i - 1] + k) % n]);
    }
    ra = nra;
    if (ra[sa[n - 1]] == n - 1)
      break;
```

```
  }
  return vector<int>(sa.begin() + 1, sa.end());
}

vector<int> kasai(string s, vector<int> sa)
{
  int n = s.size(), k = 0;
  vector<int> ra(n), lcp(n);
  for (int i = 0; i < n; i++)
  {
    ra[sa[i]] = i;
  }
  for (int i = 0; i < n; i++, k -= !!k)
  {
    if (ra[i] == n - 1)
    {
      k = 0;
      continue;
    }
    int j = sa[ra[i] + 1];
    while (i + k < n and j + k < n and s[i + k] == s[j + k])
      k++;
    lcp[ra[i]] = k;
  }
  return lcp;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s;
  cin >> s;
  int n = s.size();
  vector<int> p = suffix_array(s);
  vector<int> lcp = kasai(s, p);
  for (int i = 0; i < s.size(); i++) // sufix array
    cout << p[i] << " ";
  cout << endl;
  for (int i = 0; i + 1 < s.size(); i++) // lcp entre 2 suffixos adjacentes no
       suffix array
    cout << lcp[i] << " ";
  cout << endl;
  int q;
  cin >> q;
  while (q--)
  {
    string t;
    cin >> t;
    int i = 0, f = n - 1, m, lb, ub;
    while (i < f)
    {
      m = (i + f) / 2;
      (t <= s.substr(p[m], t.size())) ? f = m : i = m + 1;
    }
    ub = i, i = 0, f = n - 1;
    while (i < f)
    {
      m = (i + f) / 2;
      (t >= s.substr(p[m], t.size())) ? i = m + 1 : f = m;
    }
    lb = i;
    if (s.substr(p[lb], t.size()) == t)
      lb++;
    cout << lb - ub << endl;
  }
  return 0;
}
```

17.11 suffix automaton

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100001
#define mod 998244353

namespace sa
{
  struct state
  {
    int len, suf_link;
    map<char, int> nxt;
  };

  vector<int> term;
  state st[2 * MAXN];
  int dp[2 * MAXN];
  int sz, last;

  void init()
  {
    memset(dp, -1, sizeof(dp));
    st[0].len = 0;
    st[0].suf_link = -1;
    sz++;
    last = 0;
  }
  void get_link(int curr, int p, char c)
  {
    while (p != -1 && !st[p].nxt.count(c))
    {
      st[p].nxt[c] = curr;
      p = st[p].suf_link;
    }
    if (p == -1)
    {
      st[curr].suf_link = 0;
      return;
    }
    int q = st[p].nxt[c];
    if (st[p].len + 1 == st[q].len)
    {
      st[curr].suf_link = q;
      return;
    }
    int clone = sz;
    sz++;
    st[clone].len = st[p].len + 1;
    st[clone].nxt = st[q].nxt;
    st[clone].suf_link = st[q].suf_link;
    while (p != -1 && st[p].nxt[c] == q)
    {
      st[p].nxt[c] = clone;
      p = st[p].suf_link;
    }
    st[q].suf_link = clone;
    st[curr].suf_link = clone;
  }
  void build(string &s)
  {
    for (auto const &c : s)
    {
      int curr = sz;
      sz++;
      st[curr].len = st[last].len + 1;
      get_link(curr, last, c);
      last = curr;
    }
    // achar os estados terminais
    // um estado terminal e aquele que representa um sufixo da string s
    int p = last;
    while (p != -1)
```

```
    {
      term.pb(p);
      p = st[p].suf_link;
    }
  }
  void dfs2(int v)
  {
    if (dp[v] != -1)
      return;
    dp[v] = 1;
    for (auto const &u : st[v].nxt)
    {
      if (!u.sec)
        continue;
      dfs2(u.sec);
      dp[v] += dp[u.sec];
    }
  }
  void dfs(int v, int k, int &at, string &curr)
  {
    if (at == k)
      return;
    for (auto const &u : st[v].nxt)
    {
      if (!u.sec)
        continue;
      if (at + dp[u.sec] < k)
      {
        at += dp[u.sec];
        continue;
      }
      curr.pb(u.fir);
      at++;
      dfs(u.sec, k, at, curr);
      if (at == k)
        return;
      curr.pop_back();
    }
  }
  void find_kth(int k)
  {
    int at = 0;
    string curr = "";
    dfs(0, k, at, curr);
    cout << curr << endl;
  }
} // namespace sa
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s;
  cin >> s;
  sa::init();
  sa::build(s);
  sa::dfs2(0);
  int q;
  cin >> q;
  while (q--)
  {
    int k;
    cin >> k;
    sa::find_kth(k);
  }
  return 0;
}
// https://cp-algorithms.com/string/suffix-automaton.html
// suffix automaton
// definicao: um suffix automaton de uma string s e um automato finito
    deterministico
// que aceita todos os suffixos da string s.
// ou seja:
// um suffix automaton eh um grafo aciclico orientado
// tal que, um vertice representa um estado
// e uma aresta representa uma transicao (um caractere a mais em relacao ao
    estado(suffixo) atual)
// t0 -> estado inicial(string vazia), e todos os demais estados podem ser
    alcancados a partir de t0
// o suffix automaton minimiza o numero de vertices
// a propiedade mais importante de um suffix automaton eh a de que
// ele contem informacoes sobre todas as substrings de s
// pois, qualquer caminho comecando do estado t0 corresponde a uma substring de
    s

// conceitos:

// 1 - endpos
// seja t uma substring de s, endpos(t) eh o conjunto de todas os indices(
    posicoes)
// na string s no qual todas as ocorrencias de t acabam
// por exemplo, se s = "abcbc" e t = "bc"
// logo endpos(t) = {2, 4}
// com isso se duas duas substrings t1 e t2 possuem os seus endpos iguais,
// chamamos de endpos-equivalent e dai podemos extrair algumas informacoes
// info 1: se duas substrings u e w u.size() <= w.size(), se u eh um sufixo de w
    , logo endpos(u) esta contido em endpos(w)
// info 2: se duas substrings u e w u.size() <= w.size(), se u nao eh um sufixo
    de w, logo nao existe interseccao entre endpos(u) e endpos(w)

// 2 - suffix link
// seja v algum estado != t0,  sabemos que v corresponde a classe de strings que
    possui os mesmos endpos
// seja w a maior dessas strings, com isso, todas as demais sao suffixos de w
// com isso um suffix_link(v) corresponde ao maior suffix de w que esta em outra
    classe de equivalencia pelos endpos
// com isso podemos abstrair algumas informacoes:
// info 1: os suffix links foram uma arvore enraizada em t0
// info 2: se construirmos uma arvore usando os sets endpos, a estrutura sera a
    arvore com os suffix links

// com isso, vamos ao algoritimo
// 1 - vai ser online, e iremos adicionar os caracteres de 1 por 1, da esquerda
    para a direita
// 2 - com isso para adicionar um novo char, seja v o ultimo estado que
    adicionamos antes do atual, adicionamos uma aresta
// do proximo em relacao a ele e iremos procurar pelo suffix link para adicionar
// 3 - complexidade O(n) ou O(n log k), se usarmos uma map para guardar as
    transicoes partindo de um estado

// exemplos de aplicacoes:

// 1 - checar se t aparece em s como substring:
// construa o suffix automaton de s, e vamos tentar fazer um caminho partindo de
    t0
// se em algum momento, nao existir transicao, logo nao existe
// se conseguir chegar no final, existe

// 2 - numero de substrings diferentes de s
// constura o suffix automaton de s, sabemos que, cada substring de s
    corresponde a um caminho no automato
// com isso, o numero de substrings distintas eh o numero de caminhos diferentes
    que comecam de t0
// e terminam em algum canto
// isso pode ser calculado facilmente com uma dpzinha

// 3 - tamanho total de todas as substrings distintas de s
// similar a solucao passada, podemos fazer isso com uma dpzinha :)

// 4 - a k-esima menor substring lexicografica
// a k-esima menor substring lexicograficamente corresponde ao k-esimo path no
    suffix automaton
// se considerarmos as transicoes sempre indo do menor char para o maior durante
    o percurso

// 5 - o menor cyclic shift
// construa o suffix automaton da string s + s (duplicada)
// com isso o suffix automaton vai conter todos os cyclic shifts da string s
// e agora o problema eh reduzido para: encontre o menor caminho
    lexicograficamente de tamanho s.size()

// 6 - numero de ocorrencias de uma substring t em s
// construa o suffix automaton da string s
// com isso, quando criamos um no que nao seja o t0 nem um clone
// inicializamos cnt[v] = 1
// depois vamos percorrer todo os estados em ordem decrescente de len
// e aplicando cnt[link(v)] += cnt[v]
// no final, para responder a query basta fazer o caminho ate o estado que
```

```
    quisermos e printar o cnt dele

// e mais uma porrada de aplicacoes alem dessas :)

// example of a problem: https://www.spoj.com/problems/SUBLEX/
// ver qual a k-th string lexicografica sem repeticao
// note que o k pode ser gigante
// ideia: calcular dp[v] -> quantidade de caminhos que comecam em v
// dai para cada query roda um dfs, sendo que, so vou pro proximo estado se at + dp[u] >= k
// caso contrario, posso ignorar
```

17.12 z-function

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 2001
#define mod 1000000007

vector<int> z_function(string &s)
{
  int n = s.size();
  vector<int> z(n);
  z[0] = n;
  for (int i = 1, l = 0, r = 0; i < n; i++)
  {
    if (i <= r)
      z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
      z[i]++;
    if (i + z[i] - 1 > r)
      l = i, r = i + z[i] - 1;
  }
  return z;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  string s;
  cin >> s;
  vector<int> z = z_function(s);
}
// z-function
// calcula para cada i:
// z[i] = o tamanho de lcp(s, s.substr(i, n - i))
// lcp -> longest comom prefix
```

18 Structures

18.1 binary lifting

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

struct item
{
  int nxt, sum;
};

int n, q;
int v[MAXN];
item st[MAXN][21];

signed main()
{
  cin >> n >> q;
  for (int i = 0; i < n; i++)
    cin >> v[i];
  for (int i = 0; i < n; i++)
  {
    st[i][0].nxt = min(i + 1, n - 1);
    st[i][0].sum = v[st[i][0].nxt];
  }
  for (int i = 1; i < 21; i++)
  {
    for (int v = 0; v < n; v++)
    {
      st[v][i].nxt = st[st[v][i - 1].nxt][i - 1].nxt;
      st[v][i].sum = st[v][i - 1].sum + st[st[v][i - 1].nxt][i - 1].sum;
    }
  }
  while (q--)
  {
    int l, r;
    cin >> l >> r;
    r--;
    int ans = v[l], len = r - l;
    for (int i = 20; i >= 0; i--)
    {
      if (len & (1 << i))
      {
        ans += st[l][i].sum;
        l = st[l][i].nxt;
      }
    }
    cout << ans << endl;
  }
  return 0;
}
// simple range sum query with binary lifting
// https://judge.yosupo.jp/problem/static_range_sum
```

18.2 bit2d

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

```cpp
#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000001
#define mod 1000000007

// source: https://github.com/tfg50/Competitive-Programming/blob/master/
    Biblioteca/Data%20Structures/Bit2D.cpp
struct bit2d
{
  vector<int> ord;
  vector<vector<int>> t;
  vector<vector<int>> coord;

  bit2d(vector<pi> &pts) // recebe todos os pontos que vao ser inseridos pra
      construir, mas nao insere eles
  {
    sort(pts.begin(), pts.end());
    for (auto const &a : pts)
    {
      if (ord.empty() || a.fir != ord.back())
        ord.pb(a.fir);
    }
    t.resize(ord.size() + 1);
    coord.resize(t.size());
    for (auto &a : pts)
    {
      swap(a.fir, a.sec);
    }
    sort(pts.begin(), pts.end());
    for (auto &a : pts)
    {
      swap(a.fir, a.sec);
      for (int on = upper_bound(ord.begin(), ord.end(), a.fir) - ord.begin(); on
          < t.size(); on += on & -on)
      {
        if (coord[on].empty() || coord[on].back() != a.sec)
          coord[on].push_back(a.sec);
      }
    }
    for (int i = 0; i < t.size(); i++)
      t[i].assign(coord[i].size() + 1, 0);
  }
  void add(int x, int y, int v) // v[a][b] += v
  {
    for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx < t.
        size(); xx += xx & -xx)
    {
      for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) - coord[
          xx].begin(); yy < t[xx].size(); yy += yy & -yy)
        t[xx][yy] += v;
    }
  }
  int qry(int x, int y) // soma de todos os v[a][b] com (a <= x && b <= y)
  {
    int ans = 0;
    for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.begin(); xx > 0;
        xx -= xx & -xx)
    {
      for (int yy = upper_bound(coord[xx].begin(), coord[xx].end(), y) - coord[
          xx].begin(); yy > 0; yy -= yy & -yy)
        ans += t[xx][yy];
    }
    return ans;
  }
  int qry2(int x1, int y1, int x2, int y2)
  {
    return qry(x2, y2) - qry(x2, y1 - 1) - qry(x1 - 1, y2) + qry(x1 - 1, y1 - 1)
        ;
  }
  void add2(int x1, int y1, int x2, int y2, int v)
  {
    add(x1, y1, v);
    add(x1, y2 + 1, -v);
    add(x2 + 1, y1, -v);
    add(x2 + 1, y2 + 1, v);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
```

18.3 color update

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

const int inf = 1e15;

struct color_upd
{
#define left fir
#define right sec.fir
#define color sec.sec
  set<pii> ranges;
  vector<pii> erased;

  color_upd(int n) // inicialmente, todo mundo pintado com a cor inf
  {
    // nao usar cores negativas!!!!!!!!!
    ranges.insert({0, {n - 1, inf}});
  }
  int get(int i) // qual a cor do elemento na posicao i
  {
    auto it = ranges.upper_bound({i, {1e18, 1e18}});
    if (it == ranges.begin())
      return -1;
    it--;
    return (*(it)).color;
  }
  void del(int l, int r) // apaga o intervalo [l, r]
  {
    erased.clear();
    auto it = ranges.upper_bound({l, {0, 0}});
    if (it != ranges.begin())
    {
      it--;
    }
    while (it != ranges.end())
    {
      if ((*(it)).left > r)
        break;
      else if ((*(it)).right >= l)
        erased.push_back(*it);
      it++;
    }
    if (erased.size() > 0)
    {
```

```
      int sz = erased.size();
      auto it = ranges.lower_bound({erased[0].left, {0, 0}});
      auto it2 = ranges.lower_bound({erased[sz - 1].left, {0, 0}});
      pii ini = *it, fim = *it2;
      it2++;
      ranges.erase(it, it2);
      pii upd1 = {ini.left, {l - 1, ini.color}};
      pii upd2 = {r + 1, {fim.right, fim.color}};
      erased[0].left = max(erased[0].left, l);
      erased[sz - 1].right = min(erased[sz - 1].right, r);
      if (upd1.left <= upd1.right)
        ranges.insert(upd1);
      if (upd2.left <= upd2.right)
        ranges.insert(upd2);
    }
  }
  void add(int a, int b, int c) // nao ter dois intervalos adjacentes com a
       mesma cor no set de ranges
  {
    auto it = ranges.lower_bound({a, {b, 0}});
    pii aa = {-1, {-1, -1}};
    pii bb = {-1, {-1, -1}};
    if (it != ranges.end())
    {
      if ((*it).color == c && (*it).left == b + 1)
      {
        aa = *it;
        b = (*it).right;
      }
    }
    if (it != ranges.begin())
    {
      it--;
      if ((*it).color == c && (*it).right == a - 1)
      {
        bb = *it;
        a = (*it).left;
      }
    }
    ranges.erase(aa);
    ranges.erase(bb);
    ranges.insert({a, {b, c}});
  }
  void upd(int a, int b, int c) // pinta o intervalo [a, b] com a cor c
  {
    del(a, b);
    add(a, b, c);
  }
};
struct segtree
{
  vector<int> seg;
  vector<int> lazy;

  segtree(int n)
  {
    seg.resize(4 * n, 0);
    lazy.assign(4 * n, 0);
  }
  int single(int x)
  {
    return x;
  }
  int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a + b;
  }
  void add(int i, int l, int r, int diff)
  {
    seg[i] += (r - l + 1) * diff;
    if (l != r)
    {
      lazy[i << 1] += diff;
      lazy[(i << 1) | 1] += diff;
    }
    lazy[i] = 0;
  }
  void update(int i, int l, int r, int ql, int qr, int diff)
  {
    if (lazy[i])
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return;
    if (l >= ql && r <= qr)
    {
      add(i, l, r, diff);
      return;
    }
    int mid = (l + r) >> 1;
    update(i << 1, l, mid, ql, qr, diff);
    update((i << 1) | 1, mid + 1, r, ql, qr, diff);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int query(int l, int r, int ql, int qr, int i)
  {
    if (lazy[i])
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return neutral();
    if (l >= ql && r <= qr)
      return seg[i];
    int mid = (l + r) >> 1;
    return merge(query(l, mid, ql, qr, i << 1), query(mid + 1, r, ql, qr, (i <<
        1) | 1));
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  color_upd c = color_upd(n);
  segtree s = segtree(n);
  for (int i = 0; i < n; i++)
    c.upd(i, i, i + 1);
  while (q--)
  {
    int t;
    cin >> t;
    if (t == 1)
    {
      int l, r, x;
      cin >> l >> r >> x;
      l--, r--;
      c.upd(l, r, x);
      for (auto const &i : c.erased)
        s.update(1, 0, n - 1, i.left, i.right, abs(x - i.color));
    }
    else
    {
      int l, r;
      cin >> l >> r;
      l--, r--;
      cout << s.query(0, n - 1, l, r, 1) << endl;
    }
  }
  return 0;
}
// https://codeforces.com/contest/444/problem/C
```

18.4 fenwick

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
```

```
template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

struct fenw
{
  int n;
  vector<int> bit;
  fenw() {}
  fenw(int sz)
  {
    n = sz;
    bit.assign(sz + 1, 0);
  }
  int qry(int r) // query de prefixo a[0] + a[1] + ...  a[r]
  {
    int ret = 0;
    for (int i = r + 1; i > 0; i -= i & -i)
      ret += bit[i];
    return ret;
  }
  void upd(int r, int x) // a[r] += x
  {
    for (int i = r + 1; i <= n; i += i & -i)
      bit[i] += x;
  }
  int bs(int x) // retorna o maior indice i (i < n) tal que: qry(i) < x
  {
    int i = 0, k = 0;
    while (1 << (k + 1) <= n)
      k++;
    while (k >= 0)
    {
      int nxt_i = i + (1 << k);
      if (nxt_i <= n && bit[nxt_i] < x)
      {
        i = nxt_i;
        x -= bit[i];
      }
      k--;
    }
    return i - 1;
  }
};
```

18.5 fenwick2

```
// fenwick com update pro range [l, r]
// complexidade O(q * log(n)) com a criacao de duas bits ao inves de uma
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define int long long int
#define pb push_back
#define mp make_pair
#define pi pair<string, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100001
#define MAXL 20
#define mod 998244353

int n;
vector<int> bit, bit2;

void add1(int idx, int delta)
{
  for (; idx < n; idx = idx | (idx + 1))
    bit[idx] += delta;
}
void add2(int idx, int delta)
{
  for (; idx < n; idx = idx | (idx + 1))
    bit2[idx] += delta;
}
void update_range(int val, int l, int r)
{
  add1(l, val);
  add1(r + 1, -val);
  add2(l, val * (l - 1));
  add2(r + 1, -val * r);
}
int sum1(int r)
{
  int ret = 0;
  for (; r >= 0; r = (r & (r + 1)) - 1)
    ret += bit[r];
  return ret;
}
int sum2(int r)
{
  int ret = 0;
  for (; r >= 0; r = (r & (r + 1)) - 1)
    ret += bit2[r];
  return ret;
}
int sum(int x)
{
  return (sum1(x) * x) - sum2(x);
}
int range_sum(int l, int r)
{
  return sum(r) - sum(l - 1);
}
int main()
{
  bit.assign(MAXN, 0); // inicializar sempre
  bit2.assign(MAXN, 0); // inicializar sempre
  update_range(x, l, r); // pra cada elemento em [l, r] += x
  range_sum(l, r);  // soma de [l, r]
}
```

18.6 fenwick2D

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1025
#define mod 1000000007

int b[MAXN][MAXN];
int vv[MAXN][MAXN];

int qry(int x, int y)
```

```
{
  int sum = 0;
  for (; x >= 0; x = (x & (x + 1)) - 1)
  {
    for (int yy = y; yy >= 0; yy = (yy & (yy + 1)) - 1)
      sum += b[x][yy];
  }
  return sum;
}
void add(int x, int y, int v)
{
  for (; x < MAXN; x = x | (x + 1))
  {
    for (int yy = y; yy < MAXN; yy = yy | (yy + 1))
      b[x][yy] += v;
  }
}
int qry2(int x1, int y1, int x2, int y2)
{
  return qry(x2, y2) - qry(x2, y1 - 1) - qry(x1 - 1, y2) + qry(x1 - 1, y1 - 1);
}
void add2(int x1, int y1, int x2, int y2, int v)
{
  add(x1, y1, v);
  add(x1, y2 + 1, -v);
  add(x2 + 1, y1, -v);
  add(x2 + 1, y2 + 1, v);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) // reseta
    {
      for (int j = 0; j < n; j++)
      {
        add(i, j, -vv[i][j]);
        vv[i][j] = 0;
      }
    }
    while (1)
    {
      string s;
      cin >> s;
      if (s == "SET")
      {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, -vv[a][b]);
        vv[a][b] = c;
        add(a, b, vv[a][b]);
      }
      else if (s == "SUM")
      {
        int a, b, c, d;
        cin >> a >> b >> c >> d; // c >= a e d >= b
        cout << qry2(a, b, c, d) << endl;
      }
      else
      {
        break;
      }
    }
  }
  return 0;
}
// to test: https://www.spoj.com/problems/MATSUM/
```

18.7 fenwick3

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

//#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007

int v[MAXN];

namespace bit
{
  ordered_set<int> bit[MAXN];

  int query(int r, int a, int b)
  {
    int ret = 0, curr = r;
    for (; r >= 0; r = (r & (r + 1)) - 1)
      ret += (bit[r].order_of_key(b + 1) - bit[r].order_of_key(a));
    return ret;
  }
  void add(int idx, int delta)
  {
    for (; idx < MAXN; idx = idx | (idx + 1))
      bit[idx].insert(delta);
  }
  void rem(int idx, int delta)
  {
    for (; idx < MAXN; idx = idx | (idx + 1))
      bit[idx].erase(delta);
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// ideia da merge sort tree na bit (fica mais rapido)
// so fazer uma bit de ordered set ou vector(se nao tiver update)
// add -> adiciona o numero delta na posicao idx
// rem -> remove o numero delta na posicao idx
// query -> retorna o numero de elementos tal que posicao <= r && (a <= num <= b
    )
```

18.8 implicit seg

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
```

```cpp
#define MAXN 200005
#define mod 998244353

struct implcit_seg
{
  int l, r;
  int sum, lazy;
  implcit_seg *left_child = nullptr;
  implcit_seg *right_child = nullptr;

  implcit_seg(int l, int r) : l(l), r(r)
  {
    sum = 0;
    lazy = 0;
  }
  void check_childs()
  {
    if (!left_child && l != r)
    {
      int mid = (l + r) >> 1;
      left_child = new implcit_seg(l, mid);
      right_child = new implcit_seg(mid + 1, r);
    }
  }
  void add(int x)
  {
    sum += (r - l + 1) * x;
    if (l != r)
    {
      check_childs();
      left_child->lazy += x;
      right_child->lazy += x;
    }
    lazy = 0;
  }
  void upd(int ql, int qr, int x)
  {
    add(lazy);
    if (l > r || l > qr || r < ql)
      return;
    if (l >= ql && r <= qr)
    {
      add(x);
      return;
    }
    check_childs();
    left_child->upd(ql, qr, x);
    right_child->upd(ql, qr, x);
    sum = left_child->sum + right_child->sum;
  }
  void upd(int k, int x)
  {
    sum += x;
    check_childs();
    if (left_child)
    {
      if (k <= left_child->r)
        left_child->upd(k, x);
      else
        right_child->upd(k, x);
    }
  }
  int qry(int ql, int qr)
  {
    add(lazy);
    if (l > r || l > qr || r < ql)
      return 0;
    if (l >= ql && r <= qr)
      return sum;
    check_childs();
    return left_child->qry(ql, qr) + right_child->qry(ql, qr);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  implcit_seg *s = new implcit_seg(0, n - 1);
  while (q--)
  {
    int t;
    cin >> t;
    if (t == 1)
    {
      int l, r, x;
      cin >> l >> r >> x;
      if (l == r - 1) // point update
        s->upd(l, x);
      else // range update
        s->upd(l, r - 1, x);
    }
    else
    {
      int l, r;
      cin >> l >> r;
      cout << s->qry(l, r - 1) << endl; // range sum
    }
  }
  return 0;
}
```

18.9 lower bound segtree

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 998244353

const int inf = 1e18;

struct segtree
{
  int n;
  vector<int> v;
  vector<int> seg;

  segtree(vector<int> &vv)
  {
    v = vv;
    n = v.size();
    seg.assign(4 * n, 0);
    build(0, n - 1, 1);
  }
  int single(int x)
  {
    return x;
  }
  int neutral()
  {
    return 1e18;
  }
  int merge(int a, int b)
  {
    return max(a, b);
  }
  void update(int i, int l, int r, int q, int x)
  {
```

```
    if (l == r)
    {
      seg[i] = single(x);
      return;
    }
    int mid = (l + r) >> 1;
    if (q <= mid)
      update(i << 1, l, mid, q, x);
    else
      update((i << 1) | 1, mid + 1, r, q, x);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int query(int l, int r, int ql, int qr, int i, int x)
  {
    if (l == r)
      return (seg[i] >= x && l >= ql) ? l : -1;
    int mid = (l + r) >> 1, at = -1;
    if (seg[i << 1] >= x && mid >= ql)
      at = query(l, mid, ql, qr, i << 1, x);
    if (at == -1)
      at = query(mid + 1, r, ql, qr, (i << 1) | 1, x);
    return at;
  }
  void build(int l, int r, int i)
  {
    if (l == r)
    {
      seg[i] = single(v[l]);
      return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, i << 1);
    build(mid + 1, r, (i << 1) | 1);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int qry(int x, int l)
  {
    return query(0, n - 1, l, n - 1, 1, x);
  }
  void upd(int x, int l)
  {
    update(1, 0, n - 1, x, l);
  }
};
signed main()
{
  int n, q;
  cin >> n >> q;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  segtree st(v);
  while (q--)
  {
    int t;
    cin >> t;
    if (t == 2)
    {
      int x, l;
      cin >> x >> l; // find the minimum index j such that j >= l and v[j] >= x
      cout << st.qry(x, l) << endl;
    }
    else
    {
      int a, b;
      cin >> a >> b; // v[a] = b;
      st.upd(a, b);
    }
  }
}
```

18.10 mergesorttree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 100001
#define mod 1000000007

vector<int> seg[4 * MAXN];
int v[MAXN];

void update(int i, int l, int r, int q, int x)
{
  if (l == r)
  {
    seg[i].clear();
    seg[i].pb(x);
    return;
  }
  int mid = (l + r) >> 1;
  if (q <= mid)
    update(i << 1, l, mid, q, x);
  else
    update((i << 1) | 1, mid + 1, r, q, x);
  // a merge do c++ une os dois vectors, deixando ele ordenado em O(n)
  seg[i].clear();
  merge(seg[i << 1].begin(), seg[i << 1].end(), seg[(i << 1) | 1].begin(), seg[(
       i << 1) | 1].end(), back_inserter(seg[i]));
}
int query(int l, int r, int ql, int qr, int i, int x)
{
  int mid = (l + r) >> 1;
  if (l > r || l > qr || r < ql)
    return 0;
  if (l >= ql && r <= qr) // quantidade de elementos maiores do que x no range
       atual
    return seg[i].end() - upper_bound(seg[i].begin(), seg[i].end(), x);
  return query(l, mid, ql, qr, i << 1, x) + query(mid + 1, r, ql, qr, (i << 1) |
       1, x);
}
void build(int l, int r, int i)
{
  if (l == r)
  {
    seg[i].pb(v[l]);
    return;
  }
  int mid = (l + r) >> 1;
  build(l, mid, i << 1);
  build(mid + 1, r, (i << 1) | 1);
  // a merge do c++ une os dois vectors, deixando ele ordenado em O(n)
  merge(seg[i << 1].begin(), seg[i << 1].end(), seg[(i << 1) | 1].begin(), seg[(
       i << 1) | 1].end(), back_inserter(seg[i]));
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// merge sort tree
// a segment tree with ordered vectors in range nodes

// example:
// number of elements > x in a range [l, r]
```

```
// memory: O(n * log n)
// query: O(log^2 n)
```

18.11 min queue

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 1005
#define mod 998244353

namespace min_queue
{
  deque<pi> q;
  int l, r;

  void init()
  {
    l = r = 1;
    q.clear();
  }
  void push(int v)
  {
    while (!q.empty() && v < q.back().fir)
      q.pop_back();
    q.pb({v, r});
    r++;
  }
  void pop()
  {
    if (!q.empty() && q.front().sec == l)
      q.pop_front();
    l++;
  }
  int getmin()
  {
    return q.front().fir;
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, m;
  cin >> n >> m;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  int l = 0, r = m - 1;
  cout << l << " " << r << endl;
  for (int i = l; i <= r; i++)
    min_queue::push(v[i]);
  cout << min_queue::getmin() << " ";
  l++, r++;
  while (r < n)
  {
    min_queue::pop();
    min_queue::push(v[r]);
    cout << min_queue::getmin() << " ";
    l++, r++;
  }
  cout << endl;
  return 0;
}
// minimum of each subarray of length m (m <= n)
```

18.12 mo

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500005
#define mod 1000000007

int n, q;
int v[MAXN];

namespace mo
{
  struct query
  {
    int idx, l, r;
  };

  int block;
  vector<query> queries;
  vector<int> ans;

  bool cmp(query x, query y)
  {
    if (x.l / block != y.l / block)
      return x.l / block < y.l / block;
    return x.r < y.r;
  }
  void run()
  {
    block = (int)sqrt(n);
    sort(queries.begin(), queries.end(), cmp);
    ans.resize(queries.size());
    int cl = 0, cr = -1, sum = 0;
    auto add = [&](int x)
    {
      sum += x;
    };
    auto rem = [&](int x)
    {
      sum -= x;
    };
    for (int i = 0; i < queries.size(); i++)
    {
      while (cl > queries[i].l)
      {
        cl--;
        add(v[cl]);
      }
      while (cr < queries[i].r)
      {
        cr++;
        add(v[cr]);
      }
      while (cl < queries[i].l)
      {
        rem(v[cl]);
        cl++;
```

```
      }
      while (cr > queries[i].r)
      {
        rem(v[cr]);
        cr--;
      }
      ans[queries[i].idx] = sum;
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> q;
  for (int i = 0; i < n; i++)
    cin >> v[i];
  for (int i = 0; i < q; i++)
  {
    mo::query curr;
    cin >> curr.l >> curr.r;
    curr.r--;
    curr.idx = i;
    mo::queries.pb(curr);
  }
  mo::run();
  for (auto const &i : mo::ans)
    cout << i << endl;
}
// to test: https://judge.yosupo.jp/problem/static_range_sum
```

18.13 mo update

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

int n, q;
int v[MAXN];
int vv[MAXN];

namespace mo
{
  struct query
  {
    int idx, l, r, t;
  };
  struct update
  {
    int i, prevx, x;
  };

  int block;
  vector<query> queries;
  vector<update> updates;
  vector<int> ans;

  bool cmp(query x, query y)
  {
    if (x.l / block != y.l / block)
      return x.l / block < y.l / block;
    if (x.r / block != y.r / block)
      return x.r / block < y.r / block;
    return x.t < y.t;
  }
  void run()
  {
    block = 3153; // (2 * n) ^ 0.666
    sort(queries.begin(), queries.end(), cmp);
    ans.resize(queries.size());
    int cl = 0, cr = -1, sum = 0, t = 0;
    auto add = [&](int x)
    {
      sum += x;
    };
    auto rem = [&](int x)
    {
      sum -= x;
    };
    for (int i = 0; i < queries.size(); i++)
    {
      while (cl > queries[i].l)
      {
        cl--;
        add(v[cl]);
      }
      while (cr < queries[i].r)
      {
        cr++;
        add(v[cr]);
      }
      while (cl < queries[i].l)
      {
        rem(v[cl]);
        cl++;
      }
      while (cr > queries[i].r)
      {
        rem(v[cr]);
        cr--;
      }
      while (t > queries[i].t)
      {
        t--;
        if (queries[i].l <= updates[t].i && queries[i].r >= updates[t].i)
        {
          rem(updates[t].x);
          add(updates[t].prevx);
        }
        v[updates[t].i] = updates[t].prevx;
      }
      while (t < queries[i].t)
      {
        if (queries[i].l <= updates[t].i && queries[i].r >= updates[t].i)
        {
          rem(updates[t].prevx);
          add(updates[t].x);
        }
        v[updates[t].i] = updates[t].x;
        t++;
      }
      ans[queries[i].idx] = sum;
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> q;
  for (int i = 0; i < n; i++)
  {
    cin >> v[i];
    vv[i] = v[i];
  }
  for (int i = 0; i < q; i++)
  {
    int type;
```

```
    cin >> type;
    if (type == 1)
    {
      mo::update curr;
      cin >> curr.i >> curr.x;
      curr.prevx = vv[curr.i];
      vv[curr.i] = curr.x;
      mo::updates.pb(curr);
    }
    else
    {
      mo::query curr;
      cin >> curr.l >> curr.r;
      curr.r--;
      curr.idx = mo::queries.size();
      curr.t = mo::updates.size();
      mo::queries.pb(curr);
    }
  }
  mo::run();
  for (auto const &i : mo::ans)
    cout << i << endl;
}
// to test: https://codeforces.com/edu/course/2/lesson/4/1/practice/contest
    /273169/problem/A
// 1 i v - set the element with index i to v
// 2 l r - calculate the sum of elements with indices from l to r - 1
// n, q <= 100000
// runs in 467ms
```

18.14 persistent seg

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 100003
#define mod 1000000007

int v[MAXN];

namespace seg
{
  struct node
  {
    int item, lazy, lazy_status, l, r;
    node() {}
    node(int l, int r, int lazy, int lazy_status, int item) : l(l), r(r), lazy(
        lazy), lazy_status(lazy_status), item(item) {}
  };

  vector<node> seg;
  vector<int> roots;

  void init()
  {
    seg.resize(1);
  }
  int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a + b;
  }
  int newleaf(int vv)
  {
    int p = seg.size();
    seg.pb(node(0, 0, 0, 0, vv));
    return p;
  }
  int newparent(int l, int r)
  {
    int p = seg.size();
    seg.pb(node(l, r, 0, 0, merge(seg[l].item, seg[r].item)));
    return p;
  }
  int newkid(int i, int diff, int l, int r)
  {
    int p = seg.size();
    seg.pb(node(seg[i].l, seg[i].r, seg[i].lazy + diff, 1, seg[i].item + ((r - l
        + 1) * diff)));
    return p;
  }
  void add(int i, int l, int r)
  {
    if (!seg[i].lazy_status)
      return;
    if (l != r)
    {
      int mid = (l + r) >> 1;
      seg[i].l = newkid(seg[i].l, seg[i].lazy, l, mid);
      seg[i].r = newkid(seg[i].r, seg[i].lazy, mid + 1, r);
    }
    seg[i].lazy = 0;
    seg[i].lazy_status = 0;
  }
  int update(int i, int l, int r, int ql, int qr, int diff)
  {
    if (l > r || l > qr || r < ql)
      return i;
    if (l >= ql && r <= qr)
      return newkid(i, diff, l, r);
    add(i, l, r);
    int mid = (l + r) >> 1;
    return newparent(update(seg[i].l, l, mid, ql, qr, diff), update(seg[i].r,
        mid + 1, r, ql, qr, diff));
  }
  int query(int l, int r, int ql, int qr, int i)
  {
    if (l > r || l > qr || r < ql)
      return neutral();
    if (l >= ql && r <= qr)
      return seg[i].item;
    add(i, l, r);
    int mid = (l + r) >> 1;
    return merge(query(l, mid, ql, qr, seg[i].l), query(mid + 1, r, ql, qr, seg[
        i].r));
  }
  int build(int l, int r)
  {
    if (l == r)
      return newleaf(v[l]);
    int mid = (l + r) >> 1;
    return newparent(build(l, mid), build(mid + 1, r));
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  for (int i = 0; i < n; i++)
    cin >> v[i];
  seg::init();
  int root = seg::build(0, n - 1);
  seg::roots.pb(root);
  while (q--)
  {
```

```
    char t;
    cin >> t;
    if (t == 'C')
    {
      int l, r, d;
      cin >> l >> r >> d;
      l--, r--;
      int root = seg::update(seg::roots.back(), 0, n - 1, l, r, d);
      seg::roots.pb(root);
    }
    else if (t == 'Q')
    {
      int l, r;
      cin >> l >> r;
      l--, r--;
      cout << seg::query(0, n - 1, l, r, seg::roots.back()) << endl;
    }
    else if (t == 'H')
    {
      int l, r, d;
      cin >> l >> r >> d;
      l--, r--;
      cout << seg::query(0, n - 1, l, r, seg::roots[d]) << endl;
    }
    else
    {
      int d;
      cin >> d;
      while (seg::roots.size() > d + 1)
        seg::roots.pop_back();
    }
  }
  return 0;
}
// https://www.spoj.com/problems/TTM/
// rollback segtree to a time stamp t
```

18.15 persistent seg2

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

//#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007
#define PI acos(-1)

struct node
{
  int item, l, r;
  node() {}
  node(int l, int r, int item) : l(l), r(r), item(item) {}
};
int n, q;
vector<node> seg;
vector<int> roots;

void init()
{
  seg.resize(1);
}
int newleaf(int vv)
{
  int p = seg.size();
  seg.pb(node(0, 0, vv));
  return p;
}
int newpar(int l, int r)
{
  int p = seg.size();
  seg.pb(node(l, r, seg[l].item + seg[r].item));
  return p;
}
int upd(int i, int l, int r, int pos)
{
  if (l == r)
    return newleaf(seg[i].item + 1);
  int mid = (l + r) >> 1;
  if (pos <= mid)
    return newpar(upd(seg[i].l, l, mid, pos), seg[i].r);
  return newpar(seg[i].l, upd(seg[i].r, mid + 1, r, pos));
}
int build(int l, int r)
{
  if (l == r)
    return newleaf(0);
  int mid = (l + r) >> 1;
  return newpar(build(l, mid), build(mid + 1, r));
}
int qry(int vl, int vr, int l, int r, int k)
{
  if (l == r)
    return l;
  int mid = (l + r) >> 1;
  int c = seg[seg[vr].l].item - seg[seg[vl].l].item;
  if (c >= k)
    return qry(seg[vl].l, seg[vr].l, l, mid, k);
  return qry(seg[vl].r, seg[vr].r, mid + 1, r, k - c);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> q;
  vector<int> v(n);
  set<int> vals;
  for (int i = 0; i < n; i++)
  {
    cin >> v[i];
    vals.insert(v[i]);
  }
  int mx = 1;
  map<int, int> mp, mpr;
  for (auto const &i : vals)
  {
    mp[i] = mx;
    mpr[mx] = i;
    mx++;
  }
  init();
  roots.pb(build(0, mx));
  for (auto const &i : v)
    roots.pb(upd(roots.back(), 0, mx, mp[i]));
  while (q--)
  {
    char c;
    cin >> c;
    if (c == 'Q')
    {
      int l, r, k;
      cin >> l >> r >> k;
      l--, r--;
      cout << mpr[qry(roots[l], roots[r + 1], 0, mx, k)] << endl;
    }
    else
    {
      int x;
      cin >> x;
      x--;
      swap(v[x], v[x + 1]);
      int a = upd(roots[x], 0, mx, mp[v[x]]);
      int b = upd(a, 0, mx, mp[v[x + 1]]);
```

```
      roots[x + 1] = a, roots[x + 2] = b;
    }
  }
  return 0;
}
// https://neps.academy/br/exercise/127
// queries de k-esimo menor em um range
// e fazer um swap entre v[i] e v[i + 1]
```

18.16 rmq

```
#include <bits/stdc++.h>
using namespace std;

#define lli long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

struct rmq
{
  vector<int> v;
  bool is_max;
  int n;
  static const int b = 30;
  vector<int> mask, t;

  int op(int x, int y)
  {
    if (is_max)
      return v[x] >= v[y] ? x : y;
    return v[x] <= v[y] ? x : y;
  }
  int msb(int x) { return __builtin_clz(1) - __builtin_clz(x); }
  int small(int r, int sz = b) { return r - msb(mask[r] & ((1 << sz) - 1)); }
  rmq() {}
  rmq(vector<int> &v_, bool flag) : v(v_), n(v.size()), mask(n), t(n), is_max(
      flag)
  {
    for (int i = 0, at = 0; i < n; mask[i++] = at |= 1)
    {
      at = (at << 1) & ((1 << b) - 1);
      while (at and op(i - msb(at & -at), i) == i)
        at ^= at & -at;
    }
    for (int i = 0; i < n / b; i++)
      t[i] = small(b * i + b - 1);
    for (int j = 1; (1 << j) <= n / b; j++)
      for (int i = 0; i + (1 << j) <= n / b; i++)
        t[n / b * j + i] = op(t[n / b * (j - 1) + i], t[n / b * (j - 1) + i + (1
            << (j - 1))]);
  }
  int qry(int l, int r)
  {
    if (r - l + 1 <= b)
      return small(r, r - l + 1);
    int x = l / b + 1, y = r / b - 1;
    if (x > y)
      return op(small(l + b - 1), small(r));
    int j = msb(y - x + 1);
    int ans = op(small(l + b - 1), op(t[n / b * j + x], t[n / b * j + y - (1 <<
        j) + 1]));
    return op(ans, small(r));
  }
  int query(int l, int r) { return v[qry(l, r)]; }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n;
  cin >> n;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  rmq r_min(v, 0);
  rmq r_max(v, 1);
  lli ans = 0;
  {
    vector<pi> q;
    q.pb({0, n - 1});
    while (!q.empty())
    {
      int l = q.back().fir;
      int r = q.back().sec;
      int max_pos = r_max.qry(l, r);
      q.pop_back();
      lli qt = (max_pos - l + 1) * 1ll * (r - max_pos + 1);
      ans += (v[max_pos] * 1ll * qt);
      if (max_pos > l)
        q.pb({l, max_pos - 1});
      if (max_pos < r)
        q.pb({max_pos + 1, r});
    }
  }
  {
    vector<pi> q;
    q.pb({0, n - 1});
    while (!q.empty())
    {
      int l = q.back().fir;
      int r = q.back().sec;
      int min_pos = r_min.qry(l, r);
      q.pop_back();
      lli qt = (min_pos - l + 1) * 1ll * (r - min_pos + 1);
      ans -= (v[min_pos] * 1ll * qt);
      if (min_pos > l)
        q.pb({l, min_pos - 1});
      if (min_pos < r)
        q.pb({min_pos + 1, r});
    }
  }
  cout << ans << endl;
  return 0;
}
// https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/Estruturas/rmq.
    cpp
// O(n) pra buildar, query O(1)
// qry(l, r) -> retorna o indice do menor elemento no range [l, r]
// query(l, r) -> retorna o menor elemento no range [l, r]

// problema exemplo: https://codeforces.com/contest/817/problem/D
```

18.17 SegTree

```
#include <bits/stdc++.h>
using namespace std;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define mp make_pair
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 100001
#define MAXL 100
#define mod 1000000007

vector<int> seg;
vector<int> v;

int single(int x)
{
```

```
  return x;
}
int neutral()
{
  return 0;
}
int merge(int a, int b)
{
  return a + b;
}
void update(int i, int l, int r, int q, int x)
{
  if (l == r)
  {
    seg[i] = single(x);
    return;
  }
  int mid = (l + r) >> 1;
  if (q <= mid)
    update(i << 1, l, mid, q, x);
  else
    update((i << 1) | 1, mid + 1, r, q, x);
  seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
}
int query(int l, int r, int ql, int qr, int i)
{
  int mid = (l + r) >> 1;
  if (l > r || l > qr || r < ql)
    return neutral();
  if (l >= ql && r <= qr)
    return seg[i];
  return merge(query(l, mid, ql, qr, i << 1), query(mid + 1, r, ql, qr, (i << 1)
      | 1));
}
void build(int l, int r, int i)
{
  if (l == r)
  {
    seg[i] = single(v[l]);
    return;
  }
  int mid = (l + r) >> 1;
  build(l, mid, i << 1);
  build(mid + 1, r, (i << 1) | 1);
  seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  v.resize(n);
  seg.resize(4 * n);
  for (int i = 0; i < n; i++)
    cin >> v[i];
  build(0, n - 1, 1);
  while (q--)
  {
    int l, r;
    int t;
    cin >> t >> l >> r;
    if (t == 2)
      cout << query(0, n - 1, l, r - 1, 1) << endl;
    else
      update(1, 0, n - 1, l, r);
  }
}
```

18.18 Segtree2

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 500001
#define mod 1000000007

struct segtree
{
  int n;
  vector<int> seg;

  int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a + b;
  }
  void build(vector<int> &v)
  {
    n = 1;
    while (n < v.size())
      n <<= 1;
    seg.assign(n << 1, neutral());
    for (int i = 0; i < v.size(); i++)
      seg[i + n] = v[i];
    for (int i = n - 1; i; i--)
      seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  void upd(int i, int value)
  {
    seg[i += n] += value;
    for (i >>= 1; i; i >>= 1)
      seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int qry(int l, int r)
  {
    int ansl = neutral(), ansr = neutral();
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1)
    {
      if (l & 1)
        ansl = merge(ansl, seg[l++]);
      if (r & 1)
        ansr = merge(seg[--r], ansr);
    }
    return merge(ansl, ansr);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  return 0;
}
// iterative segtree without lazy propagation
```

18.19 segtree2d

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
```

```cpp
template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1003
#define mod 1000000007

struct segtree2d
{
  int n, m;
  vector<vector<int>> seg;

  int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a + b;
  }
  segtree2d(int nn, int mm)
  {
    n = nn, m = mm;
    seg = vector<vector<int>>(2 * n, vector<int>(2 * m, neutral()));
  }
  int qry(int x1, int y1, int x2, int y2)
  {
    int ret = neutral();
    int y3 = y1 + m, y4 = y2 + m;
    for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /= 2)
    {
      for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2 /= 2)
      {
        if (x1 % 2 == 1 and y1 % 2 == 1)
          ret = merge(ret, seg[x1][y1]);
        if (x1 % 2 == 1 and y2 % 2 == 0)
          ret = merge(ret, seg[x1][y2]);
        if (x2 % 2 == 0 and y1 % 2 == 1)
          ret = merge(ret, seg[x2][y1]);
        if (x2 % 2 == 0 and y2 % 2 == 0)
          ret = merge(ret, seg[x2][y2]);
      }
    }
    return ret;
  }
  void upd(int x, int y, int val)
  {
    int y2 = y += m;
    for (x += n; x; x /= 2, y = y2)
    {
      if (x >= n)
        seg[x][y] = val;
      else
        seg[x][y] = merge(seg[2 * x][y], seg[2 * x + 1][y]);
      while (y /= 2)
        seg[x][y] = merge(seg[x][2 * y], seg[x][2 * y + 1]);
    }
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int q;
  cin >> q;
  while (q--)
  {
    int n;
    cin >> n;
    segtree2d st(n, n); // matriz NxN
    while (1)
    {
      string s;
      cin >> s;
      if (s == "SET")
      {
        int a, b, c;
        cin >> a >> b >> c;
        st.upd(a, b, c);
      }
      else if (s == "SUM")
      {
        int a, b, c, d;
        cin >> a >> b >> c >> d; // c >= a e d >= b
        cout << st.qry(a, b, c, d) << endl;
      }
      else
      {
        break;
      }
    }
  }
  return 0;
}
// to test: https://www.spoj.com/problems/MATSUM/
```

18.20 segtree lazy

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 200005
#define mod 998244353

struct segtree
{
  int n;
  vector<int> v;
  vector<int> seg;
  vector<int> lazy;

  segtree(int sz)
  {
    n = sz;
    seg.assign(4 * n, 0);
    lazy.assign(4 * n, 0);
    // v = vv; // for build
    // build(0, n - 1, 1); // for build
  }
  int single(int x)
  {
    return x;
  }
  int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a + b;
  }
  void add(int i, int l, int r, int diff)
```

```
  {
    seg[i] += (r - l + 1) * diff;
    if (l != r)
    {
      lazy[i << 1] += diff;
      lazy[(i << 1) | 1] += diff;
    }
    lazy[i] = 0;
  }
  void update(int i, int l, int r, int ql, int qr, int diff)
  {
    if (lazy[i])
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return;
    if (l >= ql && r <= qr)
    {
      add(i, l, r, diff);
      return;
    }
    int mid = (l + r) >> 1;
    update(i << 1, l, mid, ql, qr, diff);
    update((i << 1) | 1, mid + 1, r, ql, qr, diff);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int query(int l, int r, int ql, int qr, int i)
  {
    if (lazy[i])
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return neutral();
    if (l >= ql && r <= qr)
      return seg[i];
    int mid = (l + r) >> 1;
    return merge(query(l, mid, ql, qr, i << 1), query(mid + 1, r, ql, qr, (i <<
        1) | 1));
  }
  void build(int l, int r, int i)
  {
    if (l == r)
    {
      seg[i] = single(v[l]);
      return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, i << 1);
    build(mid + 1, r, (i << 1) | 1);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int qry(int l, int r)
  {
    return query(0, n - 1, l, r, 1);
  }
  void upd(int l, int r, int x)
  {
    update(1, 0, n - 1, l, r, x);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  segtree s(n);
  while (q--)
  {
    int t;
    cin >> t;
    if (t == 1)
    {
      int l, r, x;
      cin >> l >> r >> x;
      s.upd(l, r, x);
    }
    else
    {
      int l, r;
      cin >> l >> r;
      cout << s.qry(l, r) << endl;
    }
  }
  return 0;
}
```

18.21 segtree max seg sum

```
#include <bits/stdc++.h>
using namespace std;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 200005
#define mod 1000000007

struct item
{
  int sg_max, pref_max, suf_max, sg_min, pref_min, suf_min, sum;
};
struct segtree
{
  int n;
  vector<item> seg;

  item single(int x)
  {
    return {max(0ll, x), max(0ll, x), max(0ll, x), min(0ll, x), min(0ll, x), min
        (0ll, x), x};
  }
  item neutral()
  {
    return {0, 0, 0, 0, 0, 0, 0};
  }
  segtree() {}
  segtree(int sz)
  {
    n = sz;
    seg.assign(4 * n, neutral());
  }
  item merge(item a, item b)
  {
    item ret;
    ret.sg_max = max({a.sg_max, b.sg_max, a.suf_max + b.pref_max});
    ret.pref_max = max(a.pref_max, a.sum + b.pref_max);
    ret.suf_max = max(b.suf_max, b.sum + a.suf_max);
    ret.sg_min = min({a.sg_min, b.sg_min, a.suf_min + b.pref_min});
    ret.pref_min = min(a.pref_min, a.sum + b.pref_min);
    ret.suf_min = min(b.suf_min, b.sum + a.suf_min);
    ret.sum = a.sum + b.sum;
    return ret;
  }
  void update(int i, int l, int r, int q, int x)
  {
    if (l == r)
    {
      seg[i] = single(x);
      return;
    }
    int mid = (l + r) >> 1;
    if (q <= mid)
      update(i << 1, l, mid, q, x);
    else
      update((i << 1) | 1, mid + 1, r, q, x);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  item query(int l, int r, int ql, int qr, int i)
```

```
  {
    int mid = (l + r) >> 1;
    if (l > r || l > qr || r < ql)
      return neutral();
    if (l >= ql && r <= qr)
      return seg[i];
    return merge(query(l, mid, ql, qr, i << 1), query(mid + 1, r, ql, qr, (i <<
        1) | 1));
  }
};
signed main()
{
  return 0;
}
// segtree for maximum segment sum
// me enrolo pra codar toda vez, e bom deixar na lib
```

18.22 SegTree pa

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 100005
#define mod 1000000007

struct lazy_node
{
  int n, a, d;

  int sum()
  {
    int an = a + (d * (n - 1));
    return ((a + an) * n) >> 1;
  }
  void merge(lazy_node to_add)
  {
    a += to_add.a;
    d += to_add.d;
  }
};
struct segtree
{
  vector<int> seg;
  vector<lazy_node> lazy;
  vector<bool> lazy_status;

  segtree(int n)
  {
    seg.resize(4 * n);
    lazy.resize(4 * n);
    lazy_status.resize(4 * n);
    build(0, n - 1, 1);
  }
  int single(int x)
  {
    return x;
  }
  int neutral()
  {
    return 0;
  }
  int merge(int a, int b)
  {
    return a + b;
  }
  void add(int i, int l, int r, lazy_node to_add)
  {
    seg[i] += to_add.sum();
    if (l != r)
    {
      int mid = (l + r) >> 1;
      lazy[i << 1].merge({mid - l + 1, to_add.a, to_add.d});
      lazy_status[i << 1] = 1;
      int diff = (mid + 1) - l, a = to_add.a, d = to_add.d;
      lazy[(i << 1) | 1].merge({r - (mid + 1) + 1, a + (d * diff), d});
      lazy_status[(i << 1) | 1] = 1;
    }
    lazy[i] = {r - l + 1, 0, 0};
    lazy_status[i] = 0;
  }
  void update(int i, int l, int r, int ql, int qr, lazy_node to_add)
  {
    if (lazy_status[i])
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return;
    if (l >= ql && r <= qr)
    {
      int diff = l - ql, a = to_add.a, d = to_add.d;
      lazy_node curr = {r - l + 1, a + (d * diff), d};
      add(i, l, r, curr);
      return;
    }
    int mid = (l + r) >> 1;
    update(i << 1, l, mid, ql, qr, to_add);
    update((i << 1) | 1, mid + 1, r, ql, qr, to_add);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int query(int l, int r, int ql, int qr, int i)
  {
    if (lazy_status[i])
      add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql)
      return neutral();
    if (l >= ql && r <= qr)
      return seg[i];
    int mid = (l + r) >> 1;
    return merge(query(l, mid, ql, qr, i << 1), query(mid + 1, r, ql, qr, (i <<
        1) | 1));
  }
  void build(int l, int r, int i)
  {
    seg[i] = 0;
    lazy_status[i] = 0;
    lazy[i] = {r - l + 1, 0, 0};
    if (l == r)
      return;
    int mid = (l + r) >> 1;
    build(l, mid, i << 1);
    build(mid + 1, r, (i << 1) | 1);
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int n, q;
  cin >> n >> q;
  segtree s(n);
  while (q--)
  {
    int t;
    cin >> t;
    if (t == 1)
    {
      int l, r, a, d;
      cin >> l >> r >> a >> d;
      l--, r--;
      s.update(1, 0, n - 1, l, r, {r - l + 1, a, d});
```

```
    }
    else
    {
      int x;
      cin >> x;
      x--;
      cout << s.query(0, n - 1, x, x, 1) << endl;
    }
  }
  return 0;
}
// queries of:
// add an arithmetic progression to a segment [l, r]
// print current value of a given element
```

18.23 sparsetable

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define PI acos(-1)
#define pb push_back
#define int long long int
#define pi pair<int, int>
#define pii pair<int, pair<int, pi>>
#define fir first
#define sec second
#define DEBUG 0
#define MAXN 10005
#define mod 1000000007

struct rmq
{
  bool is_min;
  vector<vector<int>> st;
  vector<int> log;

  int f(int a, int b) { return (is_min) ? min(a, b) : max(a, b); }
  int qry(int l, int r) { return f(st[l][log[r - l + 1]], st[r - (1 << log[r - l
      + 1]) + 1][log[r - l + 1]]); }
  rmq(vector<int> &v, bool flag)
  {
    is_min = flag;
    int n = v.size();
    log.resize(n + 1);
    log[1] = 0;
    for (int i = 2; i <= n; i++)
      log[i] = log[i / 2] + 1;
    int m = log[n] + 2;
    st.assign(n + 1, vector<int>(m, 0));
    for (int i = 0; i < n; i++)
      st[i][0] = v[i];
    for (int j = 1; j < m; j++)
    {
      for (int i = 0; i + (1 << j) <= n; i++)
        st[i][j] = f(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
  }
};
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
}
```

18.24 treap

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// #define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000005
#define mod 1000000007

struct treap
{
  int data, priority;
  int sz, lazy2;
  bool lazy;
  treap *l, *r, *parent;
};
int size(treap *node)
{
  return (!node) ? 0 : node->sz;
}
void recalc(treap *node)
{
  if (!node)
    return;
  node->sz = 1;
  node->parent = 0;
  if (node->l)
    node->sz += node->l->sz, node->l->parent = node;
  if (node->r)
    node->sz += node->r->sz, node->r->parent = node;
}
void lazy_propagation(treap *node)
{
  if (node == NULL)
    return;
  if (node->lazy2)
  {
    if (node->l)
      node->l->lazy2 += node->lazy2;
    if (node->r)
      node->r->lazy2 += node->lazy2;
    node->data += node->lazy2;
    node->lazy2 = 0;
  }
  if (node->lazy)
  {
    swap(node->l, node->r);
    if (node->l)
      node->l->lazy = !node->l->lazy;
    if (node->r)
      node->r->lazy = !node->r->lazy;
    node->lazy = 0;
  }
}
void split(treap *t, treap *&l, treap *&r, int n)
{
  if (!t)
    return void(l = r = 0);
  lazy_propagation(t);
  if (size(t->l) >= n)
    split(t->l, l, t->l, n), r = t;
  else
    split(t->r, t->r, r, n - size(t->l) - 1), l = t;
  recalc(t);
}
```

```
void merge(treap *&t, treap *l, treap *r)
{
  lazy_propagation(l);
  lazy_propagation(r);
  if (!l)
    t = r;
  else if (!r)
    t = l;
  else if (l->priority > r->priority)
    merge(l->r, l->r, r), t = l;
  else
    merge(r->l, l, r->l), t = r;
  recalc(t);
}
void troca(treap *&t, int l, int r, int ll, int rr)
{
  treap *a0, *a1, *b0, *b1, *c0, *c1, *d0, *d1;
  split(t, a0, a1, l);
  split(a1, b0, b1, r - l + 1);
  ll -= (r + 1);
  rr -= (r + 1);
  split(b1, c0, c1, ll);
  split(c1, d0, d1, rr - ll + 1);
  merge(t, a0, d0);
  merge(t, t, c0);
  merge(t, t, b0);
  merge(t, t, d1);
}
void add(treap *&t, int l, int r)
{
  treap *a0, *a1, *b0, *b1;
  split(t, a0, a1, l);
  split(a1, b0, b1, r - l + 1);
  b0->lazy ^= 1;
  b0->lazy2 += 1;
  merge(t, a0, b0);
  merge(t, t, b1);
}
void solve(int x)
{
  x = x % 26;
  char c = x + 'a';
  cout << c;
}
void dfs(treap *t)
{
  if (!t)
    return;
  lazy_propagation(t);
  dfs(t->l);
  solve(t->data);
  dfs(t->r);
}
treap *create_node(int data, int priority)
{
  treap *ret = new treap;
  ret->data = data;
  ret->priority = priority;
  ret->l = 0;
  ret->r = 0;
  ret->sz = 1;
  ret->lazy = 0;
  ret->lazy2 = 0;
  ret->parent = 0;
  return ret;
}
void goup(treap *&ans, treap *t) // vai pra raiz da arvore
{
  if (!t->parent)
  {
    ans = t;
    return;
  }
  goup(ans, t->parent);
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  srand(time(NULL));
  int q;
  cin >> q;
  while (q--)
  {
    int n, m;
    string s;
    cin >> s >> m;
    n = s.size();
    treap *t = 0;
    for (auto const &i : s)
    {
      int x = i - 'a';
      merge(t, t, create_node(x, rand()));
    }
    while (m--)
    {
      int a, b, c, d;
      cin >> a >> b >> c >> d;
      a--, b--, c--, d--;
      add(t, a, b);
      add(t, c, d);
      troca(t, a, b, c, d);
    }
    dfs(t);
    cout << endl;
  }
  return 0;
}
// https://vjudge.net/contest/478186#problem/E
// - lazy propagation
// - reverse range with lazy propagation
// - swap ranges with equal lenght

// extra:
// - save node parent
```

18.25 treap2

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define endl '\n'
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 1000005
#define mod 1000000007

vector<int> ans;

struct treap
{
  int data, priority;
  int sz;
  bool lazy;
  treap *l, *r;
};
int size(treap *node)
{
  return (!node) ? 0 : node->sz;
}
void recalc(treap *node)
{
  if (!node)
```

```
    return;
  node->sz = 1;
  if (node->l)
    node->sz += node->l->sz;
  if (node->r)
    node->sz += node->r->sz;
}
void lazy_propagation(treap *node)
{
  if (!node || !(node->lazy))
    return;
  swap(node->l, node->r);
  if (node->l)
    node->l->lazy ^= 1;
  if (node->r)
    node->r->lazy ^= 1;
  node->lazy = 0;
}
void merge(treap *&t, treap *l, treap *r)
{
  lazy_propagation(l);
  lazy_propagation(r);
  if (!l)
    t = r;
  else if (!r)
    t = l;
  else if (l->priority > r->priority)
    merge(l->r, l->r, r), t = l;
  else
    merge(r->l, l, r->l), t = r;
  recalc(t);
}
void split(treap *t, treap *&l, treap *&r, int n)
{
  if (!t)
    return void(l = r = 0);
  lazy_propagation(t);
  if (size(t->l) >= n)
    split(t->l, l, t->l, n), r = t;
  else
    split(t->r, t->r, r, n - size(t->l) - 1), l = t;
  recalc(t);
}
void reverse(treap *&t, int l, int r)
{
  treap *a0, *a1, *b0, *b1;
  split(t, a0, a1, l);
  split(a1, b0, b1, r - l + 1);
  b0->lazy ^= 1;
  merge(t, a0, b0);
  merge(t, t, b1);
}
void shift(treap *&t, int l, int r)
{
  treap *a0, *a1, *b0, *b1, *c0, *c1;
  split(t, a0, a1, l);
  split(a1, b0, b1, r - l + 1);
  split(b0, c0, c1, r - l);
  merge(t, a0, c1);
  merge(t, t, c0);
  merge(t, t, b1);
}
void dfs(treap *t)
{
  if (!t)
    return;
  lazy_propagation(t);
  dfs(t->l);
  ans.pb(t->data);
  dfs(t->r);
}
treap *create_node(int data, int priority)
{
  treap *ret = new treap;
  ret->data = data;
  ret->priority = priority;
  ret->l = 0;
  ret->r = 0;
  ret->sz = 1;
  ret->lazy = 0;
  return ret;
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  srand(time(NULL));
  treap *t = 0;
  int n, m, q;
  cin >> n >> q >> m;
  for (int i = 0; i < n; i++)
  {
    int k;
    cin >> k;
    merge(t, t, create_node(k, rand()));
  }
  while (q--)
  {
    int ty, l, r;
    cin >> ty >> l >> r;
    l--, r--;
    (ty == 1) ? shift(t, l, r) : reverse(t, l, r);
  }
  dfs(t);
  while (m--)
  {
    int i;
    cin >> i;
    i--;
    cout << ans[i] << " ";
  }
  cout << endl;
  return 0;
}
```

19 Theorems and Formulas

19.1 binomial theorem

```
## Binomial Theorem

### Theorem

$$
(x + y)^n = \sum_{k = 0}^{n} {n \choose k} x^{n - k}y^{k}
$$

in addition, we have:

$$
(x - y)^n = \sum_{k = 0}^{n} (-1)^k {n \choose k} x^{n - k}y^{k}
$$

$$
(1 + x)^n = \sum_{k = 0}^{n} {n \choose k} x^{k}
$$

### Cool Problem

[Fibonacci Fever](https://codeforces.com/gym/104412/problem/F)

Given n and k you're asked to compute (mod $10^9 + 7$):

$$
\sum\_{i = 1}^{n}f_i^k
$$

where $f_n$ is the n-th fibonacci number.

Recall that:

$$
```

```
f_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n -  \frac{1}{\sqrt
    {5}} \left(\frac{1-\sqrt{5}}{2}\right)^n
$$

This is something like that:

$$
f_n = ca^n - cb^n
$$

In the end, we end up with something like:

$$
c^k (\sum_{j = 0}^{k} (-1)^{j} {k \choose j} \sum_{i = 1}^{n} (a^{k - j}b^{j})^{
    i} )
$$

PS: To handle $\sqrt{5}$ mod $10^9 + 7$, we need to store each number in the
    form $x = a + b\sqrt{5}$

[Code](https://github.com/jonh14lk/Competitive_Programming/blob/master/Math/
    binomial_theorem.cpp)
```

19.2 chicken mcnugget

```
## Chicken McNugget Theorem

For any two coprime numbers (n > 0, m > 0), the greatest integer that cannot be
    written in the form:

an + bm, (a >= 0, b >= 0)

is (n \* m) - n - m

## Consequence of the theorem

That there are exactly ((n - 1) \* (m - 1)) / 2 positive integers which cannot
    be expressed in the form an + bm, (a >= 0, b >= 0)

## Generalization

If n and m are not coprime, so all numbers that are not multiples of gcd(n, m)
    cannot be expressed in the form an + bm, (a >= 0, b >= 0)

in addition, you can consider n = (n / gcd(n, m)) and m = (m / gcd(n, m)), to
    find how many multiples of gcd(n, m) cannot be expressed, or to find the
    greatest multiple of gcd(n, m) that cannot be expressed

## Considering a > 0, b > 0

Considering (n > 0, m > 0), n and m are coprime:

let y = ((n \* m) + min(n, m)) - 1

The number of positive integers which cannot be expressed increases by (y / n)

The number of positive integers which cannot be expressed increases by (y / m)

you must not count the multiples of (n \* m) more than once, just decrease
    number of positive integers which cannot be expressed by (y / (n \* m))

## Problems

- [Forming Compounds](https://codeforces.com/group/XrhoJtxCjm/contest/422716/
    problem/I)
```

19.3 graph notes

```
## Bipartite Graph

A bipartite graph is a graph that does not contain any odd-length cycles.

## Directed acyclic graph (DAG)

Is a directed graph with no directed cycles.

## Independent Set

Is a set of vertices in a graph, no two of which are adjacent. That is, it is a
    set S of vertices such that for every two vertices in S, there is no edge
    connecting the two.

## Clique

Is a subset of vertices of an undirected graph such that every two distinct
    vertices in the clique are adjacent.

## Vertex Cover

Is a set of vertices that includes at least one endpoint of every edge of the
    graph.

## Edge Cover

Is a set of edges such that every vertex of the graph is incident to at least
    one edge of the set.

## Path Cover

Given a directed graph G = (V, E), a path cover is a set of directed paths such
    that every vertex v belongs to at least one path.

## Koning's Theorem

In any bipartite graph, the number of edges in a maximum matching equals the
    number of vertices in a minimum vertex cover.

## Properties

- Every tree is a bipartite graph.
- Any NxM grid is a bipartite graph.
- A set of vertices is a vertex cover if and only if its complement is an
    independent set.
- The number of vertices of a graph is equal to its minimum vertex cover number
    plus the size of a maximum independent set.
- In bipartite graphs, the size of the minimum edge cover is equal to the size
    of the maximum independent set
- In bipartite graphs, the size of the minimum edge cover plus the size of the
    minimum vertex cover is equal to the number of vertices.
- In bipartite graphs, maximum clique size is two.

## Min-cut

The smallest total weight of the edges which if removed would disconnect the
    source from the sink.

## Max-flow min-cut theorem

In a flow network, the maximum amount of flow passing from the source to the
    sink is equal to the total weight of the edges in a minimum cut.

## Maximum flow with vertex capacities

In other words, the amount of flow passing through a vertex cannot exceed its
    capacity. To find the maximum flow, we can transform the problem into the
    maximum flow problem by expanding the network. Each vertex v is replaced by
     v-in and v-out, where v-in is connected by edges going into v and v-out is
     connected to edges coming out from v. Then assign capacity c(v) to the
    edge connecting v-in and v-out.

## Undirected edge-disjoint paths problem

We are given an undirected graph G = (V, E) and two vertices s and t, and we
    have to find the maximum number of edge-disjoint s-t paths in G.

## Undirected vertex-disjoint paths problem

We are given an undirected graph G = (V, E) and two vertices s and t, and we
    have to find the maximum number of vertex-disjoint (except for s and t)
    paths in G.
```

```
## Menger's theorem

The maximum number of edge-disjoint s-t paths in an undirected graph is equal to
     the minimum number of edges in an s-t cut-set.

## Undirected vertex-disjoint paths solution

We can construct a network N=(V,E) from G with vertex capacities, where the
     capacities of all vertices and all edges are 1. Then the value of the
     maximum flow is equal to the maximum number of independent paths from s to
     t.

## Minimum vertex-disjoint path cover in directed acyclic graph (DAG)

Given a directed acyclic graph G=(V, E), we are to find the minimum number of
     vertex-disjoint paths to cover each vertex in V. We can construct a
     bipartite graph G' from G. Each vertex v is replaced by v-in and v-out,
     where v-in is connected by edges going into v and v-out is connected to
     edges coming out from v. Then it can be shown that G' has a matching M of
     size m if and only if G has a vertex-disjoint path cover C of containing m
     edges and n-m paths.

## Minimum general path cover in directed acyclic graph (DAG)

A general path cover is a path cover where a vertex can belong to more than one
     path. A minimum general path cover may be smaller than a minimum vertex-
     disjoint path cover. A minimum general path cover can be found almost like
     a minimum vertex-disjoint path cover. It suffices to add some new edges to
     the matching graph so that there is an edge a - b always when there is a
     path from a to b in the original graph.

## Dilworth's theorem and maximum antichain

An antichain is a set of nodes of a graph such that there is no path from any
     node to another node using the edges of the graph. Dilworth's theorem
     states that in a directed acyclic graph, the size of a minimum general path
      cover equals the size of a maximum antichain.

Or in other words: For a DAG G that if has edges from vertex i -> vertex j and
     vertex j -> k, then it also has a edge from vertex i -> vertex k, the size
     of a minimum path cover is equal to the size of a maximum independent set.

## Maximum weighted antichain

(https://atcoder.jp/contests/abc354/tasks/abc354_g) In this problem, each vertex
      has a cost a[i]. The cost of an antichain is equal to the sum of the costs
      of the vertices present in it. We need to find the maximum cost of a
     antichain. We can construct the same bipartite of the maximum antichain
     problem from a dag G, these edges have an infinite capacity. We also need
     to create a source vertex and a sink, and we need to add edges source -> v-
     in with capacity a[v] and v-out -> sink with capacity a[v]. The answer is
     equal to the sum of all a[i] minus the maximum flow on this network.

## Hall's Theorem

Hall's theorem can be used to find out whether a bipartite graph has a matching
     that contains all left or right nodes. Assume that we want to find a
     matching that contains all left nodes. Let X be any set of left nodes and
     let f(X) be the set of their neighbors. According to Hall's theorem, a
     matching that contains all left nodes exists exactly when for
each X, the condition |X| <= |f(X)| holds.

## References

- [Competitive Programmer's Handbook](https://cses.fi/book/book.pdf)
- [(Graph Theory) - Wikipedia](https://en.wikipedia.org/wiki/Graph_theory)
- [(Medium Article) - Solving Minimum Path Cover on a DAG](https://
     towardsdatascience.com/solving-minimum-path-cover-on-a-dag-21b16ca11ac0)

## Extra (Getting Confidence Trick)

[2019-2020 ACM-ICPC Brazil Subregional Programming Contest, problem G](https://
     codeforces.com/gym/102346/problem/G)

<p>If you need to maximize a number x = (a * b * c * ...), then you can write it
      as x = (e^log(a) * e^log(b) * e^log(c) * ...), and then the number is x =
     e^(log(a) + log(b) + log(c) + ...), and the problem now becomes a problem
     of maximizing the sum of (log(a) + log(b) + log(c) + ...).<p/>

Use exp() and log() C++ functions :)
```

19.4 manhattan and chebyshev

```
## Manhattan and Chebyshev distances equivalences

It is well known that given points (x, y) and you need to calculate the
     Manhattan distances between them, instead of using:
|x1-x2|+|y1-y2|

you can first convert all points (x, y) into (x+y, x-y) (rotate 45 degrees) and
     the distances will become max(|x1-x2|, |y1-y2|) (also known as Chebyshev
     distance).

### Problems:

[Manhattan Triangle](https://codeforces.com/contest/1979/problem/E)
[https://atcoder.jp/contests/abc366/tasks/abc366_e](https://atcoder.jp/contests/
     abc366/tasks/abc366_e)
```

20 ufmg forked

21 ufmg forked/DP

21.1 dcDp

```cpp
// Divide and Conquer DP
//
// Particiona o array em k subarrays
// minimizando o somatorio das queries
//
// O(k n log n), assumindo quer query(l, r) eh O(1)

ll dp[MAX][2];

void solve(int k, int l, int r, int lk, int rk) {
        if (l > r) return;
        int m = (l+r)/2, p = -1;
        auto& ans = dp[m][k&1] = LINF;
        for (int i = max(m, lk); i <= rk; i++) {
                ll at = dp[i+1][~k&1] + query(m, i);
                if (at < ans) ans = at, p = i;
        }
        solve(k, l, m-1, lk, p), solve(k, m+1, r, p, rk);
}

ll DC(int n, int k) {
        dp[n][0] = dp[n][1] = 0;
        for (int i = 0; i < n; i++) dp[i][0] = LINF;
        for (int i = 1; i <= k; i++) solve(i, 0, n-i, 0, n-i);
        return dp[0][k&1];
}
```

21.2 lcs

```cpp
// Longest Common Subsequence
//
// Computa a LCS entre dois arrays usando
// o algoritmo de Hirschberg para recuperar
//
// O(n*m), O(n+m) de memoria

int lcs_s[MAX], lcs_t[MAX];
```

```
int dp[2][MAX];

// dp[0][j] = max lcs(s[li...ri], t[lj, lj+j])
void dp_top(int li, int ri, int lj, int rj) {
        memset(dp[0], 0, (rj-lj+1)*sizeof(dp[0][0]));
        for (int i = li; i <= ri; i++) {
                for (int j = rj; j >= lj; j--)
                        dp[0][j - lj] = max(dp[0][j - lj],
                        (lcs_s[i] == lcs_t[j]) + (j > lj ? dp[0][j-1 - lj] : 0))
                                ;
                for (int j = lj+1; j <= rj; j++)
                        dp[0][j - lj] = max(dp[0][j - lj], dp[0][j-1 -lj]);
        }
}

// dp[1][j] = max lcs(s[li...ri], t[lj+j, rj])
void dp_bottom(int li, int ri, int lj, int rj) {
        memset(dp[1], 0, (rj-lj+1)*sizeof(dp[1][0]));
        for (int i = ri; i >= li; i--) {
                for (int j = lj; j <= rj; j++)
                        dp[1][j - lj] = max(dp[1][j - lj],
                        (lcs_s[i] == lcs_t[j]) +  (j < rj ? dp[1][j+1 - lj] : 0)
                                );
                for (int j = rj-1; j >= lj; j--)
                        dp[1][j - lj] = max(dp[1][j - lj], dp[1][j+1 - lj]);
        }
}

void solve(vector<int>& ans, int li, int ri, int lj, int rj) {
        if (li == ri){
                for (int j = lj; j <= rj; j++)
                        if (lcs_s[li] == lcs_t[j]){
                                ans.push_back(lcs_t[j]);
                                break;
                        }
                return;
        }
        if (lj == rj){
                for (int i = li; i <= ri; i++){
                        if (lcs_s[i] == lcs_t[lj]){
                                ans.push_back(lcs_s[i]);
                                break;
                        }
                }
                return;
        }
        int mi = (li+ri)/2;
        dp_top(li, mi, lj, rj), dp_bottom(mi+1, ri, lj, rj);

        int j_ = 0, mx = -1;

        for (int j = lj-1; j <= rj; j++) {
                int val = 0;
                if (j >= lj) val += dp[0][j - lj];
                if (j < rj) val += dp[1][j+1 - lj];

                if (val >= mx) mx = val, j_ = j;
        }
        if (mx == -1) return;
        solve(ans, li, mi, lj, j_), solve(ans, mi+1, ri, j_+1, rj);
}

vector<int> lcs(const vector<int>& s, const vector<int>& t) {
        for (int i = 0; i < s.size(); i++) lcs_s[i] = s[i];
        for (int i = 0; i < t.size(); i++) lcs_t[i] = t[i];
        vector<int> ans;
        solve(ans, 0, s.size()-1, 0, t.size()-1);
        return ans;
}
```

21.3 mochila

```
// Mochila
//
// Resolve mochila, recuperando a resposta
//
// O(n * cap), O(n + cap) de memoria

int v[MAX], w[MAX]; // valor e peso
int dp[2][MAX_CAP];

// DP usando os itens [l, r], com capacidade = cap
void get_dp(int x, int l, int r, int cap) {
        memset(dp[x], 0, (cap+1)*sizeof(dp[x][0]));
        for (int i = l; i <= r; i++) for (int j = cap; j >= 0; j--)
                if (j - w[i] >= 0) dp[x][j] = max(dp[x][j], v[i] + dp[x][j - w[i
                    ]]);
}

void solve(vector<int>& ans, int l, int r, int cap) {
        if (l == r) {
                if (w[l] <= cap) ans.push_back(l);
                return;
        }
        int m = (l+r)/2;
        get_dp(0, l, m, cap), get_dp(1, m+1, r, cap);
        int left_cap = -1, opt = -INF;
        for (int j = 0; j <= cap; j++)
                if (int at = dp[0][j] + dp[1][cap - j]; at > opt)
                        opt = at, left_cap = j;
        solve(ans, l, m, left_cap), solve(ans, m+1, r, cap - left_cap);
}

vector<int> knapsack(int n, int cap) {
        vector<int> ans;
        solve(ans, 0, n-1, cap);
        return ans;
}
```

21.4 sosDP

```
// SOS DP [nohash]
//
// O(n 2^n)

// soma de sub-conjunto
vector<ll> sos_dp(vector<ll> f) {
        int N = __builtin_ctz(f.size());
        assert((1<<N) == f.size());

        for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N); mask++)
                if (mask>>i&1) f[mask] += f[mask^(1<<i)];
        return f;
}

// soma de super-conjunto
vector<ll> sos_dp(vector<ll> f) {
        int N = __builtin_ctz(f.size());
        assert((1<<N) == f.size());

        for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N); mask++)
                if (~mask>>i&1) f[mask] += f[mask^(1<<i)];
        return f;
}
```

21.5 subsetSum

```
// Subset sum
//
// Retorna max(x <= t tal que existe subset de w que soma x)
//
// O(n * max(w))
// O(max(w)) de memoria

int subset_sum(vector<int> w, int t) {
        int pref = 0, k = 0;
        while (k < w.size() and pref + w[k] <= t) pref += w[k++];
        if (k == w.size()) return pref;
```

```
        int W = *max_element(w.begin(), w.end());
        vector<int> last, dp(2*W, -1);
        dp[W - (t-pref)] = k;
        for (int i = k; i < w.size(); i++) {
                last = dp;
                for (int x = 0; x < W; x++) dp[x+w[i]] = max(dp[x+w[i]], last[x
                    ]);
                for (int x = 2*W - 1; x > W; x--)
                        for (int j = max(0, last[x]); j < dp[x]; j++)
                                dp[x-w[j]] = max(dp[x-w[j]], j);
        }
        int ans = t;
        while (dp[W - (t-ans)] < 0) ans--;
        return ans;
}
```

22 ufmg forked/Estruturas

22.1 bit

```
// BIT
//
// BIT de soma 0-based
//
// upper_bound(x) retorna o menor p tal que pref(p) > x
//
// Complexidades:
// build - O(n)
// update - O(log(n))
// query - O(log(n))
// upper_bound - O(log(n))

struct Bit {
        int n;
        vector<ll> bit;
        Bit(int _n=0) : n(_n), bit(n + 1) {}
        Bit(vector<int>& v) : n(v.size()), bit(n + 1) {
                for (int i = 1; i <= n; i++) {
                        bit[i] += v[i - 1];
                        int j = i + (i & -i);
                        if (j <= n) bit[j] += bit[i];
                }
        }
        void update(int i, ll x) { // soma x na posicao i
                for (i++; i <= n; i += i & -i) bit[i] += x;
        }
        ll pref(int i) { // soma [0, i]
                ll ret = 0;
                for (i++; i; i -= i & -i) ret += bit[i];
                return ret;
        }
        ll query(int l, int r) {  // soma [l, r]
                return pref(r) - pref(l - 1);
        }
        int upper_bound(ll x) {
                int p = 0;
                for (int i = __lg(n); i+1; i--)
                        if (p + (1<<i) <= n and bit[p + (1<<i)] <= x)
                                x -= bit[p += (1 << i)];
                return p;
        }
};
```

22.2 bit2d

```
// BIT 2D
//
// BIT de soma, update incrementa posicao
// Tem que construir com um vetor com todos os pontos
// que vc quer um dia atualizar (os pontos q vc vai chamar update)
//
// Complexidades:
// construir - O(n log(n))
// update e query - O(log^2(n))

template<class T = int> struct bit2d {
        vector<T> X;
        vector<vector<T>> Y, t;

        int ub(vector<T>& v, T x) {
                return upper_bound(v.begin(), v.end(), x) - v.begin();
        }
        bit2d(vector<pair<T, T>> v) {
                for (auto [x, y] : v) X.push_back(x);
                sort(X.begin(), X.end());
                X.erase(unique(X.begin(), X.end()), X.end());

                t.resize(X.size() + 1);
                Y.resize(t.size());
                sort(v.begin(), v.end(), [](auto a, auto b) {
                        return a.second < b.second; });
                for (auto [x, y] : v) for (int i = ub(X, x); i < t.size(); i +=
                     i&-i)
                        if (!Y[i].size() or Y[i].back() != y) Y[i].push_back(y);

                for (int i = 0; i < t.size(); i++) t[i].resize(Y[i].size() + 1);
        }

        void update(T x, T y, T v) {
                for (int i = ub(X, x); i < t.size(); i += i&-i)
                        for (int j = ub(Y[i], y); j < t[i].size(); j += j&-j) t[
                             i][j] += v;
        }

        T query(T x, T y) {
                T ans = 0;
                for (int i = ub(X, x); i; i -= i&-i)
                        for (int j = ub(Y[i], y); j; j -= j&-j) ans += t[i][j];
                return ans;
        }
        T query(T x1, T y1, T x2, T y2) {
                return query(x2, y2)-query(x2, y1-1)-query(x1-1, y2)+query(x1-1,
                     y1-1);
        }
};
```

22.3 bitRange

```
// BIT com update em range
//
// Operacoes 0-based
// query(l, r) retorna a soma de v[l..r]
// update(l, r, x) soma x em v[l..r]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace bit {
        ll bit[2][MAX+2];
        int n;

        void build(int n2, int* v) {
                n = n2;
                for (int i = 1; i <= n; i++)
                        bit[1][min(n+1, i+(i&-i))] += bit[1][i] += v[i-1];
        }
        ll get(int x, int i) {
                ll ret = 0;
                for (; i; i -= i&-i) ret += bit[x][i];
                return ret;
        }
        void add(int x, int i, ll val) {
                for (; i <= n; i += i&-i) bit[x][i] += val;
```

```
        }
        ll get2(int p) {
                return get(0, p) * p + get(1, p);
        }
        ll query(int l, int r) {
                return get2(r+1) - get2(l);
        }
        void update(int l, int r, ll x) {
                add(0, l+1, x), add(0, r+2, -x);
                add(1, l+1, -x*l), add(1, r+2, x*(r+1));
        }
};
```

22.4 bitSortTree

```
// BIT-Sort Tree
//
// Tipo uma MergeSort Tree usando Bit
// Apesar da complexidade ser pior, fica melhor na pratica.
//
// query(l, r, k) retorna o numero de elementos menores que k
// no intervalo [l, r]
//
// Usa O(n log(n)) de memoria
//
// Complexidades:
// construir - O(n log^2(n))
// query - O(log^2(n))

template<typename T> struct ms_bit {
    int n;
    vector<vector<T>> bit;

    ms_bit(vector<T>& v) : n(v.size()), bit(n+1) {
        for (int i = 0; i < n; i++)
            for (int j = i+1; j <= n; j += j&-j)
                bit[j].push_back(v[i]);
        for (int i = 1; i <= n; i++)
            sort(bit[i].begin(), bit[i].end());
    }

    int p_query(int i, T k) {
        int ret = 0;
        for (i++; i; i -= i&-i)
            ret += lower_bound(bit[i].begin(), bit[i].end(), k) - bit[i].begin()
                ;
        return ret;
    }
    int query(int l, int r, T k) {
        return p_query(r, k) - p_query(l-1, k);
    }
};
```

22.5 cht

```
// Convex Hull Trick Estatico
//
// adds tem que serem feitos em ordem de slope
// queries tem que ser feitas em ordem de x
//
// add O(1) amortizado, get O(1) amortizado

struct CHT {
        int it;
        vector<ll> a, b;
        CHT():it(0){}
        ll eval(int i, ll x){
                return a[i]*x + b[i];
        }
        bool useless(){
                int sz = a.size();
                int r = sz-1, m = sz-2, l = sz-3;
#warning cuidado com overflow!
                return (b[l] - b[r])*(a[m] - a[l]) <
                        (b[l] - b[m])*(a[r] - a[l]);
        }
        void add(ll A, ll B){
                a.push_back(A); b.push_back(B);
                while (!a.empty()){
                        if ((a.size() < 3) || !useless()) break;
                        a.erase(a.end() - 2);
                        b.erase(b.end() - 2);
                }
                it = min(it, int(a.size()) - 1);
        }
        ll get(ll x){
                while (it+1 < a.size()){
                        if (eval(it+1, x) > eval(it, x)) it++;
                        else break;
                }
                return eval(it, x);
        }
};
```

22.6 chtDinamico

```
// Convex Hull Trick Dinamico
//
// para double, use LINF = 1/.0, div(a, b) = a/b
// update(x) atualiza o ponto de intersecao da reta x
// overlap(x) verifica se a reta x sobrepoe a proxima
// add(a, b) adiciona reta da forma ax + b
// query(x) computa maximo de ax + b para entre as retas
//
// O(log(n)) amortizado por insercao
// O(log(n)) por query

struct Line {
        mutable ll a, b, p;
        bool operator<(const Line& o) const { return a < o.a; }
        bool operator<(ll x) const { return p < x; }
};

struct dynamic_hull : multiset<Line, less<>> {
        ll div(ll a, ll b) {
                return a / b - ((a ^ b) < 0 and a % b);
        }

        void update(iterator x) {
                if (next(x) == end()) x->p = LINF;
                else if (x->a == next(x)->a) x->p = x->b >= next(x)->b ? LINF :
                        -LINF;
                else x->p = div(next(x)->b - x->b, x->a - next(x)->a);
        }

        bool overlap(iterator x) {
                update(x);
                if (next(x) == end()) return 0;
                if (x->a == next(x)->a) return x->b >= next(x)->b;
                return x->p >= next(x)->p;
        }

        void add(ll a, ll b) {
                auto x = insert({a, b, 0});
                while (overlap(x)) erase(next(x)), update(x);
                if (x != begin() and !overlap(prev(x))) x = prev(x), update(x);
                while (x != begin() and overlap(prev(x)))
                        x = prev(x), erase(next(x)), update(x);
        }

        ll query(ll x) {
                assert(!empty());
                auto l = *lower_bound(x);
#warning cuidado com overflow!
                return l.a * x + l.b;
        }
};
```

22.7 dsu

```
// DSU
//
// Une dois conjuntos e acha a qual conjunto um elemento pertence por seu id
//
// find e unite: O(a(n)) ~= O(1) amortizado

struct dsu {
        vector<int> id, sz;

        dsu(int n) : id(n), sz(n, 1) { iota(id.begin(), id.end(), 0); }

        int find(int a) { return a == id[a] ? a : id[a] = find(id[a]); }

        void unite(int a, int b) {
                a = find(a), b = find(b);
                if (a == b) return;
                if (sz[a] < sz[b]) swap(a, b);
                sz[a] += sz[b], id[b] = a;
        }
};

// DSU de bipartido
//
// Une dois vertices e acha a qual componente um vertice pertence
// Informa se a componente de um vertice e bipartida
//
// find e unite: O(log(n))

struct dsu {
        vector<int> id, sz, bip, c;

        dsu(int n) : id(n), sz(n, 1), bip(n, 1), c(n) {
                iota(id.begin(), id.end(), 0);
        }

        int find(int a) { return a == id[a] ? a : find(id[a]); }
        int color(int a) { return a == id[a] ? c[a] : c[a] ^ color(id[a]); }

        void unite(int a, int b) {
                bool change = color(a) == color(b);
                a = find(a), b = find(b);
                if (a == b) {
                        if (change) bip[a] = 0;
                        return;
                }

                if (sz[a] < sz[b]) swap(a, b);
                if (change) c[b] = 1;
                sz[a] += sz[b], id[b] = a, bip[a] &= bip[b];
        }
};


// DSU Persistente
//
// Persistencia parcial, ou seja, tem que ir
// incrementando o 't' no une
//
// find e unite: O(log(n))

struct dsu {
        vector<int> id, sz, ti;

        dsu(int n) : id(n), sz(n, 1), ti(n, -INF) {
                iota(id.begin(), id.end(), 0);
        }

        int find(int a, int t) {
                if (id[a] == a or ti[a] > t) return a;
                return find(id[a], t);
        }

        void unite(int a, int b, int t) {
                a = find(a, t), b = find(b, t);
                if (a == b) return;
                if (sz[a] < sz[b]) swap(a, b);
                sz[a] += sz[b], id[b] = a, ti[b] = t;
        }
};

// DSU com rollback
//
// checkpoint(): salva o estado atual de todas as variaveis
// rollback(): retorna para o valor das variaveis para
// o ultimo checkpoint
//
// Sempre que uma variavel muda de valor, adiciona na stack
//
// find e unite: O(log(n))
// checkpoint: O(1)
// rollback: O(m) em que m e o numero de vezes que alguma
// variavel mudou de valor desde o ultimo checkpoint

struct dsu {
        vector<int> id, sz;
        stack<stack<pair<int&, int>>> st;

        dsu(int n) : id(n), sz(n, 1) {
                iota(id.begin(), id.end(), 0), st.emplace();
        }

        void save(int &x) { st.top().emplace(x, x); }

        void checkpoint() { st.emplace(); }

        void rollback() {
                while(st.top().size()) {
                        auto [end, val] = st.top().top(); st.top().pop();
                        end = val;
                }
                st.pop();
        }

        int find(int a) { return a == id[a] ? a : find(id[a]); }

        void unite(int a, int b) {
                a = find(a), b = find(b);
                if (a == b) return;
                if (sz[a] < sz[b]) swap(a, b);
                save(sz[a]), save(id[b]);
                sz[a] += sz[b], id[b] = a;
        }
};
```

22.8 lichao

```
// Li-Chao Tree
//
// Adiciona retas (ax+b), e computa o minimo entre as retas
// em um dado 'x'
// Cuidado com overflow!
// Se tiver overflow, tenta comprimir o 'x' ou usar
// convex hull trick
//
// O(log(MA-MI)), O(n) de memoria

template<ll MI = ll(-1e9), ll MA = ll(1e9)> struct lichao {
        struct line {
                ll a, b;
                array<int, 2> ch;
                line(ll a_ = 0, ll b_ = LINF) :
                        a(a_), b(b_), ch({-1, -1}) {}
                ll operator ()(ll x) { return a*x + b; }
        };
        vector<line> ln;

        int ch(int p, int d) {
                if (ln[p].ch[d] == -1) {
                        ln[p].ch[d] = ln.size();
                        ln.emplace_back();
```

```
        }
        return ln[p].ch[d];
    }
    lichao() { ln.emplace_back(); }

    void add(line s, ll l=MI, ll r=MA, int p=0) {
        ll m = (l+r)/2;
        bool L = s(l) < ln[p](l);
        bool M = s(m) < ln[p](m);
        bool R = s(r) < ln[p](r);
        if (M) swap(ln[p], s), swap(ln[p].ch, s.ch);
        if (s.b == LINF) return;
        if (L != M) add(s, l, m-1, ch(p, 0));
        else if (R != M) add(s, m+1, r, ch(p, 1));
    }
    ll query(int x, ll l=MI, ll r=MA, int p=0) {
        ll m = (l+r)/2, ret = ln[p](x);
        if (ret == LINF) return ret;
        if (x < m) return min(ret, query(x, l, m-1, ch(p, 0)));
        return min(ret, query(x, m+1, r, ch(p, 1)));
    }
};
```

22.9 lichaoLazy

```
// Li-Chao Tree - Lazy
//
// Sendo N = MA-MI:
// insert({a, b}) minimiza tudo com ax+b - O(log N)
// insert({a, b}, l, r) minimiza com ax+b no range [l, r] - O(log^2 N)
// shift({a, b}) soma ax+b em tudo - O(1)
// shift({a, b}, l, r) soma ax+b no range [l, r] - O(log^2 N)
// query(x) retorna o valor da posicao x - O(log N)
//
// No inicio eh tudo LINF, se inserir {0, 0} fica tudo 0
//
// O(n log N) de memoria ; O(n) de memoria se nao usar as operacoes de range

template<int MI = int(-1e9), int MA = int(1e9)> struct lichao {
    struct line {
        ll a, b;
        ll la, lb; // lazy
        array<int, 2> ch;
        line(ll a_ = 0, ll b_ = LINF) :
            a(a_), b(b_), la(0), lb(0), ch({-1, -1}) {}
        ll operator ()(ll x) { return a*x + b; }
    };
    vector<line> ln;

    int ch(int p, int d) {
        if (ln[p].ch[d] == -1) {
            ln[p].ch[d] = ln.size();
            ln.emplace_back();
        }
        return ln[p].ch[d];
    }
    lichao() { ln.emplace_back(); }

    void prop(int p, int l, int r) {
        if (ln[p].la == 0 and ln[p].lb == 0) return;
        ln[p].a += ln[p].la, ln[p].b += ln[p].lb;
        if (l != r) {
            int pl = ch(p, 0), pr = ch(p, 1);
            ln[pl].la += ln[p].la, ln[pl].lb += ln[p].lb;
            ln[pr].la += ln[p].la, ln[pr].lb += ln[p].lb;
        }
        ln[p].la = ln[p].lb = 0;
    }

    ll query(int x, int p=0, int l=MI, int r=MA) {
        prop(p, l, r);
        ll ret = ln[p](x);
        if (ln[p].ch[0] == -1 and ln[p].ch[1] == -1) return ret;
        int m = l + (r-l)/2;
        if (x <= m) return min(ret, query(x, ch(p, 0), l, m));
        return min(ret, query(x, ch(p, 1), m+1, r));
    }

    void push(line s, int p, int l, int r) {
        prop(p, l, r);
        int m = l + (r-l)/2;
        bool L = s(l) < ln[p](l);
        bool M = s(m) < ln[p](m);
        bool R = s(r) < ln[p](r);
        if (M) swap(ln[p].a, s.a), swap(ln[p].b, s.b);
        if (s.b == LINF) return;
        if (L != M) push(s, ch(p, 0), l, m);
        else if (R != M) push(s, ch(p, 1), m+1, r);
    }
    void insert(line s, int a=MI, int b=MA, int p=0, int l=MI, int r=MA) {
        prop(p, l, r);
        if (a <= l and r <= b) return push(s, p, l, r);
        if (b < l or r < a) return;
        int m = l + (r-l)/2;
        insert(s, a, b, ch(p, 0), l, m);
        insert(s, a, b, ch(p, 1), m+1, r);
    }

    void shift(line s, int a=MI, int b=MA, int p=0, int l=MI, int r=MA) {
        prop(p, l, r);
        int m = l + (r-l)/2;
        if (a <= l and r <= b) {
            ln[p].la += s.a, ln[p].lb += s.b;
            return;
        }
        if (b < l or r < a) return;
        if (ln[p].b != LINF) {
            push(ln[p], ch(p, 0), l, m);
            push(ln[p], ch(p, 1), m+1, r);
            ln[p].a = 0, ln[p].b = LINF;
        }
        shift(s, a, b, ch(p, 0), l, m);
        shift(s, a, b, ch(p, 1), m+1, r);
    }
};
```

22.10 mergeSortTree

```
// MergeSort Tree
//
// Se for construida sobre um array:
//              count(i, j, a, b) retorna quantos
//              elementos de v[i..j] pertencem a [a, b]
//              report(i, j, a, b) retorna os indices dos
//              elementos de v[i..j] que pertencem a [a, b]
//              retorna o vetor ordenado
// Se for construida sobre pontos (x, y):
//              count(x1, x2, y1, y2) retorna quantos pontos
//              pertencem ao retangulo (x1, y1), (x2, y2)
//              report(x1, x2, y1, y2) retorna os indices dos pontos que
//              pertencem ao retangulo (x1, y1), (x2, y2)
//              retorna os pontos ordenados lexicograficamente
//              (assume x1 <= x2, y1 <= y2)
//
// kth(y1, y2, k) retorna o indice do ponto com k-esimo menor
// x dentre os pontos que possuem y em [y1, y2] (0 based)
// Se quiser usar para achar k-esimo valor em range, construir
// com ms_tree t(v, true), e chamar kth(l, r, k)
//
// Usa O(n log(n)) de memoria
//
// Complexidades:
// construir - O(n log(n))
// count - O(log(n))
// report - O(log(n) + k) para k indices retornados
// kth - O(log(n))

template <typename T = int> struct ms_tree {
    vector<tuple<T, T, int>> v;
    int n;
```

```
vector<vector<tuple<T, T, int>>> t; // {y, idx, left}
vector<T> vy;

ms_tree(vector<pair<T, T>>& vv) : n(vv.size()), t(4*n), vy(n) {
        for (int i = 0; i < n; i++) v.push_back({vv[i].first, vv[i].
             second, i});
        sort(v.begin(), v.end());
        build(1, 0, n-1);
        for (int i = 0; i < n; i++) vy[i] = get<0>(t[1][i+1]);
}
ms_tree(vector<T>& vv, bool inv = false) { // inv: inverte indice e
     valor
        vector<pair<T, T>> v2;
        for (int i = 0; i < vv.size(); i++)
                inv ? v2.push_back({vv[i], i}) : v2.push_back({i, vv[i
                     ]});
        *this = ms_tree(v2);
}
void build(int p, int l, int r) {
        t[p].push_back({get<0>(v[l]), get<0>(v[r]), 0}); // {min_x,
             max_x, 0}
        if (l == r) return t[p].push_back({get<1>(v[l]), get<2>(v[l]),
             0});
        int m = (l+r)/2;
        build(2*p, l, m), build(2*p+1, m+1, r);

        int L = 0, R = 0;
        while (t[p].size() <= r-l+1) {
                int left = get<2>(t[p].back());
                if (L > m-l or (R+m+1 <= r and t[2*p+1][1+R] < t[2*p][1+
                     L])) {
                        t[p].push_back(t[2*p+1][1 + R++]);
                        get<2>(t[p].back()) = left;
                        continue;
                }
                t[p].push_back(t[2*p][1 + L++]);
                get<2>(t[p].back()) = left+1;
        }
}

int get_l(T y) { return lower_bound(vy.begin(), vy.end(), y) - vy.begin
     (); }
int get_r(T y) { return upper_bound(vy.begin(), vy.end(), y) - vy.begin
     (); }

int count(T x1, T x2, T y1, T y2) {
        function<int(int, int, int)> dfs = [&](int p, int l, int r) {
                if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0]) <
                     x1) return 0;
                if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2)
                     return r-l;
                int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
                return dfs(2*p, nl, nr) + dfs(2*p+1, l-nl, r-nr);
        };
        return dfs(1, get_l(y1), get_r(y2));
}
vector<int> report(T x1, T x2, T y1, T y2) {
        vector<int> ret;
        function<void(int, int, int)> dfs = [&](int p, int l, int r) {
                if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0]) <
                     x1) return;
                if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2) {
                        for (int i = l; i < r; i++) ret.push_back(get
                             <1>(t[p][i+1]));
                        return;
                }
                int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
                dfs(2*p, nl, nr), dfs(2*p+1, l-nl, r-nr);
        };
        dfs(1, get_l(y1), get_r(y2));
        return ret;
}
int kth(T y1, T y2, int k) {
        function<int(int, int, int)> dfs = [&](int p, int l, int r) {
                if (k >= r-l) {
                        k -= r-l;
                        return -1;
                }
                if (r-l == 1) return get<1>(t[p][l+1]);
                int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
                int left = dfs(2*p, nl, nr);
                if (left != -1) return left;
                return dfs(2*p+1, l-nl, r-nr);
        };
        return dfs(1, get_l(y1), get_r(y2));
    }
};
```

22.11 minqueueDeque

```
// Min queue - deque
//
// Tudo O(1) amortizado

template<class T> struct minqueue {
        deque<pair<T, int>> q;

        void push(T x) {
                int ct = 1;
                while (q.size() and x < q.front().first)
                        ct += q.front().second, q.pop_front();
                q.emplace_front(x, ct);
        }
        void pop() {
                if (q.back().second > 1) q.back().second--;
                else q.pop_back();
        }
        T min() { return q.back().first; }
};
```

22.12 minqueueStack

```
// Min queue - stack
//
// Tudo O(1) amortizado

template<class T> struct minstack {
        stack<pair<T, T>> s;

        void push(T x) {
                if (!s.size()) s.push({x, x});
                else s.emplace(x, std::min(s.top().second, x));
        }
        T top() { return s.top().first; }
        T pop() {
                T ans = s.top().first;
                s.pop();
                return ans;
        }
        int size() { return s.size(); }
        T min() { return s.top().second; }
};

template<class T> struct minqueue {
        minstack<T> s1, s2;

        void push(T x) { s1.push(x); }
        void move() {
                if (s2.size()) return;
                while (s1.size()) {
                        T x = s1.pop();
                        s2.push(x);
                }
        }
        T front() { return move(), s2.top(); }
        T pop() { return move(), s2.pop(); }
        int size() { return s1.size()+s2.size(); }
        T min() {
                if (!s1.size()) return s2.min();
                else if (!s2.size()) return s1.min();
                return std::min(s1.min(), s2.min());
```

```
        }
};
```

22.13 orderStatisticSet

```
// Order Statistic Set
//
// Funciona do C++11 pra cima

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
        using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;

// para declarar:
// ord_set<int> s;
// coisas do set normal funcionam:
// for (auto i : s) cout << i << endl;
// cout << s.size() << endl;
// k-esimo maior elemento O(log|s|):
// k=0: menor elemento
// cout << *s.find_by_order(k) << endl;
// quantos sao menores do que k O(log|s|):
// cout << s.order_of_key(k) << endl;

// Para fazer um multiset, tem que
// usar ord_set<pair<int, int>> com o
// segundo parametro sendo algo para diferenciar
// os ementos iguais.
// s.order_of_key({k, -INF}) vai retornar o
// numero de elementos < k
```

22.14 priorityQueueDs

```
// Priority Queue DS
//
// Mantem updates aplicados em uma estrutura de dados
// que permita rollback e nao seja amortizada.
// Cada update possui uma prioridade,
// sendo possivel remover o update com maior prioridade.
// Os updates devem ser comutativos, ou seja, o estado
// da estrutura deve ser o mesmo independente da ordem
// que eles sejam aplicados.
//
// Complexidades:
// update - O(log(n) + T(n))
// query - T(n)
// pop - O(log(n) * T(n)) amortizado
//
// onde T(n) eh a complexidade do update

// assumes all priorities are distinct
template<typename DS, typename UPD> struct priority_queue_ds {
        DS D;
        vector<tuple<UPD, int, int>> upd; // {u, p, idx_in_pos}
        set<pair<int, int>> st;
        vector<int> pos;

        priority_queue_ds(int n) : D(n) {}

        void update(UPD u, int p) {
                D.update(u);
                st.emplace(p, pos.size());
                upd.emplace_back(u, p, pos.size());
                pos.push_back(upd.size() - 1);
        }

        int query(int a) {
                return D.find(a);
        }

        void pop() {
                int k = 1, min_p; // k = number of pops we will do
                vector<tuple<UPD, int, int>> small, big;
                auto it = st.end();
                for (int qt = 0; qt++ < (k+1)/2;) {
                        it--;
                        min_p = it->first;
                        int i = pos[it->second];
                        if (qt > 1) big.push_back(upd[i]);
                        k = max<int>(k, upd.size() - i);
                }

                for (int i = 0; i < k; i++) {
                        D.rollback();
                        auto [u, p, idx] = upd.rbegin()[i];
                        if (p < min_p) small.emplace_back(u, p, idx);
                }

                st.erase(prev(st.end()));
                upd.erase(upd.end() - k, upd.end());

                small.insert(small.end(), big.rbegin(), big.rend());
                for (auto [u, p, idx] : small) {
                        D.update(u);
                        upd.emplace_back(u, p, idx);
                        pos[idx] = upd.size() - 1;
                }
        }
};
```

22.15 rangeColor

```
// Range color
//
// update(l, r, c) colore o range [l, r] com a cor c,
// e retorna os ranges que foram coloridos {l, r, cor}
// query(i) returna a cor da posicao i
//
// Complexidades (para q operacoes):
// update - O(log(q)) amortizado
// query - O(log(q))

template<typename T> struct color {
        set<tuple<int, int, T>> se;

        vector<tuple<int, int, T>> update(int l, int r, T val) {
                auto it = se.upper_bound({r, INF, val});
                if (it != se.begin() and get<1>(*prev(it)) > r) {
                        auto [L, R, V] = *--it;
                        se.erase(it);
                        se.emplace(L, r, V), se.emplace(r+1, R, V);
                }
                it = se.lower_bound({l, -INF, val});
                if (it != se.begin() and get<1>(*prev(it)) >= l) {
                        auto [L, R, V] = *--it;
                        se.erase(it);
                        se.emplace(L, l-1, V), it = se.emplace(l, R, V).first;
                }
                vector<tuple<int, int, T>> ret;
                for (; it != se.end() and get<0>(*it) <= r; it = se.erase(it))
                        ret.push_back(*it);
                se.emplace(l, r, val);
                return ret;
        }
        T query(int i) {
                auto it = se.upper_bound({i, INF, T()});
                if (it == se.begin() or get<1>(*--it) < i) return -1; // nao tem
                return get<2>(*it);
        }
};
```

22.16 rmq

```
// RMQ <O(n), O(1)> - min queue
//
// O(n) pra buildar, query O(1)
// Se tiver varios minimos, retorna
// o de menor indice

template<typename T> struct rmq {
        vector<T> v;
        int n; static const int b = 30;
        vector<int> mask, t;

        int op(int x, int y) { return v[x] <= v[y] ? x : y; }
        int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
        int small(int r, int sz = b) { return r-msb(mask[r]&((1<<sz)-1)); }
        rmq() {}
        rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
                for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
                        at = (at<<1)&((1<<b)-1);
                        while (at and op(i-msb(at&-at), i) == i) at ^= at&-at;
                }
                for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
                for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0; i+(1<<j) <=
                     n/b; i++)
                        t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j-1)+i+(1<<(j-1))
                             ]);
        }
        int index_query(int l, int r) {
                if (r-l+1 <= b) return small(r, r-l+1);
                int x = l/b+1, y = r/b-1;
                if (x > y) return op(small(l+b-1), small(r));
                int j = msb(y-x+1);
                int ans = op(small(l+b-1), op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
                return op(ans, small(r));
        }
        T query(int l, int r) { return v[index_query(l, r)]; }
};
```

22.17 slopeTrick

```
// SlopeTrick
//
// Armazena uma estrutura convexa piecewise linear
// Permite adicionar slopes sem peso e realizar query de minimo
// Comentarios acima das funcoes para explicar o que cada uma faz

template<typename T> struct SlopeTrick {
        T inf = numeric_limits<T>::max() / 3;
        T min_f;
        priority_queue<T, vector<T>, less<>> L;
        priority_queue<T, vector<T>, greater<>> R;
        T add_l, add_r;

        T top_R() {
                if (R.empty()) return inf;
                else return R.top() + add_r;
        }

        T pop_R() {
                T val = top_R();
                if (R.size()) R.pop();
                return val;
        }

        T top_L() {
                if (L.empty()) return -inf;
                else return L.top() + add_l;
        }

        T pop_L() {
                T val = top_L();
                if (L.size()) L.pop();
                return val;
        }

        size_t size() {
                return L.size() + R.size();
        }

        SlopeTrick() : min_f(0), add_l(0), add_r(0) {};

        // return {min f(x), lx, rx}
        // Em que [lx, rx] eh o intervalo que atinge o minimo
        array<T, 3> query() {
                return {min_f, top_L(), top_R()};
        }

        // f(x) += a
        void add_all(T a) {
                min_f += a;
        }

        // add \_
        // f(x) += max(a - x, 0)
        void add_a_minus_x(T a) {
                min_f += max(T(0), a - top_R());
                R.push(a - add_r);
                L.push(pop_R() - add_l);
        }

        // add _/
        // f(x) += max(x - a, 0)
        void add_x_minus_a(T a) {
                min_f += max(T(0), top_L() - a);
                L.push(a - add_l);
                R.push(pop_L() - add_r);
        }

        // add \/
        // f(x) += abs(x - a)
        void add_abs(T a) {
                add_a_minus_x(a);
                add_x_minus_a(a);
        }

        // \/ -> \_
        // f_{new} (x) = min f(y) (y <= x)
        void clear_right() {
                while (R.size()) R.pop();
        }

        // \/ -> _/
        // f_{new} (x) = min f(y) (y >= x)
        void clear_left() {
                while (L.size()) L.pop();
        }

        // \/ -> \_/
        // f_{new} (x) = min f(y) (x-b <= y <= x-a)
        void shift(T a, T b) {
                assert(a <= b);
                add_l += a;
                add_r += b;
        }

        // \/. -> .\/
        // f_{new} (x) = f(x - a)
        void shift(T a) {
                shift(a, a);
        }

        // Retorna f(x)
        // O(size)
        T get(T x) {
                auto L2 = L;
                auto R2 = R;
                T ret = min_f;
                while (L.size()) {
                        ret += max(T(0), pop_L() - x);
                }
                while (R.size()) {
                        ret += max(T(0), x - pop_R());
```

```
                }
                L = L2, R = R2;
                return ret;
        }

        // O(min(size, st.size))
        void merge(SlopeTrick &st) {
                if (st.size() > size()) {
                        swap(*this, st);
                }
                while (st.R.size()) {
                        add_x_minus_a(st.pop_R());
                }
                while (st.L.size()) {
                        add_a_minus_x(st.pop_L());
                }
                min_f += st.min_f;
        }
};
```

22.18 sparseTable

```
// Sparse Table
//
// Resolve RMQ
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)

namespace sparse {
        int m[MAX2][MAX], n;
        void build(int n2, int* v) {
                n = n2;
                for (int i = 0; i < n; i++) m[0][i] = v[i];
                for (int j = 1; (1<<j) <= n; j++) for (int i = 0; i+(1<<j) <= n;
                      i++)
                        m[j][i] = min(m[j-1][i], m[j-1][i+(1<<(j-1))]);
        }
        int query(int a, int b) {
                int j = __builtin_clz(1) - __builtin_clz(b-a+1);
                return min(m[j][a], m[j][b-(1<<j)+1]);
        }
}
```

22.19 sparseTableDisjunta

```
// Sparse Table Disjunta
//
// Resolve qualquer operacao associativa
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)

namespace sparse {
        int m[MAX2][2*MAX], n, v[2*MAX];
        int op(int a, int b) { return min(a, b); }
        void build(int n2, int* v2) {
                n = n2;
                for (int i = 0; i < n; i++) v[i] = v2[i];
                while (n&(n-1)) n++;
                for (int j = 0; (1<<j) < n; j++) {
                        int len = 1<<j;
                        for (int c = len; c < n; c += 2*len) {
                                m[j][c] = v[c], m[j][c-1] = v[c-1];
                                for (int i = c+1; i <  c+len; i++) m[j][i] = op(
                                     m[j][i-1], v[i]);
                                for (int i = c-2; i >= c-len; i--) m[j][i] = op(
                                     v[i], m[j][i+1]);
                        }
                }
        }
        int query(int l, int r) {
                if (l == r) return v[l];
                int j = __builtin_clz(l) - __builtin_clz(l^r);
                return op(m[j][l], m[j][r]);
        }
}
```

22.20 splaytree

```
// Splay Tree
//
// SEMPRE QUE DESCER NA ARVORE, DAR SPLAY NO
// NODE MAIS PROFUNDO VISITADO
// Todas as operacoes sao O(log(n)) amortizado
// Se quiser colocar mais informacao no node,
// mudar em 'update'

template<typename T> struct splaytree {
        struct node {
                node *ch[2], *p;
                int sz;
                T val;
                node(T v) {
                        ch[0] = ch[1] = p = NULL;
                        sz = 1;
                        val = v;
                }
                void update() {
                        sz = 1;
                        for (int i = 0; i < 2; i++) if (ch[i]) {
                                sz += ch[i]->sz;
                        }
                }
        };

        node* root;

        splaytree() { root = NULL; }
        splaytree(const splaytree& t) {
                throw logic_error("Nao copiar a splaytree!");
        }
        ~splaytree() {
                vector<node*> q = {root};
                while (q.size()) {
                        node* x = q.back(); q.pop_back();
                        if (!x) continue;
                        q.push_back(x->ch[0]), q.push_back(x->ch[1]);
                        delete x;
                }
        }

        void rotate(node* x) { // x vai ficar em cima
                node *p = x->p, *pp = p->p;
                if (pp) pp->ch[pp->ch[1] == p] = x;
                bool d = p->ch[0] == x;
                p->ch[!d] = x->ch[d], x->ch[d] = p;
                if (p->ch[!d]) p->ch[!d]->p = p;
                x->p = pp, p->p = x;
                p->update(), x->update();
        }
        node* splay(node* x) {
                if (!x) return x;
                root = x;
                while (x->p) {
                        node *p = x->p, *pp = p->p;
                        if (!pp) return rotate(x), x; // zig
                        if ((pp->ch[0] == p)^(p->ch[0] == x))
                                rotate(x), rotate(x); // zigzag
                        else rotate(p), rotate(x); // zigzig
                }
                return x;
        }
```

```
        node* insert(T v, bool lb=0) {
                if (!root) return lb ? NULL : root = new node(v);
                node *x = root, *last = NULL;;
                while (1) {
                        bool d = x->val < v;
                        if (!d) last = x;
                        if (x->val == v) break;
                        if (x->ch[d]) x = x->ch[d];
                        else {
                                if (lb) break;
                                x->ch[d] = new node(v);
                                x->ch[d]->p = x;
                                x = x->ch[d];
                                break;
                        }
                }
                splay(x);
                return lb ? splay(last) : x;
        }
        int size() { return root ? root->sz : 0; }
        int count(T v) { return insert(v, 1) and root->val == v; }
        node* lower_bound(T v) { return insert(v, 1); }
        void erase(T v) {
                if (!count(v)) return;
                node *x = root, *l = x->ch[0];
                if (!l) {
                        root = x->ch[1];
                        if (root) root->p = NULL;
                        return delete x;
                }
                root = l, l->p = NULL;
                while (l->ch[1]) l = l->ch[1];
                splay(l);
                l->ch[1] = x->ch[1];
                if (l->ch[1]) l->ch[1]->p = l;
                delete x;
                l->update();
        }
        int order_of_key(T v) {
                if (!lower_bound(v)) return root ? root->sz : 0;
                return root->ch[0] ? root->ch[0]->sz : 0;
        }
        node* find_by_order(int k) {
                if (k >= size()) return NULL;
                node* x = root;
                while (1) {
                        if (x->ch[0] and x->ch[0]->sz >= k+1) x = x->ch[0];
                        else {
                                if (x->ch[0]) k -= x->ch[0]->sz;
                                if (!k) return splay(x);
                                k--, x = x->ch[1];
                        }
                }
        }
        T min() {
                node* x = root;
                while (x->ch[0]) x = x->ch[0]; // max -> ch[1]
                return splay(x)->val;
        }
};
```

22.21 splaytreeImplicita

```
// Splay Tree Implicita
//
// vector da NASA
// Um pouco mais rapido q a treap
// O construtor a partir do vector
// eh linear, todas as outras operacoes
// custam O(log(n)) amortizado

template<typename T> struct splay {
        struct node {
                node *ch[2], *p;
                int sz;
                T val, sub, lazy;
                bool rev;
                node(T v) {
                        ch[0] = ch[1] = p = NULL;
                        sz = 1;
                        sub = val = v;
                        lazy = 0;
                        rev = false;
                }
                void prop() {
                        if (lazy) {
                                val += lazy, sub += lazy*sz;
                                if (ch[0]) ch[0]->lazy += lazy;
                                if (ch[1]) ch[1]->lazy += lazy;
                        }
                        if (rev) {
                                swap(ch[0], ch[1]);
                                if (ch[0]) ch[0]->rev ^= 1;
                                if (ch[1]) ch[1]->rev ^= 1;
                        }
                        lazy = 0, rev = 0;
                }
                void update() {
                        sz = 1, sub = val;
                        for (int i = 0; i < 2; i++) if (ch[i]) {
                                ch[i]->prop();
                                sz += ch[i]->sz;
                                sub += ch[i]->sub;
                        }
                }
};

node* root;

splay() { root = NULL; }
splay(node* x) {
        root = x;
        if (root) root->p = NULL;
}
splay(vector<T> v) { // O(n)
        root = NULL;
        for (T i : v) {
                node* x = new node(i);
                x->ch[0] = root;
                if (root) root->p = x;
                root = x;
                root->update();
        }
}
splay(const splay& t) {
        throw logic_error("Nao copiar a splay!");
}
~splay() {
        vector<node*> q = {root};
        while (q.size()) {
                node* x = q.back(); q.pop_back();
                if (!x) continue;
                q.push_back(x->ch[0]), q.push_back(x->ch[1]);
                delete x;
        }
}

int size(node* x) { return x ? x->sz : 0; }
void rotate(node* x) { // x vai ficar em cima
        node *p = x->p, *pp = p->p;
        if (pp) pp->ch[pp->ch[1] == p] = x;
        bool d = p->ch[0] == x;
        p->ch[!d] = x->ch[d], x->ch[d] = p;
        if (p->ch[!d]) p->ch[!d]->p = p;
        x->p = pp, p->p = x;
        p->update(), x->update();
}
node* splaya(node* x) {
        if (!x) return x;
        root = x, x->update();
        while (x->p) {
                node *p = x->p, *pp = p->p;
                if (!pp) return rotate(x), x; // zig
                if ((pp->ch[0] == p)^(p->ch[0] == x))
```

```
                        rotate(x), rotate(x); // zigzag
                else rotate(p), rotate(x); // zigzig
        }
        return x;
}
node* find(int v) {
        if (!root) return NULL;
        node *x = root;
        int key = 0;
        while (1) {
                x->prop();
                bool d = key + size(x->ch[0]) < v;
                if (key + size(x->ch[0]) != v and x->ch[d]) {
                        if (d) key += size(x->ch[0])+1;
                        x = x->ch[d];
                } else break;
        }
        return splaya(x);
}
int size() { return root ? root->sz : 0; }
void join(splay<T>& l) { // assume que l < *this
        if (!size()) swap(root, l.root);
        if (!size() or !l.size()) return;
        node* x = l.root;
        while (1) {
                x->prop();
                if (!x->ch[1]) break;
                x = x->ch[1];
        }
        l.splaya(x), root->prop(), root->update();
        x->ch[1] = root, x->ch[1]->p = x;
        root = l.root, l.root = NULL;
        root->update();
}
node* split(int v) { // retorna os elementos < v
        if (v <= 0) return NULL;
        if (v >= size()) {
                node* ret = root;
                root = NULL;
                ret->update();
                return ret;
        }
        find(v);
        node* l = root->ch[0];
        root->ch[0] = NULL;
        if (l) l->p = NULL;
        root->update();
        return l;
}
T& operator [](int i) {
        find(i);
        return root->val;
}
void push_back(T v) { // O(1)
        node* r = new node(v);
        r->ch[0] = root;
        if (root) root->p = r;
        root = r, root->update();
}
T query(int l, int r) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        T ans = M.root->sub;
        M.join(L), join(M);
        return ans;
}
void update(int l, int r, T s) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        M.root->lazy += s;
        M.join(L), join(M);
}
void reverse(int l, int r) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        M.root->rev ^= 1;
        M.join(L), join(M);
}
void erase(int l, int r) {
        splay<T> M(split(r+1));
        splay<T> L(M.split(l));
        join(L);
}
};
```

22.22 splitMergeSet

```
// Split-Merge Set
//
// Representa um conjunto de inteiros nao negativos
// Todas as operacoes custam O(log(N)),
// em que N = maior elemento do set,
// exceto o merge, que custa O(log(N)) amortizado
// Usa O(min(N, n log(N))) de memoria, sendo 'n' o
// numero de elementos distintos no set

template<typename T, bool MULTI=false, typename SIZE_T=int> struct sms {
        struct node {
                node *l, *r;
                SIZE_T cnt;
                node() : l(NULL), r(NULL), cnt(0) {}
                void update() {
                        cnt = 0;
                        if (l) cnt += l->cnt;
                        if (r) cnt += r->cnt;
                }
        };

        node* root;
        T N;

        sms() : root(NULL), N(0) {}
        sms(T v) : sms() { while (v >= N) N = 2*N+1; }
        sms(const sms& t) : root(NULL), N(t.N) {
                for (SIZE_T i = 0; i < t.size(); i++) {
                        T at = t[i];
                        SIZE_T qt = t.count(at);
                        insert(at, qt);
                        i += qt-1;
                }
        }
        sms(initializer_list<T> v) : sms() { for (T i : v) insert(i); }
        ~sms() {
                vector<node*> q = {root};
                while (q.size()) {
                        node* x = q.back(); q.pop_back();
                        if (!x) continue;
                        q.push_back(x->l), q.push_back(x->r);
                        delete x;
                }
        }

        friend void swap(sms& a, sms& b) {
                swap(a.root, b.root), swap(a.N, b.N);
        }
        sms& operator =(const sms& v) {
                sms tmp = v;
                swap(tmp, *this);
                return *this;
        }
        SIZE_T size() const { return root ? root->cnt : 0; }
        SIZE_T count(node* x) const { return x ? x->cnt : 0; }
        void clear() {
                sms tmp;
                swap(*this, tmp);
        }
        void expand(T v) {
                for (; N < v; N = 2*N+1) if (root) {
                        node* nroot = new node();
                        nroot->l = root;
                        root = nroot;
                        root->update();
                }
        }
```

```
node* insert(node* at, T idx, SIZE_T qt, T l, T r) {
        if (!at) at = new node();
        if (l == r) {
                at->cnt += qt;
                if (!MULTI) at->cnt = 1;
                return at;
        }
        T m = l + (r-l)/2;
        if (idx <= m) at->l = insert(at->l, idx, qt, l, m);
        else at->r = insert(at->r, idx, qt, m+1, r);
        return at->update(), at;
}
void insert(T v, SIZE_T qt=1) { // insere 'qt' ocorrencias de 'v'
        if (qt <= 0) return erase(v, -qt);
        assert(v >= 0);
        expand(v);
        root = insert(root, v, qt, 0, N);
}

node* erase(node* at, T idx, SIZE_T qt, T l, T r) {
        if (!at) return at;
        if (l == r) at->cnt = at->cnt < qt ? 0 : at->cnt - qt;
        else {
                T m = l + (r-l)/2;
                if (idx <= m) at->l = erase(at->l, idx, qt, l, m);
                else at->r = erase(at->r, idx, qt, m+1, r);
                at->update();
        }
        if (!at->cnt) delete at, at = NULL;
        return at;
}
void erase(T v, SIZE_T qt=1) { // remove 'qt' ocorrencias de 'v'
        if (v < 0 or v > N or !qt) return;
        if (qt < 0) insert(v, -qt);
        root = erase(root, v, qt, 0, N);
}
void erase_all(T v) { // remove todos os 'v'
        if (v < 0 or v > N) return;
        root = erase(root, v, numeric_limits<SIZE_T>::max(), 0, N);
}

SIZE_T count(node* at, T a, T b, T l, T r) const {
        if (!at or b < l or r < a) return 0;
        if (a <= l and r <= b) return at->cnt;
        T m = l + (r-l)/2;
        return count(at->l, a, b, l, m) + count(at->r, a, b, m+1, r);
}
SIZE_T count(T v) const { return count(root, v, v, 0, N); }
SIZE_T order_of_key(T v) { return count(root, 0, v-1, 0, N); }
SIZE_T lower_bound(T v) { return order_of_key(v); }

const T operator [](SIZE_T i) const { // i-esimo menor elemento
        assert(i >= 0 and i < size());
        node* at = root;
        T l = 0, r = N;
        while (l < r) {
                T m = l + (r-l)/2;
                if (count(at->l) > i) at = at->l, r = m;
                else {
                        i -= count(at->l);
                        at = at->r; l = m+1;
                }
        }
        return l;
}

node* merge(node* l, node* r) {
        if (!l or !r) return l ? l : r;
        if (!l->l and !l->r) { // folha
                if (MULTI) l->cnt += r->cnt;
                delete r;
                return l;
        }
        l->l = merge(l->l, r->l), l->r = merge(l->r, r->r);
        l->update(), delete r;
        return l;
}
void merge(sms& s) { // mergeia dois sets
        if (N > s.N) swap(*this, s);
        expand(s.N);
        root = merge(root, s.root);
        s.root = NULL;
}

node* split(node*& x, SIZE_T k) {
        if (k <= 0 or !x) return NULL;
        node* ret = new node();
        if (!x->l and !x->r) x->cnt -= k, ret->cnt += k;
        else {
                if (k <= count(x->l)) ret->l = split(x->l, k);
                else {
                        ret->r = split(x->r, k - count(x->l));
                        swap(x->l, ret->l);
                }
                ret->update(), x->update();
        }
        if (!x->cnt) delete x, x = NULL;
        return ret;
}
void split(SIZE_T k, sms& s) { // pega os 'k' menores
        s.clear();
        s.root = split(root, min(k, size()));
        s.N = N;
}
// pega os menores que 'k'
void split_val(T k, sms& s) { split(order_of_key(k), s); }
};
```

22.23 splitMergeSetLazy

```
// Split-Merge Set - Lazy [noprint]
//
// Representa um conjunto de inteiros nao negativos
// Todas as operacoes custam O(log(N)),
// em que N = maior elemento do set,
// exceto o merge e o insert_range, que custa O(log(N)) amortizado
// Usa O(min(N, n log(N))) de memoria, sendo 'n' o
// numero de elementos distintos no set

template<typename T> struct sms {
        struct node {
                node *l, *r;
                int cnt;
                bool flip;
                node() : l(NULL), r(NULL), cnt(0), flip(0) {}
                void update() {
                        cnt = 0;
                        if (l) cnt += l->cnt;
                        if (r) cnt += r->cnt;
                }
        };

        void prop(node* x, int size) {
                if (!x or !x->flip) return;
                x->flip = 0;
                x->cnt = size - x->cnt;
                if (size > 1) {
                        if (!x->l) x->l = new node();
                        if (!x->r) x->r = new node();
                        x->l->flip ^= 1;
                        x->r->flip ^= 1;
                }
        }

        node* root;
        T N;

        sms() : root(NULL), N(0) {}
        sms(T v) : sms() { while (v >= N) N = 2*N+1; }
        sms(sms& t) : root(NULL), N(t.N) {
                for (int i = 0; i < t.size(); i++) insert(t[i]);
        }
        sms(initializer_list<T> v) : sms() { for (T i : v) insert(i); }
```

```
void destroy(node* r) {
        vector<node*> q = {r};
        while (q.size()) {
                node* x = q.back(); q.pop_back();
                if (!x) continue;
                q.push_back(x->l), q.push_back(x->r);
                delete x;
        }
}
~sms() { destroy(root); }

friend void swap(sms& a, sms& b) {
        swap(a.root, b.root), swap(a.N, b.N);
}
sms& operator =(const sms& v) {
        sms tmp = v;
        swap(tmp, *this);
        return *this;
}
int count(node* x, T size) {
        if (!x) return 0;
        prop(x, size);
        return x->cnt;
}
int size() { return count(root, N+1); }
void clear() {
        sms tmp;
        swap(*this, tmp);
}
void expand(T v) {
        for (; N < v; N = 2*N+1) if (root) {
                prop(root, N+1);
                node* nroot = new node();
                nroot->l = root;
                root = nroot;
                root->update();
        }
}

node* insert(node* at, T idx, T l, T r) {
        if (!at) at = new node();
        else prop(at, r-l+1);
        if (l == r) {
                at->cnt = 1;
                return at;
        }
        T m = l + (r-l)/2;
        if (idx <= m) at->l = insert(at->l, idx, l, m);
        else at->r = insert(at->r, idx, m+1, r);
        return at->update(), at;
}
void insert(T v) {
        assert(v >= 0);
        expand(v);
        root = insert(root, v, 0, N);
}

node* erase(node* at, T idx, T l, T r) {
        if (!at) return at;
        prop(at, r-l+1);
        if (l == r) at->cnt = 0;
        else {
                T m = l + (r-l)/2;
                if (idx <= m) at->l = erase(at->l, idx, l, m);
                else at->r = erase(at->r, idx, m+1, r);
                at->update();
        }
        return at;
}
void erase(T v) {
        if (v < 0 or v > N) return;
        root = erase(root, v, 0, N);
}

int count(node* at, T a, T b, T l, T r) {
        if (!at or b < l or r < a) return 0;
        prop(at, r-l+1);
        if (a <= l and r <= b) return at->cnt;
        T m = l + (r-l)/2;
        return count(at->l, a, b, l, m) + count(at->r, a, b, m+1, r);
}
int count(T v) { return count(root, v, v, 0, N); }
int order_of_key(T v) { return count(root, 0, v-1, 0, N); }
int lower_bound(T v) { return order_of_key(v); }

const T operator [](int i) { // i-esimo menor elemento
        assert(i >= 0 and i < size());
        node* at = root;
        T l = 0, r = N;
        while (l < r) {
                prop(at, r-l+1);
                T m = l + (r-l)/2;
                if (count(at->l, m-l+1) > i) at = at->l, r = m;
                else {
                        i -= count(at->l, r-m);
                        at = at->r; l = m+1;
                }
        }
        return l;
}

node* merge(node* a, node* b, T tam) {
        if (!a or !b) return a ? a : b;
        prop(a, tam), prop(b, tam);
        if (b->cnt == tam) swap(a, b);
        if (tam == 1 or a->cnt == tam) {
                destroy(b);
                return a;
        }
        a->l = merge(a->l, b->l, tam>>1), a->r = merge(a->r, b->r, tam
            >>1);
        a->update(), delete b;
        return a;
}
void merge(sms& s) { // mergeia dois sets
        if (N > s.N) swap(*this, s);
        expand(s.N);
        root = merge(root, s.root, N+1);
        s.root = NULL;
}

node* split(node*& x, int k, T tam) {
        if (k <= 0 or !x) return NULL;
        prop(x, tam);
        node* ret = new node();
        if (tam == 1) x->cnt = 0, ret->cnt = 1;
        else {
                if (k <= count(x->l, tam>>1)) ret->l = split(x->l, k,
                    tam>>1);
                else {
                        ret->r = split(x->r, k - count(x->l, tam>>1),
                            tam>>1);
                        swap(x->l, ret->l);
                }
                ret->update(), x->update();
        }
        return ret;
}
void split(int k, sms& s) { // pega os 'k' menores
        s.clear();
        s.root = split(root, min(k, size()), N+1);
        s.N = N;
}
// pega os menores que 'k'
void split_val(T k, sms& s) { split(order_of_key(k), s); }

void flip(node*& at, T a, T b, T l, T r) {
        if (!at) at = new node();
        else prop(at, r-l+1);
        if (a <= l and r <= b) {
                at->flip ^= 1;
                prop(at, r-l+1);
                return;
        }
        if (r < a or b < l) return;
        T m = l + (r-l)/2;
        flip(at->l, a, b, l, m), flip(at->r, a, b, m+1, r);
        at->update();
```

```
        }
        void flip(T l, T r) { // flipa os valores em [l, r]
                assert(l >= 0 and l <= r);
                expand(r);
                flip(root, l, r, 0, N);
        }
        // complemento considerando que o universo eh [0, lim]
        void complement(T lim) {
                assert(lim >= 0);
                if (lim > N) expand(lim);
                flip(root, 0, lim, 0, N);
                sms tmp;
                split_val(lim+1, tmp);
                swap(*this, tmp);
        }
        void insert_range(T l, T r) { // insere todo os valores em [l, r]
                sms tmp;
                tmp.flip(l, r);
                merge(tmp);
        }
};
```

22.24 sqrtTree

```
// SQRT Tree
//
// RMQ em O(log log n) com O(n log log n) pra buildar
// Funciona com qualquer operacao associativa
// Tao rapido quanto a sparse table, mas usa menos memoria
// (log log (1e9) < 5, entao a query eh praticamente O(1))
//
// build - O(n log log n)
// query - O(log log n)

namespace sqrtTree {
        int n, *v;
        int pref[4][MAX], sulf[4][MAX], getl[4][MAX], entre[4][MAX], sz[4];

        int op(int a, int b) { return min(a, b); }
        inline int getblk(int p, int i) { return (i-getl[p][i])/sz[p]; }
        void build(int p, int l, int r) {
                if (l+1 >= r) return;
                for (int i = l; i <= r; i++) getl[p][i] = l;
                for (int L = l; L <= r; L += sz[p]) {
                        int R = min(L+sz[p]-1, r);
                        pref[p][L] = v[L], sulf[p][R] = v[R];
                        for (int i = L+1; i <= R; i++) pref[p][i] = op(pref[p][i
                             -1], v[i]);
                        for (int i = R-1; i >= L; i--) sulf[p][i] = op(v[i],
                             sulf[p][i+1]);
                        build(p+1, L, R);
                }
                for (int i = 0; i <= sz[p]; i++) {
                        int at = entre[p][l+i*sz[p]+i] = sulf[p][l+i*sz[p]];
                        for (int j = i+1; j <= sz[p]; j++) entre[p][l+i*sz[p]+j]
                             = at =
                                        op(at, sulf[p][l+j*sz[p]]);
                }
        }
        void build(int n2, int* v2) {
                n = n2, v = v2;
                for (int p = 0; p < 4; p++) sz[p] = n2 = sqrt(n2);
                build(0, 0, n-1);
        }
        int query(int l, int r) {
                if (l+1 >= r) return l == r ? v[l] : op(v[l], v[r]);
                int p = 0;
                while (getblk(p, l) == getblk(p, r)) p++;
                int ans = sulf[p][l], a = getblk(p, l)+1, b = getblk(p, r)-1;
                if (a <= b) ans = op(ans, entre[p][getl[p][l]+a*sz[p]+b]);
                return op(ans, pref[p][r]);
        }
}
```

22.25 treap

```
// Treap
//
// Todas as operacoes custam
// O(log(n)) com alta probabilidade, exceto meld
// meld custa O(log^2 n) amortizado com alta prob.,
// e permite unir duas treaps sem restricao adicional
// Na pratica, esse meld tem constante muito boa e
// o pior caso eh meio estranho de acontecer

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

template<typename T> struct treap {
        struct node {
                node *l, *r;
                int p, sz;
                T val, mi;
                node(T v) : l(NULL), r(NULL), p(rng()), sz(1), val(v), mi(v) {}
                void update() {
                        sz = 1;
                        mi = val;
                        if (l) sz += l->sz, mi = min(mi, l->mi);
                        if (r) sz += r->sz, mi = min(mi, r->mi);
                }
        };

        node* root;

        treap() { root = NULL; }
        treap(const treap& t) {
                throw logic_error("Nao copiar a treap!");
        }
        ~treap() {
                vector<node*> q = {root};
                while (q.size()) {
                        node* x = q.back(); q.pop_back();
                        if (!x) continue;
                        q.push_back(x->l), q.push_back(x->r);
                        delete x;
                }
        }

        int size(node* x) { return x ? x->sz : 0; }
        int size() { return size(root); }
        void join(node* l, node* r, node*& i) { // assume que l < r
                if (!l or !r) return void(i = l ? l : r);
                if (l->p > r->p) join(l->r, r, l->r), i = l;
                else join(l, r->l, r->l), i = r;
                i->update();
        }
        void split(node* i, node*& l, node*& r, T v) {
                if (!i) return void(r = l = NULL);
                if (i->val < v) split(i->r, i->r, r, v), l = i;
                else split(i->l, l, i->l, v), r = i;
                i->update();
        }
        void split_leq(node* i, node*& l, node*& r, T v) {
                if (!i) return void(r = l = NULL);
                if (i->val <= v) split_leq(i->r, i->r, r, v), l = i;
                else split_leq(i->l, l, i->l, v), r = i;
                i->update();
        }
        int count(node* i, T v) {
                if (!i) return 0;
                if (i->val == v) return 1;
                if (v < i->val) return count(i->l, v);
                return count(i->r, v);
        }
        void index_split(node* i, node*& l, node*& r, int v, int key = 0) {
                if (!i) return void(r = l = NULL);
                if (key + size(i->l) < v) index_split(i->r, i->r, r, v, key+size
                     (i->l)+1), l = i;
                else index_split(i->l, l, i->l, v, key), r = i;
                i->update();
        }
        int count(T v) {
                return count(root, v);
```

```
        }
        void insert(T v) {
                if (count(v)) return;
                node *L, *R;
                split(root, L, R, v);
                node* at = new node(v);
                join(L, at, L);
                join(L, R, root);
        }
        void erase(T v) {
                node *L, *M, *R;
                split_leq(root, M, R, v), split(M, L, M, v);
                if (M) delete M;
                M = NULL;
                join(L, R, root);
        }
        void meld(treap& t) { // segmented merge
                node *L = root, *R = t.root;
                root = NULL;
                while (L or R) {
                        if (!L or (L and R and L->mi > R->mi)) std::swap(L, R);
                        if (!R) join(root, L, root), L = NULL;
                        else if (L->mi == R->mi) {
                                node* LL;
                                split(L, LL, L, R->mi+1);
                                delete LL;
                        } else {
                                node* LL;
                                split(L, LL, L, R->mi);
                                join(root, LL, root);
                        }
                }
                t.root = NULL;
        }
};
```

22.26 treapImplicita

```
// Treap Implicita
//
// Todas as operacoes custam
// O(log(n)) com alta probabilidade

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

template<typename T> struct treap {
        struct node {
                node *l, *r;
                int p, sz;
                T val, sub, lazy;
                bool rev;
                node(T v) : l(NULL), r(NULL), p(rng()), sz(1), val(v), sub(v),
                    lazy(0), rev(0) {}
                void prop() {
                        if (lazy) {
                                val += lazy, sub += lazy*sz;
                                if (l) l->lazy += lazy;
                                if (r) r->lazy += lazy;
                        }
                        if (rev) {
                                swap(l, r);
                                if (l) l->rev ^= 1;
                                if (r) r->rev ^= 1;
                        }
                        lazy = 0, rev = 0;
                }
                void update() {
                        sz = 1, sub = val;
                        if (l) l->prop(), sz += l->sz, sub += l->sub;
                        if (r) r->prop(), sz += r->sz, sub += r->sub;
                }
        };

        node* root;
        treap() { root = NULL; }
        treap(const treap& t) {
                throw logic_error("Nao copiar a treap!");
        }
        ~treap() {
                vector<node*> q = {root};
                while (q.size()) {
                        node* x = q.back(); q.pop_back();
                        if (!x) continue;
                        q.push_back(x->l), q.push_back(x->r);
                        delete x;
                }
        }

        int size(node* x) { return x ? x->sz : 0; }
        int size() { return size(root); }
        void join(node* l, node* r, node*& i) { // assume que l < r
                if (!l or !r) return void(i = l ? l : r);
                l->prop(), r->prop();
                if (l->p > r->p) join(l->r, r, l->r), i = l;
                else join(l, r->l, r->l), i = r;
                i->update();
        }
        void split(node* i, node*& l, node*& r, int v, int key = 0) {
                if (!i) return void(r = l = NULL);
                i->prop();
                if (key + size(i->l) < v) split(i->r, i->r, r, v, key+size(i->l)
                    +1), l = i;
                else split(i->l, l, i->l, v, key), r = i;
                i->update();
        }
        void push_back(T v) {
                node* i = new node(v);
                join(root, i, root);
        }
        T query(int l, int r) {
                node *L, *M, *R;
                split(root, M, R, r+1), split(M, L, M, l);
                T ans = M->sub;
                join(L, M, M), join(M, R, root);
                return ans;
        }
        void update(int l, int r, T s) {
                node *L, *M, *R;
                split(root, M, R, r+1), split(M, L, M, l);
                M->lazy += s;
                join(L, M, M), join(M, R, root);
        }
        void reverse(int l, int r) {
                node *L, *M, *R;
                split(root, M, R, r+1), split(M, L, M, l);
                M->rev ^= 1;
                join(L, M, M), join(M, R, root);
        }
};
```

22.27 treapPersistent

```
// Treap Persistent Implicita
//
// Todas as operacoes custam
// O(log(n)) com alta probabilidade

mt19937_64 rng((int) chrono::steady_clock::now().time_since_epoch().count());

struct node {
        node *l, *r;
        ll sz, val, sub;
        node(ll v) : l(NULL), r(NULL), sz(1), val(v), sub(v) {}
        node(node* x) : l(x->l), r(x->r), sz(x->sz), val(x->val), sub(x->sub) {}
        void update() {
                sz = 1, sub = val;
                if (l) sz += l->sz, sub += l->sub;
                if (r) sz += r->sz, sub += r->sub;
                sub %= MOD;
```

```
        }
};

ll size(node* x) { return x ? x->sz : 0; }
void update(node* x) { if (x) x->update(); }
node* copy(node* x) { return x ? new node(x) : NULL; }

node* join(node* l, node* r) {
        if (!l or !r) return l ? copy(l) : copy(r);
        node* ret;
        if (rng() % (size(l) + size(r)) < size(l)) {
                ret = copy(l);
                ret->r = join(ret->r, r);
        } else {
                ret = copy(r);
                ret->l = join(l, ret->l);
        }
        return update(ret), ret;
}

void split(node* x, node*& l, node*& r, ll v, ll key = 0) {
        if (!x) return void(l = r = NULL);
        if (key + size(x->l) < v) {
                l = copy(x);
                split(l->r, l->r, r, v, key+size(l->l)+1);
        } else {
                r = copy(x);
                split(r->l, l, r->l, v, key);
        }
        update(l), update(r);
}

vector<node*> treap;

void init(const vector<ll>& v) {
        treap = {NULL};
        for (auto i : v) treap[0] = join(treap[0], new node(i));
}
```

22.28 waveletTree

```
// Wavelet Tree
//
// Usa O(sigma + n log(sigma)) de memoria,
// onde sigma = MAXN - MINN
// Depois do build, o v fica ordenado
// count(i, j, x, y) retorna o numero de elementos de
// v[i, j) que pertencem a [x, y]
// kth(i, j, k) retorna o elemento que estaria
// na poscicao k-1 de v[i, j), se ele fosse ordenado
// sum(i, j, x, y) retorna a soma dos elementos de
// v[i, j) que pertencem a [x, y]
// sumk(i, j, k) retorna a soma dos k-esimos menores
// elementos de v[i, j) (sum(i, j, 1) retorna o menor)
//
// Complexidades:
// build - O(n log(sigma))
// count - O(log(sigma))
// kth   - O(log(sigma))
// sum   - O(log(sigma))
// sumk  - O(log(sigma))

int n, v[MAX];
vector<int> esq[4*(MAXN-MINN)], pref[4*(MAXN-MINN)];

void build(int b = 0, int e = n, int p = 1, int l = MINN, int r = MAXN) {
        int m = (l+r)/2; esq[p].push_back(0); pref[p].push_back(0);
        for (int i = b; i < e; i++) {
                esq[p].push_back(esq[p].back()+(v[i]<=m));
                pref[p].push_back(pref[p].back()+v[i]);
        }
        if (l == r) return;
        int m2 = stable_partition(v+b, v+e, [=](int i){return i <= m;}) - v;
        build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
}

int count(int i, int j, int x, int y, int p = 1, int l = MINN, int r = MAXN) {
        if (y < l or r < x) return 0;
        if (x <= l and r <= y) return j-i;
        int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
        return count(ei, ej, x, y, 2*p, l, m)+count(i-ei, j-ej, x, y, 2*p+1, m
             +1, r);
}

int kth(int i, int j, int k, int p=1, int l = MINN, int r = MAXN) {
        if (l == r) return l;
        int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
        if (k <= ej-ei) return kth(ei, ej, k, 2*p, l, m);
        return kth(i-ei, j-ej, k-(ej-ei), 2*p+1, m+1, r);
}

int sum(int i, int j, int x, int y, int p = 1, int l = MINN, int r = MAXN) {
        if (y < l or r < x) return 0;
        if (x <= l and r <= y) return pref[p][j]-pref[p][i];
        int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
        return sum(ei, ej, x, y, 2*p, l, m) + sum(i-ei, j-ej, x, y, 2*p+1, m+1,
             r);
}

int sumk(int i, int j, int k, int p = 1, int l = MINN, int r = MAXN) {
        if (l == r) return l*k;
        int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
        if (k <= ej-ei) return sumk(ei, ej, k, 2*p, l, m);
        return pref[2*p][ej]-pref[2*p][ei]+sumk(i-ei, j-ej, k-(ej-ei), 2*p+1, m
             +1, r);
}
```

23 ufmg forked/Estruturas/Segtree

23.1 segTreap

```
// SegTreap
//
// Muda uma posicao do plano, e faz query de operacao
// associativa e comutativa em retangulo
// Mudar ZERO e op
// Esparso nas duas coordenadas, inicialmente eh tudo ZERO
//
// Para query com distancia de manhattan <= d, faca
// nx = x+y, ny = x-y
// Update em (nx, ny), query em ((nx-d, ny-d), (nx+d, ny+d))
//
// Valores no X tem que ser de 0 ateh NX
// Para q operacoes, usa O(q log(NX)) de memoria, e as
// operacoes custa O(log(q) log(NX))

const int ZERO = INF;
const int op(int l, int r) { return min(l, r); }

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

template<typename T> struct treap {
        struct node {
                node *l, *r;
                int p;
                pair<ll, ll> idx; // {y, x}
                T val, mi;
                node(ll x, ll y, T val_) : l(NULL), r(NULL), p(rng()),
                        idx(pair(y, x)), val(val_), mi(val) {}
                void update() {
                        mi = val;
                        if (l) mi = op(mi, l->mi);
                        if (r) mi = op(mi, r->mi);
                }
        };

        node* root;

        treap() { root = NULL; }
```

```
        ~treap() {
                vector<node*> q = {root};
                while (q.size()) {
                        node* x = q.back(); q.pop_back();
                        if (!x) continue;
                        q.push_back(x->l), q.push_back(x->r);
                        delete x;
                }
        }
        treap(treap&& t) : treap() { swap(root, t.root); }

        void join(node* l, node* r, node*& i) { // assume que l < r
                if (!l or !r) return void(i = l ? l : r);
                if (l->p > r->p) join(l->r, r, l->r), i = l;
                else join(l, r->l, r->l), i = r;
                i->update();
        }
        void split(node* i, node*& l, node*& r, pair<ll, ll> idx) {
                if (!i) return void(r = l = NULL);
                if (i->idx < idx) split(i->r, i->r, r, idx), l = i;
                else split(i->l, l, i->l, idx), r = i;
                i->update();
        }
        void update(ll x, ll y, T v) {
                node *L, *M, *R;
                split(root, M, R, pair(y, x+1)), split(M, L, M, pair(y, x));
                if (M) M->val = M->mi = v;
                else M = new node(x, y, v);
                join(L, M, M), join(M, R, root);
        }
        T query(ll ly, ll ry) {
                node *L, *M, *R;
                split(root, M, R, pair(ry, LINF)), split(M, L, M, pair(ly, 0));
                T ret = M ? M->mi : ZERO;
                join(L, M, M), join(M, R, root);
                return ret;
        }
};

template<typename T> struct segtreap {
        vector<treap<T>> seg;
        vector<int> ch[2];
        ll NX;

        segtreap(ll NX_) : seg(1), NX(NX_) { ch[0].push_back(-1), ch[1].
             push_back(-1); }

        int get_ch(int i, int d){
                if (ch[d][i] == -1) {
                        ch[d][i] = seg.size();
                        seg.emplace_back();
                        ch[0].push_back(-1), ch[1].push_back(-1);
                }
                return ch[d][i];
        }

        T query(ll lx, ll rx, ll ly, ll ry, int p, ll l, ll r) {
                if (rx < l or r < lx) return ZERO;
                if (lx <= l and r <= rx) return seg[p].query(ly, ry);

                ll m = l + (r-l)/2;
                return op(query(lx, rx, ly, ry, get_ch(p, 0), l, m),
                                query(lx, rx, ly, ry, get_ch(p, 1), m+1, r));
        }
        T query(ll lx, ll rx, ll ly, ll ry) { return query(lx, rx, ly, ry, 0, 0,
             NX); }

        void update(ll x, ll y, T val, int p, ll l, ll r) {
                if (l == r) return seg[p].update(x, y, val);
                ll m = l + (r-l)/2;
                if (x <= m) update(x, y, val, get_ch(p, 0), l, m);
                else update(x, y, val, get_ch(p, 1), m+1, r);
                seg[p].update(x, y, val);
        }
        void update(ll x, ll y, T val) { update(x, y, val, 0, 0, NX); }
};
```

23.2 segTree

```
// SegTree
//
// Recursiva com Lazy Propagation
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
// Pode usar a seguinte funcao para indexar os nohs:
// f(l, r) = (l+r)|(l!=r), usando 2N de memoria
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace seg {
        ll seg[4*MAX], lazy[4*MAX];
        int n, *v;

        ll build(int p=1, int l=0, int r=n-1) {
                lazy[p] = 0;
                if (l == r) return seg[p] = v[l];
                int m = (l+r)/2;
                return seg[p] = build(2*p, l, m) + build(2*p+1, m+1, r);
        }
        void build(int n2, int* v2) {
                n = n2, v = v2;
                build();
        }
        void prop(int p, int l, int r) {
                seg[p] += lazy[p]*(r-l+1);
                if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
                lazy[p] = 0;
        }
        ll query(int a, int b, int p=1, int l=0, int r=n-1) {
                prop(p, l, r);
                if (a <= l and r <= b) return seg[p];
                if (b < l or r < a) return 0;
                int m = (l+r)/2;
                return query(a, b, 2*p, l, m) + query(a, b, 2*p+1, m+1, r);
        }
        ll update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
                prop(p, l, r);
                if (a <= l and r <= b) {
                        lazy[p] += x;
                        prop(p, l, r);
                        return seg[p];
                }
                if (b < l or r < a) return seg[p];
                int m = (l+r)/2;
                return seg[p] = update(a, b, x, 2*p, l, m) +
                        update(a, b, x, 2*p+1, m+1, r);
        }
};


// Se tiver uma seg de max, da pra descobrir em O(log(n))
// o primeiro e ultimo elemento >= val numa range:

// primeira posicao >= val em [a, b] (ou -1 se nao tem)
int get_left(int a, int b, int val, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (b < l or r < a or seg[p] < val) return -1;
        if (r == l) return l;
        int m = (l+r)/2;
        int x = get_left(a, b, val, 2*p, l, m);
        if (x != -1) return x;
        return get_left(a, b, val, 2*p+1, m+1, r);
}

// ultima posicao >= val em [a, b] (ou -1 se nao tem)
int get_right(int a, int b, int val, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (b < l or r < a or seg[p] < val) return -1;
        if (r == l) return l;
        int m = (l+r)/2;
        int x = get_right(a, b, val, 2*p+1, m+1, r);
        if (x != -1) return x;
```

```
        return get_right(a, b, val, 2*p, l, m);
}

// Se tiver uma seg de soma sobre um array nao negativo v, da pra
// descobrir em O(log(n)) o maior j tal que v[i]+v[i+1]+...+v[j-1] < val
int lower_bound(int i, ll& val, int p, int l, int r) {
        prop(p, l, r);
        if (r < i) return n;
        if (i <= l and seg[p] < val) {
                val -= seg[p];
                return n;
        }
        if (l == r) return l;
        int m = (l+r)/2;
        int x = lower_bound(i, val, 2*p, l, m);
        if (x != n) return x;
        return lower_bound(i, val, 2*p+1, m+1, r);
}
```

23.3 segTree2D

```
// SegTree 2D Iterativa
//
// Consultas 0-based
// Um valor inicial em (x, y) deve ser colocado em seg[x+n][y+n]
// Query: soma do retangulo ((x1, y1), (x2, y2))
// Update: muda o valor da posicao (x, y) para val
// Nao pergunte como que essa coisa funciona
//
// Para query com distancia de manhattan <= d, faca
// nx = x+y, ny = x-y
// Update em (nx, ny), query em ((nx-d, ny-d), (nx+d, ny+d))
//
// Se for de min/max, pode tirar os if's da 'query', e fazer
// sempre as 4 operacoes. Fica mais rapido
//
// Complexidades:
// build - O(n^2)
// query - O(log^2(n))
// update - O(log^2(n))

int seg[2*MAX][2*MAX], n;

void build() {
        for (int x = 2*n; x; x--) for (int y = 2*n; y; y--) {
                if (x < n) seg[x][y] = seg[2*x][y] + seg[2*x+1][y];
                if (y < n) seg[x][y] = seg[x][2*y] + seg[x][2*y+1];
        }
}

int query(int x1, int y1, int x2, int y2) {
        int ret = 0, y3 = y1 + n, y4 = y2 + n;
        for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /= 2)
                for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2 /= 2) {
                        if (x1%2 == 1 and y1%2 == 1) ret += seg[x1][y1];
                        if (x1%2 == 1 and y2%2 == 0) ret += seg[x1][y2];
                        if (x2%2 == 0 and y1%2 == 1) ret += seg[x2][y1];
                        if (x2%2 == 0 and y2%2 == 0) ret += seg[x2][y2];
                }

        return ret;
}

void update(int x, int y, int val) {
        int y2 = y += n;
        for (x += n; x; x /= 2, y = y2) {
                if (x >= n) seg[x][y] = val;
                else seg[x][y] = seg[2*x][y] + seg[2*x+1][y];

                while (y /= 2) seg[x][y] = seg[x][2*y] + seg[x][2*y+1];
        }
}
```

23.4 segTreeBeats

```
// SegTree Beats
//
// query(a, b) - {{min(v[a..b]), max(v[a..b])}, sum(v[a..b])}
// updatemin(a, b, x) faz com que v[i] <- min(v[i], x),
// para i em [a, b]
// updatemax faz o mesmo com max, e updatesum soma x
// em todo mundo do intervalo [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log^2 (n)) amortizado
// (se nao usar updatesum, fica log(n) amortizado)

#define f first
#define s second

namespace beats {
        struct node {
                int tam;
                ll sum, lazy; // lazy pra soma
                ll mi1, mi2, mi; // mi = #mi1
                ll ma1, ma2, ma; // ma = #ma1

                node(ll x = 0) {
                        sum = mi1 = ma1 = x;
                        mi2 = LINF, ma2 = -LINF;
                        mi = ma = tam = 1;
                        lazy = 0;
                }
                node(const node& l, const node& r) {
                        sum = l.sum + r.sum, tam = l.tam + r.tam;
                        lazy = 0;
                        if (l.mi1 > r.mi1) {
                                mi1 = r.mi1, mi = r.mi;
                                mi2 = min(l.mi1, r.mi2);
                        } else if (l.mi1 < r.mi1) {
                                mi1 = l.mi1, mi = l.mi;
                                mi2 = min(r.mi1, l.mi2);
                        } else {
                                mi1 = l.mi1, mi = l.mi+r.mi;
                                mi2 = min(l.mi2, r.mi2);
                        }
                        if (l.ma1 < r.ma1) {
                                ma1 = r.ma1, ma = r.ma;
                                ma2 = max(l.ma1, r.ma2);
                        } else if (l.ma1 > r.ma1) {
                                ma1 = l.ma1, ma = l.ma;
                                ma2 = max(r.ma1, l.ma2);
                        } else {
                                ma1 = l.ma1, ma = l.ma+r.ma;
                                ma2 = max(l.ma2, r.ma2);
                        }
                }
                void setmin(ll x) {
                        if (x >= ma1) return;
                        sum += (x - ma1)*ma;
                        if (mi1 == ma1) mi1 = x;
                        if (mi2 == ma1) mi2 = x;
                        ma1 = x;
                }
                void setmax(ll x) {
                        if (x <= mi1) return;
                        sum += (x - mi1)*mi;
                        if (ma1 == mi1) ma1 = x;
                        if (ma2 == mi1) ma2 = x;
                        mi1 = x;
                }
                void setsum(ll x) {
                        mi1 += x, mi2 += x, ma1 += x, ma2 += x;
                        sum += x*tam;
                        lazy += x;
                }
        };

        node seg[4*MAX];
```

```
        int n, *v;

        node build(int p=1, int l=0, int r=n-1) {
                if (l == r) return seg[p] = {v[l]};
                int m = (l+r)/2;
                return seg[p] = {build(2*p, l, m), build(2*p+1, m+1, r)};
        }
        void build(int n2, int* v2) {
                n = n2, v = v2;
                build();
        }
        void prop(int p, int l, int r) {
                if (l == r) return;
                for (int k = 0; k < 2; k++) {
                        if (seg[p].lazy) seg[2*p+k].setsum(seg[p].lazy);
                        seg[2*p+k].setmin(seg[p].ma1);
                        seg[2*p+k].setmax(seg[p].mi1);
                }
                seg[p].lazy = 0;
        }
        pair<pair<ll, ll>, ll> query(int a, int b, int p=1, int l=0, int r=n-1)
             {
                if (b < l or r < a) return {{LINF, -LINF}, 0};
                if (a <= l and r <= b) return {{seg[p].mi1, seg[p].ma1}, seg[p].
                     sum};
                prop(p, l, r);
                int m = (l+r)/2;
                auto L = query(a, b, 2*p, l, m), R = query(a, b, 2*p+1, m+1, r);
                return {{min(L.f.f, R.f.f), max(L.f.s, R.f.s)}, L.s+R.s};
        }
        node updatemin(int a, int b, ll x, int p=1, int l=0, int r=n-1) {
                if (b < l or r < a or seg[p].ma1 <= x) return seg[p];
                if (a <= l and r <= b and seg[p].ma2 < x) {
                        seg[p].setmin(x);
                        return seg[p];
                }
                prop(p, l, r);
                int m = (l+r)/2;
                return seg[p] = {updatemin(a, b, x, 2*p, l, m),
                                                 updatemin(a, b, x, 2*p+1, m+1, r
                                                      )};
        }
        node updatemax(int a, int b, ll x, int p=1, int l=0, int r=n-1) {
                if (b < l or r < a or seg[p].mi1 >= x) return seg[p];
                if (a <= l and r <= b and seg[p].mi2 > x) {
                        seg[p].setmax(x);
                        return seg[p];
                }
                prop(p, l, r);
                int m = (l+r)/2;
                return seg[p] = {updatemax(a, b, x, 2*p, l, m),
                                                 updatemax(a, b, x, 2*p+1, m+1, r
                                                      )};
        }
        node updatesum(int a, int b, ll x, int p=1, int l=0, int r=n-1) {
                if (b < l or r < a) return seg[p];
                if (a <= l and r <= b) {
                        seg[p].setsum(x);
                        return seg[p];
                }
                prop(p, l, r);
                int m = (l+r)/2;
                return seg[p] = {updatesum(a, b, x, 2*p, l, m),
                                                 updatesum(a, b, x, 2*p+1, m+1, r
                                                      )};
        }
};
```

23.5 segTreeColor

```
// SegTree Colorida
//
// Cada posicao tem um valor e uma cor
// O construtor receve um vector de {valor, cor}
// e o numero de cores (as cores devem estar em [0, c-1])
// query(c, a, b) retorna a soma dos valores
// de todo mundo em [a, b] que tem cor c
// update(c, a, b, x) soma x em todo mundo em
// [a, b] que tem cor c
// paint(c1, c2, a, b) faz com que todo mundo
// em [a, b] que tem cor c1 passe a ter cor c2
//
// Complexidades:
// construir - O(n log(n)) espaco e tempo
// query - O(log(n))
// update - O(log(n))
// paint - O(log(n)) amortizado

struct seg_color {
        struct node {
                node *l, *r;
                int cnt;
                ll val, lazy;
                node() : l(NULL), r(NULL), cnt(0), val(0), lazy(0) {}
                void update() {
                        cnt = 0, val = 0;
                        for (auto i : {l, r}) if (i) {
                                i->prop();
                                cnt += i->cnt, val += i->val;
                        }
                }
                void prop() {
                        if (!lazy) return;
                        val += lazy*(ll)cnt;
                        for (auto i : {l, r}) if (i) i->lazy += lazy;
                        lazy = 0;
                }
        };

        int n;
        vector<node*> seg;

        seg_color(vector<pair<int, int>>& v, int c) : n(v.size()), seg(c, NULL)
             {
                for (int i = 0; i < n; i++)
                        seg[v[i].second] = insert(seg[v[i].second], i, v[i].
                             first, 0, n-1);
        }
        ~seg_color() {
                queue<node*> q;
                for (auto i : seg) q.push(i);
                while (q.size()) {
                        auto i = q.front(); q.pop();
                        if (!i) continue;
                        q.push(i->l), q.push(i->r);
                        delete i;
                }
        }

        node* insert(node* at, int idx, int val, int l, int r) {
                if (!at) at = new node();
                if (l == r) return at->cnt = 1, at->val = val, at;
                int m = (l+r)/2;
                if (idx <= m) at->l = insert(at->l, idx, val, l, m);
                else at->r = insert(at->r, idx, val, m+1, r);
                return at->update(), at;
        }
        ll query(node* at, int a, int b, int l, int r) {
                if (!at or b < l or r < a) return 0;
                at->prop();
                if (a <= l and r <= b) return at->val;
                int m = (l+r)/2;
                return query(at->l, a, b, l, m) + query(at->r, a, b, m+1, r);
        }
        ll query(int c, int a, int b) { return query(seg[c], a, b, 0, n-1); }
        void update(node* at, int a, int b, int x, int l, int r) {
                if (!at or b < l or r < a) return;
                at->prop();
                if (a <= l and r <= b) {
                        at->lazy += x;
                        return void(at->prop());
                }
                int m = (l+r)/2;
                update(at->l, a, b, x, l, m), update(at->r, a, b, x, m+1, r);
```

```
            at->update();
        }
        void update(int c, int a, int b, int x) { update(seg[c], a, b, x, 0, n
            -1); }
        void paint(node*& from, node*& to, int a, int b, int l, int r) {
            if (to == from or !from or b < l or r < a) return;
            from->prop();
            if (to) to->prop();
            if (a <= l and r <= b) {
                if (!to) {
                    to = from;
                    from = NULL;
                    return;
                }
                int m = (l+r)/2;
                paint(from->l, to->l, a, b, l, m), paint(from->r, to->r,
                    a, b, m+1, r);
                to->update();
                delete from;
                from = NULL;
                return;
            }
            if (!to) to = new node();
            int m = (l+r)/2;
            paint(from->l, to->l, a, b, l, m), paint(from->r, to->r, a, b, m
                +1, r);
            from->update(), to->update();
        }
        void paint(int c1, int c2, int a, int b) { paint(seg[c1], seg[c2], a, b,
            0, n-1); }
};
```

23.6 segTreeEsparsa

```
// SegTree Esparsa - Lazy
//
// Query: soma do range [a, b]
// Update: flipa os valores de [a, b]
// O MAX tem q ser Q log N para Q updates
//
// Complexidades:
// build - O(1)
// query - O(log(n))
// update - O(log(n))

namespace seg {
        int seg[MAX], lazy[MAX], R[MAX], L[MAX], ptr;
        int get_l(int i){
            if (L[i] == 0) L[i] = ptr++;
            return L[i];
        }
        int get_r(int i){
            if (R[i] == 0) R[i] = ptr++;
            return R[i];
        }

        void build() { ptr = 2; }

        void prop(int p, int l, int r) {
            if (!lazy[p]) return;
            seg[p] = r-l+1 - seg[p];
            if (l != r) lazy[get_l(p)]^=lazy[p], lazy[get_r(p)]^=lazy[p];
            lazy[p] = 0;
        }

        int query(int a, int b, int p=1, int l=0, int r=N-1) {
            prop(p, l, r);
            if (b < l or r < a) return 0;
            if (a <= l and r <= b) return seg[p];

            int m = (l+r)/2;
            return query(a, b, get_l(p), l, m)+query(a, b, get_r(p), m+1, r)
                ;
        }
        int update(int a, int b, int p=1, int l=0, int r=N-1) {
            prop(p, l, r);
            if (b < l or r < a) return seg[p];
            if (a <= l and r <= b) {
                lazy[p] ^= 1;
                prop(p, l, r);
                return seg[p];
            }
            int m = (l+r)/2;
            return seg[p] = update(a, b, get_l(p), l, m)+update(a, b, get_r(
                p), m+1, r);
        }
};
```

23.7 segTreeEsparsa2

```
// SegTree Esparsa - O(q) memoria
//
// Query: min do range [a, b]
// Update: troca o valor de uma posicao
// Usa O(q) de memoria para q updates
//
// Complexidades:
// query - O(log(n))
// update - O(log(n))

template<typename T> struct seg {
        struct node {
            node* ch[2];
            char d;
            T v;

            T mi;

            node(int d_, T v_, T val) : d(d_), v(v_) {
                ch[0] = ch[1] = NULL;
                mi = val;
            }
            node(node* x) : d(x->d), v(x->v), mi(x->mi) {
                ch[0] = x->ch[0], ch[1] = x->ch[1];
            }
            void update() {
                mi = numeric_limits<T>::max();
                for (int i = 0; i < 2; i++) if (ch[i])
                    mi = min(mi, ch[i]->mi);
            }
        };

        node* root;
        char n;

        seg() : root(NULL), n(0) {}
        ~seg() {
            std::vector<node*> q = {root};
            while (q.size()) {
                node* x = q.back(); q.pop_back();
                if (!x) continue;
                q.push_back(x->ch[0]), q.push_back(x->ch[1]);
                delete x;
            }
        }

        char msb(T v, char l, char r) { // msb in range (l, r]
            for (char i = r; i > l; i--) if (v>>i&1) return i;
            return -1;
        }
        void cut(node* at, T v, char i) {
            char d = msb(v ^ at->v, at->d, i);
            if (d == -1) return; // no need to split
            node* nxt = new node(at);
            at->ch[v>>d&1] = NULL;
            at->ch[!(v>>d&1)] = nxt;
            at->d = d;
        }
```

```
        node* update(node* at, T idx, T val, char i) {
                if (!at) return new node(-1, idx, val);
                cut(at, idx, i);
                if (at->d == -1) { // leaf
                        at->mi = val;
                        return at;
                }
                bool dir = idx>>at->d&1;
                at->ch[dir] = update(at->ch[dir], idx, val, at->d-1);
                at->update();
                return at;
        }
        void update(T idx, T val) {
                while (idx>>n) n++;
                root = update(root, idx, val, n-1);
        }

        T query(node* at, T a, T b, T l, T r, char i) {
                if (!at or b < l or r < a) return numeric_limits<T>::max();
                if (a <= l and r <= b) return at->mi;
                T m = l + (r-l)/2;
                if (at->d < i) {
                        if ((at->v>>i&1) == 0) return query(at, a, b, l, m, i-1)
                                ;
                        else return query(at, a, b, m+1, r, i-1);
                }
                return min(query(at->ch[0], a, b, l, m, i-1), query(at->ch[1], a
                     , b, m+1, r, i-1));
        }
        T query(T l, T r) { return query(root, l, r, 0, (T(1)<<n)-1, n-1); }
};
```

23.8 segTreeIterativa

```
// SegTree Iterativa
//
// Consultas 0-based
// Valores iniciais devem estar em (seg[n], ... , seg[2*n-1])
// Query: soma do range [a, b]
// Update: muda o valor da posicao p para x
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

int seg[2 * MAX];
int n;

void build() {
        for (int i = n - 1; i; i--) seg[i] = seg[2*i] + seg[2*i+1];
}

int query(int a, int b) {
        int ret = 0;
        for(a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
                if (a % 2 == 1) ret += seg[a];
                if (b % 2 == 0) ret += seg[b];
        }
        return ret;
}

void update(int p, int x) {
        seg[p += n] = x;
        while (p /= 2) seg[p] = seg[2*p] + seg[2*p+1];
}
```

23.9 segTreeIterativaComLazy

```
// SegTree Iterativa com Lazy Propagation
//
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
// Para mudar, mudar as funcoes junta, poe e query
// LOG = ceil(log2(MAX))
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace seg {
        ll seg[2*MAX], lazy[2*MAX];
        int n;

        ll junta(ll a, ll b) {
                return a+b;
        }

        // soma x na posicao p de tamanho tam
        void poe(int p, ll x, int tam, bool prop=1) {
                seg[p] += x*tam;
                if (prop and p < n) lazy[p] += x;
        }

        // atualiza todos os pais da folha p
        void sobe(int p) {
                for (int tam = 2; p /= 2; tam *= 2) {
                        seg[p] = junta(seg[2*p], seg[2*p+1]);
                        poe(p, lazy[p], tam, 0);
                }
        }

        // propaga o caminho da raiz ate a folha p
        void prop(int p) {
                int tam = 1 << (LOG-1);
                for (int s = LOG; s; s--, tam /= 2) {
                        int i = p >> s;
                        if (lazy[i]) {
                                poe(2*i, lazy[i], tam);
                                poe(2*i+1, lazy[i], tam);
                                lazy[i] = 0;
                        }
                }
        }

        void build(int n2, int* v) {
                n = n2;
                for (int i = 0; i < n; i++) seg[n+i] = v[i];
                for (int i = n-1; i; i--) seg[i] = junta(seg[2*i], seg[2*i+1]);
                for (int i = 0; i < 2*n; i++) lazy[i] = 0;
        }

        ll query(int a, int b) {
                ll ret = 0;
                for (prop(a+=n), prop(b+=n); a <= b; ++a/=2, --b/=2) {
                        if (a%2 == 1) ret = junta(ret, seg[a]);
                        if (b%2 == 0) ret = junta(ret, seg[b]);
                }
                return ret;
        }

        void update(int a, int b, int x) {
                int a2 = a += n, b2 = b += n, tam = 1;
                for (; a <= b; ++a/=2, --b/=2, tam *= 2) {
                        if (a%2 == 1) poe(a, x, tam);
                        if (b%2 == 0) poe(b, x, tam);
                }
                sobe(a2), sobe(b2);
        }
};
```

23.10 segTreePa

```
// SegTree PA
//
// Segtree de PA
```

```
// update_set(l, r, A, R) seta [l, r] para PA(A, R),
// update_add soma PA(A, R) em [l, r]
// query(l, r) retorna a soma de [l, r]
//
// PA(A, R) eh a PA: [A+R, A+2R, A+3R, ... ]
//
// Complexidades:
// construir - O(n)
// update_set, update_add, query - O(log(n))

struct seg_pa {
        struct Data {
                ll sum;
                ll set_a, set_r, add_a, add_r;
                Data() : sum(0), set_a(LINF), set_r(0), add_a(0), add_r(0) {}
        };
        vector<Data> seg;
        int n;

        seg_pa(int n_) {
                n = n_;
                seg = vector<Data>(4*n);
        }

        void prop(int p, int l, int r) {
                int tam = r-l+1;
                ll &sum = seg[p].sum, &set_a = seg[p].set_a, &set_r = seg[p].
                     set_r,
                        &add_a = seg[p].add_a, &add_r = seg[p].add_r;

                if (set_a != LINF) {
                        set_a += add_a, set_r += add_r;
                        sum = set_a*tam + set_r*tam*(tam+1)/2;
                        if (l != r) {
                                int m = (l+r)/2;

                                seg[2*p].set_a = set_a;
                                seg[2*p].set_r = set_r;
                                seg[2*p].add_a = seg[2*p].add_r = 0;

                                seg[2*p+1].set_a = set_a + set_r * (m-l+1);
                                seg[2*p+1].set_r = set_r;
                                seg[2*p+1].add_a = seg[2*p+1].add_r = 0;
                        }
                        set_a = LINF, set_r = 0;
                        add_a = add_r = 0;
                } else if (add_a or add_r) {
                        sum += add_a*tam + add_r*tam*(tam+1)/2;
                        if (l != r) {
                                int m = (l+r)/2;

                                seg[2*p].add_a += add_a;
                                seg[2*p].add_r += add_r;

                                seg[2*p+1].add_a += add_a + add_r * (m-l+1);
                                seg[2*p+1].add_r += add_r;
                        }
                        add_a = add_r = 0;
                }
        }

        int inter(pair<int, int> a, pair<int, int> b) {
                if (a.first > b.first) swap(a, b);
                return max(0, min(a.second, b.second) - b.first + 1);
        }
        ll set(int a, int b, ll aa, ll rr, int p, int l, int r) {
                prop(p, l, r);
                if (b < l or r < a) return seg[p].sum;
                if (a <= l and r <= b) {
                        seg[p].set_a = aa;
                        seg[p].set_r = rr;
                        prop(p, l, r);
                        return seg[p].sum;
                }
                int m = (l+r)/2;
                int tam_l = inter({l, m}, {a, b});
                return seg[p].sum = set(a, b, aa, rr, 2*p, l, m) +
                        set(a, b, aa + rr * tam_l, rr, 2*p+1, m+1, r);
        }
        void update_set(int l, int r, ll aa, ll rr) {
                set(l, r, aa, rr, 1, 0, n-1);
        }
        ll add(int a, int b, ll aa, ll rr, int p, int l, int r) {
                prop(p, l, r);
                if (b < l or r < a) return seg[p].sum;
                if (a <= l and r <= b) {
                        seg[p].add_a += aa;
                        seg[p].add_r += rr;
                        prop(p, l, r);
                        return seg[p].sum;
                }
                int m = (l+r)/2;
                int tam_l = inter({l, m}, {a, b});
                return seg[p].sum = add(a, b, aa, rr, 2*p, l, m) +
                        add(a, b, aa + rr * tam_l, rr, 2*p+1, m+1, r);
        }
        void update_add(int l, int r, ll aa, ll rr) {
                add(l, r, aa, rr, 1, 0, n-1);
        }
        ll query(int a, int b, int p, int l, int r) {
                prop(p, l, r);
                if (b < l or r < a) return 0;
                if (a <= l and r <= b) return seg[p].sum;
                int m = (l+r)/2;
                return query(a, b, 2*p, l, m) + query(a, b, 2*p+1, m+1, r);
        }
        ll query(int l, int r) { return query(l, r, 1, 0, n-1); }
};
```

23.11 segTreePersistent

```
// SegTree Persistente
//
// SegTree de soma, update de somar numa posicao
//
// query(a, b, t) retorna a query de [a, b] na versao t
// update(a, x, t) faz um update v[a]+=x a partir da
// versao de t, criando uma nova versao e retornando seu id
// Por default, faz o update a partir da ultima versao
//
// build - O(n)
// query - O(log(n))
// update - O(log(n))

const int MAX = 1e5+10, UPD = 1e5+10, LOG = 18;
const int MAXS = 2*MAX+UPD*LOG;

namespace perseg {
        ll seg[MAXS];
        int rt[UPD], L[MAXS], R[MAXS], cnt, t;
        int n, *v;

        ll build(int p, int l, int r) {
                if (l == r) return seg[p] = v[l];
                L[p] = cnt++, R[p] = cnt++;
                int m = (l+r)/2;
                return seg[p] = build(L[p], l, m) + build(R[p], m+1, r);
        }
        void build(int n2, int* v2) {
                n = n2, v = v2;
                rt[0] = cnt++;
                build(0, 0, n-1);
        }
        ll query(int a, int b, int p, int l, int r) {
                if (b < l or r < a) return 0;
                if (a <= l and r <= b) return seg[p];
                int m = (l+r)/2;
                return query(a, b, L[p], l, m) + query(a, b, R[p], m+1, r);
        }
        ll query(int a, int b, int tt) {
                return query(a, b, rt[tt], 0, n-1);
        }
        ll update(int a, int x, int lp, int p, int l, int r) {
                if (l == r) return seg[p] = seg[lp]+x;
```

```
                int m = (l+r)/2;
                if (a <= m)
                        return seg[p] = update(a, x, L[lp], L[p]=cnt++, l, m) +
                             seg[R[p]=R[lp]];
                return seg[p] = seg[L[p]=L[lp]] + update(a, x, R[lp], R[p]=cnt
                     ++, m+1, r);
        }
        int update(int a, int x, int tt=t) {
                update(a, x, rt[tt], rt[++t]=cnt++, 0, n-1);
                return t;
        }
};
```

23.12 segTreePersistentComLazy

```
// SegTree Persistente com Lazy
//
// Nao propaga, meio estranho de mexer, mas da
//
// query(a, b, t) retorna a query de [a, b] na versao t
// update(a, b, x, t) faz um update v[a..b]+=x a partir da
// versao de t, criando uma nova versao e retornando seu id
// Por default, faz o update a partir da ultima versao
//
// build - O(n)
// query - O(log(n))
// update - O(log(n))

const int MAX = 1e5+10, UPD = 1e5+10, LOG = 18;
const int MAXS = 2*MAX + 4*UPD*LOG;

namespace perseg {
        int seg[MAXS];
        int rt[UPD], L[MAXS], R[MAXS], cnt, t;
        int n, *v;

        int build(int p, int l, int r) {
                if (l == r) return seg[p] = v[l];
                L[p] = cnt++, R[p] = cnt++;
                int m = (l+r)/2;
                return seg[p] = max(build(L[p], l, m), build(R[p], m+1, r));
        }
        void build(int n2, int *v2) {
                n = n2, v = v2;
                rt[0] = cnt++;
                build(0, 0, n-1);
        }
        int query(int a, int b, int p, int l, int r) {
                if (b < l or r < a) return -INF;
                if (a <= l and r <= b) return lazy[p] + seg[p];
                int m = (l+r)/2;
                int ret = lazy[p] + max(query(a, b, L[p], l, m), query(a, b, R[p
                     ], m+1, r));
                return ret;
        }
        int query(int a, int b, int tt) {
                return query(a, b, rt[tt], 0, n-1);
        }
        int update(int a, int b, int x, int lp, int p, int l, int r) {
                tie(seg[p], lazy[p], L[p], R[p]) = {seg[lp], lazy[lp], L[lp], R[
                     lp]};
                if (b < l or r < a) return seg[p] + lazy[p];
                if (a <= l and r <= b) return seg[p] + (lazy[p] += x);

                int m = (l+r)/2;
                seg[p] = max(update(a, b, x, L[lp], L[p] = cnt++, l, m),
                                update(a, b, x, R[lp], R[p] = cnt++, m
                                     +1, r));
                lazy[p] = lazy[lp];
                return seg[p] + lazy[p];
        }
        int update(int a, int b, int x, int tt=t) {
                assert(tt <= t);
                update(a, b, x, rt[tt], rt[++t]=cnt++, 0, n-1);
                return t;
        }
};
```

24 ufmg forked/Extra

24.1 debug

```
void debug_out(string s, int line) { cerr << endl; }
template<typename H, typename... T>
void debug_out(string s, int line, H h, T... t) {
    if (s[0] != ',') cerr << "Line(" << line << ") ";
    do { cerr << s[0]; s = s.substr(1);
    } while (s.size() and s[0] != ',');
    cerr << " = " << h;
    debug_out(s, line, t...);
}
#ifdef DEBUG
#define debug(...) debug_out(#__VA_ARGS__, __LINE__, __VA_ARGS__)
#else
#define debug(...) 42
#endif
```

24.2 fastIO

```
int read_int() {
    bool minus = false;
    int result = 0;
    char ch;
    ch = getchar();
    while (1) {
        if (ch == '-') break;
        if (ch >= '0' && ch <= '9') break;
        ch = getchar();
    }
    if (ch == '-') minus = true;
    else result = ch-'0';
    while (1) {
        ch = getchar();
        if (ch < '0' || ch > '9') break;
        result = result*10 + (ch - '0');
    }
    if (minus) return -result;
    else return result;
}
```

24.3 hash

```
# Para usar (hash das linhas [l1, l2]):
# bash hash.sh arquivo.cpp l1 l2
sed -n $2','$3' p' $1 | sed '/^#w/d' | cpp -dD -P -fpreprocessed | tr -d '[:
     space:]' | md5sum | cut -c-6
```

24.4 pragma

```
// Otimizacoes agressivas, pode deixar mais rapido ou mais devagar
#pragma GCC optimize("Ofast")
// Auto explicativo
#pragma GCC optimize("unroll-loops")
// Vetorizacao
#pragma GCC target("avx2")
// Para operacoes com bits
#pragma GCC target("bmi,bmi2,popcnt,lzcnt")
```

24.5 rand

```
mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

int uniform(int l, int r){
        uniform_int_distribution<int> uid(l, r);
        return uid(rng);
}
```

24.6 stress

```
P=a
make ${P} ${P}2 gen || exit 1
for ((i = 1; ; i++)) do
        ./gen $i > in
        ./${P} < in > out
        ./${P}2 < in > out2
        if (! cmp -s out out2) then
                echo "--> entrada:"
                cat in
                echo "--> saida1:"
                cat out
                echo "--> saida2:"
                cat out2
                break;
        fi
        echo $i
done
```

24.7 template

```
#include <bits/stdc++.h>

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'

typedef long long ll;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3fll;

int main() { _
        exit(0);
}
```

24.8 timer

```
// timer T; T() -> retorna o tempo em ms desde que declarou
using namespace chrono;
struct timer : high_resolution_clock {
        const time_point start;
        timer(): start(now()) {}
        int operator()() {
                return duration_cast<milliseconds>(now() - start).count();
        }
};
```

25 ufmg forked/Grafos

25.1 articulationPoints

```
// Articulation Points
//
// Computa os pontos de articulacao (vertices criticos) de um grafo
//
// art[i] armazena o numero de novas componentes criadas ao deletar vertice i
// se art[i] >= 1, entao vertice i eh ponto de articulacao
//
// O(n+m)

int n;
vector<vector<int>> g;
stack<int> s;
vector<int> id, art;

int dfs_art(int i, int& t, int p = -1) {
        int lo = id[i] = t++;
        s.push(i);
        for (int j : g[i]) if (j != p) {
                if (id[j] == -1) {
                        int val = dfs_art(j, t, i);
                        lo = min(lo, val);

                        if (val >= id[i]) {
                                art[i]++;
                                while (s.top() != j) s.pop();
                                s.pop();
                        }
                        // if (val > id[i]) aresta i-j eh ponte
                }
                else lo = min(lo, id[j]);
        }
        if (p == -1 and art[i]) art[i]--;
        return lo;
}

void compute_art_points() {
        id = vector<int>(n, -1);
        art = vector<int>(n, 0);
        int t = 0;
        for (int i = 0; i < n; i++) if (id[i] == -1)
                dfs_art(i, t, -1);
}
```

25.2 bellmanFord

```
// Bellman-Ford
//
// Calcula a menor distancia
// entre a e todos os vertices e
// detecta ciclo negativo
// Retorna 1 se ha ciclo negativo
// Nao precisa representar o grafo,
// soh armazenar as arestas
//
// O(nm)

int n, m;
int d[MAX];
vector<pair<int, int>> ar; // vetor de arestas
vector<int> w;             // peso das arestas

bool bellman_ford(int a) {
        for (int i = 0; i < n; i++) d[i] = INF;
        d[a] = 0;

        for (int i = 0; i <= n; i++)
                for (int j = 0; j < m; j++) {
                        if (d[ar[j].second] > d[ar[j].first] + w[j]) {
                                if (i == n) return 1;

                                d[ar[j].second] = d[ar[j].first] + w[j];
                        }
                }

        return 0;
}
```

25.3 blockCutTree

```
// Block-Cut Tree
//
// Cria a block-cut tree, uma arvore com os blocos
// e os pontos de articulacao
// Blocos sao componentes 2-vertice-conexos maximais
// Uma 2-coloracao da arvore eh tal que uma cor sao
// os blocos, e a outra cor sao os pontos de art.
// Funciona para grafo nao conexo
//
// art[i] responde o numero de novas componentes conexas
// criadas apos a remocao de i do grafo g
// Se art[i] >= 1, i eh ponto de articulacao
//
// Para todo i <= blocks.size()
// blocks[i] eh uma componente 2-vertce-conexa maximal
// edgblocks[i] sao as arestas do bloco i
// tree[i] eh um vertice da arvore que corresponde ao bloco i
//
// pos[i] responde a qual vertice da arvore vertice i pertence
// Arvore tem no maximo 2n vertices
//
// O(n+m)

struct block_cut_tree {
        vector<vector<int>> g, blocks, tree;
        vector<vector<pair<int, int>>> edgblocks;
        stack<int> s;
        stack<pair<int, int>> s2;
        vector<int> id, art, pos;

        block_cut_tree(vector<vector<int>> g_) : g(g_) {
                int n = g.size();
                id.resize(n, -1), art.resize(n), pos.resize(n);
                build();
        }

        int dfs(int i, int& t, int p = -1) {
                int lo = id[i] = t++;
                s.push(i);

                if (p != -1) s2.emplace(i, p);
                for (int j : g[i]) if (j != p and id[j] != -1) s2.emplace(i, j);

                for (int j : g[i]) if (j != p) {
                        if (id[j] == -1) {
                                int val = dfs(j, t, i);
                                lo = min(lo, val);

                                if (val >= id[i]) {
                                        art[i]++;
                                        blocks.emplace_back(1, i);
                                        while (blocks.back().back() != j)
                                                blocks.back().push_back(s.top())
                                                        , s.pop();

                                        edgblocks.emplace_back(1, s2.top()), s2.
                                                pop();
                                        while (edgblocks.back().back() != pair(j
                                                , i))
                                                edgblocks.back().push_back(s2.
                                                        top()), s2.pop();
                                }
                                // if (val > id[i]) aresta i-j eh ponte
                        }
                        else lo = min(lo, id[j]);
                }

                if (p == -1 and art[i]) art[i]--;
                return lo;
        }

        void build() {
                int t = 0;
                for (int i = 0; i < g.size(); i++) if (id[i] == -1) dfs(i, t,
                        -1);

                tree.resize(blocks.size());
                for (int i = 0; i < g.size(); i++) if (art[i])
                        pos[i] = tree.size(), tree.emplace_back();

                for (int i = 0; i < blocks.size(); i++) for (int j : blocks[i])
                        {
                                if (!art[j]) pos[j] = i;
                                else tree[i].push_back(pos[j]), tree[pos[j]].push_back(i
                                        );
                        }
        }
};
```

25.4 blossom

```
// Blossom
//
// Matching maximo em grafo geral
//
// O(n^3)
// Se for bipartido, nao precisa da funcao
// 'contract', e roda em O(nm)

vector<int> g[MAX];
int match[MAX]; // match[i] = com quem i esta matchzado ou -1
int n, pai[MAX], base[MAX], vis[MAX];
queue<int> q;

void contract(int u, int v, bool first = 1) {
        static vector<bool> bloss;
        static int l;
        if (first) {
                bloss = vector<bool>(n, 0);
                vector<bool> teve(n, 0);
                int k = u; l = v;
                while (1) {
                        teve[k = base[k]] = 1;
                        if (match[k] == -1) break;
                        k = pai[match[k]];
                }
                while (!teve[l = base[l]]) l = pai[match[l]];
        }
        while (base[u] != l) {
                bloss[base[u]] = bloss[base[match[u]]] = 1;
                pai[u] = v;
                v = match[u];
                u = pai[match[u]];
        }
        if (!first) return;
        contract(v, u, 0);
        for (int i = 0; i < n; i++) if (bloss[base[i]]) {
                base[i] = l;
                if (!vis[i]) q.push(i);
                vis[i] = 1;
        }
}

int getpath(int s) {
        for (int i = 0; i < n; i++) base[i] = i, pai[i] = -1, vis[i] = 0;
        vis[s] = 1; q = queue<int>(); q.push(s);
        while (q.size()) {
                int u = q.front(); q.pop();
                for (int i : g[u]) {
                        if (base[i] == base[u] or match[u] == i) continue;
                        if (i == s or (match[i] != -1 and pai[match[i]] != -1))
                                contract(u, i);
                        else if (pai[i] == -1) {
                                pai[i] = u;
                                if (match[i] == -1) return i;
                                i = match[i];
                                vis[i] = 1; q.push(i);
                        }
                }
        }
```

```
        return -1;
}

int blossom() {
        int ans = 0;
        memset(match, -1, sizeof(match));
        for (int i = 0; i < n; i++) if (match[i] == -1)
                for (int j : g[i]) if (match[j] == -1) {
                        match[i] = j;
                        match[j] = i;
                        ans++;
                        break;
                }
        for (int i = 0; i < n; i++) if (match[i] == -1) {
                int j = getpath(i);
                if (j == -1) continue;
                ans++;
                while (j != -1) {
                        int p = pai[j], pp = match[p];
                        match[p] = j;
                        match[j] = p;
                        j = pp;
                }
        }
        return ans;
}
```

25.5 center

```
// Centro de arvore
//
// Retorna o diametro e o(s) centro(s) da arvore
// Uma arvore tem sempre um ou dois centros e estes estao no meio do diametro
//
// O(n)

vector<int> g[MAX];
int d[MAX], par[MAX];

pair<int, vector<int>> center() {
        int f, df;
        function<void(int)> dfs = [&] (int v) {
                if (d[v] > df) f = v, df = d[v];
                for (int u : g[v]) if (u != par[v])
                        d[u] = d[v] + 1, par[u] = v, dfs(u);
        };

        f = df = par[0] = -1, d[0] = 0;
        dfs(0);
        int root = f;
        f = df = par[root] = -1, d[root] = 0;
        dfs(root);

        vector<int> c;
        while (f != -1) {
                if (d[f] == df/2 or d[f] == (df+1)/2) c.push_back(f);
                f = par[f];
        }

        return {df, c};
}
```

25.6 centroid

```
// Centroid
//
// Computa os 2 centroids da arvore
//
// O(n)

int n, subsize[MAX];
vector<int> g[MAX];

void dfs(int k, int p=-1) {
        subsize[k] = 1;
        for (int i : g[k]) if (i != p) {
                dfs(i, k);
                subsize[k] += subsize[i];
        }
}

int centroid(int k, int p=-1, int size=-1) {
        if (size == -1) size = subsize[k];
        for (int i : g[k]) if (i != p) if (subsize[i] > size/2)
                return centroid(i, k, size);
        return k;
}

pair<int, int> centroids(int k=0) {
        dfs(k);
        int i = centroid(k), i2 = i;
        for (int j : g[i]) if (2*subsize[j] == subsize[k]) i2 = j;
        return {i, i2};
}
```

25.7 centroidDecomp

```
// Centroid decomposition
//
// decomp(0, k) computa numero de caminhos com 'k' arestas
// Mudar depois do comentario
//
// O(n log(n))

vector<int> g[MAX];
int sz[MAX], rem[MAX];

void dfs(vector<int>& path, int i, int l=-1, int d=0) {
        path.push_back(d);
        for (int j : g[i]) if (j != l and !rem[j]) dfs(path, j, i, d+1);
}

int dfs_sz(int i, int l=-1) {
        sz[i] = 1;
        for (int j : g[i]) if (j != l and !rem[j]) sz[i] += dfs_sz(j, i);
        return sz[i];
}

int centroid(int i, int l, int size) {
        for (int j : g[i]) if (j != l and !rem[j] and sz[j] > size / 2)
                return centroid(j, i, size);
        return i;
}

ll decomp(int i, int k) {
        int c = centroid(i, i, dfs_sz(i));
        rem[c] = 1;

        // gasta O(n) aqui - dfs sem ir pros caras removidos
        ll ans = 0;
        vector<int> cnt(sz[i]);
        cnt[0] = 1;
        for (int j : g[c]) if (!rem[j]) {
                vector<int> path;
                dfs(path, j);
                for (int d : path) if (0 <= k-d-1 and k-d-1 < sz[i])
                        ans += cnt[k-d-1];
                for (int d : path) cnt[d+1]++;
        }

        for (int j : g[c]) if (!rem[j]) ans += decomp(j, k);
        rem[c] = 0;
        return ans;
}
```

25.8 centroidTree

```cpp
// Centroid Tree
//
// Constroi a centroid tree
// p[i] eh o pai de i na centroid-tree
// dist[i][k] = distancia na arvore original entre i
// e o k-esimo ancestral na arvore da centroid
//
// O(n log(n)) de tempo e memoria

vector<int> g[MAX], dist[MAX];
int sz[MAX], rem[MAX], p[MAX];

int dfs_sz(int i, int l=-1) {
        sz[i] = 1;
        for (int j : g[i]) if (j != l and !rem[j]) sz[i] += dfs_sz(j, i);
        return sz[i];
}

int centroid(int i, int l, int size) {
        for (int j : g[i]) if (j != l and !rem[j] and sz[j] > size / 2)
                return centroid(j, i, size);
        return i;
}

void dfs_dist(int i, int l, int d=0) {
        dist[i].push_back(d);
        for (int j : g[i]) if (j != l and !rem[j])
                dfs_dist(j, i, d+1);
}

void decomp(int i, int l = -1) {
        int c = centroid(i, i, dfs_sz(i));
        rem[c] = 1, p[c] = l;
        dfs_dist(c, c);
        for (int j : g[c]) if (!rem[j]) decomp(j, c);
}

void build(int n) {
        for (int i = 0; i < n; i++) rem[i] = 0, dist[i].clear();
        decomp(0);
        for (int i = 0; i < n; i++) reverse(dist[i].begin(), dist[i].end());
}
```

25.9 cover

```cpp
// Vertex cover
//
// Encontra o tamanho do vertex cover minimo
// Da pra alterar facil pra achar os vertices
// Parece rodar com < 2 s pra N = 90
//
// O(n * 1.38^n)

namespace cover {
        const int MAX = 96;
        vector<int> g[MAX];
        bitset<MAX> bs[MAX];
        int n;

        void add(int i, int j) {
                if (i == j) return;
                n = max({n, i+1, j+1});
                bs[i][j] = bs[j][i] = 1;
        }

        int rec(bitset<MAX> m) {
                int ans = 0;
                for (int x = 0; x < n; x++) if (m[x]) {
                        bitset<MAX> comp;
                        function<void(int)> dfs = [&](int i) {
                                comp[i] = 1, m[i] = 0;
                                for (int j : g[i]) if (m[j]) dfs(j);
                        };
                        dfs(x);

                        int ma, deg = -1, cyc = 1;
                        for (int i = 0; i < n; i++) if (comp[i]) {
                                int d = (bs[i]&comp).count();
                                if (d <= 1) cyc = 0;
                                if (d > deg) deg = d, ma = i;
                        }
                        if (deg <= 2) { // caminho ou ciclo
                                ans += (comp.count() + cyc) / 2;
                                continue;
                        }
                        comp[ma] = 0;

                        // ou ta no cover, ou nao ta no cover
                        ans += min(1 + rec(comp), deg + rec(comp & ~bs[ma]));
                }
                return ans;
        }
        int solve() {
                bitset<MAX> m;
                for (int i = 0; i < n; i++) {
                        m[i] = 1;
                        for (int j = 0; j < n; j++)
                                if (bs[i][j]) g[i].push_back(j);
                }
                return rec(m);
        }
}
```

25.10 dijkstra

```cpp
// Dijkstra
//
// encontra menor distancia de x
// para todos os vertices
// se ao final do algoritmo d[i] = LINF,
// entao x nao alcanca i
//
// O(m log(n))

ll d[MAX];
vector<pair<int, int>> g[MAX]; // {vizinho, peso}

int n;

void dijkstra(int v) {
        for (int i = 0; i < n; i++) d[i] = LINF;
        d[v] = 0;
        priority_queue<pair<ll, int>> pq;
        pq.emplace(0, v);

        while (pq.size()) {
                auto [ndist, u] = pq.top(); pq.pop();
                if (-ndist > d[u]) continue;

                for (auto [idx, w] : g[u]) if (d[idx] > d[u] + w) {
                        d[idx] = d[u] + w;
                        pq.emplace(-d[idx], idx);
                }
        }
}
```

25.11 dinitz

```cpp
// Dinitz
//
// O(min(m * max_flow, n^2 m))
// Grafo com capacidades 1: O(min(m sqrt(m), m * n^(2/3)))
// Todo vertice tem grau de entrada ou saida 1: O(m sqrt(n))

struct dinitz {
```

```
        const bool scaling = false; // com scaling -> O(nm log(MAXCAP)),
        int lim;                    // com constante alta
        struct edge {
                int to, cap, rev, flow;
                bool res;
                edge(int to_, int cap_, int rev_, bool res_)
                        : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
        };

        vector<vector<edge>> g;
        vector<int> lev, beg;
        ll F;
        dinitz(int n) : g(n), F(0) {}

        void add(int a, int b, int c) {
                g[a].emplace_back(b, c, g[b].size(), false);
                g[b].emplace_back(a, 0, g[a].size()-1, true);
        }
        bool bfs(int s, int t) {
                lev = vector<int>(g.size(), -1); lev[s] = 0;
                beg = vector<int>(g.size(), 0);
                queue<int> q; q.push(s);
                while (q.size()) {
                        int u = q.front(); q.pop();
                        for (auto& i : g[u]) {
                                if (lev[i.to] != -1 or (i.flow == i.cap))
                                        continue;
                                if (scaling and i.cap - i.flow < lim) continue;
                                lev[i.to] = lev[u] + 1;
                                q.push(i.to);
                        }
                }
                return lev[t] != -1;
        }
        int dfs(int v, int s, int f = INF) {
                if (!f or v == s) return f;
                for (int& i = beg[v]; i < g[v].size(); i++) {
                        auto& e = g[v][i];
                        if (lev[e.to] != lev[v] + 1) continue;
                        int foi = dfs(e.to, s, min(f, e.cap - e.flow));
                        if (!foi) continue;
                        e.flow += foi, g[e.to][e.rev].flow -= foi;
                        return foi;
                }
                return 0;
        }
        ll max_flow(int s, int t) {
                for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
                        while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
                return F;
        }
};

// Recupera as arestas do corte s-t
vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {
        g.max_flow(s, t);
        vector<pair<int, int>> cut;
        vector<int> vis(g.g.size(), 0), st = {s};
        vis[s] = 1;
        while (st.size()) {
                int u = st.back(); st.pop_back();
                for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
                        vis[e.to] = 1, st.push_back(e.to);
        }
        for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
                if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
        return cut;
}
```

25.12 directedMst

```
// AGM Direcionada
//
// Fala o menor custo para selecionar arestas tal que
// o vertice 'r' alcance todos
// Se nao tem como, retorna LINF
//
// O(m log(n))

struct node {
        pair<ll, int> val;
        ll lazy;
        node *l, *r;
        node() {}
        node(pair<int, int> v) : val(v), lazy(0), l(NULL), r(NULL) {}

        void prop() {
                val.first += lazy;
                if (l) l->lazy += lazy;
                if (r) r->lazy += lazy;
                lazy = 0;
        }
};
void merge(node*& a, node* b) {
        if (!a) swap(a, b);
        if (!b) return;
        a->prop(), b->prop();
        if (a->val > b->val) swap(a, b);
        merge(rand()%2 ? a->l : a->r, b);
}
pair<ll, int> pop(node*& R) {
        R->prop();
        auto ret = R->val;
        node* tmp = R;
        merge(R->l, R->r);
        R = R->l;
        if (R) R->lazy -= ret.first;
        delete tmp;
        return ret;
}
void apaga(node* R) { if (R) apaga(R->l), apaga(R->r), delete R; }

ll dmst(int n, int r, vector<pair<pair<int, int>, int>>& ar) {
        vector<int> p(n); iota(p.begin(), p.end(), 0);
        function<int(int)> find = [&](int k) { return p[k]==k?k:p[k]=find(p[k]);
                };
        vector<node*> h(n);
        for (auto e : ar) merge(h[e.first.second], new node({e.second, e.first.
             first}));
        vector<int> pai(n, -1), path(n);
        pai[r] = r;
        ll ans = 0;

        for (int i = 0; i < n; i++) { // vai conectando todo mundo
                int u = i, at = 0;
                while (pai[u] == -1) {
                        if (!h[u]) { // nao tem
                                for (auto i : h) apaga(i);
                                return LINF;
                        }
                        path[at++] = u, pai[u] = i;
                        auto [mi, v] = pop(h[u]);
                        ans += mi;

                        if (pai[u = find(v)] == i) { // ciclo
                                while (find(v = path[--at]) != u)
                                        merge(h[u], h[v]), h[v] = NULL, p[find(v
                                             )] = u;
                                pai[u] = -1;
                        }
                }
        }
        for (auto i : h) apaga(i);
        return ans;
}
```

25.13 dominatorTree

```
// Dominator Tree
//
// Codigo do Kawakami. Se vira pra usar ai
//
```

```cpp
// build - O(m log(n))
// dominates - O(1)

int n;

namespace d_tree {
	vector<int> g[MAX];

	// The dominator tree
	vector<int> tree[MAX];
	int dfs_l[MAX], dfs_r[MAX];

	// Auxiliary data
	vector<int> rg[MAX], bucket[MAX];
	int idom[MAX], sdom[MAX], prv[MAX], pre[MAX];
	int ancestor[MAX], label[MAX];
	vector<int> preorder;

	void dfs(int v) {
		static int t = 0;
		pre[v] = ++t;
		sdom[v] = label[v] = v;
		preorder.push_back(v);
		for (int nxt: g[v]) {
			if (sdom[nxt] == -1) {
				prv[nxt] = v;
				dfs(nxt);
			}
			rg[nxt].push_back(v);
		}
	}
	int eval(int v) {
		if (ancestor[v] == -1) return v;
		if (ancestor[ancestor[v]] == -1) return label[v];
		int u = eval(ancestor[v]);
		if (pre[sdom[u]] < pre[sdom[label[v]]]) label[v] = u;
		ancestor[v] = ancestor[u];
		return label[v];
	}
	void dfs2(int v) {
		static int t = 0;
		dfs_l[v] = t++;
		for (int nxt: tree[v]) dfs2(nxt);
		dfs_r[v] = t++;
	}
	void build(int s) {
		for (int i = 0; i < n; i++) {
			sdom[i] = pre[i] = ancestor[i] = -1;
			rg[i].clear();
			tree[i].clear();
			bucket[i].clear();
		}
		preorder.clear();
		dfs(s);
		if (preorder.size() == 1) return;
		for (int i = int(preorder.size()) - 1; i >= 1; i--) {
			int w = preorder[i];
			for (int v: rg[w]) {
				int u = eval(v);
				if (pre[sdom[u]] < pre[sdom[w]]) sdom[w] = sdom[
				    u];
			}
			bucket[sdom[w]].push_back(w);
			ancestor[w] = prv[w];
			for (int v: bucket[prv[w]]) {
				int u = eval(v);
				idom[v] = (u == v) ? sdom[v] : u;
			}
			bucket[prv[w]].clear();
		}
		for (int i = 1; i < preorder.size(); i++) {
			int w = preorder[i];
			if (idom[w] != sdom[w]) idom[w] = idom[idom[w]];
			tree[idom[w]].push_back(w);
		}
		idom[s] = sdom[s] = -1;
		dfs2(s);
	}
	// Whether every path from s to v passes through u
	bool dominates(int u, int v) {
		if (pre[v] == -1) return 1; // vacuously true
		return dfs_l[u] <= dfs_l[v] && dfs_r[v] <= dfs_r[u];
	}
};
```

25.14 eulerPath

```cpp
// Euler Path / Euler Cycle
//
// Para declarar: 'euler<true> E(n);' se quiser
// direcionado e com 'n' vertices
// As funcoes retornam um par com um booleano
// indicando se possui o cycle/path que voce pediu,
// e um vector de {vertice, id da aresta para chegar no vertice}
// Se for get_path, na primeira posicao o id vai ser -1
// get_path(src) tenta achar um caminho ou ciclo euleriano
// comecando no vertice 'src'.
// Se achar um ciclo, o primeiro e ultimo vertice serao 'src'.
// Se for um P3, um possiveo retorno seria [0, 1, 2, 0]
// get_cycle() acha um ciclo euleriano se o grafo for euleriano.
// Se for um P3, um possivel retorno seria [0, 1, 2]
// (vertie inicial nao repete)
//
// O(n+m)

template<bool directed=false> struct euler {
	int n;
	vector<vector<pair<int, int>>> g;
	vector<int> used;

	euler(int n_) : n(n_), g(n) {}
	void add(int a, int b) {
		int at = used.size();
		used.push_back(0);
		g[a].emplace_back(b, at);
		if (!directed) g[b].emplace_back(a, at);
	}
#warning chamar para o src certo!
	pair<bool, vector<pair<int, int>>> get_path(int src) {
		if (!used.size()) return {true, {}};
		vector<int> beg(n, 0);
		for (int& i : used) i = 0;
		// {{vertice, anterior}, label}
		vector<pair<pair<int, int>, int>> ret, st = {{{src, -1}, -1}};
		while (st.size()) {
			int at = st.back().first.first;
			int& it = beg[at];
			while (it < g[at].size() and used[g[at][it].second]) it
			    ++;
			if (it == g[at].size()) {
				if (ret.size() and ret.back().first.second != at
				    )
					return {false, {}};
				ret.push_back(st.back()), st.pop_back();
			} else {
				st.push_back({{g[at][it].first, at}, g[at][it].
				    second});
				used[g[at][it].second] = 1;
			}
		}
		if (ret.size() != used.size()+1) return {false, {}};
		vector<pair<int, int>> ans;
		for (auto i : ret) ans.emplace_back(i.first.first, i.second);
		reverse(ans.begin(), ans.end());
		return {true, ans};
	}
	pair<bool, vector<pair<int, int>>> get_cycle() {
		if (!used.size()) return {true, {}};
		int src = 0;
		while (!g[src].size()) src++;
		auto ans = get_path(src);
		if (!ans.first or ans.second[0].first != ans.second.back().first
		    )
			return {false, {}};
```

```
                ans.second[0].second = ans.second.back().second;
                ans.second.pop_back();
                return ans;
        }
};
```

25.15 eulerTourTree

```
// Euler Tour Tree
//
// Mantem uma floresta enraizada dinamicamente
// e permite queries/updates em sub-arvore
//
// Chamar ETT E(n, v), passando n = numero de vertices
// e v = vector com os valores de cada vertice (se for vazio,
// constroi tudo com 0
//
// link(v, u) cria uma aresta de v pra u, de forma que u se torna
// o pai de v (eh preciso que v seja raiz anteriormente)
// cut(v) corta a resta de v para o pai
// query(v) retorna a soma dos valores da sub-arvore de v
// update(v, val) soma val em todos os vertices da sub-arvore de v
// update_v(v, val) muda o valor do vertice v para val
// is_in_subtree(v, u) responde se o vertice u esta na sub-arvore de v
//
// Tudo O(log(n)) com alta probabilidade

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

template<typename T> struct ETT {
        // treap
        struct node {
                node *l, *r, *p;
                int pr, sz;
                T val, sub, lazy;
                int id;
                bool f; // se eh o 'first'
                int qt_f; // numero de firsts na subarvore
                node(int id_, T v, bool f_ = 0) : l(NULL), r(NULL), p(NULL), pr(
                     rng()),
                        sz(1), val(v), sub(v), lazy(), id(id_), f(f_), qt_f(f_)
                             {}
                void prop() {
                        if (lazy != T()) {
                                if (f) val += lazy;
                                sub += lazy*sz;
                                if (l) l->lazy += lazy;
                                if (r) r->lazy += lazy;
                        }
                        lazy = T();
                }
                void update() {
                        sz = 1, sub = val, qt_f = f;
                        if (l) l->prop(), sz += l->sz, sub += l->sub, qt_f += l
                             ->qt_f;
                        if (r) r->prop(), sz += r->sz, sub += r->sub, qt_f += r
                             ->qt_f;
                }
        };

        node* root;

        int size(node* x) { return x ? x->sz : 0; }
        void join(node* l, node* r, node*& i) { // assume que l < r
                if (!l or !r) return void(i = l ? l : r);
                l->prop(), r->prop();
                if (l->pr > r->pr) join(l->r, r, l->r), l->r->p = i = l;
                else join(l, r->l, r->l), r->l->p = i = r;
                i->update();
        }
        void split(node* i, node*& l, node*& r, int v, int key = 0) {
                if (!i) return void(r = l = NULL);
                i->prop();
                if (key + size(i->l) < v) {
                        split(i->r, i->r, r, v, key+size(i->l)+1), l = i;
                        if (r) r->p = NULL;
                        if (i->r) i->r->p = i;
                } else {
                        split(i->l, l, i->l, v, key), r = i;
                        if (l) l->p = NULL;
                        if (i->l) i->l->p = i;
                }
                i->update();
        }
        int get_idx(node* i) {
                int ret = size(i->l);
                for (; i->p; i = i->p) {
                        node* pai = i->p;
                        if (i != pai->l) ret += size(pai->l) + 1;
                }
                return ret;
        }
        node* get_min(node* i) {
                if (!i) return NULL;
                return i->l ? get_min(i->l) : i;
        }
        node* get_max(node* i) {
                if (!i) return NULL;
                return i->r ? get_max(i->r) : i;
        }
        // fim da treap

        vector<node*> first, last;

        ETT(int n, vector<T> v = {}) : root(NULL), first(n), last(n) {
                if (!v.size()) v = vector<T>(n);
                for (int i = 0; i < n; i++) {
                        first[i] = last[i] = new node(i, v[i], 1);
                        join(root, first[i], root);
                }
        }
        ETT(const ETT& t) { throw logic_error("Nao copiar a ETT!"); }
        ~ETT() {
                vector<node*> q = {root};
                while (q.size()) {
                        node* x = q.back(); q.pop_back();
                        if (!x) continue;
                        q.push_back(x->l), q.push_back(x->r);
                        delete x;
                }
        }

        pair<int, int> get_range(int i) {
                return {get_idx(first[i]), get_idx(last[i])};
        }
        void link(int v, int u) { // 'v' tem que ser raiz
                auto [lv, rv] = get_range(v);
                int ru = get_idx(last[u]);

                node* V;
                node *L, *M, *R;
                split(root, M, R, rv+1), split(M, L, M, lv);
                V = M;
                join(L, R, root);

                split(root, L, R, ru+1);
                join(L, V, L);
                join(L, last[u] = new node(u, T() /* elemento neutro */), L);
                join(L, R, root);
        }
        void cut(int v) {
                auto [l, r] = get_range(v);

                node *L, *M, *R;
                split(root, M, R, r+1), split(M, L, M, l);
                node *LL = get_max(L), *RR = get_min(R);
                if (LL and RR and LL->id == RR->id) { // remove duplicata
                        if (last[RR->id] == RR) last[RR->id] = LL;
                        node *A, *B;
                        split(R, A, B, 1);
                        delete A;
                        R = B;
                }
                join(L, R, root);
                join(root, M, root);
```

```
        }
        T query(int v) {
                auto [l, r] = get_range(v);
                node *L, *M, *R;
                split(root, M, R, r+1), split(M, L, M, l);
                T ans = M->sub;
                join(L, M, M), join(M, R, root);
                return ans;
        }
        void update(int v, T val) { // soma val em todo mundo da subarvore
                auto [l, r] = get_range(v);
                node *L, *M, *R;
                split(root, M, R, r+1), split(M, L, M, l);
                M->lazy += val;
                join(L, M, M), join(M, R, root);
        }
        void update_v(int v, T val) { // muda o valor de v pra val
                int l = get_idx(first[v]);
                node *L, *M, *R;
                split(root, M, R, l+1), split(M, L, M, l);
                M->val = M->sub = val;
                join(L, M, M), join(M, R, root);
        }
        bool is_in_subtree(int v, int u) { // se u ta na subtree de v
                auto [lv, rv] = get_range(v);
                auto [lu, ru] = get_range(u);
                return lv <= lu and ru <= rv;
        }

        void print(node* i) {
                if (!i) return;
                print(i->l);
                cout << i->id+1 << " ";
                print(i->r);
        }
        void print() { print(root); cout << endl; }
};
```

25.16 floydWarshall

```
// Floyd-Warshall
//
// encontra o menor caminho entre todo
// par de vertices e detecta ciclo negativo
// returna 1 sse ha ciclo negativo
// d[i][i] deve ser 0
// para i != j, d[i][j] deve ser w se ha uma aresta
// (i, j) de peso w, INF caso contrario
//
// O(n^3)

int n;
int d[MAX][MAX];

bool floyd_warshall() {
        for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

        for (int i = 0; i < n; i++)
                if (d[i][i] < 0) return 1;

        return 0;
}
```

25.17 functionalGraph

```
// Functional Graph
//
// rt[i] fala o ID da raiz associada ao vertice i
// d[i] fala a profundidade (0 sse ta no ciclo)
// pos[i] fala a posicao de i no array que eh a concat. dos ciclos
// build(f, val) recebe a funcao f e o custo de ir de
// i para f[i] (por default, val = f)
// f_k(i, k) fala onde i vai parar se seguir k arestas
// path(i, k) fala o custo (soma) seguir k arestas a partir de i
// Se quiser outra operacao, da pra alterar facil o codigo
// Codigo um pouco louco, tenho que admitir
//
// build - O(n)
// f_k   - O(log(min(n, k)))
// path  - O(log(min(n, k)))

namespace func_graph {
        int n;
        int f[MAX], vis[MAX], d[MAX];
        int p[MAX], pp[MAX], rt[MAX], pos[MAX];
        int sz[MAX], comp;
        vector<vector<int>> ciclo;
        ll val[MAX], jmp[MAX], seg[2*MAX];

        ll op(ll a, ll b) { return a+b; }; // mudar a operacao aqui
        void dfs(int i, int t = 2) {
                vis[i] = t;
                if (vis[f[i]] >= 2) { // comeca ciclo - f[i] eh o rep.
                        d[i] = 0, rt[i] = comp;
                        sz[comp] = t - vis[f[i]] + 1;
                        p[i] = pp[i] = i, jmp[i] = val[i];
                        ciclo.emplace_back();
                        ciclo.back().push_back(i);
                } else {
                        if (!vis[f[i]]) dfs(f[i], t+1);
                        rt[i] = rt[f[i]];
                        if (sz[comp]+1) { // to no ciclo
                                d[i] = 0;
                                p[i] = pp[i] = i, jmp[i] = val[i];
                                ciclo.back().push_back(i);
                        } else { // nao to no ciclo
                                d[i] = d[f[i]]+1, p[i] = f[i];
                                pp[i] = 2*d[pp[f[i]]] == d[pp[pp[f[i]]]]+d[f[i]]
                                        ? pp[pp[f[i]]] : f[i];
                                jmp[i] = pp[i] == f[i] ? val[i] : op(val[i], op(
                                        jmp[f[i]], jmp[pp[f[i]]]));
                        }
                }
                if (f[ciclo[rt[i]][0]] == i) comp++; // fim do ciclo
                vis[i] = 1;
        }
        void build(vector<int> f_, vector<int> val_ = {}) {
                n = f_.size(), comp = 0;
                if (!val_.size()) val_ = f_;
                for (int i = 0; i < n; i++)
                        f[i] = f_[i], val[i] = val_[i], vis[i] = 0, sz[i] = -1;

                ciclo.clear();
                for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);
                int t = 0;
                for (auto& c : ciclo) {
                        reverse(c.begin(), c.end());
                        for (int j : c) {
                                pos[j] = t;
                                seg[n+t] = val[j];
                                t++;
                        }
                }
                for (int i = n-1; i; i--) seg[i] = op(seg[2*i], seg[2*i+1]);
        }

        int f_k(int i, ll k) {
                while (d[i] and k) {
                        int big = d[i] - d[pp[i]];
                        if (big <= k) k -= big, i = pp[i];
                        else k--, i = p[i];
                }
                if (!k) return i;
                return ciclo[rt[i]][(pos[i] - pos[ciclo[rt[i]][0]] + k) % sz[rt[
                    i]]];
        }
        ll path(int i, ll k) {
                auto query = [&](int l, int r) {
```

```
                ll q = 0;
                for (l += n, r += n; l <= r; ++l/=2, --r/=2) {
                        if (l%2 == 1) q = op(q, seg[l]);
                        if (r%2 == 0) q = op(q, seg[r]);
                }
                return q;
        };
        ll ret = 0;
        while (d[i] and k) {
                int big = d[i] - d[pp[i]];
                if (big <= k) k -= big, ret = op(ret, jmp[i]), i = pp[i
                    ];
                else k--, ret = op(ret, val[i]), i = p[i];
        }
        if (!k) return ret;
        int first = pos[ciclo[rt[i]][0]], last = pos[ciclo[rt[i]].back()
            ];

        // k/sz[rt[i]] voltas completas
        if (k/sz[rt[i]]) ret = op(ret, k/sz[rt[i]] * query(first, last))
            ;

        k %= sz[rt[i]];
        if (!k) return ret;
        int l = pos[i], r = first + (pos[i] - first + k - 1) % sz[rt[i
            ]];
        if (l <= r) return op(ret, query(l, r));
        return op(ret, op(query(l, last), query(first, r)));
    }
}
```

25.18 hopcroftKarp

```
// Hopcroft Karp
//
// Computa matching maximo em grafo bipartido
// 'n' e 'm' sao quantos vertices tem em cada particao
// chamar add(i, j) para add aresta entre o cara i
// da particao A, e o cara j da particao B
// (entao i < n, j < m)
//
// O(|E| * sqrt(|V|)) com constante baixa
// Para grafos esparsos gerados aleatoriamente, roda em O(|E| * log(|V|))
// com alta probabilidade

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

struct hopcroft_karp {
        int n, m;
        vector<vector<int>> g;
        vector<int> dist, nxt, ma, mb;

        hopcroft_karp(int n_, int m_) : n(n_), m(m_), g(n),
                dist(n), nxt(n), ma(n, -1), mb(m, -1) {}

        void add(int a, int b) { g[a].push_back(b); }

        bool dfs(int i) {
                for (int &id = nxt[i]; id < g[i].size(); id++) {
                        int j = g[i][id];
                        if (mb[j] == -1 or (dist[mb[j]] == dist[i] + 1 and dfs(
                            mb[j]))) {
                                ma[i] = j, mb[j] = i;
                                return true;
                        }
                }
                return false;
        }
        bool bfs() {
                for (int i = 0; i < n; i++) dist[i] = n;
                queue<int> q;
                for (int i = 0; i < n; i++) if (ma[i] == -1) {
                        dist[i] = 0;
                        q.push(i);
                }
                bool rep = 0;
                while (q.size()) {
                        int i = q.front(); q.pop();
                        for (int j : g[i]) {
                                if (mb[j] == -1) rep = 1;
                                else if (dist[mb[j]] > dist[i] + 1) {
                                        dist[mb[j]] = dist[i] + 1;
                                        q.push(mb[j]);
                                }
                        }
                }
                return rep;
        }
        int matching() {
                int ret = 0;
                for (auto& i : g) shuffle(i.begin(), i.end(), rng);
                while (bfs()) {
                        for (int i = 0; i < n; i++) nxt[i] = 0;
                        for (int i = 0; i < n; i++)
                                if (ma[i] == -1 and dfs(i)) ret++;
                }
                return ret;
        }
};
```

25.19 johnson

```
// Johnson
//
// funciona igual ao Floyd-Warshall
// encontra o menor caminho entre todo
// par de vertices e retorna 1 sse tem
// ciclo negativo no grafo
//
// O(nm log(m))

vector<pair<int, ll>> g[MAX]; // {vizinho, peso}
ll d[MAX][MAX];

bool johnson(int n) {
        vector<ll> h(n, 0);
        for (int i = 0; i <= n; i++)
                for (int v = 0; v < n; v++)
                        for (auto [u, w] : g[v]) if (h[u] > h[v] + w) {
                                if (i == n) return 1;
                                h[u] = h[v] + w;
                        }

        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) d[i][j] = LINF;
                d[i][i] = 0;
                priority_queue<pair<ll, int>> pq;
                pq.emplace(0, i);
                while (pq.size()) {
                        auto [ndist, v] = pq.top(); pq.pop();
                        if (-ndist > d[i][v]) continue;

                        for (auto [u, w] : g[v]) {
                                w += h[v] - h[u];
                                if (d[i][u] > d[i][v] + w) {
                                        d[i][u] = d[i][v] + w;
                                        pq.emplace(-d[i][u], u);
                                }
                        }
                }
                for (int j = 0; j < n; j++)
                        d[i][j] += h[j] - h[i];
        }

        return 0;
}
```

25.20 kosaraju

```
// Kosaraju
//
// O(n + m)

int n;
vector<int> g[MAX];
vector<int> gi[MAX]; // grafo invertido
int vis[MAX];
stack<int> S;
int comp[MAX]; // componente conexo de cada vertice

void dfs(int k) {
        vis[k] = 1;
        for (int i = 0; i < (int) g[k].size(); i++)
                if (!vis[g[k][i]]) dfs(g[k][i]);

        S.push(k);
}

void scc(int k, int c) {
        vis[k] = 1;
        comp[k] = c;
        for (int i = 0; i < (int) gi[k].size(); i++)
                if (!vis[gi[k][i]]) scc(gi[k][i], c);
}

void kosaraju() {
        for (int i = 0; i < n; i++) vis[i] = 0;
        for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

        for (int i = 0; i < n; i++) vis[i] = 0;
        while (S.size()) {
                int u = S.top();
                S.pop();
                if (!vis[u]) scc(u, u);
        }
}
```

25.21 kruskal

```
// Kruskal
//
// Gera e retorna uma AGM e seu custo total a partir do vetor de arestas (edg)
// do grafo
//
// O(m log(m) + m a(m))
// 864875

vector<tuple<int, int, int>> edg; // {peso,[x,y]}

// DSU em O(a(n))
void dsu_build();
int find(int a);
void unite(int a, int b);

pair<ll, vector<tuple<int, int, int>>> kruskal(int n) {
        dsu_build(n);
        sort(edg.begin(), edg.end());

        ll cost = 0;
        vector<tuple<int, int, int>> mst;
        for (auto [w,x,y] : edg) if (find(x) != find(y)) {
                mst.emplace_back(w, x, y);
                cost += w;
                unite(x,y);
        }
        return {cost, mst};
}
```

25.22 kuhn

```
// Kuhn
//
// Computa matching maximo em grafo bipartido
// 'n' e 'm' sao quantos vertices tem em cada particao
// chamar add(i, j) para add aresta entre o cara i
// da particao A, e o cara j da particao B
// (entao i < n, j < m)
// Para recuperar o matching, basta olhar 'ma' e 'mb'
// 'recover' recupera o min vertex cover como um par de
// {caras da particao A, caras da particao B}
//
// O(|V| * |E|)
// Na pratica, parece rodar tao rapido quanto o Dinitz

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

struct kuhn {
        int n, m;
        vector<vector<int>> g;
        vector<int> vis, ma, mb;

        kuhn(int n_, int m_) : n(n_), m(m_), g(n),
                vis(n+m), ma(n, -1), mb(m, -1) {}

        void add(int a, int b) { g[a].push_back(b); }

        bool dfs(int i) {
                vis[i] = 1;
                for (int j : g[i]) if (!vis[n+j]) {
                        vis[n+j] = 1;
                        if (mb[j] == -1 or dfs(mb[j])) {
                                ma[i] = j, mb[j] = i;
                                return true;
                        }
                }
                return false;
        }
        int matching() {
                int ret = 0, aum = 1;
                for (auto& i : g) shuffle(i.begin(), i.end(), rng);
                while (aum) {
                        for (int j = 0; j < m; j++) vis[n+j] = 0;
                        aum = 0;
                        for (int i = 0; i < n; i++)
                                if (ma[i] == -1 and dfs(i)) ret++, aum = 1;
                }
                return ret;
        }
};

pair<vector<int>, vector<int>> recover(kuhn& K) {
        K.matching();
        int n = K.n, m = K.m;
        for (int i = 0; i < n+m; i++) K.vis[i] = 0;
        for (int i = 0; i < n; i++) if (K.ma[i] == -1) K.dfs(i);
        vector<int> ca, cb;
        for (int i = 0; i < n; i++) if (!K.vis[i]) ca.push_back(i);
        for (int i = 0; i < m; i++) if (K.vis[n+i]) cb.push_back(i);
        return {ca, cb};
}
```

25.23 linetree

```
// Line Tree
//
// Reduz min-query em arvore para RMQ
// Se o grafo nao for uma arvore, as queries
// sao sobre a arvore geradora maxima
// Queries de minimo
//
// build - O(n log(n))
// query - O(log(n))

int n;

namespace linetree {
```

```
        int id[MAX], seg[2*MAX], pos[MAX];
        vector<int> v[MAX], val[MAX];
        vector<pair<int, pair<int, int> > > ar;

        void add(int a, int b, int p) { ar.push_back({p, {a, b}}); }
        void build() {
                sort(ar.rbegin(), ar.rend());
                for (int i = 0; i < n; i++) id[i] = i, v[i] = {i}, val[i].clear
                     ();
                for (auto i : ar) {
                        int a = id[i.second.first], b = id[i.second.second];
                        if (a == b) continue;
                        if (v[a].size() < v[b].size()) swap(a, b);
                        for (auto j : v[b]) id[j] = a, v[a].push_back(j);
                        val[a].push_back(i.first);
                        for (auto j : val[b]) val[a].push_back(j);
                        v[b].clear(), val[b].clear();
                }
                vector<int> vv;
                for (int i = 0; i < n; i++) for (int j = 0; j < v[i].size(); j
                     ++) {
                        pos[v[i][j]] = vv.size();
                        if (j + 1 < v[i].size()) vv.push_back(val[i][j]);
                        else vv.push_back(0);
                }
                for (int i = n; i < 2*n; i++) seg[i] = vv[i-n];
                for (int i = n-1; i; i--) seg[i] = min(seg[2*i], seg[2*i+1]);
        }
        int query(int a, int b) {
                if (id[a] != id[b]) return 0; // nao estao conectados
                a = pos[a], b = pos[b];
                if (a > b) swap(a, b);
                b--;
                int ans = INF;
                for (a += n, b += n; a <= b; ++a/=2, --b/=2) ans = min({ans, seg
                     [a], seg[b]});
                return ans;
        }
};
```

25.24 lowerBoundMaxFlow

```
// Max flow com lower bound
//
// add(a, b, l, r):
//      adiciona aresta de a pra b, onde precisa passar f de fluxo, l <= f <= r
// add(a, b, c):
//      adiciona aresta de a pra b com capacidade c
//
// Mesma complexidade do Dinitz

struct lb_max_flow : dinitz {
        vector<int> d;
        lb_max_flow(int n) : dinitz(n + 2), d(n, 0) {}
        void add(int a, int b, int l, int r) {
                d[a] -= l;
                d[b] += l;
                dinitz::add(a, b, r - l);
        }
        void add(int a, int b, int c) {
                dinitz::add(a, b, c);
        }
        bool has_circulation() {
                int n = d.size();

                ll cost = 0;
                for (int i = 0; i < n; i++) {
                        if (d[i] > 0) {
                                cost += d[i];
                                dinitz::add(n, i, d[i]);
                        } else if (d[i] < 0) {
                                dinitz::add(i, n+1, -d[i]);
                        }
                }

                return (dinitz::max_flow(n, n+1) == cost);
        }
        bool has_flow(int src, int snk) {
                dinitz::add(snk, src, INF);
                return has_circulation();
        }
        ll max_flow(int src, int snk) {
                if (!has_flow(src, snk)) return -1;
                dinitz::F = 0;
                return dinitz::max_flow(src, snk);
        }
};
```

25.25 minCostMaxFlow

```
// MinCostMaxFlow
//
// min_cost_flow(s, t, f) computa o par (fluxo, custo)
// com max(fluxo) <= f que tenha min(custo)
// min_cost_flow(s, t) -> Fluxo maximo de custo minimo de s pra t
// Se for um dag, da pra substituir o SPFA por uma DP pra nao
// pagar O(nm) no comeco
// Se nao tiver aresta com custo negativo, nao precisa do SPFA
//
// O(nm + f * m log n)

template<typename T> struct mcmf {
        struct edge {
                int to, rev, flow, cap; // para, id da reversa, fluxo,
                     capacidade
                bool res; // se eh reversa
                T cost; // custo da unidade de fluxo
                edge() : to(0), rev(0), flow(0), cap(0), cost(0), res(false) {}
                edge(int to_, int rev_, int flow_, int cap_, T cost_, bool res_)
                        : to(to_), rev(rev_), flow(flow_), cap(cap_), res(res_),
                              cost(cost_) {}
        };

        vector<vector<edge>> g;
        vector<int> par_idx, par;
        T inf;
        vector<T> dist;

        mcmf(int n) : g(n), par_idx(n), par(n), inf(numeric_limits<T>::max()/3)
             {}

        void add(int u, int v, int w, T cost) { // de u pra v com cap w e custo
             cost
                edge a = edge(v, g[v].size(), 0, w, cost, false);
                edge b = edge(u, g[u].size(), 0, 0, -cost, true);

                g[u].push_back(a);
                g[v].push_back(b);
        }

        vector<T> spfa(int s) { // nao precisa se nao tiver custo negativo
                deque<int> q;
                vector<bool> is_inside(g.size(), 0);
                dist = vector<T>(g.size(), inf);

                dist[s] = 0;
                q.push_back(s);
                is_inside[s] = true;

                while (!q.empty()) {
                        int v = q.front();
                        q.pop_front();
                        is_inside[v] = false;

                        for (int i = 0; i < g[v].size(); i++) {
                                auto [to, rev, flow, cap, res, cost] = g[v][i];
                                if (flow < cap and dist[v] + cost < dist[to]) {
                                        dist[to] = dist[v] + cost;

                                        if (is_inside[to]) continue;
                                        if (!q.empty() and dist[to] > dist[q.
                                             front()]) q.push_back(to);
```

```
                                else q.push_front(to);
                                is_inside[to] = true;
                        }
                }
        }
        return dist;
}
bool dijkstra(int s, int t, vector<T>& pot) {
        priority_queue<pair<T, int>, vector<pair<T, int>>, greater<>> q;
        dist = vector<T>(g.size(), inf);
        dist[s] = 0;
        q.emplace(0, s);
        while (q.size()) {
                auto [d, v] = q.top();
                q.pop();
                if (dist[v] < d) continue;
                for (int i = 0; i < g[v].size(); i++) {
                        auto [to, rev, flow, cap, res, cost] = g[v][i];
                        cost += pot[v] - pot[to];
                        if (flow < cap and dist[v] + cost < dist[to]) {
                                dist[to] = dist[v] + cost;
                                q.emplace(dist[to], to);
                                par_idx[to] = i, par[to] = v;
                        }
                }
        }
        return dist[t] < inf;
}

pair<int, T> min_cost_flow(int s, int t, int flow = INF) {
        vector<T> pot(g.size(), 0);
        pot = spfa(s); // mudar algoritmo de caminho minimo aqui

        int f = 0;
        T ret = 0;
        while (f < flow and dijkstra(s, t, pot)) {
                for (int i = 0; i < g.size(); i++)
                        if (dist[i] < inf) pot[i] += dist[i];

                int mn_flow = flow - f, u = t;
                while (u != s){
                        mn_flow = min(mn_flow,
                                g[par[u]][par_idx[u]].cap - g[par[u]][
                                        par_idx[u]].flow);
                        u = par[u];
                }

                ret += pot[t] * mn_flow;

                u = t;
                while (u != s) {
                        g[par[u]][par_idx[u]].flow += mn_flow;
                        g[u][g[par[u]][par_idx[u]].rev].flow -= mn_flow;
                        u = par[u];
                }

                f += mn_flow;
        }

        return make_pair(f, ret);
}

// Opcional: retorna as arestas originais por onde passa flow = cap
vector<pair<int,int>> recover() {
        vector<pair<int,int>> used;
        for (int i = 0; i < g.size(); i++) for (edge e : g[i])
                if(e.flow == e.cap && !e.res) used.push_back({i, e.to});
        return used;
}
};
```

25.26 prufer

```
// Prufer code
//
// Traduz de lista de arestas para prufer code
// e vice-versa
// Os vertices tem label de 0 a n-1
// Todo array com n-2 posicoes e valores de
// 0 a n-1 sao prufer codes validos
//
// O(n)

vector<int> to_prufer(vector<pair<int, int>> tree) {
        int n = tree.size()+1;
        vector<int> d(n, 0);
        vector<vector<int>> g(n);
        for (auto [a, b] : tree) d[a]++, d[b]++,
                g[a].push_back(b), g[b].push_back(a);
        vector<int> pai(n, -1);
        queue<int> q; q.push(n-1);
        while (q.size()) {
                int u = q.front(); q.pop();
                for (int v : g[u]) if (v != pai[u])
                        pai[v] = u, q.push(v);
        }
        int idx, x;
        idx = x = find(d.begin(), d.end(), 1) - d.begin();
        vector<int> ret;
        for (int i = 0; i < n-2; i++) {
                int y = pai[x];
                ret.push_back(y);
                if (--d[y] == 1 and y < idx) x = y;
                else idx = x = find(d.begin()+idx+1, d.end(), 1) - d.begin();
        }
        return ret;
}

vector<pair<int, int>> from_prufer(vector<int> p) {
        int n = p.size()+2;
        vector<int> d(n, 1);
        for (int i : p) d[i]++;
        p.push_back(n-1);
        int idx, x;
        idx = x = find(d.begin(), d.end(), 1) - d.begin();
        vector<pair<int, int>> ret;
        for (int y : p) {
                ret.push_back({x, y});
                if (--d[y] == 1 and y < idx) x = y;
                else idx = x = find(d.begin()+idx+1, d.end(), 1) - d.begin();
        }
        return ret;
}
```

25.27 sack

```
// Sack (DSU em arvores)
//
// Responde queries de todas as sub-arvores
// offline
//
// O(n log(n))

int sz[MAX], cor[MAX], cnt[MAX];
vector<int> g[MAX];

void build(int k, int d=0) {
        sz[k] = 1;
        for (auto& i : g[k]) {
                build(i, d+1); sz[k] += sz[i];
                if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
        }
}

void compute(int k, int x, bool dont=1) {
        cnt[cor[k]] += x;
        for (int i = dont; i < g[k].size(); i++)
                compute(g[k][i], x, 0);
}

void solve(int k, bool keep=0) {
```

```
        for (int i = int(g[k].size())-1; i >= 0; i--)
                solve(g[k][i], !i);
        compute(k, 1);

        // agora cnt[i] tem quantas vezes a cor
        // i aparece na sub-arvore do k

        if (!keep) compute(k, -1, 0);
}
```

25.28 stableMarriage

```
// Stable Marriage
//
// Emparelha todos os elementos de A com elementos de B
// de forma que nao exista um par x \in A, y \in B
// e x nao pareado com y tal que x prefira parear com y
// e y prefira parear com x.
//
// a[i] contem os elementos de B ordenados por preferencia de i
// b[j] contem os elementos de A ordenados por preferencia de j
// |A| <= |B|
//
// Retorna um vetor v de tamanho |A| onde v[i] guarda o match de i.
//
// O(|A| * |B|)

vector<int> stable_marriage(vector<vector<int>> &a, vector<vector<int>> &b) {
        int n = a.size(), m = b.size();
        assert(a[0].size() == m and b[0].size() == n and n <= m);
        vector<int> match(m, -1), it(n, 0);
        vector inv_b(m, vector<int>(n));
        for (int i = 0; i < m; i++) for (int j = 0; j < n; j++)
                inv_b[i][b[i][j]] = j;

        queue<int> q;
        for (int i = 0; i < n; i++) q.push(i);
        while (q.size()) {
                int i = q.front(); q.pop();
                int j = a[i][it[i]];

                if (match[j] == -1) match[j] = i;
                else if (inv_b[j][i] < inv_b[j][match[j]]) {
                        q.emplace(match[j]);
                        it[match[j]]++;
                        match[j] = i;
                } else q.emplace(i), it[i]++;
        }

        vector<int> ret(n);
        for (int i = 0; i < m; i++) if (match[i] != -1) ret[match[i]] = i;
        return ret;
}
```

25.29 tarjan

```
// Tarjan para SCC
//
// O(n + m)

vector<int> g[MAX];
stack<int> s;
int vis[MAX], comp[MAX];
int id[MAX];

// se quiser comprimir ciclo ou achar ponte em grafo nao direcionado,
// colocar um if na dfs para nao voltar pro pai da DFS tree
int dfs(int i, int& t) {
        int lo = id[i] = t++;
        s.push(i);
        vis[i] = 2;
        for (int j : g[i]) {
                if (!vis[j]) lo = min(lo, dfs(j, t));
                else if (vis[j] == 2) lo = min(lo, id[j]);
        }

        // aresta de i pro pai eh uma ponte (no caso nao direcionado)
        if (lo == id[i]) while (1) {
                int u = s.top(); s.pop();
                vis[u] = 1, comp[u] = i;
                if (u == i) break;
        }

        return lo;
}

void tarjan(int n) {
        int t = 0;
        for (int i = 0; i < n; i++) vis[i] = 0;

        for (int i = 0; i < n; i++) if (!vis[i]) dfs(i, t);
}
```

25.30 topoSort

```
// Topological Sort
//
// Retorna uma ordenacaoo topologica de g
// Se g nao for DAG retorna um vetor vazio
//
// O(n + m)

vector<int> g[MAX];

vector<int> topo_sort(int n) {
        vector<int> ret(n,-1), vis(n,0);

        int pos = n-1, dag = 1;
        function<void(int)> dfs = [&](int v) {
                vis[v] = 1;
                for (auto u : g[v]) {
                        if (vis[u] == 1) dag = 0;
                        else if (!vis[u]) dfs(u);
                }
                ret[pos--] = v, vis[v] = 2;
        };

        for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

        if (!dag) ret.clear();
        return ret;
}
```

25.31 treeIsomorf

```
// Isomorfismo de arvores
//
// thash() retorna o hash da arvore (usando centroids como vertices especiais).
// Duas arvores sao isomorfas sse seu hash eh o mesmo
//
// O(|V|.log(|V|))

map<vector<int>, int> mphash;

struct tree {
        int n;
        vector<vector<int>> g;
        vector<int> sz, cs;

        tree(int n_) : n(n_), g(n_), sz(n_) {}

        void dfs_centroid(int v, int p) {
```

```
                sz[v] = 1;
                bool cent = true;
                for (int u : g[v]) if (u != p) {
                        dfs_centroid(u, v), sz[v] += sz[u];
                        if(sz[u] > n/2) cent = false;
                }
                if (cent and n - sz[v] <= n/2) cs.push_back(v);
        }
        int fhash(int v, int p) {
                vector<int> h;
                for (int u : g[v]) if (u != p) h.push_back(fhash(u, v));
                sort(h.begin(), h.end());
                if (!mphash.count(h)) mphash[h] = mphash.size();
                return mphash[h];
        }
        ll thash() {
                cs.clear();
                dfs_centroid(0, -1);
                if (cs.size() == 1) return fhash(cs[0], -1);
                ll h1 = fhash(cs[0], cs[1]), h2 = fhash(cs[1], cs[0]);
                return (min(h1, h2) << 30) + max(h1, h2);
        }
};


// Versao mais rapida com hash, ideal para hash de floresta.
// subtree_hash(v, p) retorna o hash da subarvore enraizada em v com pai p.
// tree_hash() retorna o hash da arvore.
// forest_hash() retorna o hash da floresta.
// use o vetor forb[] para marcar vertices que nao podem ser visitados.
//
// O(|V|.log(|V|))

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

int uniform(ll l, ll r) {
        uniform_int_distribution<ll> uid(l, r);
        return uid(rng);
}

const int MOD = 1e9 + 7;
const int H = 13;
const int P = uniform(1, MOD-1);
const int P2 = uniform(1, MOD-1);

struct tree {
        int fn;
        vector<vector<int>> g;
        vector<int> sz, cs;
        vector<bool> forb;

        tree(int n_) : fn(n_), g(n_), sz(n_), forb(n_) {}

        void dfs_size(int v, int p) {
                sz[v] = 1;
                for (int u : g[v]) if (u != p and !forb[u]) {
                        dfs_size(u, v), sz[v] += sz[u];
                }
        }
        void dfs_centroid(int v, int p, int n) {
                bool cent = true;
                for (int u : g[v]) if (u != p and !forb[u]) {
                        dfs_centroid(u, v, n);
                        if(sz[u] > n/2) cent = false;
                }
                if (cent and n - sz[v] <= n/2) cs.push_back(v);
        }
        int subtree_hash(int v, int p) {
                int h = H;
                for (int u : g[v]) if (u != p and !forb[u]) {
                        h = ll(h) * (P + subtree_hash(u, v)) %  MOD;
                }
                return h;
        }
        int tree_hash(int v=0) {
                cs.clear();
                dfs_size(v, -1);
                dfs_centroid(v, -1, sz[v]);
                if (cs.size() == 1) return subtree_hash(cs[0], -1);
                assert (cs.size() == 2);
                int h1 = subtree_hash(cs[0], cs[1]);
                int h2 = subtree_hash(cs[1], cs[0]);
                return ll(P + h1) * (P + h2) % MOD;
        }
        int forest_hash() {
                fill(sz.begin(), sz.end(), 0);
                int hash = 1;
                for (int v = 0; v < fn; v++) if (!sz[v] and !forb[v]) {
                        hash = hash * ll(P2 + tree_hash(v)) % MOD;
                }
                return hash;
        }
};
```

25.32 virtualTree

```
// Virtual Tree
//
// Comprime uma arvore dado um conjunto S de vertices, de forma que
// o conjunto de vertices da arvore comprimida contenha S e seja
// minimal e fechado sobre a operacao de LCA
// Se |S| = k, a arvore comprimida tem menos que 2k vertices
// As arestas de virt possuem a distancia do vertice ate o vizinho
// Retorna a raiz da virtual tree
//
// lca::pos deve ser a ordem de visitacao no dfs
// voce pode usar o LCAcomHLD, por exemplo
//
// O(k log(k))

vector<pair<int, int>> virt[MAX];

#warning lembrar de buildar o LCA antes
int build_virt(vector<int> v) {
        auto cmp = [&](int i, int j) { return lca::pos[i] < lca::pos[j]; };
        sort(v.begin(), v.end(), cmp);
        for (int i = v.size()-1; i; i--) v.push_back(lca::lca(v[i], v[i-1]));
        sort(v.begin(), v.end(), cmp);
        v.erase(unique(v.begin(), v.end()), v.end());
        for (int i = 0; i < v.size(); i++) virt[v[i]].clear();
        for (int i = 1; i < v.size(); i++) virt[lca::lca(v[i-1], v[i])].clear();
        for (int i = 1; i < v.size(); i++) {
                int parent = lca::lca(v[i-1], v[i]);
                int d = lca::dist(parent, v[i]);
#warning soh to colocando aresta descendo
                virt[parent].emplace_back(v[i], d);
        }
        return v[0];
}
```

26 ufmg forked/Grafos/LCA-HLD

26.1 hldAresta

```
// HLD - aresta
//
// SegTree de soma
// query / update de soma das arestas
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

// namespace seg { ... }
```

```
namespace hld {
        vector<pair<int, int> > g[MAX];
        int pos[MAX], sz[MAX];
        int sobe[MAX], pai[MAX];
        int h[MAX], v[MAX], t;

        void build_hld(int k, int p = -1, int f = 1) {
                v[pos[k] = t++] = sobe[k]; sz[k] = 1;
                for (auto& i : g[k]) if (i.first != p) {
                        auto [u, w] = i;
                        sobe[u] = w; pai[u] = k;
                        h[u] = (i == g[k][0] ? h[k] : u);
                        build_hld(u, k, f); sz[k] += sz[u];

                        if (sz[u] > sz[g[k][0].first] or g[k][0].first == p)
                                swap(i, g[k][0]);
                }
                if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
        }
        void build(int root = 0) {
                t = 0;
                build_hld(root);
                seg::build(t, v);
        }
        ll query_path(int a, int b) {
                if (a == b) return 0;
                if (pos[a] < pos[b]) swap(a, b);

                if (h[a] == h[b]) return seg::query(pos[b]+1, pos[a]);
                return seg::query(pos[h[a]], pos[a]) + query_path(pai[h[a]], b);
        }
        void update_path(int a, int b, int x) {
                if (a == b) return;
                if (pos[a] < pos[b]) swap(a, b);

                if (h[a] == h[b]) return (void)seg::update(pos[b]+1, pos[a], x);
                seg::update(pos[h[a]], pos[a], x); update_path(pai[h[a]], b, x);
        }
        ll query_subtree(int a) {
                if (sz[a] == 1) return 0;
                return seg::query(pos[a]+1, pos[a]+sz[a]-1);
        }
        void update_subtree(int a, int x) {
                if (sz[a] == 1) return;
                seg::update(pos[a]+1, pos[a]+sz[a]-1, x);
        }
        int lca(int a, int b) {
                if (pos[a] < pos[b]) swap(a, b);
                return h[a] == h[b] ? b : lca(pai[h[a]], b);
        }
}
```

26.2 hldSemUpdate

```
// HLD sem Update
//
// query de min do caminho
//
// Complexidades:
// build - O(n)
// query_path - O(log(n))

namespace hld {
        vector<pair<int, int> > g[MAX];
        int pos[MAX], sz[MAX];
        int sobe[MAX], pai[MAX];
        int h[MAX], v[MAX], t;
        int men[MAX], seg[2*MAX];

        void build_hld(int k, int p = -1, int f = 1) {
                v[pos[k] = t++] = sobe[k]; sz[k] = 1;
                for (auto& i : g[k]) if (i.first != p) {
                        sobe[i.first] = i.second; pai[i.first] = k;
                        h[i.first] = (i == g[k][0] ? h[k] : i.first);
                        men[i.first] = (i == g[k][0] ? min(men[k], i.second) : i
                             .second);
                        build_hld(i.first, k, f); sz[k] += sz[i.first];

                        if (sz[i.first] > sz[g[k][0].first] or g[k][0].first ==
                             p)
                                swap(i, g[k][0]);
                }
                if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
        }
        void build(int root = 0) {
                t = 0;
                build_hld(root);
                for (int i = 0; i < t; i++) seg[i+t] = v[i];
                for (int i = t-1; i; i--) seg[i] = min(seg[2*i], seg[2*i+1]);
        }
        int query_path(int a, int b) {
                if (a == b) return INF;
                if (pos[a] < pos[b]) swap(a, b);

                if (h[a] != h[b]) return min(men[a], query_path(pai[h[a]], b));
                int ans = INF, x = pos[b]+1+t, y = pos[a]+t;
                for (; x <= y; ++x/=2, --y/=2) ans = min({ans, seg[x], seg[y]});
                return ans;
        }
};
```

26.3 hldVertice

```
// HLD - vertice
//
// SegTree de soma
// query / update de soma dos vertices
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

// namespace seg { ... }

namespace hld {
        vector<int> g[MAX];
        int pos[MAX], sz[MAX];
        int peso[MAX], pai[MAX];
        int h[MAX], v[MAX], t;

        void build_hld(int k, int p = -1, int f = 1) {
                v[pos[k] = t++] = peso[k]; sz[k] = 1;
                for (auto& i : g[k]) if (i != p) {
                        pai[i] = k;
                        h[i] = (i == g[k][0] ? h[k] : i);
                        build_hld(i, k, f); sz[k] += sz[i];

                        if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i, g[k
                             ][0]);
                }
                if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
        }
        void build(int root = 0) {
                t = 0;
                build_hld(root);
                seg::build(t, v);
        }
        ll query_path(int a, int b) {
                if (pos[a] < pos[b]) swap(a, b);

                if (h[a] == h[b]) return seg::query(pos[b], pos[a]);
                return seg::query(pos[h[a]], pos[a]) + query_path(pai[h[a]], b);
        }
        void update_path(int a, int b, int x) {
                if (pos[a] < pos[b]) swap(a, b);

                if (h[a] == h[b]) return (void)seg::update(pos[b], pos[a], x);
                seg::update(pos[h[a]], pos[a], x); update_path(pai[h[a]], b, x);
        }
```

```
        ll query_subtree(int a) {
                return seg::query(pos[a], pos[a]+sz[a]-1);
        }
        void update_subtree(int a, int x) {
                seg::update(pos[a], pos[a]+sz[a]-1, x);
        }
        int lca(int a, int b) {
                if (pos[a] < pos[b]) swap(a, b);
                return h[a] == h[b] ? b : lca(pai[h[a]], b);
        }
}
```

26.4 lca

```
// LCA com RMQ
//
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// dist(a, b) retorna a distancia entre a e b
//
// Complexidades:
// build - O(n)
// lca - O(1)
// dist - O(1)

template<typename T> struct rmq {
        vector<T> v;
        int n; static const int b = 30;
        vector<int> mask, t;

        int op(int x, int y) { return v[x] < v[y] ? x : y; }
        int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
        rmq() {}
        rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
                for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
                        at = (at<<1)&((1<<b)-1);
                        while (at and op(i, i-msb(at&-at)) == i) at ^= at&-at;
                }
                for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-msb(mask[b*i+b-1]);
                for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0; i+(1<<j) <=
                     n/b; i++)
                        t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j-1)+i+(1<<(j-1))
                             ]);
        }
        int small(int r, int sz = b) { return r-msb(mask[r]&((1<<sz)-1)); }
        T query(int l, int r) {
                if (r-l+1 <= b) return small(r, r-l+1);
                int ans = op(small(l+b-1), small(r));
                int x = l/b+1, y = r/b-1;
                if (x <= y) {
                        int j = msb(y-x+1);
                        ans = op(ans, op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
                }
                return ans;
        }
};

namespace lca {
        vector<int> g[MAX];
        int v[2*MAX], pos[MAX], dep[2*MAX];
        int t;
        rmq<int> RMQ;

        void dfs(int i, int d = 0, int p = -1) {
                v[t] = i, pos[i] = t, dep[t++] = d;
                for (int j : g[i]) if (j != p) {
                        dfs(j, d+1, i);
                        v[t] = i, dep[t++] = d;
                }
        }
        void build(int n, int root) {
                t = 0;
                dfs(root);
                RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
        }
        int lca(int a, int b) {
                a = pos[a], b = pos[b];
                return v[RMQ.query(min(a, b), max(a, b))];
        }
        int dist(int a, int b) {
                return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a, b)]];
        }
}
```

26.5 lcaComBinaryLifting

```
// LCA com binary lifting
//
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// MAX2 = ceil(log(MAX))
//
// Complexidades:
// build - O(n log(n))
// lca - O(log(n))

vector<vector<int> > g(MAX);
int n, p;
int pai[MAX2][MAX];
int in[MAX], out[MAX];

void dfs(int k) {
        in[k] = p++;
        for (int i = 0; i < (int) g[k].size(); i++)
                if (in[g[k][i]] == -1) {
                        pai[0][g[k][i]] = k;
                        dfs(g[k][i]);
                }
        out[k] = p++;
}

void build(int raiz) {
        for (int i = 0; i < n; i++) pai[0][i] = i;
        p = 0, memset(in, -1, sizeof in);
        dfs(raiz);

        // pd dos pais
        for (int k = 1; k < MAX2; k++) for (int i = 0; i < n; i++)
                pai[k][i] = pai[k - 1][pai[k - 1][i]];
}

bool anc(int a, int b) { // se a eh ancestral de b
        return in[a] <= in[b] and out[a] >= out[b];
}

int lca(int a, int b) {
        if (anc(a, b)) return a;
        if (anc(b, a)) return b;

        // sobe a
        for (int k = MAX2 - 1; k >= 0; k--)
                if (!anc(pai[k][a], b)) a = pai[k][a];

        return pai[0][a];
}

// Alternativamente:
// 'binary lifting' gastando O(n) de memoria
// Da pra add folhas e fazer queries online
// 3 vezes o tempo do binary lifting normal
//
// build - O(n)
// kth, lca, dist - O(log(n))

int d[MAX], p[MAX], pp[MAX];

void set_root(int i) { p[i] = pp[i] = i, d[i] = 0; }

void add_leaf(int i, int u) {
        p[i] = u, d[i] = d[u]+1;
```

```
        pp[i] = 2*d[pp[u]] == d[pp[pp[u]]]+d[u] ? pp[pp[u]] : u;
}

int kth(int i, int k) {
        int dd = max(0, d[i]-k);
        while (d[i] > dd) i = d[pp[i]] >= dd ? pp[i] : p[i];
        return i;
}

int lca(int a, int b) {
        if (d[a] < d[b]) swap(a, b);
        while (d[a] > d[b]) a = d[pp[a]] >= d[b] ? pp[a] : p[a];
        while (a != b) {
                if (pp[a] != pp[b]) a = pp[a], b = pp[b];
                else a = p[a], b = p[b];
        }
        return a;
}

int dist(int a, int b) { return d[a]+d[b]-2*d[lca(a,b)]; }

vector<int> g[MAX];

void build(int i, int pai=-1) {
        if (pai == -1) set_root(i);
        for (int j : g[i]) if (j != pai) {
                add_leaf(j, i);
                build(j, i);
        }
}
```

26.6 lcaComHld

```
// LCA com HLD
//
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// Para buildar pasta chamar build(root)
// anc(a, b) responde se 'a' eh ancestral de 'b'
//
// Complexidades:
// build - O(n)
// lca - O(log(n))
// anc - O(1)

vector<int> g[MAX];
int pos[MAX], h[MAX], sz[MAX];
int pai[MAX], t;

void build(int k, int p = -1, int f = 1) {
        pos[k] = t++; sz[k] = 1;
        for (int& i : g[k]) if (i != p) {
                pai[i] = k;
                h[i] = (i == g[k][0] ? h[k] : i);
                build(i, k, f); sz[k] += sz[i];

                if (sz[i] > sz[g[k][0]] or g[k][0] == p) swap(i, g[k][0]);
        }
        if (p*f == -1) t = 0, h[k] = k, build(k, -1, 0);
}

int lca(int a, int b) {
        if (pos[a] < pos[b]) swap(a, b);
        return h[a] == h[b] ? b : lca(pai[h[a]], b);
}

bool anc(int a, int b) {
        return pos[a] <= pos[b] and pos[b] <= pos[a]+sz[a]-1;
}
```

27 ufmg forked/Grafos/LCT

27.1 lct

```
// Link-cut Tree
//
// Link-cut tree padrao
//
// Todas as operacoes sao O(log(n)) amortizado

namespace lct {
        struct node {
                int p, ch[2];
                node() { p = ch[0] = ch[1] = -1; }
        };

        node t[MAX];

        bool is_root(int x) {
                return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1]
                        != x);
        }
        void rotate(int x) {
                int p = t[x].p, pp = t[p].p;
                if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
                bool d = t[p].ch[0] == x;
                t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
                if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
                t[x].p = pp, t[p].p = x;
        }
        void splay(int x) {
                while (!is_root(x)) {
                        int p = t[x].p, pp = t[p].p;
                        if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
                                == x) ? x : p);
                        rotate(x);
                }
        }
        int access(int v) {
                int last = -1;
                for (int w = v; w+1; last = w, splay(v), w = t[v].p)
                        splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
                return last;
        }
        int find_root(int v) {
                access(v);
                while (t[v].ch[0]+1) v = t[v].ch[0];
                return splay(v), v;
        }
        void link(int v, int w) { // v deve ser raiz
                access(v);
                t[v].p = w;
        }
        void cut(int v) { // remove aresta de v pro pai
                access(v);
                t[v].ch[0] = t[t[v].ch[0]].p = -1;
        }
        int lca(int v, int w) {
                return access(v), access(w);
        }
}
```

27.2 lctAresta

```
// Link-cut Tree - aresta
//
// Valores nas arestas
// rootify(v) torna v a raiz de sua arvore
// query(v, w) retorna a soma do caminho v--w
// update(v, w, x) soma x nas arestas do caminho v--w
//
// Todas as operacoes sao O(log(n)) amortizado
```

```
namespace lct {
	struct node {
		int p, ch[2];
		ll val, sub;
		bool rev;
		int sz, ar;
		ll lazy;
		node() {}
		node(int v, int ar_) :
		p(-1), val(v), sub(v), rev(0), sz(ar_), ar(ar_), lazy(0) {
			ch[0] = ch[1] = -1;
		}
	};

	node t[2*MAX]; // MAXN + MAXQ
	map<pair<int, int>, int> aresta;
	int sz;

	void prop(int x) {
		if (t[x].lazy) {
			if (t[x].ar) t[x].val += t[x].lazy;
			t[x].sub += t[x].lazy*t[x].sz;
			if (t[x].ch[0]+1) t[t[x].ch[0]].lazy += t[x].lazy;
			if (t[x].ch[1]+1) t[t[x].ch[1]].lazy += t[x].lazy;
		}
		if (t[x].rev) {
			swap(t[x].ch[0], t[x].ch[1]);
			if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
			if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
		}
		t[x].lazy = 0, t[x].rev = 0;
	}
	void update(int x) {
		t[x].sz = t[x].ar, t[x].sub = t[x].val;
		for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
			prop(t[x].ch[i]);
			t[x].sz += t[t[x].ch[i]].sz;
			t[x].sub += t[t[x].ch[i]].sub;
		}
	}
	bool is_root(int x) {
		return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1]
		    != x);
	}
	void rotate(int x) {
		int p = t[x].p, pp = t[p].p;
		if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
		bool d = t[p].ch[0] == x;
		t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
		if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
		t[x].p = pp, t[p].p = x;
		update(p), update(x);
	}
	int splay(int x) {
		while (!is_root(x)) {
			int p = t[x].p, pp = t[p].p;
			if (!is_root(p)) prop(pp);
			prop(p), prop(x);
			if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
			    == x) ? x : p);
			rotate(x);
		}
		return prop(x), x;
	}
	int access(int v) {
		int last = -1;
		for (int w = v; w+1; update(last = w), splay(v), w = t[v].p)
			splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
		return last;
	}
	void make_tree(int v, int w=0, int ar=0) { t[v] = node(w, ar); }
	int find_root(int v) {
		access(v), prop(v);
		while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
		return splay(v);
	}
	bool conn(int v, int w) {
		access(v), access(w);
		return v == w ? true : t[v].p != -1;
	}
	void rootify(int v) {
		access(v);
		t[v].rev ^= 1;
	}
	ll query(int v, int w) {
		rootify(w), access(v);
		return t[v].sub;
	}
	void update(int v, int w, int x) {
		rootify(w), access(v);
		t[v].lazy += x;
	}
	void link_(int v, int w) {
		rootify(w);
		t[w].p = v;
	}
	void link(int v, int w, int x) { // v--w com peso x
		int id = MAX + sz++;
		aresta[make_pair(v, w)] = id;
		make_tree(id, x, 1);
		link_(v, id), link_(id, w);
	}
	void cut_(int v, int w) {
		rootify(w), access(v);
		t[v].ch[0] = t[t[v].ch[0]].p = -1;
	}
	void cut(int v, int w) {
		int id = aresta[make_pair(v, w)];
		cut_(v, id), cut_(id, w);
	}
	int lca(int v, int w) {
		access(v);
		return access(w);
	}
}
```

27.3 lctVertice

```
// Link-cut Tree - vertice
//
// Valores nos vertices
// make_tree(v, w) cria uma nova arvore com um
// vertice soh com valor 'w'
// rootify(v) torna v a raiz de sua arvore
// query(v, w) retorna a soma do caminho v--w
// update(v, w, x) soma x nos vertices do caminho v--w
//
// Todas as operacoes sao O(log(n)) amortizado

namespace lct {
	struct node {
		int p, ch[2];
		ll val, sub;
		bool rev;
		int sz;
		ll lazy;
		node() {}
		node(int v) : p(-1), val(v), sub(v), rev(0), sz(1), lazy(0) {
			ch[0] = ch[1] = -1;
		}
	};

	node t[MAX];

	void prop(int x) {
		if (t[x].lazy) {
			t[x].val += t[x].lazy, t[x].sub += t[x].lazy*t[x].sz;
			if (t[x].ch[0]+1) t[t[x].ch[0]].lazy += t[x].lazy;
			if (t[x].ch[1]+1) t[t[x].ch[1]].lazy += t[x].lazy;
		}
		if (t[x].rev) {
			swap(t[x].ch[0], t[x].ch[1]);
			if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
			if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
		}
```

```
        t[x].lazy = 0, t[x].rev = 0;
    }
    void update(int x) {
        t[x].sz = 1, t[x].sub = t[x].val;
        for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
            prop(t[x].ch[i]);
            t[x].sz += t[t[x].ch[i]].sz;
            t[x].sub += t[t[x].ch[i]].sub;
        }
    }
    bool is_root(int x) {
        return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1]
            != x);
    }
    void rotate(int x) {
        int p = t[x].p, pp = t[p].p;
        if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
        bool d = t[p].ch[0] == x;
        t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
        if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
        t[x].p = pp, t[p].p = x;
        update(p), update(x);
    }
    int splay(int x) {
        while (!is_root(x)) {
            int p = t[x].p, pp = t[p].p;
            if (!is_root(p)) prop(pp);
            prop(p), prop(x);
            if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
                == x) ? x : p);
            rotate(x);
        }
        return prop(x), x;
    }
    int access(int v) {
        int last = -1;
        for (int w = v; w+1; update(last = w), splay(v), w = t[v].p)
            splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
        return last;
    }
    void make_tree(int v, int w) { t[v] = node(w); }
    int find_root(int v) {
        access(v), prop(v);
        while (t[v].ch[0]+1) v = t[v].ch[0], prop(v);
        return splay(v);
    }
    bool connected(int v, int w) {
        access(v), access(w);
        return v == w ? true : t[v].p != -1;
    }
    void rootify(int v) {
        access(v);
        t[v].rev ^= 1;
    }
    ll query(int v, int w) {
        rootify(w), access(v);
        return t[v].sub;
    }
    void update(int v, int w, int x) {
        rootify(w), access(v);
        t[v].lazy += x;
    }
    void link(int v, int w) {
        rootify(w);
        t[w].p = v;
    }
    void cut(int v, int w) {
        rootify(w), access(v);
        t[v].ch[0] = t[t[v].ch[0]].p = -1;
    }
    int lca(int v, int w) {
        access(v);
        return access(w);
    }
}
```

28 ufmg forked/Matematica

28.1 2sat

```
// 2-SAT
//
// solve() retorna um par, o first fala se eh possivel
// atribuir, o second fala se cada variavel eh verdadeira
//
// O(|V|+|E|) = O(#variaveis + #restricoes)

struct sat {
    int n, tot;
    vector<vector<int>> g;
    vector<int> vis, comp, id, ans;
    stack<int> s;

    sat() {}
    sat(int n_) : n(n_), tot(n), g(2*n) {}

    int dfs(int i, int& t) {
        int lo = id[i] = t++;
        s.push(i), vis[i] = 2;
        for (int j : g[i]) {
            if (!vis[j]) lo = min(lo, dfs(j, t));
            else if (vis[j] == 2) lo = min(lo, id[j]);
        }
        if (lo == id[i]) while (1) {
            int u = s.top(); s.pop();
            vis[u] = 1, comp[u] = i;
            if ((u>>1) < n and ans[u>>1] == -1) ans[u>>1] = ~u&1;
            if (u == i) break;
        }
        return lo;
    }

    void add_impl(int x, int y) { // x -> y = !x ou y
        x = x >= 0 ? 2*x : -2*x-1;
        y = y >= 0 ? 2*y : -2*y-1;
        g[x].push_back(y);
        g[y^1].push_back(x^1);
    }
    void add_cl(int x, int y) { // x ou y
        add_impl(~x, y);
    }
    void add_xor(int x, int y) { // x xor y
        add_cl(x, y), add_cl(~x, ~y);
    }
    void add_eq(int x, int y) { // x = y
        add_xor(~x, y);
    }
    void add_true(int x) { // x = T
        add_impl(~x, x);
    }
    void at_most_one(vector<int> v) { // no max um verdadeiro
        g.resize(2*(tot+v.size()));
        for (int i = 0; i < v.size(); i++) {
            add_impl(tot+i, ~v[i]);
            if (i) {
                add_impl(tot+i, tot+i-1);
                add_impl(v[i], tot+i-1);
            }
        }
        tot += v.size();
    }

    pair<bool, vector<int>> solve() {
        ans = vector<int>(n, -1);
        int t = 0;
        vis = comp = id = vector<int>(2*tot, 0);
        for (int i = 0; i < 2*tot; i++) if (!vis[i]) dfs(i, t);
        for (int i = 0; i < tot; i++)
            if (comp[2*i] == comp[2*i+1]) return {false, {}};
        return {true, ans};
    }
};
```

28.2 berlekampMassey

```cpp
// Berlekamp-Massey
//
// guess_kth(s, k) chuta o k-esimo (0-based) termo
// de uma recorrencia linear que gera s
// Para uma rec. lin. de ordem x, se passar 2x termos
// vai gerar a certa
// Usar aritmetica modular
//
// Pro fast_evaluate, precisa de ntt e divmod (powerSeries.cpp)
//
// Complexidades: (n = |s|)
// evaluate: O(n^2 log k)
// fast_evaluate: O(n log n log k)
// berlekampMassey: O(n^2 + O(evaluate))

template<typename T> T evaluate(vector<T> c, vector<T> s, ll k) {
	int n = c.size();
	assert(c.size() <= s.size());

	auto mul = [&](const vector<T> &a, const vector<T> &b) {
		vector<T> ret(a.size() + b.size() - 1);
		for (int i = 0; i < a.size(); i++) for (int j = 0; j < b.size();
		     j++)
			ret[i+j] += a[i] * b[j];
		for (int i = ret.size()-1; i >= n; i--) for (int j = n-1; j >=
		     0; j--)
			ret[i-j-1] += ret[i] * c[j];
		ret.resize(min<int>(ret.size(), n));
		return ret;
	};

	vector<T> a = n == 1 ? vector<T>({c[0]}) : vector<T>({0, 1}), x = {1};
	while (k) {
		if (k&1) x = mul(x, a);
		a = mul(a, a), k >>= 1;
	}
	x.resize(n);

	T ret = 0;
	for (int i = 0; i < n; i++) ret += x[i] * s[i];
	return ret;
}

mint fast_evaluate(poly c, poly s, ll k) {
	if (k < s.size()) return s[k];
	int n = c.size();
	assert(c.size() <= s.size());

	auto f = poly(n + 1, 1);
	for (int i = 0; i < n; i++) f[i] = -c[n-i-1];

	poly a = n == 1 ? poly({c[0]}) : poly({0, 1}), x = {1};
	while (k) {
		if (k&1) x = divmod(convolution(x, a), f).second;
		a = divmod(convolution(a, a), f).second, k >>= 1;
	}

	mint ret = 0;
	for (int i = 0; i < n; i++) ret += x[i] * s[i];
	return ret;
}

template<typename T> vector<T> berlekamp_massey(vector<T> s) {
	int n = s.size(), l = 0, m = 1;
	vector<T> b(n), c(n);
	T ld = b[0] = c[0] = 1;
	for (int i = 0; i < n; i++, m++) {
		T d = s[i];
		for (int j = 1; j <= l; j++) d += c[j] * s[i-j];
		if (d == 0) continue;
		vector<T> temp = c;
		T coef = d / ld;
		for (int j = m; j < n; j++) c[j] -= coef * b[j-m];
		if (2 * l <= i) l = i + 1 - l, b = temp, ld = d, m = 0;
	}
	c.resize(l + 1);
	c.erase(c.begin());
	for (T& x : c) x = -x;
	return c;
}

template<typename T> T guess_kth(const vector<T>& s, ll k) {
	auto c = berlekamp_massey(s);
	return evaluate(c, s, k);
}
```

28.3 chinese

```cpp
// Teorema Chines do Resto
//
// Combina equacoes modulares lineares: x = a (mod m)
// O m final eh o lcm dos m's, e a resposta eh unica mod o lcm
// Os m nao precisam ser coprimos
// Se nao tiver solucao, o 'a' vai ser -1

template<typename T> tuple<T, T, T> ext_gcd(T a, T b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd(b%a, a);
    return {g, y - b/a*x, x};
}

template<typename T = ll> struct crt {
	T a, m;

	crt() : a(0), m(1) {}
	crt(T a_, T m_) : a(a_), m(m_) {}
	crt operator * (crt C) {
		auto [g, x, y] = ext_gcd(m, C.m);
		if ((a - C.a) % g) a = -1;
		if (a == -1 or C.a == -1) return crt(-1, 0);
		T lcm = m/g*C.m;
		T ans = a + (x*(C.a-a)/g % (C.m/g))*m;
		return crt((ans % lcm + lcm) % lcm, lcm);
	}
};
```

28.4 convolution

```cpp
// FFT
//
// Chamar convolution com vector<complex<double>> para FFT
// Precisa do mint para NTT
//
// O(n log(n))

// Para FFT
void get_roots(bool f, int n, vector<complex<double>>& roots) {
	const static double PI = acosl(-1);
	for (int i = 0; i < n/2; i++) {
		double alpha = i*((2*PI)/n);
		if (f) alpha = -alpha;
		roots[i] = {cos(alpha), sin(alpha)};
	}
}

// Para NTT
template<int p>
void get_roots(bool f, int n, vector<mod_int<p>>& roots) {
	mod_int<p> r;
	int ord;
	if (p == 998244353) {
		r = 102292;
		ord = (1 << 23);
	} else if (p == 754974721) {
		r = 739831874;
		ord = (1 << 24);
```

```
        } else if (p == 167772161) {
                r = 243;
                ord = (1 << 25);
        } else assert(false);

        if (f) r = r^(p - 1 -ord/n);
        else r = r^(ord/n);
        roots[0] = 1;
        for (int i = 1; i < n/2; i++) roots[i] = roots[i-1]*r;
}

template<typename T> void fft(vector<T>& a, bool f, int N, vector<int>& rev) {
        for (int i = 0; i < N; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);
        int l, r, m;
        vector<T> roots(N);
        for (int n = 2; n <= N; n *= 2) {
                get_roots(f, n, roots);
                for (int pos = 0; pos < N; pos += n) {
                        l = pos + 0, r = pos + n/2, m = 0;
                        while (m < n/2) {
                                auto t = roots[m] * a[r];
                                a[r] = a[l] - t;
                                a[l] = a[l] + t;
                                l++, r++, m++;
                        }
                }
        }
        if (f) {
                auto invN = T(1) / T(N);
                for (int i = 0; i < N; i++) a[i] = a[i] * invN;
        }
}

template<typename T> vector<T> convolution(vector<T>& a, vector<T>& b) {
        vector<T> l(a.begin(), a.end()), r(b.begin(), b.end());
        int N = l.size()+r.size()-1;
        int n = 1, log_n = 0;
        while (n <= N) n *= 2, log_n++;
        vector<int> rev(n);
        for (int i = 0; i < n; i++) {
                rev[i] = 0;
                for (int j = 0; j < log_n; j++) if (i>>j&1)
                        rev[i] |= 1 << (log_n-1-j);
        }
        assert(N <= n);
        l.resize(n);
        r.resize(n);
        fft(l, false, n, rev);
        fft(r, false, n, rev);
        for (int i = 0; i < n; i++) l[i] *= r[i];
        fft(l, true, n, rev);
        l.resize(N);
        return l;
}

// NTT
template<int p, typename T> vector<mod_int<p>> ntt(vector<T>& a, vector<T>& b) {
        vector<mod_int<p>> A(a.begin(), a.end()), B(b.begin(), b.end());
        return convolution(A, B);
}

// Convolucao de inteiro
//
// Precisa do CRT
//
// Tabela de valores:
// [0,1]       - <int, 1>
// [-1e5, 1e5] - <ll, 2>
// [-1e9, 1e9] - <__int128, 3>
template<typename T, int mods>
vector<T> int_convolution(vector<int>& a, vector<int>& b) {
        static const int M1 = 998244353, M2 = 754974721, M3 = 167772161;

        auto c1 = ntt<M1>(a, b);
        auto c2 = (mods >= 2 ? ntt<M2>(a, b) : vector<mod_int<M2>>());
        auto c3 = (mods >= 3 ? ntt<M3>(a, b) : vector<mod_int<M3>>());

        vector<T> ans;

        for (int i = 0; i < c1.size(); i++) {
                crt<T> at(c1[i].v, M1);
                if (mods >= 2) at = at * crt<T>(c2[i].v, M2);
                if (mods >= 3) at = at * crt<T>(c3[i].v, M3);
                ans.push_back(at.a);
                if (at.a > at.m/2) ans.back() -= at.m;
        }
        return ans;
}
```

28.5 coprimeBasis

```
// Coprime Basis
//
// Dado um conjunto de elementos A constroi uma base B
// de fatores coprimos tal que todo elemento A[i]
// pode ser fatorado como A[i] = \prod B[j]^p_ij
//
// Sendo n o numero de inserts, a complexidade esperada fica
// O(n*(n*loglog(MAX) + log(MAX)^2))
//
// No pior caso, podemos trocar n*loglog(MAX) por
//  8n, se MAX <= 1e6
// 10n, se MAX <= 1e9
// 16n, se MAX <= 1e18
// 26n, se MAX <= 1e36

template <typename T> struct coprime_basis {
        vector<T> basis;

        coprime_basis() {}
        coprime_basis(vector<T> v) { for (T i : v) insert(i); }

        void insert(T z) {
                int n = basis.size();
                basis.push_back(z);
                for (int i = n; i < basis.size(); i++) {
                        for (int j = (i != n) ? i+1 : 0; j < basis.size(); j++)
                                {
                                if (i == j) continue;
                                T &x = basis[i];
                                if (x == 1) {
                                        j = INF;
                                        continue;
                                }
                                T &y = basis[j];
                                T g = gcd(x, y);
                                if (g == 1) continue;
                                y /= g, x /= g;
                                basis.push_back(g);
                        }
                }
                basis.erase(remove(basis.begin(), basis.end(), 1), basis.end());
        }

        vector<int> factor(T x) {
                vector<int> fat(basis.size());
                for (int i = 0; i < basis.size(); i++) {
                        while (x % basis[i] == 0) x /= basis[i], fat[i]++;
                }
                return fat;
        }
};
```

28.6 crivo

```
// Crivo de Eratosthenes

// "O" crivo
//
// Encontra maior divisor primo
// Um numero eh primo sse divi[x] == x
```

```
// fact fatora um numero <= lim
// A fatoracao sai ordenada
//
// crivo - O(n log(log(n)))
// fact - O(log(n))

int divi[MAX];

void crivo(int lim) {
        for (int i = 1; i <= lim; i++) divi[i] = 1;

        for (int i = 2; i <= lim; i++) if (divi[i] == 1)
                for (int j = i; j <= lim; j += i) divi[j] = i;
}

#warning A funcao fact ira adicionar o 1 no vetor se voce tentar fatorar
     especificamente o numero 1
void fact(vector<int>& v, int n) {
        if (n != divi[n]) fact(v, n/divi[n]);
        v.push_back(divi[n]);
}

// Crivo linear
//
// Mesma coisa que o de cima, mas tambem
// calcula a lista de primos
//
// O(n)

int divi[MAX];
vector<int> primes;

void crivo(int lim) {
        divi[1] = 1;
        for (int i = 2; i <= lim; i++) {
                if (divi[i] == 0) divi[i] = i, primes.push_back(i);
                for (int j : primes) {
                        if (j > divi[i] or i*j > lim) break;
                        divi[i*j] = j;
                }
        }
}

// Crivo de divisores
//
// Encontra numero de divisores
// ou soma dos divisores
//
// O(n log(n))

int divi[MAX];

void crivo(int lim) {
        for (int i = 1; i <= lim; i++) divi[i] = 1;

        for (int i = 2; i <= lim; i++)
                for (int j = i; j <= lim; j += i) {
                        // para numero de divisores
                        divi[j]++;
                        // para soma dos divisores
                        divi[j] += i;
                }
}

// Crivo de totiente
//
// Encontra o valor da funcao
// totiente de Euler
//
// O(n log(log(n)))

int tot[MAX];

void crivo(int lim) {
        for (int i = 1; i <= lim; i++) {
                tot[i] += i;
                for (int j = 2*i; j <= lim; j += i)
                        tot[j] -= tot[i];
        }
}

// Crivo de funcao de mobius
//
// O(n log(log(n)))

char meb[MAX];

void crivo(int lim) {
        for (int i = 2; i <= lim; i++) meb[i] = 2;
        meb[1] = 1;
        for (int i = 2; i <= lim; i++) if (meb[i] == 2)
                for (int j = i; j <= lim; j += i) if (meb[j]) {
                        if (meb[j] == 2) meb[j] = 1;
                        meb[j] *= j/i%i ? -1 : 0;
                }
}

// Crivo linear de funcao multiplicativa
//
// Computa f(i) para todo 1 <= i <= n, sendo f
// uma funcao multiplicativa (se gcd(a,b) = 1,
// entao f(a*b) = f(a)*f(b))
// f_prime tem que computar f de um primo, e
// add_prime tem que computar f(p^(k+1)) dado f(p^k) e p
// Se quiser computar f(p^k) dado p e k, usar os comentarios
//
// O(n)

vector<int> primes;
int f[MAX], pot[MAX];
//int expo[MAX];

void sieve(int lim) {
        // Funcoes para soma dos divisores:
        auto f_prime = [](int p) { return p+1; };
        auto add_prime = [](int fpak, int p) { return fpak*p+1; };
        //auto f_pak = [](int p, int k) {};

        f[1] = 1;
        for (int i = 2; i <= lim; i++) {
                if (!pot[i]) {
                        primes.push_back(i);
                        f[i] = f_prime(i), pot[i] = i;
                        //expo[i] = 1;
                }
                for (int p : primes) {
                        if (i*p > lim) break;
                        if (i%p == 0) {
                                f[i*p] = f[i / pot[i]] * add_prime(f[pot[i]], p)
                                    ;
                                // se for descomentar, tirar a linha de cima
                                    tambem
                                //f[i*p] = f[i / pot[i]] * f_pak(p, expo[i]+1);
                                //expo[i*p] = expo[i]+1;
                                pot[i*p] = pot[i] * p;
                                break;
                        } else {
                                f[i*p] = f[i] * f[p];
                                pot[i*p] = p;
                                //expo[i*p] = 1;
                        }
                }
        }
}
```

28.7 cycleDetection

```
// Deteccao de ciclo - Tortoise and Hare
//
// Linear no tanto que tem que andar pra ciclar,
// O(1) de memoria
// Retorna um par com o tanto que tem que andar
// do f0 ate o inicio do ciclo e o tam do ciclo
```

```
pair<ll, ll> find_cycle() {
        ll tort = f(f0);
        ll hare = f(f(f0));
        ll t = 0;
        while (tort != hare) {
                tort = f(tort);
                hare = f(f(hare));
                t++;
        }
        ll st = 0;
        tort = f0;
        while (tort != hare) {
                tort = f(tort);
                hare = f(hare);
                st++;
        }

        ll len = 1;
        hare = f(tort);
        while (tort != hare) {
                hare = f(hare);
                len++;
        }
        return {st, len};
}
```

28.8 diofantina

```
// Equacao Diofantina Linear
//
// Encontra o numero de solucoes de a*x + b*y = c,
// em que x \in [lx, rx] e y \in [ly, ry]
// Usar o comentario para recuperar as solucoes
// (note que o b ao final eh b/gcd(a, b))
// Cuidado com overflow! Tem que caber o quadrado dos valores
//
// O(log(min(a, b)))

template<typename T> tuple<ll, T, T> ext_gcd(ll a, ll b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd<T>(b%a, a);
    return {g, y - b/a*x, x};
}

// numero de solucoes de a*[lx, rx] + b*[ly, ry] = c
template<typename T = ll> // usar __int128 se for ate 1e18
ll diophantine(ll a, ll b, ll c, ll lx, ll rx, ll ly, ll ry) {
        if (lx > rx or ly > ry) return 0;
        if (a == 0 and b == 0) return c ? 0 : (rx-lx+1)*(ry-ly+1);
        auto [g, x, y] = ext_gcd<T>(abs(a), abs(b));
        if (c % g != 0) return 0;
        if (a == 0) return (rx-lx+1)*(ly <= c/b and c/b <= ry);
        if (b == 0) return (ry-ly+1)*(lx <= c/a and c/a <= rx);
        x *= a/abs(a) * c/g, y *= b/abs(b) * c/g, a /= g, b /= g;

        auto shift = [&](T qt) { x += qt*b, y -= qt*a; };
        auto test = [&](T& k, ll mi, ll ma, ll coef, int t) {
                shift((mi - k)*t / coef);
                if (k < mi) shift(coef > 0 ? t : -t);
                if (k > ma) return pair<T, T>(rx+2, rx+1);
                T x1 = x;
                shift((ma - k)*t / coef);
                if (k > ma) shift(coef > 0 ? -t : t);
                return pair<T, T>(x1, x);
        };

        auto [l1, r1] = test(x, lx, rx, b, 1);
        auto [l2, r2] = test(y, ly, ry, a, -1);
        if (l2 > r2) swap(l2, r2);
        T l = max(l1, l2), r = min(r1, r2);
        if (l > r) return 0;
        ll k = (r-l) / abs(b) + 1;
        return k; // solucoes: x = l + [0, k)*|b|
}
```

28.9 divisionTrick

```
// Division Trick
//
// Gera o conjunto n/i, pra todo i, em O(sqrt(n))
// copiei do github do tfg50

for(int l = 1, r; l <= n; l = r + 1) {
        r = n / (n / l);
        // n / i has the same value for l <= i <= r
}
```

28.10 evalInterpol

```
// Avaliacao de Interpolacao
//
// Dado 'n' pontos (i, y[i]), i \in [0, n),
// avalia o polinomio de grau n-1 que passa
// por esses pontos em 'x'
// Tudo modular, precisa do mint
//
// O(n)

mint evaluate_interpolation(int x, vector<mint> y) {
        int n = y.size();

        vector<mint> sulf(n+1, 1), fat(n, 1), ifat(n);
        for (int i = n-1; i >= 0; i--) sulf[i] = sulf[i+1] * (x - i);
        for (int i = 1; i < n; i++) fat[i] = fat[i-1] * i;
        ifat[n-1] = 1/fat[n-1];
        for (int i = n-2; i >= 0; i--) ifat[i] = ifat[i+1] * (i + 1);

        mint pref = 1, ans = 0;
        for (int i = 0; i < n; pref *= (x - i++)) {
                mint num = pref * sulf[i+1];

                mint den = ifat[i] * ifat[n-1 - i];
                if ((n-1 - i)%2) den *= -1;

                ans += y[i] * num * den;
        }
        return ans;
}
```

28.11 fastPow

```
// Exponenciacao rapida
//
// (x^y mod m) em O(log(y))

ll pow(ll x, ll y, ll m) { // iterativo
        ll ret = 1;
        while (y) {
                if (y & 1) ret = (ret * x) % m;
                y >>= 1;
                x = (x * x) % m;
        }
        return ret;
}

ll pow(ll x, ll y, ll m) { // recursivo
        if (!y) return 1;
        ll ans = pow(x*x%m, y/2, m);
        return y%2 ? x*ans%m : ans;
}
```

28.12 fwht

```
// Fast Walsh Hadamard Transform
//
// FWHT<'|'>(f) eh SOS DP
// FWHT<'&'>(f) eh soma de superset DP
// Se chamar com ^, usar tamanho potencia de 2!!
//
// O(n log(n))

template<char op, class T> vector<T> FWHT(vector<T> f, bool inv = false) {
        int n = f.size();
        for (int k = 0; (n-1)>>k; k++) for (int i = 0; i < n; i++) if (i>>k&1) {
                int j = i^(1<<k);
                if (op == '^') f[j] += f[i], f[i] = f[j] - 2*f[i];
                if (op == '|') f[i] += (inv ? -1 : 1) * f[j];
                if (op == '&') f[j] += (inv ? -1 : 1) * f[i];
        }
        if (op == '^' and inv) for (auto& i : f) i /= n;
        return f;
}

// Generalizacao de FWHT de Xor
//
// Convolucao de soma mod B, usar tamanho potencia de B!!
// Precisa definir o tipo T e a raiz primitiva g
// satisfazendo g^b == g
//
// Se possivel, hardcodar a multiplicacao de matriz
// feita em cada iteracao faz ficar bem mais rapido
//
// O(n b log_b(n))

template<class T>
vector<T> FWHT(vector<T> f, int b, T g, bool inv = false) {
        int n = f.size();

        vector<T> w(b);
        w[1] = g;
        for (int i = 2; i < b; i++) w[i] = w[i - 1] * g;
        w[0] = w[b - 1] * g;

        if (inv) reverse(w.begin() + 1, w.end());

        for (int pot = 1; pot < n; pot *= b) {
                for (int i = 0; i < n; i++) if (!(i / pot % b)) {
                        vector<T> res(b);
                        for (int j = 0; j < b; j++) {
                                for (int k = 0; k < b; k++)
                                        res[j] = res[j] + w[j * k % b] * f[i + k
                                                * pot];
                                if (inv) res[j] = res[j] / b;
                        }
                        for (int j = 0; j < b; j++) f[i + j * pot] = res[j];
                }
        }

        return f;
}

// Exemplos da FWHT Generalizada:
//
// mod 7, resposta mod 998244353:
// T = mint, g = 14553391
//
// mod 3, resposta cabe em um long long:
// T = array<ll, 2>, g = {0, 1};
//
// using T = array<ll, 2>;
// T operator +(const T& a, const T& b) {
//       return T{a[0] + b[0], a[1] + b[1]};
// }
// T operator *(const T& a, const T& b) {
//       return T{a[0] * b[0] - a[1] * b[1],
//                       a[0] * b[1] + a[1] * b[0] - a[1] * b[1]};
// };
// T operator /(const T& a, const int& b) {
//       return T{a[0] / b, a[1] / b};
// }
```

28.13 gauss

```
// Gauss
//
// Resolve sistema linear
// Retornar um par com o numero de solucoes
// e alguma solucao, caso exista
//
// O(n^2 * m)

template<typename T>
pair<int, vector<T>> gauss(vector<vector<T>> a, vector<T> b) {
        const double eps = 1e-6;
        int n = a.size(), m = a[0].size();
        for (int i = 0; i < n; i++) a[i].push_back(b[i]);

        vector<int> where(m, -1);
        for (int col = 0, row = 0; col < m and row < n; col++) {
                int sel = row;
                for (int i=row; i<n; ++i)
                        if (abs(a[i][col]) > abs(a[sel][col])) sel = i;
                if (abs(a[sel][col]) < eps) continue;
                for (int i = col; i <= m; i++)
                        swap(a[sel][i], a[row][i]);
                where[col] = row;

                for (int i = 0; i < n; i++) if (i != row) {
                        T c = a[i][col] / a[row][col];
                        for (int j = col; j <= m; j++)
                                a[i][j] -= a[row][j] * c;
                }
                row++;
        }

        vector<T> ans(m, 0);
        for (int i = 0; i < m; i++) if (where[i] != -1)
                ans[i] = a[where[i]][m] / a[where[i]][i];
        for (int i = 0; i < n; i++) {
                T sum = 0;
                for (int j = 0; j < m; j++)
                        sum += ans[j] * a[i][j];
                if (abs(sum - a[i][m]) > eps)
                        return pair(0, vector<T>());
        }

        for (int i = 0; i < m; i++) if (where[i] == -1)
                return pair(INF, ans);
        return pair(1, ans);
}
```

28.14 gaussZ2

```
// Gauss - Z2
//
// D eh dimensao do espaco vetorial
// add(v) - adiciona o vetor v na base (retorna se ele jah pertencia ao span da
    base)
// coord(v) - retorna as coordenadas (c) de v na base atual (basis^T.c = v)
// recover(v) - retorna as coordenadas de v nos vetores na ordem em que foram
    inseridos
// coord(v).first e recover(v).first - se v pertence ao span
//
// Complexidade:
// add, coord, recover: O(D^2 / 64)

template<int D> struct gauss_z2 {
        bitset<D> basis[D], keep[D];
        int rk, in;
        vector<int> id;

        gauss_z2 () : rk(0), in(-1), id(D, -1) {};
```

```
        bool add(bitset<D> v) {
                in++;
                bitset<D> k;
                for (int i = D - 1; i >= 0; i--) if (v[i]) {
                        if (basis[i][i]) v ^= basis[i], k ^= keep[i];
                        else {
                                k[i] = true, id[i] = in, keep[i] = k;
                                basis[i] = v, rk++;
                                return true;
                        }
                }
                return false;
        }
        pair<bool, bitset<D>> coord(bitset<D> v) {
                bitset<D> c;
                for (int i = D - 1; i >= 0; i--) if (v[i]) {
                        if (basis[i][i]) v ^= basis[i], c[i] = true;
                        else return {false, bitset<D>()};
                }
                return {true, c};
        }
        pair<bool, vector<int>> recover(bitset<D> v) {
                auto [span, bc] = coord(v);
                if (not span) return {false, {}};
                bitset<D> aux;
                for (int i = D - 1; i >= 0; i--) if (bc[i]) aux ^= keep[i];
                vector<int> oc;
                for (int i = D - 1; i >= 0; i--) if (aux[i]) oc.push_back(id[i])
                     ;
                return {true, oc};
        }
};
```

28.15 gcdEstendido

```
// Euclides estendido
//
// Acha x e y tal que ax + by = mdc(a, b)  (nao eh unico)
// Assume a, b >= 0
//
// O(log(min(a, b)))

tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd(b%a, a);
    return {g, y - b/a*x, x};
}
```

28.16 gcdLcmConvolution

```
// Convolucao de GCD / LCM
//
// O(n log(n))

// multiple_transform(a)[i] = \sum_d a[d * i]
template<typename T> void multiple_transform(vector<T>& v, bool inv = false) {
        vector<int> I(v.size()-1);
        iota(I.begin(), I.end(), 1);
        if (inv) reverse(I.begin(), I.end());
        for (int i : I) for (int j = 2; i*j < v.size(); j++)
                v[i] += (inv ? -1 : 1) * v[i*j];
}

// gcd_convolution(a, b)[k] = \sum_{gcd(i, j) = k} a_i * b_j
template<typename T> vector<T> gcd_convolution(vector<T> a, vector<T> b) {
        multiple_transform(a), multiple_transform(b);
        for (int i = 0; i < a.size(); i++) a[i] *= b[i];
        multiple_transform(a, true);
        return a;
}

// divisor_transform(a)[i] = \sum_{d|i} a[i/d]
template<typename T> void divisor_transform(vector<T>& v, bool inv = false) {
        vector<int> I(v.size()-1);
        iota(I.begin(), I.end(), 1);
        if (!inv) reverse(I.begin(), I.end());
        for (int i : I) for (int j = 2; i*j < v.size(); j++)
                v[i*j] += (inv ? -1 : 1) * v[i];
}

// lcm_convolution(a, b)[k] = \sum_{lcm(i, j) = k} a_i * b_j
template<typename T> vector<T> lcm_convolution(vector<T> a, vector<T> b) {
        divisor_transform(a), divisor_transform(b);
        for (int i = 0; i < a.size(); i++) a[i] *= b[i];
        divisor_transform(a, true);
        return a;
}
```

28.17 integral

```
// Integracao Numerica
//
// Metodo de Simpson 3/8
// Integra f no intervalo [a, b], erro cresce proporcional a (b - a)^5

const int N = 3*100; // multiplo de 3
ld integrate(ld a, ld b, function<ld(ld)> f) {
    ld s = 0, h = (b - a)/N;
    for (int i = 1 ; i < N; i++) s += f(a + i*h)*(i%3 ? 3 : 2);
    return (f(a) + s + f(b))*3*h/8;
}
```

28.18 karatsuba

```
// Karatsuba
//
// Os pragmas podem ajudar
// Para n ~ 2e5, roda em < 1 s
//
// O(n^1.58)

//#pragma GCC optimize("Ofast")
//#pragma GCC target ("avx,avx2")
template<typename T> void kar(T* a, T* b, int n, T* r, T* tmp) {
        if (n <= 64) {
                for (int i = 0; i < n; i++) for (int j = 0; j < n; j++)
                        r[i+j] += a[i] * b[j];
                return;
        }
        int mid = n/2;
        T *atmp = tmp, *btmp = tmp+mid, *E = tmp+n;
        memset(E, 0, sizeof(E[0])*n);
        for (int i = 0; i < mid; i++) {
                atmp[i] = a[i] + a[i+mid];
                btmp[i] = b[i] + b[i+mid];
        }
        kar(atmp, btmp, mid, E, tmp+2*n);
        kar(a, b, mid, r, tmp+2*n);
        kar(a+mid, b+mid, mid, r+n, tmp+2*n);
        for (int i = 0; i < mid; i++) {
                T temp = r[i+mid];
                r[i+mid] += E[i] - r[i] - r[i+2*mid];
                r[i+2*mid] += E[i+mid] - temp - r[i+3*mid];
        }
}

template<typename T> vector<T> karatsuba(vector<T> a, vector<T> b) {
        int n = max(a.size(), b.size());
        while (n&(n-1)) n++;
        a.resize(n), b.resize(n);
        vector<T> ret(2*n), tmp(4*n);
        kar(&a[0], &b[0], n, &ret[0], &tmp[0]);
        return ret;
}
```

28.19 logDiscreto

```cpp
// Logaritmo Discreto
//
// Resolve logaritmo discreto com o algoritmo baby step giant step
// Encontra o menor x tal que a^x = b (mod m)
// Se nao tem, retorna -1
//
// O(sqrt(m) * log(sqrt(m))

int dlog(int b, int a, int m) {
	if (a == 0) return b ? -1 : 1; // caso nao definido

	a %= m, b %= m;
	int k = 1, shift = 0;
	while (1) {
		int g = gcd(a, m);
		if (g == 1) break;

		if (b == k) return shift;
		if (b % g) return -1;
		b /= g, m /= g, shift++;
		k = (ll) k * a / g % m;
	}

	int sq = sqrt(m)+1, giant = 1;
	for (int i = 0; i < sq; i++) giant = (ll) giant * a % m;

	vector<pair<int, int>> baby;
	for (int i = 0, cur = b; i <= sq; i++) {
		baby.emplace_back(cur, i);
		cur = (ll) cur * a % m;
	}
	sort(baby.begin(), baby.end());

	for (int j = 1, cur = k; j <= sq; j++) {
		cur = (ll) cur * giant % m;
		auto it = lower_bound(baby.begin(), baby.end(), pair(cur, INF));
		if (it != baby.begin() and (--it)->first == cur)
			return sq * j - it->second + shift;
	}

	return -1;
}
```

28.20 millerRabin

```cpp
// Miller-Rabin
//
// Testa se n eh primo, n <= 3 * 10^18
//
// O(log(n)), considerando multiplicacao
// e exponenciacao constantes

ll mul(ll a, ll b, ll m) {
	ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
	return ret < 0 ? ret+m : ret;
}

ll pow(ll x, ll y, ll m) {
	if (!y) return 1;
	ll ans = pow(mul(x, x, m), y/2, m);
	return y%2 ? mul(x, ans, m) : ans;
}

bool prime(ll n) {
	if (n < 2) return 0;
	if (n <= 3) return 1;
	if (n % 2 == 0) return 0;
	ll r = __builtin_ctzll(n - 1), d = n >> r;

	// com esses primos, o teste funciona garantido para n <= 2^64
	// funciona para n <= 3*10^24 com os primos ate 41
	for (int a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
		ll x = pow(a, d, n);
		if (x == 1 or x == n - 1 or a % n == 0) continue;

		for (int j = 0; j < r - 1; j++) {
			x = mul(x, x, n);
			if (x == n - 1) break;
		}
		if (x != n - 1) return 0;
	}
	return 1;
}
```

28.21 modInverse

```cpp
// Inverso Modular
//
// Computa o inverso de a modulo b
// Se b eh primo, basta fazer
// a^(b-2)

ll inv(ll a, ll b) {
	return a > 1 ? b - inv(b%a, a)*b/a : 1;
}

// computa o inverso modular de 1..MAX-1 modulo um primo
ll inv[MAX]:
inv[1] = 1;
for (int i = 2; i < MAX; i++) inv[i] = MOD - MOD/i*inv[MOD%i]%MOD;
```

28.22 mulmod

```cpp
// Produto de dois long long mod m
//
// O(1)

ll mul(ll a, ll b, ll m) { // a*b % m
	ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
	return ret < 0 ? ret+m : ret;
}
```

28.23 multipointEvaluation

```cpp
// Multipoint Evaluation And Interpolation
//
// Evaluation:
//	Avalia o polinomio f(x) nos pontos p[0], p[1], ..., p[n-1]
//
// Interpolation:
//	Retorna o polinomio f(x) de grau n que
//	satisfaz f(x) = y pra o conjunto de pontos x, y
//
// Precisa do ntt e
// - do divmod pro evaluate
// - da derivada pro interpolate
//
// O divmod e a derivada estao no arquivo powerSeries.cpp
//
// O(n log^2(n))

namespace multipoint {
	vector<poly> tree;
	void build(vector<mint>& p) {
		int n = p.size();
		tree.resize(2*n);
		for (int i = 0; i < n; i++) tree[n + i] = {-p[i], 1};
		for (int i = n - 1; i > 0; i--)
			tree[i] = convolution(tree[2*i], tree[2*i + 1]);
	}
```

```
        vector<mint> evaluate(poly& f, vector<mint>& p) {
                build(p);
                int n = p.size();
                vector<poly> ans(2 * n);
                ans[1] = divmod(f, tree[1]).second;
                for (int i = 2; i < 2 * n; i++)
                        ans[i] = divmod(ans[i/2], tree[i]).second;
                vector<mint> results(n);
                for (int i = 0; i < n; i++) results[i] = ans[n + i][0];
                return results;
        }
        poly prod(vector<mint>& p, int l, int r) {
                if (l == r) return {-p[l], 1};
                int m = (l + r) / 2;
                return convolution(prod(p, l, m), prod(p, m + 1, r));
        }
        poly interpolate(vector<mint>& x, vector<mint>& y) {
                int n = x.size();
                poly p = D(prod(x, 0, n - 1));
                auto d = evaluate(p, x);
                vector<poly> ans(2 * n);
                for (int i = 0; i < n; i++) ans[n + i] = {y[i] / d[i]};
                for (int i = n - 1; i > 0; i--) {
                        poly p1 = convolution(tree[2*i], ans[2*i + 1]);
                        poly p2 = convolution(tree[2*i + 1], ans[2*i]);
                        ans[i] = p1;
                        for (int j = 0; j < p1.size(); j++) ans[i][j] += p2[j];
                }
                return ans[1];
        }
}
```

28.24 ntt

```
// NTT
//
// Precisa do mint (primitivas de aritmetica modular)
//
// O(n log (n))

const int MOD = 998244353;
typedef mod_int<MOD> mint;

void ntt(vector<mint>& a, bool rev) {
        int n = a.size(); auto b = a;
        assert(!(n&(n-1)));
        mint g = 1;
        while ((g^(MOD / 2)) == 1) g += 1;
        if (rev) g = 1 / g;

        for (int step = n / 2; step; step /= 2) {
                mint w = g^(MOD / (n / step)), wn = 1;
                for (int i = 0; i < n/2; i += step) {
                        for (int j = 0; j < step; j++) {
                                auto u = a[2 * i + j], v = wn * a[2 * i + j +
                                    step];
                                b[i+j] = u + v; b[i + n/2 + j] = u - v;
                        }
                        wn = wn * w;
                }
                swap(a, b);
        }
        if (rev) {
                auto n1 = mint(1) / n;
                for (auto& x : a) x *= n1;
        }
}

vector<mint> convolution(const vector<mint>& a, const vector<mint>& b) {
        vector<mint> l(a.begin(), a.end()), r(b.begin(), b.end());
        int N = l.size()+r.size()-1, n = 1;
        while (n <= N) n *= 2;
        l.resize(n);
        r.resize(n);
        ntt(l, false);
        ntt(r, false);
        for (int i = 0; i < n; i++) l[i] *= r[i];
        ntt(l, true);
        l.resize(N);
        return l;
}
```

28.25 pollardrho

```
// Pollard's Rho Alg
//
// Usa o algoritmo de deteccao de ciclo de Floyd
// com uma otimizacao na qual o gcd eh acumulado
// A fatoracao nao sai necessariamente ordenada
// O algoritmo rho encontra um fator de n,
// e funciona muito bem quando n possui um fator pequeno
//
// Complexidades (considerando mul constante):
// rho - esperado O(n^(1/4)) no pior caso
// fact - esperado menos que O(n^(1/4) log(n)) no pior caso

ll mul(ll a, ll b, ll m) {
        ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
        return ret < 0 ? ret+m : ret;
}

ll pow(ll x, ll y, ll m) {
        if (!y) return 1;
        ll ans = pow(mul(x, x, m), y/2, m);
        return y%2 ? mul(x, ans, m) : ans;
}

bool prime(ll n) {
        if (n < 2) return 0;
        if (n <= 3) return 1;
        if (n % 2 == 0) return 0;

        ll r = __builtin_ctzll(n - 1), d = n >> r;
        for (int a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
                ll x = pow(a, d, n);
                if (x == 1 or x == n - 1 or a % n == 0) continue;

                for (int j = 0; j < r - 1; j++) {
                        x = mul(x, x, n);
                        if (x == n - 1) break;
                }
                if (x != n - 1) return 0;
        }
        return 1;
}

ll rho(ll n) {
        if (n == 1 or prime(n)) return n;
        auto f = [n](ll x) {return mul(x, x, n) + 1;};

        ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
        while (t % 40 != 0 or gcd(prd, n) == 1) {
                if (x==y) x = ++x0, y = f(x);
                q = mul(prd, abs(x-y), n);
                if (q != 0) prd = q;
                x = f(x), y = f(f(y)), t++;
        }
        return gcd(prd, n);
}

vector<ll> fact(ll n) {
        if (n == 1) return {};
        if (prime(n)) return {n};
        ll d = rho(n);
        vector<ll> l = fact(d), r = fact(n / d);
        l.insert(l.end(), r.begin(), r.end());
        return l;
}
```

28.26 powerSeries

```
// Operacoes em Polinomios e Series de Potencias
//
// Precisa do NTT
// O exp nao foi bem testado
//
// Fonte: github.com/celiopassos/competitive-programming/blob/master/algorithms/
    mathematics/formal_power_series.hpp
//
// D, I: O(n)
// inv, divmod, log e exp: O(n log(n))

using poly = vector<mint>;

const int MAGIC = 512;

poly D(poly p) {
        if (p.empty()) return p;
        for (int i = 0; i + 1 < p.size(); i++)
                p[i] = (i + 1) * p[i + 1];
        p.pop_back();
        return p;
}

poly I(poly p) {
        int n = p.size();
        p.push_back(0);
        for (int i = n - 1; i >= 0; i--)
                p[i + 1] = p[i] / (i + 1);
        p[0] = 0;
        return p;
}

poly inv(poly p) {
        assert(!p.empty() && p[0] == 1);
        poly q = {mint(1) / p[0]};
        int n = p.size(), k = 1;
        while (k < n) {
                k *= 2;
                q.resize(2 * k);
                ntt(q, false);
                poly p0(2 * k);
                copy_n(p.begin(), min(k, n), p0.begin());
                ntt(p0, false);
                for (int i = 0; i < 2 * k; i++)
                        q[i] *= 2 - p0[i] * q[i];
                ntt(q, true);
                q.resize(k);
        }
        q.resize(n);
        return q;
}

pair<poly, poly> divslow(const poly& a, const poly& b) {
        poly q, r = a;
        while (r.size() >= b.size()) {
                q.push_back(r.back() / b.back());
                if (q.back() != 0)
                        for (int i = 0; i < b.size(); i++)
                                r.end()[-i-1] -= q.back() * b.end()[-i-1];
                r.pop_back();
        }
        reverse(q.begin(), q.end());
        return {q, r};
}

// retorna (q, r) : a(x) = b(x) * q(x) + r(x)
pair<poly, poly> divmod(const poly& a, const poly& b) {
        if (a.size() < b.size()) return {{}, a};
        if (max(b.size(), a.size() - b.size()) < MAGIC) return divslow(a, b);
        poly ra = poly(a.rbegin(), a.rend());
        poly rb = poly(b.rbegin(), b.rend());
        int k = a.size() - b.size() + 1;
        rb.resize(k);
        poly irb = inv(move(rb)), q = convolution(ra, irb);
        q = poly(q.rend() - k, q.rend());
        poly r = convolution(move(q), b);
        for (int i = 0; i < r.size(); i++) r[i] = a[i] - r[i];
        while (r.size() > 1 && r.back() == 0) r.pop_back();
        return {q, r};
}

poly log(poly p) {
        assert(!p.empty() && p[0] == 1);
        int n = p.size();
        auto d = D(p), i = inv(p);
        auto r = convolution(d, i);
        r.resize(n - 1);
        return I(move(r));
}

poly exp(poly p) {
        assert(p.empty() || p[0] == 0);
        poly q = {1};
        int n = p.size(), k = 1;
        while (k < n) {
                k *= 2;
                q.resize(k);
                poly b = log(q);
                for (int i = 0; i < k; i++) b[i] *= -1;
                b[0] += 1;
                for (int i = 0; i < min(n, k); i++) b[i] += p[i];
                q = convolution(q, b);
                q.resize(k);
        }
        q.resize(n);
        return q;
}
```

28.27 probabilityBinomial

```
// Binomial Distribution
//
// binom(n, k, p) retorna a probabilidade de k sucessos
// numa binomial(n, p)

double logfact[MAX];

void calc() {
        logfact[0] = 0;
        for (int i = 1; i < MAX; i++) logfact[i] = logfact[i-1] + log(i);
}

double binom(int n, int k, double p) {
        return exp(logfact[n] - logfact[k] - logfact[n-k] + k * log(p) + (n-k) *
              log(1 - p));
}
```

28.28 simplex

```
// Simplex
//
// Maximiza c^T x s.t. Ax <= b, x >= 0
//
// O(2^n), porem executa em O(n^3) no caso medio

const double eps = 1e-7;

namespace Simplex {
        vector<vector<double>> T;
        int n, m;
        vector<int> X, Y;

        void pivot(int x, int y) {
                swap(X[y], Y[x-1]);
                for (int i = 0; i <= m; i++) if (i != y) T[x][i] /= T[x][y];
                T[x][y] = 1/T[x][y];
                for (int i = 0; i <= n; i++) if (i != x and abs(T[i][y]) > eps)
                     {
```

```
                for (int j = 0; j <= m; j++) if (j != y) T[i][j] -= T[i
                    ][y] * T[x][j];
                T[i][y] = -T[i][y] * T[x][y];
            }
        }

        // Retorna o par (valor maximo, vetor solucao)
        pair<double, vector<double>> simplex(
                vector<vector<double>> A, vector<double> b, vector<
                    double> c) {
            n = b.size(), m = c.size();
            T = vector(n + 1, vector<double>(m + 1));
            X = vector<int>(m);
            Y = vector<int>(n);
            for (int i = 0; i < m; i++) X[i] = i;
            for (int i = 0; i < n; i++) Y[i] = i+m;
            for (int i = 0; i < m; i++) T[0][i] = -c[i];
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) T[i+1][j] = A[i][j];
                T[i+1][m] = b[i];
            }
            while (true) {
                int x = -1, y = -1;
                double mn = -eps;
                for (int i = 1; i <= n; i++) if (T[i][m] < mn) mn = T[i
                    ][m], x = i;
                if (x < 0) break;
                for (int i = 0; i < m; i++) if (T[x][i] < -eps) { y = i;
                     break; }

                if (y < 0) return {-1e18, {}}; // sem solucao para  Ax
                    <= b
                pivot(x, y);
            }
            while (true) {
                int x = -1, y = -1;
                double mn = -eps;
                for (int i = 0; i < m; i++) if (T[0][i] < mn) mn = T[0][
                    i], y = i;
                if (y < 0) break;
                mn = 1e200;
                for (int i = 1; i <= n; i++) if (T[i][y] > eps and T[i][
                    m] / T[i][y] < mn)
                        mn = T[i][m] / T[i][y], x = i;

                if (x < 0) return {1e18, {}}; // c^T x eh ilimitado
                pivot(x, y);
            }
            vector<double> r(m);
            for(int i = 0; i < n; i++) if (Y[i] < m) r[Y[i]] = T[i+1][m];
            return {T[0][m], r};
        }
}
```

28.29 totiente

```
// Totiente
//
// O(sqrt(n))

int tot(int n) {
        int ret = n;

        for (int i = 2; i*i <= n; i++) if (n % i == 0) {
            while (n % i == 0) n /= i;
            ret -= ret / i;
        }
        if (n > 1) ret -= ret / n;

        return ret;
}
```

29 ufmg forked/Primitivas

29.1 bigint

```
// Big Integer
//
// Complexidades: (para n digitos)
// Soma, subtracao, comparacao - O(n)
// Multiplicacao - O(n log(n))
// Divisao, resto - O(n^2)

struct bint {
        static const int BASE = 1e9;
        vector<int> v;
        bool neg;

        bint() : neg(0) {}
        bint(int val) : bint() { *this = val; }
        bint(long long val) : bint() { *this = val; }

        void trim() {
            while (v.size() and v.back() == 0) v.pop_back();
            if (!v.size()) neg = 0;
        }

        // converter de/para string | cin/cout
        bint(const char* s) : bint() { from_string(string(s)); }
        bint(const string& s) : bint() { from_string(s); }
        void from_string(const string& s) {
            v.clear(), neg = 0;
            int ini = 0;
            while (ini < s.size() and (s[ini] == '-' or s[ini] == '+' or s[
                ini] == '0'))
                if (s[ini++] == '-') neg = 1;
            for (int i = s.size()-1; i >= ini; i -= 9) {
                int at = 0;
                for (int j = max(ini, i - 8); j <= i; j++) at = 10*at +
                    (s[j]-'0');
                v.push_back(at);
            }
            if (!v.size()) neg = 0;
        }
        string to_string() const {
            if (!v.size()) return "0";
            string ret;
            if (neg) ret += '-';
            for (int i = v.size()-1; i >= 0; i--) {
                string at = ::to_string(v[i]);
                int add = 9 - at.size();
                if (i+1 < v.size()) for (int j = 0; j < add; j++) ret +=
                    '0';
                ret += at;
            }
            return ret;
        }
        friend istream& operator>>(istream& in, bint& val) {
            string s; in >> s;
            val = s;
            return in;
        }
        friend ostream& operator<<(ostream& out, const bint& val) {
            string s = val.to_string();
            out << s;
            return out;
        }

        // operators
        friend bint abs(bint val) {
            val.neg = 0;
            return val;
        }
        friend bint operator-(bint val) {
            if (val != 0) val.neg ^= 1;
            return val;
        }
        bint& operator=(const bint& val) { v = val.v, neg = val.neg; return *
```

```
    this; }
bint& operator=(long long val) {
        v.clear(), neg = 0;
        if (val < 0) neg = 1, val *= -1;
        for (; val; val /= BASE) v.push_back(val % BASE);
        return *this;
}
int cmp(const bint& r) const { // menor: -1 | igual: 0 | maior: 1
        if (neg != r.neg) return neg ? -1 : 1;
        if (v.size() != r.v.size()) {
                int ret = v.size() < r.v.size() ? -1 : 1;
                return neg ? -ret : ret;
        }
        for (int i = int(v.size())-1; i >= 0; i--) {
                if (v[i] != r.v[i]) {
                        int ret = v[i] < r.v[i] ? -1 : 1;
                        return neg ? -ret : ret;
                }
        }
        return 0;
}
friend bool operator<(const bint& l, const bint& r) { return l.cmp(r) ==
     -1; }
friend bool operator>(const bint& l, const bint& r) { return l.cmp(r) ==
     1; }
friend bool operator<=(const bint& l, const bint& r) { return l.cmp(r)
    <= 0; }
friend bool operator>=(const bint& l, const bint& r) { return l.cmp(r)
    >= 0; }
friend bool operator==(const bint& l, const bint& r) { return l.cmp(r)
    == 0; }
friend bool operator!=(const bint& l, const bint& r) { return l.cmp(r)
    != 0; }

bint& operator +=(const bint& r) {
        if (!r.v.size()) return *this;
        if (neg != r.neg) return *this -= -r;
        for (int i = 0, c = 0; i < r.v.size() or c; i++) {
                if (i == v.size()) v.push_back(0);
                v[i] += c + (i < r.v.size() ? r.v[i] : 0);
                if ((c = v[i] >= BASE)) v[i] -= BASE;
        }
        return *this;
}
friend bint operator+(bint a, const bint& b) { return a += b; }
bint& operator -=(const bint& r) {
        if (!r.v.size()) return *this;
        if (neg != r.neg) return *this += -r;
        if ((!neg and *this < r) or (neg and r < *this)) {
                *this = r - *this;
                neg ^= 1;
                return *this;
        }
        for (int i = 0, c = 0; i < r.v.size() or c; i++) {
                v[i] -= c + (i < r.v.size() ? r.v[i] : 0);
                if ((c = v[i] < 0)) v[i] += BASE;
        }
        trim();
        return *this;
}
friend bint operator-(bint a, const bint& b) { return a -= b; }

// operators de * / %
bint& operator *=(int val) {
        if (val < 0) val *= -1, neg ^= 1;
        for (int i = 0, c = 0; i < v.size() or c; i++) {
                if (i == v.size()) v.push_back(0);
                long long at = (long long) v[i] * val + c;
                v[i] = at % BASE;
                c = at / BASE;
        }
        trim();
        return *this;
}
friend bint operator *(bint a, int b) { return a *= b; }
friend bint operator *(int a, bint b) { return b *= a; }
using cplx = complex<double>;
void fft(vector<cplx>& a, bool f, int N, vector<int>& rev) const {
        for (int i = 0; i < N; i++) if (i < rev[i]) swap(a[i], a[rev[i
            ]]);
        vector<cplx> roots(N);
        for (int n = 2; n <= N; n *= 2) {
                const static double PI = acos(-1);
                for (int i = 0; i < n/2; i++) {
                        double alpha = (2*PI*i)/n;
                        if (f) alpha = -alpha;
                        roots[i] = cplx(cos(alpha), sin(alpha));
                }
                for (int pos = 0; pos < N; pos += n)
                        for (int l = pos, r = pos+n/2, m = 0; m < n/2; l
                            ++, r++, m++) {
                                auto t = roots[m]*a[r];
                                a[r] = a[l] - t;
                                a[l] = a[l] + t;
                        }
        }
        if (!f) return;
        auto invN = cplx(1)/cplx(N);
        for (int i = 0; i < N; i++) a[i] *= invN;
}
vector<long long> convolution(const vector<int>& a, const vector<int>& b
    ) const {
        vector<cplx> l(a.begin(), a.end()), r(b.begin(), b.end());
        int ln = l.size(), rn = r.size(), N = ln+rn+1, n = 1, log_n = 0;
        while (n <= N) n <<= 1, log_n++;
        vector<int> rev(n);
        for (int i = 0; i < n; i++) {
                rev[i] = 0;
                for (int j = 0; j < log_n; j++) if (i>>j&1)
                        rev[i] |= 1 << (log_n-1-j);
        }
        l.resize(n), r.resize(n);
        fft(l, false, n, rev), fft(r, false, n, rev);
        for (int i = 0; i < n; i++) l[i] *= r[i];
        fft(l, true, n, rev);
        vector<long long> ret;
        for (auto& i : l) ret.push_back(round(i.real()));
        return ret;
}
vector<int> convert_base(const vector<int>& a, int from, int to) const {
        static vector<long long> pot(10, 1);
        if (pot[1] == 1) for (int i = 1; i < 10; i++) pot[i] = 10*pot[i
            -1];
        vector<int> ret;
        long long at = 0;
        int digits = 0;
        for (int i : a) {
                at += i * pot[digits];
                digits += from;
                while (digits >= to) {
                        ret.push_back(at % pot[to]);
                        at /= pot[to];
                        digits -= to;
                }
        }
        ret.push_back(at);
        while (ret.size() and ret.back() == 0) ret.pop_back();
        return ret;
}
bint operator*(const bint& r) const { // O(n log(n))
        bint ret;
        ret.neg = neg ^ r.neg;
        auto conv = convolution(convert_base(v, 9, 4), convert_base(r.v,
             9, 4));
        long long c = 0;
        for (auto i : conv) {
                long long at = i+c;
                ret.v.push_back(at % 10000);
                c = at / 10000;
        }
        for (; c; c /= 10000) ret.v.push_back(c%10000);
        ret.v = convert_base(ret.v, 4, 9);
        if (!ret.v.size()) ret.neg = 0;
        return ret;
}
bint& operator*=(const bint& r) { return *this = *this * r; };
```

```
        bint& operator/=(int val) {
                if (val < 0) neg ^= 1, val *= -1;
                for (int i = int(v.size())-1, c = 0; i >= 0; i--) {
                        long long at = v[i] + c * (long long) BASE;
                        v[i] = at / val;
                        c = at % val;
                }
                trim();
                return *this;
        }
        friend bint operator/(bint a, int b) { return a /= b; }
        int operator %=(int val) {
                if (val < 0) val *= -1;
                long long at = 0;
                for (int i = int(v.size())-1; i >= 0; i--)
                        at = (BASE * at + v[i]) % val;
                if (neg) at *= -1;
                return at;
        }
        friend int operator%(bint a, int b) { return a %= b; }
        friend pair<bint, bint> divmod(const bint& a_, const bint& b_) { // O(n
             ^2)
                if (a_ == 0) return {0, 0};
                int norm = BASE / (b_.v.back() + 1);
                bint a = abs(a_) * norm;
                bint b = abs(b_) * norm;
                bint q, r;
                for (int i = a.v.size() - 1; i >= 0; i--) {
                        r *= BASE, r += a.v[i];
                        long long upper = b.v.size() < r.v.size() ? r.v[b.v.size
                             ()] : 0;
                        int lower = b.v.size() - 1 < r.v.size() ? r.v[b.v.size()
                              - 1] : 0;
                        int d = (upper * BASE + lower) / b.v.back();
                        r -= b*d;
                        while (r < 0) r += b, d--; // roda O(1) vezes
                        q.v.push_back(d);
                }
                reverse(q.v.begin(), q.v.end());
                q.neg = a_.neg ^ b_.neg;
                r.neg = a_.neg;
                q.trim(), r.trim();
                return {q, r / norm};
        }
        bint operator/(const bint& val) { return divmod(*this, val).first; }
        bint& operator/=(const bint& val) { return *this = *this / val; }
        bint operator%(const bint& val) { return divmod(*this, val).second; }
        bint& operator%=(const bint& val) { return *this = *this % val; }
};
```

29.2 calendario

```
// Calendario
//
// Congruencia de Zeller
//
// Os dias da semana correspondem aos restos % 7
// Segunda=0, Terca=1, ..., Domingo=6

int get_id(int d, int m, int y) {
    if (m < 3) y--, m += 12;
    return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d -
         307;
}

tuple<int, int, int> date(int id) {
    int x = id + 1789995, n = 4 * x / 146097, i, j, d, m, y;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447, d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x, y = 100 * (n - 49) + i + x;
    return {d, m, y};
}
```

29.3 frac

```
// Fracao
//
// Funciona com o Big Int

template<typename T = int> struct frac {
        T num, den;
        template<class U, class V>
        frac(U num_ = 0, V den_ = 1) : num(num_), den(den_) {
                assert(den != 0);
                if (den < 0) num *= -1, den *= -1;
                T g = gcd(abs(num), den);
                num /= g, den /= g;
        }

        friend bool operator<(const frac& l, const frac& r) {
                return l.num * r.den < r.num * l.den;
        }
        friend frac operator+(const frac& l, const frac& r) {
                return {l.num*r.den + l.den*r.num, l.den*r.den};
        }
        friend frac operator-(const frac& l, const frac& r) {
                return {l.num*r.den - l.den*r.num, l.den*r.den};
        }
        friend frac operator*(const frac& l, const frac& r) {
                return {l.num*r.num, l.den*r.den};
        }
        friend frac operator/(const frac& l, const frac& r) {
                return {l.num*r.den, l.den*r.num};
        }
        friend ostream& operator<<(ostream& out, frac f) {
                out << f.num << '/' << f.den;
                return out;
        }
};
```

29.4 geometria

```
// Geometria

typedef double ld;
const ld DINF = 1e18;
const ld pi = acos(-1.0);
const ld eps = 1e-9;

#define sq(x) ((x)*(x))

bool eq(ld a, ld b) {
        return abs(a - b) <= eps;
}

struct pt { // ponto
        ld x, y;
        pt(ld x_ = 0, ld y_ = 0) : x(x_), y(y_) {}
        bool operator < (const pt p) const {
                if (!eq(x, p.x)) return x < p.x;
                if (!eq(y, p.y)) return y < p.y;
                return 0;
        }
        bool operator == (const pt p) const {
                return eq(x, p.x) and eq(y, p.y);
        }
        pt operator + (const pt p) const { return pt(x+p.x, y+p.y); }
        pt operator - (const pt p) const { return pt(x-p.x, y-p.y); }
        pt operator * (const ld c) const { return pt(x*c  , y*c  ); }
        pt operator / (const ld c) const { return pt(x/c  , y/c  ); }
        ld operator * (const pt p) const { return x*p.x + y*p.y; }
        ld operator ^ (const pt p) const { return x*p.y - y*p.x; }
        friend istream& operator >> (istream& in, pt& p) {
                return in >> p.x >> p.y;
        }
};
```

```
struct line { // reta
        pt p, q;
        line() {}
        line(pt p_, pt q_) : p(p_), q(q_) {}
        friend istream& operator >> (istream& in, line& r) {
                return in >> r.p >> r.q;
        }
};

// PONTO & VETOR

ld dist(pt p, pt q) { // distancia
        return hypot(p.y - q.y, p.x - q.x);
}

ld dist2(pt p, pt q) { // quadrado da distancia
        return sq(p.x - q.x) + sq(p.y - q.y);
}

ld norm(pt v) { // norma do vetor
        return dist(pt(0, 0), v);
}

ld angle(pt v) { // angulo do vetor com o eixo x
        ld ang = atan2(v.y, v.x);
        if (ang < 0) ang += 2*pi;
        return ang;
}

ld sarea(pt p, pt q, pt r) { // area com sinal
        return ((q-p)^(r-q))/2;
}

bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
        return eq(sarea(p, q, r), 0);
}

bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
        return sarea(p, q, r) > eps;
}

pt rotate(pt p, ld th) { // rotaciona o ponto th radianos
        return pt(p.x * cos(th) - p.y * sin(th),
                  p.x * sin(th) + p.y * cos(th));
}

pt rotate90(pt p) { // rotaciona 90 graus
        return pt(-p.y, p.x);
}

// RETA

bool isvert(line r) { // se r eh vertical
        return eq(r.p.x, r.q.x);
}

bool isinseg(pt p, line r) { // se p pertence ao seg de r
        pt a = r.p - p, b = r.q - p;
        return eq((a ^ b), 0) and (a * b) < eps;
}

ld get_t(pt v, line r) { // retorna t tal que t*v pertence a reta r
        return (r.p^r.q) / ((r.p-r.q)^v);
}

pt proj(pt p, line r) { // projecao do ponto p na reta r
        if (r.p == r.q) return r.p;
        r.q = r.q - r.p; p = p - r.p;
        pt proj = r.q * ((p*r.q) / (r.q*r.q));
        return proj + r.p;
}

pt inter(line r, line s) { // r inter s
        if (eq((r.p - r.q) ^ (s.p - s.q), 0)) return pt(DINF, DINF);
        r.q = r.q - r.p, s.p = s.p - r.p, s.q = s.q - r.p;
        return r.q * get_t(r.q, s) + r.p;
}

bool interseg(line r, line s) { // se o seg de r intersecta o seg de s
        if (isinseg(r.p, s) or isinseg(r.q, s)
                or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

        return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
                        ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
}

ld disttoline(pt p, line r) { // distancia do ponto a reta
        return 2 * abs(sarea(p, r.p, r.q)) / dist(r.p, r.q);
}

ld disttoseg(pt p, line r) { // distancia do ponto ao seg
        if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p, p);
        if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q, p);
        return disttoline(p, r);
}

ld distseg(line a, line b) { // distancia entre seg
        if (interseg(a, b)) return 0;

        ld ret = DINF;
        ret = min(ret, disttoseg(a.p, b));
        ret = min(ret, disttoseg(a.q, b));
        ret = min(ret, disttoseg(b.p, a));
        ret = min(ret, disttoseg(b.q, a));

        return ret;
}

// POLIGONO

// corta poligono com a reta r deixando os pontos p tal que
// ccw(r.p, r.q, p)
vector<pt> cut_polygon(vector<pt> v, line r) { // O(n)
        vector<pt> ret;
        for (int j = 0; j < v.size(); j++) {
                if (ccw(r.p, r.q, v[j])) ret.push_back(v[j]);
                if (v.size() == 1) continue;
                line s(v[j], v[(j+1)%v.size()]);
                pt p = inter(r, s);
                if (isinseg(p, s)) ret.push_back(p);
        }
        ret.erase(unique(ret.begin(), ret.end()), ret.end());
        if (ret.size() > 1 and ret.back() == ret[0]) ret.pop_back();
        return ret;
}

// distancia entre os retangulos a e b (lados paralelos aos eixos)
// assume que ta representado (inferior esquerdo, superior direito)
ld dist_rect(pair<pt, pt> a, pair<pt, pt> b) {
        ld hor = 0, vert = 0;
        if (a.second.x < b.first.x) hor = b.first.x - a.second.x;
        else if (b.second.x < a.first.x) hor = a.first.x - b.second.x;
        if (a.second.y < b.first.y) vert = b.first.y - a.second.y;
        else if (b.second.y < a.first.y) vert = a.first.y - b.second.y;
        return dist(pt(0, 0), pt(hor, vert));
}

ld polarea(vector<pt> v) { // area do poligono
        ld ret = 0;
        for (int i = 0; i < v.size(); i++)
                ret += sarea(pt(0, 0), v[i], v[(i + 1) % v.size()]);
        return abs(ret);
}

// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
int inpol(vector<pt>& v, pt p) { // O(n)
        int qt = 0;
        for (int i = 0; i < v.size(); i++) {
                if (p == v[i]) return 2;
                int j = (i+1)%v.size();
                if (eq(p.y, v[i].y) and eq(p.y, v[j].y)) {
                        if ((v[i]-p)*(v[j]-p) < eps) return 2;
                        continue;
                }
                bool baixo = v[i].y+eps < p.y;
                if (baixo == (v[j].y+eps < p.y)) continue;
```

```
                auto t = (p-v[i])^(v[j]-v[i]);
                if (eq(t, 0)) return 2;
                if (baixo == (t > eps)) qt += baixo ? 1 : -1;
        }
        return qt != 0;
}

bool interpol(vector<pt> v1, vector<pt> v2) { // se dois poligonos se
    intersectam - O(n*m)
        int n = v1.size(), m = v2.size();
        for (int i = 0; i < n; i++) if (inpol(v2, v1[i])) return 1;
        for (int i = 0; i < n; i++) if (inpol(v1, v2[i])) return 1;
        for (int i = 0; i < n; i++) for (int j = 0; j < m; j++)
                if (interseg(line(v1[i], v1[(i+1)%n]), line(v2[j], v2[(j+1)%m]))
                    ) return 1;
        return 0;
}

ld distpol(vector<pt> v1, vector<pt> v2) { // distancia entre poligonos
        if (interpol(v1, v2)) return 0;

        ld ret = DINF;

        for (int i = 0; i < v1.size(); i++) for (int j = 0; j < v2.size(); j++)
                ret = min(ret, distseg(line(v1[i], v1[(i + 1) % v1.size()]),
                                       line(v2[j], v2[(j + 1) % v2.size()])));
        return ret;
}

vector<pt> convex_hull(vector<pt> v) { // convex hull - O(n log(n))
        sort(v.begin(), v.end());
        v.erase(unique(v.begin(), v.end()), v.end());
        if (v.size() <= 1) return v;
        vector<pt> l, u;
        for (int i = 0; i < v.size(); i++) {
                while (l.size() > 1 and !ccw(l.end()[-2], l.end()[-1], v[i]))
                        l.pop_back();
                l.push_back(v[i]);
        }
        for (int i = v.size() - 1; i >= 0; i--) {
                while (u.size() > 1 and !ccw(u.end()[-2], u.end()[-1], v[i]))
                        u.pop_back();
                u.push_back(v[i]);
        }
        l.pop_back(); u.pop_back();
        for (pt i : u) l.push_back(i);
        return l;
}

struct convex_pol {
        vector<pt> pol;

        // nao pode ter ponto colinear no convex hull
        convex_pol() {}
        convex_pol(vector<pt> v) : pol(convex_hull(v)) {}

        // se o ponto ta dentro do hull - O(log(n))
        bool is_inside(pt p) {
                if (pol.size() == 0) return false;
                if (pol.size() == 1) return p == pol[0];
                int l = 1, r = pol.size();
                while (l < r) {
                        int m = (l+r)/2;
                        if (ccw(p, pol[0], pol[m])) l = m+1;
                        else r = m;
                }
                if (l == 1) return isinseg(p, line(pol[0], pol[1]));
                if (l == pol.size()) return false;
                return !ccw(p, pol[l], pol[l-1]);
        }
        // ponto extremo em relacao a cmp(p, q) = p mais extremo q
        // (copiado de https://github.com/gustavoM32/caderno-zika)
        int extreme(const function<bool(pt, pt)>& cmp) {
                int n = pol.size();
                auto extr = [&](int i, bool& cur_dir) {
                        cur_dir = cmp(pol[(i+1)%n], pol[i]);
                        return !cur_dir and !cmp(pol[(i+n-1)%n], pol[i]);
                };
                bool last_dir, cur_dir;
                if (extr(0, last_dir)) return 0;
                int l = 0, r = n;
                while (l+1 < r) {
                        int m = (l+r)/2;
                        if (extr(m, cur_dir)) return m;
                        bool rel_dir = cmp(pol[m], pol[l]);
                        if ((!last_dir and cur_dir) or
                                        (last_dir == cur_dir and rel_dir ==
                                             cur_dir)) {
                                l = m;
                                last_dir = cur_dir;
                        } else r = m;
                }
                return l;
        }
        int max_dot(pt v) {
                return extreme([&](pt p, pt q) { return p*v > q*v; });
        }
        pair<int, int> tangents(pt p) {
                auto L = [&](pt q, pt r) { return ccw(p, r, q); };
                auto R = [&](pt q, pt r) { return ccw(p, q, r); };
                return {extreme(L), extreme(R)};
        }
};

// CIRCUNFERENCIA

pt getcenter(pt a, pt b, pt c) { // centro da circunf dado 3 pontos
        b = (a + b) / 2;
        c = (a + c) / 2;
        return inter(line(b, b + rotate90(a - b)),
                     line(c, c + rotate90(a - c)));
}

vector<pt> circ_line_inter(pt a, pt b, pt c, ld r) { // intersecao da circunf (c
    , r) e reta ab
        vector<pt> ret;
        b = b-a, a = a-c;
        ld A = b*b;
        ld B = a*b;
        ld C = a*a - r*r;
        ld D = B*B - A*C;
        if (D < -eps) return ret;
        ret.push_back(c+a+b*(-B+sqrt(D+eps))/A);
        if (D > eps) ret.push_back(c+a+b*(-B-sqrt(D))/A);
        return ret;
}

vector<pt> circ_inter(pt a, pt b, ld r, ld R) { // intersecao da circunf (a, r)
    e (b, R)
        vector<pt> ret;
        ld d = dist(a, b);
        if (d > r+R or d+min(r, R) < max(r, R)) return ret;
        ld x = (d*d-R*R+r*r)/(2*d);
        ld y = sqrt(r*r-x*x);
        pt v = (b-a)/d;
        ret.push_back(a+v*x + rotate90(v)*y);
        if (y > 0) ret.push_back(a+v*x - rotate90(v)*y);
        return ret;
}

bool operator <(const line& a, const line& b) { // comparador pra reta
        // assume que as retas tem p < q
        pt v1 = a.q - a.p, v2 = b.q - b.p;
        if (!eq(angle(v1), angle(v2))) return angle(v1) < angle(v2);
        return ccw(a.p, a.q, b.p); // mesmo angulo
}
bool operator ==(const line& a, const line& b) {
        return !(a < b) and !(b < a);
}

// comparador pro set pra fazer sweep line com segmentos
struct cmp_sweepline {
        bool operator () (const line& a, const line& b) const {
                // assume que os segmentos tem p < q
                if (a.p == b.p) return ccw(a.p, a.q, b.q);
                if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps < b.p.x
                    ))
                        return ccw(a.p, a.q, b.p);
```

```
                return ccw(a.p, b.q, b.p);
        }
};

// comparador pro set pra fazer sweep angle com segmentos
pt dir;
struct cmp_sweepangle {
        bool operator () (const line& a, const line& b) const {
                return get_t(dir, a) + eps < get_t(dir, b);
        }
};
```

29.5 geometria3d

```
// Geometria 3D

typedef double ld;
const ld DINF = 1e18;
const ld eps = 1e-9;

#define sq(x) ((x)*(x))

bool eq(ld a, ld b) {
        return abs(a - b) <= eps;
}

struct pt { // ponto
        ld x, y, z;
        pt(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_), z(z_) {}
        bool operator < (const pt p) const {
                if (!eq(x, p.x)) return x < p.x;
                if (!eq(y, p.y)) return y < p.y;
                if (!eq(z, p.z)) return z < p.z;
                return 0;
        }
        bool operator == (const pt p) const {
                return eq(x, p.x) and eq(y, p.y) and eq(z, p.z);
        }
        pt operator + (const pt p) const { return pt(x+p.x, y+p.y, z+p.z); }
        pt operator - (const pt p) const { return pt(x-p.x, y-p.y, z-p.z); }
        pt operator * (const ld c) const { return pt(x*c  , y*c  , z*c  ); }
        pt operator / (const ld c) const { return pt(x/c  , y/c  , z/c  ); }
        ld operator * (const pt p) const { return x*p.x + y*p.y + z*p.z; }
        pt operator ^ (const pt p) const { return pt(y*p.z - z*p.y, z*p.x - x*p.
            z, x*p.y - y*p.x); }
        friend istream& operator >> (istream& in, pt& p) {
                return in >> p.x >> p.y >> p.z;
        }
};

struct line { // reta
        pt p, q;
        line() {}
        line(pt p_, pt q_) : p(p_), q(q_) {}
        friend istream& operator >> (istream& in, line& r) {
                return in >> r.p >> r.q;
        }
};

struct plane { // plano
        array<pt, 3> p;  // pontos que definem o plano
        array<ld, 4> eq; // equacao do plano
        plane() {}
        plane(pt p_, pt q_, pt r_) : p({p_, q_, r_}) { build(); }

        friend istream& operator >> (istream& in, plane& P) {
                return in >> P.p[0] >> P.p[1] >> P.p[2];
                P.build();
        }
        void build() {
                pt dir = (p[1] - p[0]) ^ (p[2] - p[0]);
                eq = {dir.x, dir.y, dir.z, dir*p[0]*(-1)};
        }
};

// converte de coordenadas polares para cartesianas
// (angulos devem estar em radianos)
// phi eh o angulo com o eixo z (cima) theta eh o angulo de rotacao ao redor de
    z
pt convert(ld rho, ld th, ld phi) {
        return pt(sin(phi) * cos(th), sin(phi) * sin(th), cos(phi)) * rho;
}

// projecao do ponto p na reta r
pt proj(pt p, line r) {
        if (r.p == r.q) return r.p;
        r.q = r.q - r.p; p = p - r.p;
        pt proj = r.q * ((p*r.q) / (r.q*r.q));
        return proj + r.p;
}

// projecao do ponto p no plano P
pt proj(pt p, plane P) {
        p = p - P.p[0], P.p[1] = P.p[1] - P.p[0], P.p[2] = P.p[2] - P.p[0];
        pt norm = P.p[1] ^ P.p[2];
        pt proj = p - (norm * (norm * p) / (norm*norm));
        return proj + P.p[0];
}

// distancia
ld dist(pt a, pt b) {
        return sqrt(sq(a.x-b.x) + sq(a.y-b.y) + sq(a.z-b.z));
}

// distancia ponto reta
ld distline(pt p, line r) {
        return dist(p, proj(p, r));
}

// distancia de ponto para segmento
ld distseg(pt p, line r) {
        if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p, p);
        if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q, p);
        return distline(p, r);
}

// distancia de ponto a plano com sinal
ld sdist(pt p, plane P) {
        return P.eq[0]*p.x + P.eq[1]*p.y + P.eq[2]*p.z +  P.eq[3];
}

// distancia de ponto a plano
ld distplane(pt p, plane P) {
        return abs(sdist(p, P));
}

// se ponto pertence a reta
bool isinseg(pt p, line r) {
        return eq(distseg(p, r), 0);
}

// se ponto pertence ao triangulo definido por P.p
bool isinpol(pt p, vector<pt> v) {
        assert(v.size() >= 3);
        pt norm = (v[1]-v[0]) ^ (v[2]-v[1]);
        bool inside = true;
        int sign = -1;
        for (int i = 0; i < v.size(); i++) {
                line r(v[(i+1)%3], v[i]);
                if (isinseg(p, r)) return true;

                pt ar = v[(i+1)%3] - v[i];
                if (sign == -1) sign = ((ar^(p-v[i]))*norm > 0);
                else if (((ar^(p-v[i]))*norm > 0) != sign) inside = false;
        }
        return inside;
}

// distancia de ponto ate poligono
ld distpol(pt p, vector<pt> v) {
        pt p2 = proj(p, plane(v[0], v[1], v[2]));
        if (isinpol(p2, v)) return dist(p, p2);
        ld ret = DINF;
        for (int i = 0; i < v.size(); i++) {
```

```
                int j = (i+1)%v.size();
                ret = min(ret, distseg(p, line(v[i], v[j])));
        }
        return ret;
}

// intersecao de plano e segmento
// BOTH = o segmento esta no plano
// ONE = um dos pontos do segmento esta no plano
// PARAL = segmento paralelo ao plano
// CONCOR = segmento concorrente ao plano
enum RETCODE {BOTH, ONE, PARAL, CONCOR};
pair<RETCODE, pt> intersect(plane P, line r) {
    ld d1 = sdist(r.p, P);
    ld d2 = sdist(r.q, P);
    if (eq(d1, 0) and eq(d2, 0))
                return pair(BOTH, r.p);
    if (eq(d1, 0))
                return pair(ONE, r.p);
    if (eq(d2, 0))
                return pair(ONE, r.q);
    if ((d1 > 0 and d2 > 0) or (d1 < 0 and d2 < 0)) {
        if (eq(d1-d2, 0)) return pair(PARAL, pt());
        return pair(CONCOR, pt());
    }
    ld frac = d1 / (d1 - d2);
    pt res = r.p + ((r.q - r.p) * frac);
    return pair(ONE, res);
}

// rotaciona p ao redor do eixo u por um angulo a
pt rotate(pt p, pt u, ld a) {
        u = u / dist(u, pt());
        return u * (u * p) + (u ^ p ^ u) * cos(a) + (u ^ p) * sin(a);
}
```

29.6 geometriaInt

```
// Geometria - inteiro

#define sq(x) ((x)*(ll)(x))

struct pt { // ponto
        int x, y;
        pt(int x_ = 0, int y_ = 0) : x(x_), y(y_) {}
        bool operator < (const pt p) const {
                if (x != p.x) return x < p.x;
                return y < p.y;
        }
        bool operator == (const pt p) const {
                return x == p.x and y == p.y;
        }
        pt operator + (const pt p) const { return pt(x+p.x, y+p.y); }
        pt operator - (const pt p) const { return pt(x-p.x, y-p.y); }
        pt operator * (const int c) const { return pt(x*c, y*c); }
        ll operator * (const pt p) const { return x*(ll)p.x + y*(ll)p.y; }
        ll operator ^ (const pt p) const { return x*(ll)p.y - y*(ll)p.x; }
        friend istream& operator >> (istream& in, pt& p) {
                return in >> p.x >> p.y;
        }
};

struct line { // reta
        pt p, q;
        line() {}
        line(pt p_, pt q_) : p(p_), q(q_) {}
        friend istream& operator >> (istream& in, line& r) {
                return in >> r.p >> r.q;
        }
};

// PONTO & VETOR

ll dist2(pt p, pt q) { // quadrado da distancia
        return sq(p.x - q.x) + sq(p.y - q.y);
}

ll sarea2(pt p, pt q, pt r) { // 2 * area com sinal
        return (q-p)^(r-q);
}

bool col(pt p, pt q, pt r) { // se p, q e r sao colin.
        return sarea2(p, q, r) == 0;
}

bool ccw(pt p, pt q, pt r) { // se p, q, r sao ccw
        return sarea2(p, q, r) > 0;
}

int quad(pt p) { // quadrante de um ponto
        return (p.x<0)^3*(p.y<0);
}

bool compare_angle(pt p, pt q) { // retorna se ang(p) < ang(q)
        if (quad(p) != quad(q)) return quad(p) < quad(q);
        return ccw(q, pt(0, 0), p);
}

pt rotate90(pt p) { // rotaciona 90 graus
        return pt(-p.y, p.x);
}

// RETA

bool isinseg(pt p, line r) { // se p pertence ao seg de r
        pt a = r.p - p, b = r.q - p;
        return (a ^ b) == 0 and (a * b) <= 0;
}

bool interseg(line r, line s) { // se o seg de r intersecta o seg de s
        if (isinseg(r.p, s) or isinseg(r.q, s)
                or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

        return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
                        ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
}

int segpoints(line r) { // numero de pontos inteiros no segmento
        return 1 + __gcd(abs(r.p.x - r.q.x), abs(r.p.y - r.q.y));
}

double get_t(pt v, line r) { // retorna t tal que t*v pertence a reta r
        return (r.p^r.q) / (double) ((r.p-r.q)^v);
}

// POLIGONO

// quadrado da distancia entre os retangulos a e b (lados paralelos aos eixos)
// assume que ta representado (inferior esquerdo, superior direito)
ll dist2_rect(pair<pt, pt> a, pair<pt, pt> b) {
        int hor = 0, vert = 0;
        if (a.second.x < b.first.x) hor = b.first.x - a.second.x;
        else if (b.second.x < a.first.x) hor = a.first.x - b.second.x;
        if (a.second.y < b.first.y) vert = b.first.y - a.second.y;
        else if (b.second.y < a.first.y) vert = a.first.y - b.second.y;
        return sq(hor) + sq(vert);
}

ll polarea2(vector<pt> v) { // 2 * area do poligono
        ll ret = 0;
        for (int i = 0; i < v.size(); i++)
                ret += sarea2(pt(0, 0), v[i], v[(i + 1) % v.size()]);
        return abs(ret);
}

// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
int inpol(vector<pt>& v, pt p) { // O(n)
        int qt = 0;
        for (int i = 0; i < v.size(); i++) {
                if (p == v[i]) return 2;
                int j = (i+1)%v.size();
                if (p.y == v[i].y and p.y == v[j].y) {
                        if ((v[i]-p)*(v[j]-p) <= 0) return 2;
```

```
                        continue;
                }
                bool baixo = v[i].y < p.y;
                if (baixo == (v[j].y < p.y)) continue;
                auto t = (p-v[i])^(v[j]-v[i]);
                if (!t) return 2;
                if (baixo == (t > 0)) qt += baixo ? 1 : -1;
        }
        return qt != 0;
}

vector<pt> convex_hull(vector<pt> v) { // convex hull - O(n log(n))
        sort(v.begin(), v.end());
        v.erase(unique(v.begin(), v.end()), v.end());
        if (v.size() <= 1) return v;
        vector<pt> l, u;
        for (int i = 0; i < v.size(); i++) {
                while (l.size() > 1 and !ccw(l.end()[-2], l.end()[-1], v[i]))
                        l.pop_back();
                l.push_back(v[i]);
        }
        for (int i = v.size() - 1; i >= 0; i--) {
                while (u.size() > 1 and !ccw(u.end()[-2], u.end()[-1], v[i]))
                        u.pop_back();
                u.push_back(v[i]);
        }
        l.pop_back(); u.pop_back();
        for (pt i : u) l.push_back(i);
        return l;
}

ll interior_points(vector<pt> v) { // pontos inteiros dentro de um poligono
    simples
        ll b = 0;
        for (int i = 0; i < v.size(); i++)
                b += segpoints(line(v[i], v[(i+1)%v.size()])) - 1;
        return (polarea2(v) - b) / 2 + 1;
}

struct convex_pol {
        vector<pt> pol;

        // nao pode ter ponto colinear no convex hull
        convex_pol() {}
        convex_pol(vector<pt> v) : pol(convex_hull(v)) {}

        // se o ponto ta dentro do hull - O(log(n))
        bool is_inside(pt p) {
                if (pol.size() == 0) return false;
                if (pol.size() == 1) return p == pol[0];
                int l = 1, r = pol.size();
                while (l < r) {
                        int m = (l+r)/2;
                        if (ccw(p, pol[0], pol[m])) l = m+1;
                        else r = m;
                }
                if (l == 1) return isinseg(p, line(pol[0], pol[1]));
                if (l == pol.size()) return false;
                return !ccw(p, pol[l], pol[l-1]);
        }
        // ponto extremo em relacao a cmp(p, q) = p mais extremo q
        // (copiado de https://github.com/gustavoM32/caderno-zika)
        int extreme(const function<bool(pt, pt)>& cmp) {
                int n = pol.size();
                auto extr = [&](int i, bool& cur_dir) {
                        cur_dir = cmp(pol[(i+1)%n], pol[i]);
                        return !cur_dir and !cmp(pol[(i+n-1)%n], pol[i]);
                };
                bool last_dir, cur_dir;
                if (extr(0, last_dir)) return 0;
                int l = 0, r = n;
                while (l+1 < r) {
                        int m = (l+r)/2;
                        if (extr(m, cur_dir)) return m;
                        bool rel_dir = cmp(pol[m], pol[l]);
                        if ((!last_dir and cur_dir) or
                                        (last_dir == cur_dir and rel_dir ==
                                            cur_dir)) {
                                l = m;
                                last_dir = cur_dir;
                        } else r = m;
                }
                return l;
        }
        int max_dot(pt v) {
                return extreme([&](pt p, pt q) { return p*v > q*v; });
        }
        pair<int, int> tangents(pt p) {
                auto L = [&](pt q, pt r) { return ccw(p, r, q); };
                auto R = [&](pt q, pt r) { return ccw(p, q, r); };
                return {extreme(L), extreme(R)};
        }
};

bool operator <(const line& a, const line& b) { // comparador pra reta
        // assume que as retas tem p < q
        pt v1 = a.q - a.p, v2 = b.q - b.p;
        bool b1 = compare_angle(v1, v2), b2 = compare_angle(v2, v1);
        if (b1 or b2) return b1;
        return ccw(a.p, a.q, b.p); // mesmo angulo
}
bool operator ==(const line& a, const line& b) {
        return !(a < b) and !(b < a);
}

// comparador pro set pra fazer sweep line com segmentos
struct cmp_sweepline {
        bool operator () (const line& a, const line& b) const {
                // assume que os segmentos tem p < q
                if (a.p == b.p) return ccw(a.p, a.q, b.q);
                if (a.p.x != a.q.x and (b.p.x == b.q.x or a.p.x < b.p.x))
                        return ccw(a.p, a.q, b.p);
                return ccw(a.p, b.q, b.p);
        }
};

// comparador pro set pra fazer sweep angle com segmentos
pt dir;
struct cmp_sweepangle {
    bool operator () (const line& a, const line& b) const {
        return get_t(dir, a) < get_t(dir, b);
    }
};
```

29.7 matrix

```
// Matriz

#define MODULAR false
template<typename T> struct matrix : vector<vector<T>> {
        int n, m;

        void print() {
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < m; j++) cout << (*this)[i][j] << " "
                            ;
                        cout << endl;
                }
        }

        matrix(int n_, int m_, bool ident = false) :
                        vector<vector<T>>(n_, vector<T>(m_, 0)), n(n_), m(m_) {
                if (ident) {
                        assert(n == m);
                        for (int i = 0; i < n; i++) (*this)[i][i] = 1;
                }
        }
        matrix(const vector<vector<T>>& c) : vector<vector<T>>(c),
                n(c.size()), m(c[0].size()) {}
        matrix(const initializer_list<initializer_list<T>>& c) {
                vector<vector<T>> val;
                for (auto& i : c) val.push_back(i);
                *this = matrix(val);
        }
```

```
        matrix<T> operator*(matrix<T>& r) {
                assert(m == r.n);
                matrix<T> M(n, r.m);
                for (int i = 0; i < n; i++) for (int k = 0; k < m; k++)
                        for (int j = 0; j < r.m; j++) {
                                T add = (*this)[i][k] * r[k][j];
#if MODULAR
#warning Usar matrix<ll> e soh colocar valores em [0, MOD) na matriz!
                                M[i][j] += add%MOD;
                                if (M[i][j] >= MOD) M[i][j] -= MOD;
#else
                                M[i][j] += add;
#endif
                        }
                return M;
        }
        matrix<T> operator^(ll e){
                matrix<T> M(n, n, true), at = *this;
                while (e) {
                        if (e&1) M = M*at;
                        e >>= 1;
                        at = at*at;
                }
                return M;
        }
        void apply_transform(matrix M, ll e){
                auto& v = *this;
                while (e) {
                        if (e&1) v = M*v;
                        e >>= 1;
                        M = M*M;
                }
        }
};
```

29.8 matroid

```
// Matroid
//
// Matroids de Grafo e Particao
// De modo geral, toda Matroid contem um build() linear
// e uma funcao constante oracle()
// oracle(i) responde se o conjunto continua independente
// apos adicao do elemento i
// oracle(i, j) responde se o conjunto continua indepente
// apos trocar o elemento i pelo elemento j
//
// Intersecao sem peso O(r^2 n)
// em que n eh o tamanho do conjunto e r eh o tamanho da resposta

// Matroid Grafica
// Matroid das florestas de um grafo
// Um conjunto de arestas eh independente se formam uma floresta
//
// build() : O(n)
// oracle() : O(1)

struct graphic_matroid {
        int n, m, t;
        vector<array<int, 2>> edges;
        vector<vector<int>> g;
        vector<int> comp, in, out;
        graphic_matroid(int n_, vector<array<int, 2>> edges_)
                : n(n_), m(edges_.size()), edges(edges_), g(n), comp(n), in(n),
                  out(n) {}
        void dfs(int u) {
                in[u] = t++;
                for (auto v : g[u]) if (in[v] == -1)
                        comp[v] = comp[u], dfs(v);
                out[u] = t;
        }
        void build(vector<int> I) {
                t = 0;
                for (int u = 0; u < n; u++) g[u].clear(), in[u] = -1;
                for (int e : I) {
                        auto [u, v] = edges[e];
                        g[u].push_back(v), g[v].push_back(u);
                }
                for (int u = 0; u < n; u++) if (in[u] == -1)
                        comp[u] = u, dfs(u);
        }
        bool is_ancestor(int u, int v) {
                return in[u] <= in[v] and in[v] < out[u];
        }
        bool oracle(int e) {
                return comp[edges[e][0]] != comp[edges[e][1]];
        }
        bool oracle(int e, int f) {
                if (oracle(f)) return true;
                int u = edges[e][in[edges[e][0]] < in[edges[e][1]]];
                return is_ancestor(u, edges[f][0]) != is_ancestor(u, edges[f
                    ][1]);
        }
};

// Matroid de particao ou cores
// Um conjunto eh independente se a quantidade de elementos
// de cada cor nao excede a capacidade da cor
// Quando todas as capacidades sao 1, um conjunto eh independente
// se todas as suas cores sao distintas
//
// build() : O(n)
// oracle() : O(1)

struct partition_matroid {
        vector<int> cap, color, d;
        partition_matroid(vector<int> cap_, vector<int> color_)
                : cap(cap_), color(color_), d(cap.size()) {}
        void build(vector<int> I) {
                fill(d.begin(), d.end(), 0);
                for (int u : I) d[color[u]]++;
        }
        bool oracle(int u) {
                return d[color[u]] < cap[color[u]];
        }
        bool oracle(int u, int v) {
                return color[u] == color[v] or oracle(v);
        }
};

// Intersecao de matroid sem pesos
// Dadas duas matroids M1 e M2 definidas sobre o mesmo
// conjunto I, retorna o maior subconjunto de I
// que eh independente tanto para M1 quanto para M2
//
// O(r^2*n)

// Matroid "pesada" deve ser a M2
template<typename Matroid1, typename Matroid2>
vector<int> matroid_intersection(int n, Matroid1 M1, Matroid2 M2) {
        vector<bool> b(n);
        vector<int> I[2];
        bool converged = false;
        while (!converged) {
                I[0].clear(), I[1].clear();
                for (int u = 0; u < n; u++) I[b[u]].push_back(u);

                M1.build(I[1]), M2.build(I[1]);
                vector<bool> target(n), pushed(n);
                queue<int> q;
                for (int u : I[0]) {
                        target[u] = M2.oracle(u);
                        if (M1.oracle(u)) pushed[u] = true, q.push(u);
                }
                vector<int> p(n, -1);
                converged = true;
                while (q.size()) {
                        int u = q.front(); q.pop();
                        if (target[u]) {
                                converged = false;
                                for (int v = u; v != -1; v = p[v]) b[v] = !b[v];
                                break;
                        }
                        for (int v : I[!b[u]]) if (!pushed[v]) {
                                if ((b[u] and M1.oracle(u, v)) or (b[v] and M2.
```

```
                                oracle(v, u)))
                                    p[v] = u, pushed[v] = true, q.push(v);
                        }
                }
        }
        return I[1];
}

// Intersecao de matroid com pesos
// Dadas duas matroids M1 e M2 e uma funcao de pesos w, todas definidas sobre
// um conjunto I retorna o maior subconjunto de I (desempatado pelo menor peso)
// que eh independente tanto para M1 quanto para M2
// A resposta eh construida incrementando o tamanho conjunto I de 1 em 1
// Se nao tiver custo negativo, nao precisa de SPFA
//
// O(r^3*n) com SPFA
// O(r^2*n*log(n)) com Dijkstra e potencial

template<typename T, typename Matroid1, typename Matroid2>
vector<int> weighted_matroid_intersection(int n, vector<T> w, Matroid1 M1,
     Matroid2 M2) {
        vector<bool> b(n), target(n), is_inside(n);
        vector<int> I[2], from(n);
        vector<pair<T, int>> d(n);
        auto check_edge = [&](int u, int v) {
                return (b[u] and M1.oracle(u, v)) or (b[v] and M2.oracle(v, u));
        };
        while (true) {
                I[0].clear(), I[1].clear();
                for (int u = 0; u < n; u++) I[b[u]].push_back(u);
                // I[1] contem o conjunto de tamanho I[1].size() de menor peso
                M1.build(I[1]), M2.build(I[1]);
                for (int u = 0; u < n; u++) {
                        target[u] = false, is_inside[u] = false, from[u] = -1;
                        d[u] = {numeric_limits<T>::max(), INF};
                }
                deque<T> q;
                sort(I[0].begin(), I[0].end(), [&](int i, int j){ return w[i] <
                     w[j]; });
                for (int u : I[0]) {
                        target[u] = M2.oracle(u);
                        if (M1.oracle(u)) {
                                if (is_inside[u]) continue;
                                d[u] = {w[u], 0};
                                if (!q.empty() and d[u] > d[q.front()]) q.
                                     push_back(u);
                                else q.push_front(u);
                                is_inside[u] = true;
                        }
                }
                while (q.size()) {
                        int u = q.front(); q.pop_front();
                        is_inside[u] = false;
                        for (int v : I[!b[u]]) if (check_edge(u, v)) {
                                pair<T, int> nd(d[u].first + w[v], d[u].second +
                                     1);
                                if (nd < d[v]) {
                                        from[v] = u, d[v] = nd;
                                        if (is_inside[v]) continue;
                                        if (q.size() and d[v] > d[q.front()]) q.
                                             push_back(v);
                                        else q.push_front(v);
                                        is_inside[v] = true;
                                }
                        }
                }
                pair<T, int> mini = pair(numeric_limits<T>::max(), INF);
                int targ = -1;
                for (int u : I[0]) if (target[u] and d[u] < mini)
                        mini = d[u], targ = u;
                if (targ != -1) for (int u = targ; u != -1; u = from[u])
                        b[u] = !b[u], w[u] *= -1;
                else break;
        }
        return I[1];
}
```

29.9 modularArithmetic

```
// Aritmetica Modular
//
// O mod tem q ser primo

template<int p> struct mod_int {
        ll expo(ll b, ll e) {
                ll ret = 1;
                while (e) {
                        if (e % 2) ret = ret * b % p;
                        e /= 2, b = b * b % p;
                }
                return ret;
        }
        ll inv(ll b) { return expo(b, p-2); }

        using m = mod_int;
        int v;
        mod_int() : v(0) {}
        mod_int(ll v_) {
                if (v_ >= p or v_ <= -p) v_ %= p;
                if (v_ < 0) v_ += p;
                v = v_;
        }
        m& operator +=(const m& a) {
                v += a.v;
                if (v >= p) v -= p;
                return *this;
        }
        m& operator -=(const m& a) {
                v -= a.v;
                if (v < 0) v += p;
                return *this;
        }
        m& operator *=(const m& a) {
                v = v * ll(a.v) % p;
                return *this;
        }
        m& operator /=(const m& a) {
                v = v * inv(a.v) % p;
                return *this;
        }
        m operator -(){ return m(-v); }
        m& operator ^=(ll e) {
                if (e < 0) {
                        v = inv(v);
                        e = -e;
                }
                v = expo(v, e);
                // possivel otimizacao:
                // cuidado com 0^0
                // v = expo(v, e%(p-1));
                return *this;
        }
        bool operator ==(const m& a) { return v == a.v; }
        bool operator !=(const m& a) { return v != a.v; }

        friend istream& operator >>(istream& in, m& a) {
                ll val; in >> val;
                a = m(val);
                return in;
        }
        friend ostream& operator <<(ostream& out, m a) {
                return out << a.v;
        }
        friend m operator +(m a, m b) { return a += b; }
        friend m operator -(m a, m b) { return a -= b; }
        friend m operator *(m a, m b) { return a *= b; }
        friend m operator /(m a, m b) { return a /= b; }
        friend m operator ^(m a, ll e) { return a ^= e; }
};

typedef mod_int<(int)1e9+7> mint;
```

30 ufmg forked/Problemas

30.1 additionChain

```
// Shortest Addition Chain
//
// Computa o menor numero de adicoes para construir
// cada valor, comecando com 1 (e podendo salvar variaveis)
// Retorna um par com a dp e o pai na arvore
// A arvore eh tao que o taminho da raiz (1) ate x
// contem os valores que devem ser criados para gerar x
// A profundidade de x na arvore eh dp[x]
// DP funciona para ateh 300, mas a arvore soh funciona
// para ateh 148

// recuperacao certa soh ateh 148 (erra para 149, 233, 298)
pair<vector<int>, vector<int>> addition_chain() {
        int MAX = 301;
        vector<int> dp(MAX), p(MAX);
        for (int n = 2; n < MAX; n++) {
                pair<int, int> val = {INF, -1};
                for (int i = 1; i < n; i++) for (int j = i; j; j = p[j])
                        if (j == n-i) val = min(val, pair(dp[i]+1, i));
                tie(dp[n], p[n]) = val;
                if (n == 9) p[n] = 8;
                if (n == 149 or n == 233) dp[n]--;
        }
        return {dp, p};
}
```

30.2 angleRange

```
// Angle Range Intersection
//
// Computa intersecao de angulos
// Os angulos (arcos) precisam ter comprimeiro < pi
// (caso contrario a intersecao eh estranha)
//
// Tudo O(1)

struct angle_range {
        static constexpr ld ALL = 1e9, NIL = -1e9;
        ld l, r;
        angle_range() : l(ALL), r(ALL) {}
        angle_range(ld l_, ld r_) : l(l_), r(r_) { fix(l), fix(r); }

        void fix(ld& theta) {
                if (theta == ALL or theta == NIL) return;
                if (theta > 2*pi) theta -= 2*pi;
                if (theta < 0) theta += 2*pi;
        }
        bool empty() { return l == NIL; }
        bool contains(ld q) {
                fix(q);
                if (l == ALL) return true;
                if (l == NIL) return false;
                if (l < r) return l < q and q < r;
                return q > l or q < r;
        }
        friend angle_range operator &(angle_range p, angle_range q) {
                if (p.l == ALL or q.l == NIL) return q;
                if (q.l == ALL or p.l == NIL) return p;
                if (p.l > p.r and q.l > q.r) return {max(p.l, q.l) , min(p.r, q.
                    r)};
                if (q.l > q.r) swap(p.l, q.l), swap(p.r, q.r);
                if (p.l > p.r) {
                        if (q.r > p.l) return {max(q.l, p.l) , q.r};
                        else if (q.l < p.r) return {q.l, min(q.r, p.r)};
                        return {NIL, NIL};
                }
                if (max(p.l, q.l) > min(p.r, q.r)) return {NIL, NIL};
                return {max(p.l, q.l), min(p.r, q.r)};
        }
};
```

30.3 areaHistograma

```
// Area Maxima de Histograma
//
// Assume que todas as barras tem largura 1,
// e altura dada no vetor v
//
// O(n)

ll area(vector<int> v) {
        ll ret = 0;
        stack<int> s;
        // valores iniciais pra dar tudo certo
        v.insert(v.begin(), -1);
        v.insert(v.end(), -1);
        s.push(0);

        for(int i = 0; i < (int) v.size(); i++) {
                while (v[s.top()] > v[i]) {
                        ll h = v[s.top()]; s.pop();
                        ret = max(ret, h * (i - s.top() - 1));
                }
                s.push(i);
        }

        return ret;
}
```

30.4 areaUniaoRetangulo

```
// Area da Uniao de Retangulos
//
// O(n log(n))
// 5d8d2f

namespace seg {
        pair<int, ll> seg[4*MAX];
        ll lazy[4*MAX], *v;
        int n;

        pair<int, ll> merge(pair<int, ll> l, pair<int, ll> r){
                if (l.second == r.second) return {l.first+r.first, l.second};
                else if (l.second < r.second) return l;
                else return r;
        }

        pair<int, ll> build(int p=1, int l=0, int r=n-1) {
                lazy[p] = 0;
                if (l == r) return seg[p] = {1, v[l]};
                int m = (l+r)/2;
                return seg[p] = merge(build(2*p, l, m), build(2*p+1, m+1, r));
        }
        void build(int n2, ll* v2) {
                n = n2, v = v2;
                build();
        }
        void prop(int p, int l, int r) {
                seg[p].second += lazy[p];
                if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
                lazy[p] = 0;
        }
        pair<int, ll> query(int a, int b, int p=1, int l=0, int r=n-1) {
                prop(p, l, r);
                if (a <= l and r <= b) return seg[p];
                if (b < l or r < a) return {0, LINF};
                int m = (l+r)/2;
                return merge(query(a, b, 2*p, l, m), query(a, b, 2*p+1, m+1, r))
                    ;
        }
        pair<int, ll> update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
```

```
                prop(p, l, r);
                if (a <= l and r <= b) {
                        lazy[p] += x;
                        prop(p, l, r);
                        return seg[p];
                }
                if (b < l or r < a) return seg[p];
                int m = (l+r)/2;
                return seg[p] = merge(update(a, b, x, 2*p, l, m),
                                update(a, b, x, 2*p+1, m+1, r));
        }
};

ll seg_vec[MAX];

ll area_sq(vector<pair<pair<int, int>, pair<int, int>>> &sq){
        vector<pair<pair<int, int>, pair<int, int>>> up;
        for (auto it : sq){
                int x1, y1, x2, y2;
                tie(x1, y1) = it.first;
                tie(x2, y2) = it.second;
                up.push_back({{x1+1,  1}, {y1, y2}});
                up.push_back({{x2+1, -1}, {y1, y2}});
        }
        sort(up.begin(), up.end());
        memset(seg_vec, 0, sizeof seg_vec);
        ll H_MAX = MAX;
        seg::build(H_MAX-1, seg_vec);
        auto it = up.begin();
        ll ans = 0;
        while (it != up.end()){
                ll L = (*it).first.first;
                while (it != up.end() && (*it).first.first == L){
                        int x, inc, y1, y2;
                        tie(x, inc) = it->first;
                        tie(y1, y2) = it->second;
                        seg::update(y1+1, y2, inc);
                        it++;
                }
                if (it == up.end()) break;
                ll R = (*it).first.first;

                ll W = R-L;
                auto jt = seg::query(0, H_MAX-1);
                ll H = H_MAX - 1;
                if (jt.second == 0) H -= jt.first;
                ans += W*H;
        }
        return ans;
}
```

30.5 binomial

```
// Binomial modular
//
// Computa C(n, k) mod m em O(m + log(m) log(n))
// = O(rapido)

ll divi[MAX];

ll expo(ll a, ll b, ll m) {
        if (!b) return 1;
        ll ans = expo(a*a%m, b/2, m);
        if (b%2) ans *= a;
        return ans%m;
}

ll inv(ll a, ll b){
        return 1<a ? b - inv(b%a,a)*b/a : 1;
}

template<typename T> tuple<T, T, T> ext_gcd(T a, T b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd(b%a, a);
    return {g, y - b/a*x, x};
}

template<typename T = ll> struct crt {
        T a, m;

        crt() : a(0), m(1) {}
        crt(T a_, T m_) : a(a_), m(m_) {}
        crt operator * (crt C) {
                auto [g, x, y] = ext_gcd(m, C.m);
                if ((a - C.a) % g) a = -1;
                if (a == -1 or C.a == -1) return crt(-1, 0);
                T lcm = m/g*C.m;
                T ans = a + (x*(C.a-a)/g % (C.m/g))*m;
                return crt((ans % lcm + lcm) % lcm, lcm);
        }
};

pair<ll, ll> divide_show(ll n, int p, int k, int pak) {
        if (n == 0) return {0, 1};
        ll blocos = n/pak, falta = n%pak;
        ll periodo = divi[pak], resto = divi[falta];
        ll r = expo(periodo, blocos, pak)*resto%pak;

        auto rec = divide_show(n/p, p, k, pak);
        ll y = n/p + rec.first;
        r = r*rec.second % pak;

        return {y, r};
}

ll solve_pak(ll n, ll x, int p, int k, int pak) {
        divi[0] = 1;
        for (int i = 1; i <= pak; i++) {
                divi[i] = divi[i-1];
                if (i%p) divi[i] = divi[i] * i % pak;
        }

        auto dn = divide_show(n, p, k, pak), dx = divide_show(x, p, k, pak),
             dnx = divide_show(n-x, p, k, pak);
        ll y = dn.first-dx.first-dnx.first, r =
             (dn.second*inv(dx.second, pak)%pak)*inv(dnx.second, pak)%pak;
        return expo(p, y, pak) * r % pak;
}

ll solve(ll n, ll x, int mod) {
        vector<pair<int, int>> f;
        int mod2 = mod;
        for (int i = 2; i*i <= mod2; i++) if (mod2%i==0) {
                int c = 0;
                while (mod2%i==0) mod2 /= i, c++;
                f.push_back({i, c});
        }
        if (mod2 > 1) f.push_back({mod2, 1});
        crt ans(0, 1);
        for (int i = 0; i < f.size(); i++) {
                int pak = 1;
                for (int j = 0; j < f[i].second; j++) pak *= f[i].first;
                ans = ans * crt(solve_pak(n, x, f[i].first, f[i].second, pak),
                    pak);
        }
        return ans.a;
}
```

30.6 closestPairOfPoints

```
// Closest pair of points
//
// O(nlogn)

pair<pt, pt> closest_pair_of_points(vector<pt> v) {
        int n = v.size();
        sort(v.begin(), v.end());
        for (int i = 1; i < n; i++) if (v[i] == v[i-1]) return {v[i-1], v[i]};
        auto cmp_y = [&](const pt &l, const pt &r) {
                if (l.y != r.y) return l.y < r.y;
                return l.x < r.x;
        };
```

```
        set<pt, decltype(cmp_y)> s(cmp_y);
        int l = 0, r = -1;
        ll d2_min = numeric_limits<ll>::max();
        pt pl, pr;
        const int magic = 5;
        while (r+1 < n) {
                auto it = s.insert(v[++r]).first;
                int cnt = magic/2;
                while (cnt-- and it != s.begin()) it--;
                cnt = 0;
                while (cnt++ < magic and it != s.end()) {
                        if (!((*it) == v[r])) {
                                ll d2 = dist2(*it, v[r]);
                                if (d2_min > d2) {
                                        d2_min = d2;
                                        pl = *it;
                                        pr = v[r];
                                }
                        }
                        it++;
                }
                while (l < r and sq(v[l].x-v[r].x) > d2_min) s.erase(v[l++]);
        }
        return {pl, pr};
}
```

30.7 conectividadeDinamica

```
// Conectividade Dinamica DC
//
// Offline com Divide and Conquer e
// DSU com rollback
// O(n log^2(n))

typedef pair<int, int> T;

namespace data {
        int n, ans;
        int p[MAX], sz[MAX];
        stack<int> S;

        void build(int n2) {
                n = n2;
                for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
                ans = n;
        }
        int find(int k) {
                while (p[k] != k) k = p[k];
                return k;
        }
        void add(T x) {
                int a = x.first, b = x.second;
                a = find(a), b = find(b);
                if (a == b) return S.push(-1);
                ans--;
                if (sz[a] > sz[b]) swap(a, b);
                S.push(a);
                sz[b] += sz[a];
                p[a] = b;
        }
        int query() {
                return ans;
        }
        void rollback() {
                int u = S.top(); S.pop();
                if (u == -1) return;
                sz[p[u]] -= sz[u];
                p[u] = u;
                ans++;
        }
};

int ponta[MAX]; // outra ponta do intervalo ou -1 se for query
int ans[MAX], n, q;
T qu[MAX];

void solve(int l = 0, int r = q-1) {
        if (l >= r) {
                ans[l] = data::query(); // agora a estrutura ta certa
                return;
        }
        int m = (l+r)/2, qnt = 1;
        for (int i = m+1; i <= r; i++) if (ponta[i]+1 and ponta[i] < l)
                data::add(qu[i]), qnt++;
        solve(l, m);
        while (--qnt) data::rollback();
        for (int i = l; i <= m; i++) if (ponta[i]+1 and ponta[i] > r)
                data::add(qu[i]), qnt++;
        solve(m+1, r);
        while (qnt--) data::rollback();
}
```

30.8 conectividadeDinamica2

```
// Conectividade Dinamica LCT
//
// Offline com link-cut trees
// O(n log(n))

namespace lct {
        struct node {
                int p, ch[2];
                int val, sub;
                bool rev;
                node() {}
                node(int v) : p(-1), val(v), sub(v), rev(0) { ch[0] = ch[1] =
                     -1; }
        };

        node t[2*MAX]; // MAXN + MAXQ
        map<pair<int, int>, int> aresta;
        int sz;

        void prop(int x) {
                if (t[x].rev) {
                        swap(t[x].ch[0], t[x].ch[1]);
                        if (t[x].ch[0]+1) t[t[x].ch[0]].rev ^= 1;
                        if (t[x].ch[1]+1) t[t[x].ch[1]].rev ^= 1;
                }
                t[x].rev = 0;
        }
        void update(int x) {
                t[x].sub = t[x].val;
                for (int i = 0; i < 2; i++) if (t[x].ch[i]+1) {
                        prop(t[x].ch[i]);
                        t[x].sub = min(t[x].sub, t[t[x].ch[i]].sub);
                }
        }
        bool is_root(int x) {
                return t[x].p == -1 or (t[t[x].p].ch[0] != x and t[t[x].p].ch[1]
                     != x);
        }
        void rotate(int x) {
                int p = t[x].p, pp = t[p].p;
                if (!is_root(p)) t[pp].ch[t[pp].ch[1] == p] = x;
                bool d = t[p].ch[0] == x;
                t[p].ch[!d] = t[x].ch[d], t[x].ch[d] = p;
                if (t[p].ch[!d]+1) t[t[p].ch[!d]].p = p;
                t[x].p = pp, t[p].p = x;
                update(p), update(x);
        }
        int splay(int x) {
                while (!is_root(x)) {
                        int p = t[x].p, pp = t[p].p;
                        if (!is_root(p)) prop(pp);
                        prop(p), prop(x);
                        if (!is_root(p)) rotate((t[pp].ch[0] == p)^(t[p].ch[0]
                             == x) ? x : p);
                        rotate(x);
                }
                return prop(x), x;
        }
```

```
        int access(int v) {
                int last = -1;
                for (int w = v; w+1; update(last = w), splay(v), w = t[v].p)
                        splay(w), t[w].ch[1] = (last == -1 ? -1 : v);
                return last;
        }
        void make_tree(int v, int w=INF) { t[v] = node(w); }
        bool conn(int v, int w) {
                access(v), access(w);
                return v == w ? true : t[v].p != -1;
        }
        void rootify(int v) {
                access(v);
                t[v].rev ^= 1;
        }
        int query(int v, int w) {
                rootify(w), access(v);
                return t[v].sub;
        }
        void link_(int v, int w) {
                rootify(w);
                t[w].p = v;
        }
        void link(int v, int w, int x) { // v--w com peso x
                int id = MAX + sz++;
                aresta[make_pair(v, w)] = id;
                make_tree(id, x);
                link_(v, id), link_(id, w);
        }
        void cut_(int v, int w) {
                rootify(w), access(v);
                t[v].ch[0] = t[t[v].ch[0]].p = -1;
        }
        void cut(int v, int w) {
                int id = aresta[make_pair(v, w)];
                cut_(v, id), cut_(id, w);
        }
}

void dyn_conn() {
        int n, q; cin >> n >> q;
        vector<int> p(2*q, -1); // outra ponta do intervalo
        for (int i = 0; i < n; i++) lct::make_tree(i);
        vector<pair<int, int>> qu(q);
        map<pair<int, int>, int> m;
        for (int i = 0; i < q; i++) {
                char c; cin >> c;
                if (c == '?') continue;
                int a, b; cin >> a >> b; a--, b--;
                if (a > b) swap(a, b);
                qu[i] = {a, b};
                if (c == '+') {
                        p[i] = i+q, p[i+q] = i;
                        m[make_pair(a, b)] = i;
                } else {
                        int j = m[make_pair(a, b)];
                        p[i] = j, p[j] = i;
                }
        }
        int ans = n;
        for (int i = 0; i < q; i++) {
                if (p[i] == -1) {
                        cout << ans << endl; // numero de comp conexos
                        continue;
                }
                int a = qu[i].first, b = qu[i].second;
                if (p[i] > i) { // +
                        if (lct::conn(a, b)) {
                                int mi = lct::query(a, b);
                                if (p[i] < mi) {
                                        p[p[i]] = p[i];
                                        continue;
                                }
                                lct::cut(qu[p[mi]].first, qu[p[mi]].second), ans
                                    ++;
                                p[mi] = mi;
                        }
                        lct::link(a, b, p[i]), ans--;
                } else if (p[i] != i) lct::cut(a, b), ans++; // -
        }
}
```

30.9 deBrujin

```
// Sequencia de de Brujin
//
// Se passar sem o terceiro parametro, gera um vetor com valores
// em [0, k) de tamanho k^n de forma que todos os subarrays ciclicos
// de tamanho n ocorrem exatamente uma vez
// Se passar com um limite lim, gera o menor vetor com valores
// em [0, k) que possui lim subarrays de tamanho n distintos
// (assume que lim <= k^n)
//
// Linear no tamanho da resposta

vector<int> de_brujin(int n, int k, int lim = INF) {
        if (k == 1) return vector<int>(lim == INF ? 1 : n, 0);
        vector<int> l = {0}, ret; // l eh lyndon word
        while (true) {
                if (l.size() == 0) {
                        if (lim == INF) break;
                        l.push_back(0);
                }
                if (n % l.size() == 0) for (int i : l) {
                        ret.push_back(i);
                        if (ret.size() == n+lim-1) return ret;
                }
                int p = l.size();
                while (l.size() < n) l.push_back(l[l.size()%p]);
                while (l.size() and l.back() == k-1) l.pop_back();
                if (l.size()) l.back()++;
        }
        return ret;
}
```

30.10 delaunay

```
// Triangulacao de Delaunay
//
// Computa a triangulacao de Delaunay, o dual
// do diagrama de Voronoi (a menos de casos degenerados)
// Retorna um grafo indexado pelos indices dos pontos, e as arestas
// sao as arestas da triangulacao
// As arestas partindo de um vertice ja vem ordenadas por angulo,
// ou seja, se o vertice v nao esta no convex hull, (v, v_i, v_{i+1})
// eh um triangulo da triangulacao, em que v_i eh o i-esimo vizinho
// Usa o alg d&c, precisa representar MAX_COOR^4, por isso __int128
// pra aguentar valores ateh 1e9
//
// Propriedades:
// 1 - O grafo tem no max 3n-6 arestas
// 2 - Para todo triangulo, a circunf. que passa pelos 3 pontos
//     nao contem estritamente nenhum ponto
// 3 - A MST euclidiana eh subgrafo desse grafo
// 4 - Cada ponto eh vizinho do ponto mais proximo dele
//
// O(n log n)

typedef struct QuadEdge* Q;
struct QuadEdge {
        int id;
        pt o;
        Q rot, nxt;
        bool used;

        QuadEdge(int id_ = -1, pt o_ = pt(INF, INF)) :
                id(id_), o(o_), rot(nullptr), nxt(nullptr), used(false) {}

        Q rev() const { return rot->rot; }
        Q next() const { return nxt; }
        Q prev() const { return rot->next()->rot; }
```

```
        pt dest() const { return rev()->o; }
};

Q edge(pt from, pt to, int id_from, int id_to) {
        Q e1 = new QuadEdge(id_from, from);
        Q e2 = new QuadEdge(id_to, to);
        Q e3 = new QuadEdge;
        Q e4 = new QuadEdge;
        tie(e1->rot, e2->rot, e3->rot, e4->rot) = {e3, e4, e2, e1};
        tie(e1->nxt, e2->nxt, e3->nxt, e4->nxt) = {e1, e2, e4, e3};
        return e1;
}

void splice(Q a, Q b) {
        swap(a->nxt->rot->nxt, b->nxt->rot->nxt);
        swap(a->nxt, b->nxt);
}

void del_edge(Q& e, Q ne) { // delete e and assign e <- ne
        splice(e, e->prev());
        splice(e->rev(), e->rev()->prev());
        delete e->rev()->rot, delete e->rev();
        delete e->rot; delete e;
        e = ne;
}

Q conn(Q a, Q b) {
        Q e = edge(a->dest(), b->o, a->rev()->id, b->id);
        splice(e, a->rev()->prev());
        splice(e->rev(), b);
        return e;
}

bool in_c(pt a, pt b, pt c, pt p) { // p ta na circunf. (a, b, c) ?
        __int128 p2 = p*p, A = a*a - p2, B = b*b - p2, C = c*c - p2;
        return sarea2(p, a, b) * C + sarea2(p, b, c) * A + sarea2(p, c, a) * B >
             0;
}

pair<Q, Q> build_tr(vector<pt>& p, int l, int r) {
        if (r-l+1 <= 3) {
                Q a = edge(p[l], p[l+1], l, l+1), b = edge(p[l+1], p[r], l+1, r)
                     ;
                if (r-l+1 == 2) return {a, a->rev()};
                splice(a->rev(), b);
                ll ar = sarea2(p[l], p[l+1], p[r]);
                Q c = ar ? conn(b, a) : 0;
                if (ar >= 0) return {a, b->rev()};
                return {c->rev(), c};
        }
        int m = (l+r)/2;
        auto [la, ra] = build_tr(p, l, m);
        auto [lb, rb] = build_tr(p, m+1, r);
        while (true) {
                if (ccw(lb->o, ra->o, ra->dest())) ra = ra->rev()->prev();
                else if (ccw(lb->o, ra->o, lb->dest())) lb = lb->rev()->next();
                else break;
        }
        Q b = conn(lb->rev(), ra);
        auto valid = [&](Q e) { return ccw(e->dest(), b->dest(), b->o); };
        if (ra->o == la->o) la = b->rev();
        if (lb->o == rb->o) rb = b;
        while (true) {
                Q L = b->rev()->next();
                if (valid(L)) while (in_c(b->dest(), b->o, L->dest(), L->next()
                     ->dest()))
                        del_edge(L, L->next());
                Q R = b->prev();
                if (valid(R)) while (in_c(b->dest(), b->o, R->dest(), R->prev()
                     ->dest()))
                        del_edge(R, R->prev());
                if (!valid(L) and !valid(R)) break;
                if (!valid(L) or (valid(R) and in_c(L->dest(), L->o, R->o, R->
                     dest())))
                        b = conn(R, b->rev());
                else b = conn(b->rev(), L->rev());
        }
        return {la, rb};
}
```

```
vector<vector<int>> delaunay(vector<pt> v) {
        int n = v.size();
        auto tmp = v;
        vector<int> idx(n);
        iota(idx.begin(), idx.end(), 0);
        sort(idx.begin(), idx.end(), [&](int l, int r) { return v[l] < v[r]; });
        for (int i = 0; i < n; i++) v[i] = tmp[idx[i]];
        assert(unique(v.begin(), v.end()) == v.end());
        vector<vector<int>> g(n);
        bool col = true;
        for (int i = 2; i < n; i++) if (sarea2(v[i], v[i-1], v[i-2])) col =
             false;
        if (col) {
                for (int i = 1; i < n; i++)
                        g[idx[i-1]].push_back(idx[i]), g[idx[i]].push_back(idx[i
                             -1]);
                return g;
        }
        Q e = build_tr(v, 0, n-1).first;
        vector<Q> edg = {e};
        for (int i = 0; i < edg.size(); e = edg[i++]) {
                for (Q at = e; !at->used; at = at->next()) {
                        at->used = true;
                        g[idx[at->id]].push_back(idx[at->rev()->id]);
                        edg.push_back(at->rev());
                }
        }
        return g;
}
```

30.11 distinct

```
// Distinct Range Query
//
// build - O(n (log n + log(sigma)))
// query - O(log(sigma))

namespace perseg { };

int qt[MAX];

void build(vector<int>& v) {
        int n = v.size();
        perseg::build(n);
        map<int, int> last;
        int at = 0;
        for (int i = 0; i < n; i++) {
                if (last.count(v[i])) {
                        perseg::update(last[v[i]], -1);
                        at++;
                }
                perseg::update(i, 1);
                qt[i] = ++at;
                last[v[i]] = i;
        }
}

int query(int l, int r) {
        return perseg::query(l, r, qt[r]);
}
```

30.12 distinctUpdate

```
// Distinct Range Query com Update
//
// build - O(n log(n))
// query - O(log^2(n))
// update - O(log^2(n))

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
```

```
using namespace __gnu_pbds;
template <class T>
        using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;

int v[MAX], n, nxt[MAX], prv[MAX];
map<int, set<int> > ocor;

namespace bit {
        ord_set<pair<int, int>> bit[MAX];

        void build() {
                for (int i = 1; i <= n; i++) bit[i].insert({nxt[i-1], i-1});
                for (int i = 1; i <= n; i++) {
                        int j = i + (i&-i);
                        if (j <= n) for (auto x : bit[i]) bit[j].insert(x);
                }
        }
        int pref(int p, int x) {
                int ret = 0;
                for (; p; p -= p&-p) ret += bit[p].order_of_key({x, -INF});
                return ret;
        }
        int query(int l, int r, int x) {
                return pref(r+1, x) - pref(l, x);
        }
        void update(int p, int x) {
                int p2 = p;
                for (p++; p <= n; p += p&-p) {
                        bit[p].erase({nxt[p2], p2});
                        bit[p].insert({x, p2});
                }
        }
}

void build() {
        for (int i = 0; i < n; i++) nxt[i] = INF;
        for (int i = 0; i < n; i++) prv[i] = -INF;
        vector<pair<int, int>> t;
        for (int i = 0; i < n; i++) t.push_back({v[i], i});
        sort(t.begin(), t.end());
        for (int i = 0; i < n; i++) {
                if (i and t[i].first == t[i-1].first)
                        prv[t[i].second] = t[i-1].second;
                if (i+1 < n and t[i].first == t[i+1].first)
                        nxt[t[i].second] = t[i+1].second;
        }

        for (int i = 0; i < n; i++) ocor[v[i]].insert(i);

        bit::build();
}

void muda(int p, int x) {
        bit::update(p, x);
        nxt[p] = x;
}

int query(int a, int b) {
        return b-a+1 - bit::query(a, b, b+1);
}

void update(int p, int x) { // mudar valor na pos. p para x
        if (prv[p] > -INF) muda(prv[p], nxt[p]);
        if (nxt[p] < INF) prv[nxt[p]] = prv[p];

        ocor[v[p]].erase(p);
        if (!ocor[x].size()) {
                muda(p, INF);
                prv[p] = -INF;
        } else if (*ocor[x].rbegin() < p) {
                int i = *ocor[x].rbegin();
                prv[p] = i;
                muda(p, INF);
                muda(i, p);
        } else {
                int i = *ocor[x].lower_bound(p);
                if (prv[i] > -INF) {
                        muda(prv[i], p);
                        prv[p] = prv[i];
                } else prv[p] = -INF;
                prv[i] = p;
                muda(p, i);
        }
        v[p] = x; ocor[x].insert(p);
}
```

30.13 dominacao3d

```
// DP de Dominacao 3D
//
// Computa para todo ponto i,
// dp[i] = 1 + max_{j dominado por i} dp[j]
// em que ser dominado eh ter as 3 coordenadas menores
// Da pra adaptar facil para outras dps
//
// O(n log^2 n), O(n) de memoria

void lis2d(vector<vector<tuple<int, int, int>>>& v, vector<int>& dp, int l, int
     r) {
        if (l == r) {
                for (int i = 0; i < v[l].size(); i++) {
                        int ii = get<2>(v[l][i]);
                        dp[ii] = max(dp[ii], 1);
                }
                return;
        }
        int m = (l+r)/2;
        lis2d(v, dp, l, m);

        vector<tuple<int, int, int>> vv[2];
        vector<int> Z;
        for (int i = l; i <= r; i++) for (auto it : v[i]) {
                vv[i > m].push_back(it);
                Z.push_back(get<1>(it));
        }
        sort(vv[0].begin(), vv[0].end());
        sort(vv[1].begin(), vv[1].end());
        sort(Z.begin(), Z.end());
        auto get_z = [&](int z) { return lower_bound(Z.begin(), Z.end(), z) - Z.
             begin(); };
        vector<int> bit(Z.size());

        int i = 0;
        for (auto [y, z, id] : vv[1]) {
                while (i < vv[0].size() and get<0>(vv[0][i]) < y) {
                        auto [y2, z2, id2] = vv[0][i++];
                        for (int p = get_z(z2)+1; p <= Z.size(); p += p&-p)
                                bit[p-1] = max(bit[p-1], dp[id2]);
                }
                int q = 0;
                for (int p = get_z(z); p; p -= p&-p) q = max(q, bit[p-1]);
                dp[id] = max(dp[id], q + 1);
        }
        lis2d(v, dp, m+1, r);
}

vector<int> solve(vector<tuple<int, int, int>> v) {
        int n = v.size();
        vector<tuple<int, int, int, int>> vv;
        for (int i = 0; i < n; i++) {
                auto [x, y, z] = v[i];
                vv.emplace_back(x, y, z, i);
        }
        sort(vv.begin(), vv.end());

        vector<vector<tuple<int, int, int>>> V;
        for (int i = 0; i < n; i++) {
                int j = i;
                V.emplace_back();
                while (j < n and get<0>(vv[j]) == get<0>(vv[i])) {
                        auto [x, y, z, id] = vv[j++];
                        V.back().emplace_back(y, z, id);
                }
                i = j-1;
```

```
        }
        vector<int> dp(n);
        lis2d(V, dp, 0, V.size()-1);
        return dp;
}
```

30.14 dominatorPoints

```
// Dominator Points
//
// Se um ponto A tem ambas as coordenadas >= B, dizemos
// que A domina B
// is_dominated(p) fala se existe algum ponto no conjunto
// que domina p
// insert(p) insere p no conjunto
// (se p for dominado por alguem, nao vai inserir)
// o multiset 'quina' guarda informacao sobre os pontos
// nao dominados por um elemento do conjunto que nao dominam
// outro ponto nao dominado por um elemento do conjunto
// No caso, armazena os valores de x+y esses pontos
//
// Complexidades:
// is_dominated - O(log(n))
// insert - O(log(n)) amortizado
// query - O(1)

struct dominator_points {
        set<pair<int, int>> se;
        multiset<int> quina;

        bool is_dominated(pair<int, int> p) {
                auto it = se.lower_bound(p);
                if (it == se.end()) return 0;
                return it->second >= p.second;
        }
        void mid(pair<int, int> a, pair<int, int> b, bool rem) {
                pair<int, int> m = {a.first+1, b.second+1};
                int val = m.first + m.second;
                if (!rem) quina.insert(val);
                else quina.erase(quina.find(val));
        }
        bool insert(pair<int, int> p) {
                if (is_dominated(p)) return 0;
                auto it = se.lower_bound(p);
                if (it != se.begin() and it != se.end())
                        mid(*prev(it), *it, 1);
                while (it != se.begin()) {
                        it--;
                        if (it->second > p.second) break;
                        if (it != se.begin()) mid(*prev(it), *it, 1);
                        it = se.erase(it);
                }
                it = se.insert(p).first;
                if (it != se.begin()) mid(*prev(it), *it, 0);
                if (next(it) != se.end()) mid(*it, *next(it), 0);
                return 1;
        }
        int query() {
                if (!quina.size()) return INF;
                return *quina.begin();
        }
};
```

30.15 dynamicHull

```
// Convex Hull Dinamico
//
// insert - O(log n) amortizado
// is_inside - O(log n)

struct upper {
        set<pt> se;
        set<pt>::iterator it;

        int is_under(pt p) { // 1 -> inside ; 2 -> border
                it = se.lower_bound(p);
                if (it == se.end()) return 0;
                if (it == se.begin()) return p == *it ? 2 : 0;
                if (ccw(p, *it, *prev(it))) return 1;
                return ccw(p, *prev(it), *it) ? 0 : 2;
        }
        void insert(pt p) {
                if (is_under(p)) return;

                if (it != se.end()) while (next(it) != se.end() and !ccw(*next(
                     it), *it, p))
                        it = se.erase(it);
                if (it != se.begin()) while (--it != se.begin() and !ccw(p, *it,
                      *prev(it)))
                        it = se.erase(it);

                se.insert(p);
        }
};

struct dyn_hull {
        upper U, L;

        int is_inside(pt p) {
                int u = U.is_under(p), l = L.is_under({-p.x, -p.y});
                if (!u or !l) return 0;
                return max(u, l);
        }
        void insert(pt p) {
                U.insert(p);
                L.insert({-p.x, -p.y});
        }
        int size() {
                int ans = U.se.size() + L.se.size();
                return ans <= 2 ? ans/2 : ans-2;
        }
};
```

30.16 graphTriangles

```
// Triangulos em Grafos
//
// get_triangles(i) encontra todos os triangulos ijk no grafo
// Custo nas arestas
// retorna {custo do triangulo, {j, k}}
//
// O(m sqrt(m) log(n)) se chamar para todos os vertices

vector<pair<int, int>> g[MAX]; // {para, peso}

#warning o 'g' deve estar ordenado
vector<pair<int, pair<int, int>>> get_triangles(int i) {
        vector<pair<int, pair<int, int>>> tri;
        for (pair<int, int> j : g[i]) {
                int a = i, b = j.first;
                if (g[a].size() > g[b].size()) swap(a, b);
                for (pair<int, int> c : g[a]) if (c.first != b and c.first > j.
                     first) {
                        auto it = lower_bound(g[b].begin(), g[b].end(),
                             make_pair(c.first, -INF));
                        if (it == g[b].end() or it->first != c.first) continue;
                        tri.push_back({j.second+c.second+it->second, {a == i ? b
                              : a, c.first}});
                }
        }
        return tri;
}
```

30.17 grayCode

```
// Gray Code
//
// Gera uma permutacao de 0 a 2^n-1, de forma que
// duas posicoes adjacentes diferem em exatamente 1 bit
//
// O(2^n)

vector<int> gray_code(int n) {
        vector<int> ret(1<<n);
        for (int i = 0; i < (1<<n); i++) ret[i] = i^(i>>1);
        return ret;
}
```

30.18 halfPlaneIntersection

```
// Half-plane intersection
// Computa a regiao convexa formada pela intersecao de n half-planes
// Cada half-plane eh identificado por uma reta e a regiao ccw a ela
//
// O(n log n)

vector<pt> hp_intersection(vector<line> &v) {
        deque<pt> dq = {{INF, INF}, {-INF, INF}, {-INF, -INF}, {INF, -INF}};

#warning considerar trocar por compare_angle
        sort(v.begin(), v.end(), [&](line r, line s) { return angle(r.q-r.p) <
             angle(s.q-s.p); });

        for(int i = 0; i < v.size() and dq.size() > 1; i++) {
                pt p1 = dq.front(), p2 = dq.back();
                while (dq.size() and !ccw(v[i].p, v[i].q, dq.back()))
                        p1 = dq.back(), dq.pop_back();
                while (dq.size() and !ccw(v[i].p, v[i].q, dq.front()))
                        p2 = dq.front(), dq.pop_front();

                if (!dq.size()) break;
                if (p1 == dq.front() and p2 == dq.back()) continue;
                dq.push_back(inter(v[i], line(dq.back(), p1)));
                dq.push_front(inter(v[i], line(dq.front(), p2)));

                if (dq.size() > 1 and dq.back() == dq.front()) dq.pop_back();
        }
        return vector<pt>(dq.begin(), dq.end());
}
```

30.19 heapSort

```
// Heap Sort
//
// O(n log n)

void down(vector<int>& v, int n, int i) {
        while ((i = 2*i+1) < n) {
                if (i+1 < n and v[i] < v[i+1]) i++;
                if (v[i] < v[(i-1)/2]) break;
                swap(v[i], v[(i-1)/2]);
        }
}
void heap_sort(vector<int>& v) {
        int n = v.size();
        for (int i = n/2-1; i >= 0; i--) down(v, n, i);
        for (int i = n-1; i > 0; i--)
                swap(v[0], v[i]), down(v, i, 0);
}
```

30.20 hungarian

```
// Hungaro
//
// Resolve o problema de assignment (matriz n x n)
// Colocar os valores da matriz em 'a' (pode < 0)
// assignment() retorna um par com o valor do
// assignment minimo, e a coluna escolhida por cada linha
//
// O(n^3)

template<typename T> struct hungarian {
        int n;
        vector<vector<T>> a;
        vector<T> u, v;
        vector<int> p, way;
        T inf;

        hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1), way(n+1) {
                a = vector<vector<T>>(n, vector<T>(n));
                inf = numeric_limits<T>::max();
        }
        pair<T, vector<int>> assignment() {
                for (int i = 1; i <= n; i++) {
                        p[0] = i;
                        int j0 = 0;
                        vector<T> minv(n+1, inf);
                        vector<int> used(n+1, 0);
                        do {
                                used[j0] = true;
                                int i0 = p[j0], j1 = -1;
                                T delta = inf;
                                for (int j = 1; j <= n; j++) if (!used[j]) {
                                        T cur = a[i0-1][j-1] - u[i0] - v[j];
                                        if (cur < minv[j]) minv[j] = cur, way[j]
                                             = j0;
                                        if (minv[j] < delta) delta = minv[j], j1
                                             = j;
                                }
                                for (int j = 0; j <= n; j++)
                                        if (used[j]) u[p[j]] += delta, v[j] -=
                                             delta;
                                        else minv[j] -= delta;
                                j0 = j1;
                        } while (p[j0] != 0);
                        do {
                                int j1 = way[j0];
                                p[j0] = p[j1];
                                j0 = j1;
                        } while (j0);
                }
                vector<int> ans(n);
                for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
                return make_pair(-v[0], ans);
        }
};
```

30.21 intervalGraphColoring

```
// Coloracao de Grafo de Intervalo
//
// Colore os intervalos com o numero minimo
// de cores de tal forma que dois intervalos
// que se interceptam tem cores diferentes
// As cores vao de 1 ate n
//
// O(n log(n))

vector<int> coloring(vector<pair<int, int>>& v) {
        int n = v.size();
        vector<pair<int, pair<int, int>>> ev;
        for (int i = 0; i < n; i++) {
                ev.push_back({v[i].first, {1, i}});
                ev.push_back({v[i].second, {0, i}});
        }
        sort(ev.begin(), ev.end());
        vector<int> ans(n), avl(n);
        for (int i = 0; i < n; i++) avl.push_back(n-i);
        for (auto i : ev) {
```

```
                if (i.second.first == 1) {
                        ans[i.second.second] = avl.back();
                        avl.pop_back();
                } else avl.push_back(ans[i.second.second]);
        }
        return ans;
}
```

30.22 intervalGraphIndSet

```
// Conj. Indep. Maximo com Peso em Grafo de Intervalo
//
// Retorna os indices ordenados dos intervalos selecionados
// Se tiver empate, retorna o que minimiza o comprimento total
//
// O(n log(n))

vector<int> ind_set(vector<tuple<int, int, int>>& v) {
        vector<tuple<int, int, int>> w;
        for (int i = 0; i < v.size(); i++) {
                w.push_back(tuple(get<0>(v[i]), 0, i));
                w.push_back(tuple(get<1>(v[i]), 1, i));
        }
        sort(w.begin(), w.end());

        vector<int> nxt(v.size());
        vector<pair<ll, int>> dp(v.size());
        int last = -1;
        for (auto [fim, t, i] : w) {
                if (t == 0) {
                        nxt[i] = last;
                        continue;
                }
                dp[i] = {0, 0};
                if (last != -1) dp[i] = max(dp[i], dp[last]);
                pair<ll, int> pega = {get<2>(v[i]), -(get<1>(v[i]) - get<0>(v[i
                     ]) + 1)};
                if (nxt[i] != -1) pega.first += dp[nxt[i]].first, pega.second +=
                      dp[nxt[i]].second;
                if (pega > dp[i]) dp[i] = pega;
                else nxt[i] = last;
                last = i;
        }
        pair<ll, int> ans = {0, 0};
        int idx = -1;
        for (int i = 0; i < v.size(); i++) if (dp[i] > ans) ans = dp[i], idx = i
             ;
        vector<int> ret;
        while (idx != -1) {
                if (get<2>(v[idx]) > 0 and
                        (nxt[idx] == -1 or get<1>(v[nxt[idx]]) < get<0>(v[idx]))
                             ) ret.push_back(idx);
                idx = nxt[idx];
        }
        sort(ret.begin(), ret.end());
        return ret;
}
```

30.23 inversionCount

```
// Inversion Count
//
// Computa o numero de inversoes para transformar
// l em r (se nao tem como, retorna -1)
//
// O(n log(n))

template<typename T> ll inv_count(vector<T> l, vector<T> r = {}) {
        if (!r.size()) {
                r = l;
                sort(r.begin(), r.end());
        }
        int n = l.size();
        vector<int> v(n), bit(n);
        vector<pair<T, int>> w;
        for (int i = 0; i < n; i++) w.push_back({r[i], i+1});
        sort(w.begin(), w.end());
        for (int i = 0; i < n; i++) {
                auto it = lower_bound(w.begin(), w.end(), make_pair(l[i], 0));
                if (it == w.end() or it->first != l[i]) return -1; // nao da
                v[i] = it->second;
                it->second = -1;
        }

        ll ans = 0;
        for (int i = n-1; i >= 0; i--) {
                for (int j = v[i]-1; j; j -= j&-j) ans += bit[j];
                for (int j = v[i]; j < n; j += j&-j) bit[j]++;
        }
        return ans;
}
```

30.24 lis

```
// LIS - recupera
//
// Calcula e retorna uma LIS
//
// O(n.log(n))

template<typename T> vector<T> lis(vector<T>& v) {
        int n = v.size(), m = -1;
        vector<T> d(n+1, INF);
        vector<int> l(n);
        d[0] = -INF;

        for (int i = 0; i < n; i++) {
                // Para non-decreasing use upper_bound()
                int t = lower_bound(d.begin(), d.end(), v[i]) - d.begin();
                d[t] = v[i], l[i] = t, m = max(m, t);
        }

        int p = n;
        vector<T> ret;
        while (p--) if (l[p] == m) {
                ret.push_back(v[p]);
                m--;
        }
        reverse(ret.begin(),ret.end());

        return ret;
}
```

30.25 lis2

```
// LIS - tamanho
//
// Calcula o tamanho da LIS
//
// O(n log(n))

template<typename T> int lis(vector<T> &v){
        vector<T> ans;
        for (T t : v){
                // Para non-decreasing use upper_bound()
                auto it = lower_bound(ans.begin(), ans.end(), t);
                if (it == ans.end()) ans.push_back(t);
                else *it = t;
        }
        return ans.size();
}
```

30.26 maxDist

```
// Distancia maxima entre dois pontos
//
// max_dist2(v) - O(n log(n))
// max_dist_manhattan - O(n)

// Quadrado da Distancia Euclidiana (precisa copiar convex_hull, ccw e pt)
ll max_dist2(vector<pt> v) {
        v = convex_hull(v);
        if (v.size() <= 2) return dist2(v[0], v[1%v.size()]);
        ll ans = 0;
        int n = v.size(), j = 0;
        for (int i = 0; i < n; i++) {
                while (!ccw(v[(i+1)%n]-v[i], pt(0, 0), v[(j+1)%n]-v[j])) j = (j
                    +1)%n;
                ans = max({ans, dist2(v[i], v[j]), dist2(v[(i+1)%n], v[j])});
        }
        return ans;
}

// Distancia de Manhattan
template<typename T> T max_dist_manhattan(vector<pair<T, T>> v) {
        T min_sum, max_sum, min_dif, max_dif;
        min_sum = max_sum = v[0].first + v[0].second;
        min_dif = max_dif = v[0].first - v[0].second;
        for (auto [x, y] : v) {
                min_sum = min(min_sum, x+y);
                max_sum = max(max_sum, x+y);
                min_dif = min(min_dif, x-y);
                max_dif = max(max_dif, x-y);
        }
        return max(max_sum - min_sum, max_dif - min_dif);
}
```

30.27 minCirc

```
// Minimum Enclosing Circle
//
// O(n) com alta probabilidade

const double EPS = 1e-12;
mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

struct pt {
        double x, y;
        pt(double x_ = 0, double y_ = 0) : x(x_), y(y_) {}
        pt operator + (const pt& p) const { return pt(x+p.x, y+p.y); }
        pt operator - (const pt& p) const { return pt(x-p.x, y-p.y); }
        pt operator * (double c) const { return pt(x*c, y*c); }
        pt operator / (double c) const { return pt(x/c, y/c); }
};

double dot(pt p, pt q) { return p.x*q.x+p.y*q.y; }
double cross(pt p, pt q) { return p.x*q.y-p.y*q.x; }
double dist(pt p, pt q) { return sqrt(dot(p-q, p-q)); }

pt center(pt p, pt q, pt r) {
        pt a = p-r, b = q-r;
        pt c = pt(dot(a, p+r)/2, dot(b, q+r)/2);
        return pt(cross(c, pt(a.y, b.y)), cross(pt(a.x, b.x), c)) / cross(a, b);
}

struct circle {
        pt cen;
        double r;
        circle(pt cen_, double r_) : cen(cen_), r(r_) {}
        circle(pt a, pt b, pt c) {
                cen = center(a, b, c);
                r = dist(cen, a);
        }
        bool inside(pt p) { return dist(p, cen) < r+EPS; }
};

circle minCirc(vector<pt> v) {
        shuffle(v.begin(), v.end(), rng);
        circle ret = circle(pt(0, 0), 0);
        for (int i = 0; i < v.size(); i++) if (!ret.inside(v[i])) {
                ret = circle(v[i], 0);
                for (int j = 0; j < i; j++) if (!ret.inside(v[j])) {
                        ret = circle((v[i]+v[j])/2, dist(v[i], v[j])/2);
                        for (int k = 0; k < j; k++) if (!ret.inside(v[k]))
                                ret = circle(v[i], v[j], v[k]);
                }
        }
        return ret;
}
```

30.28 minkowski

```
// Minkowski Sum
//
// Computa A+B = {a+b : a \in A, b \in B}, em que
// A e B sao poligonos convexos
// A+B eh um poligono convexo com no max |A|+|B| pontos
//
// O(|A|+|B|)

vector<pt> minkowski(vector<pt> p, vector<pt> q) {
        auto fix = [](vector<pt>& P) {
                rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
                P.push_back(P[0]), P.push_back(P[1]);
        };
        fix(p), fix(q);
        vector<pt> ret;
        int i = 0, j = 0;
        while (i < p.size()-2 or j < q.size()-2) {
                ret.push_back(p[i] + q[j]);
                auto c = ((p[i+1] - p[i]) ^ (q[j+1] - q[j]));
                if (c >= 0) i = min<int>(i+1, p.size()-2);
                if (c <= 0) j = min<int>(j+1, q.size()-2);
        }
        return ret;
}

ld dist_convex(vector<pt> p, vector<pt> q) {
        for (pt& i : p) i = i * -1;
        auto s = minkowski(p, q);
        if (inpol(s, pt(0, 0))) return 0;
        ld ans = DINF;
        for (int i = 0; i < s.size(); i++) ans = min(ans,
                        disttoseg(pt(0, 0), line(s[(i+1)%s.size()], s[i])));
        return ans;
}
```

30.29 mo

```
// MO
//
// Para ter o bound abaixo, escolher
// SQ = n / sqrt(q)
//
// O(n * sqrt(q))

const int MAX = 1e5+10;
const int SQ = sqrt(MAX);
int v[MAX];

int ans, freq[MAX];

inline void insert(int p) {
        int o = v[p];
        freq[o]++;
        ans += (freq[o] == 1);
}

inline void erase(int p) {
```

```
        int o = v[p];
        ans -= (freq[o] == 1);
        freq[o]--;
}

inline ll hilbert(int x, int y) {
        static int N = 1 << (__builtin_clz(0) - __builtin_clz(MAX));
        int rx, ry, s;
        ll d = 0;
        for (s = N/2; s > 0; s /= 2) {
                rx = (x & s) > 0, ry = (y & s) > 0;
                d += s * ll(s) * ((3 * rx) ^ ry);
                if (ry == 0) {
                        if (rx == 1) x = N-1 - x, y = N-1 - y;
                        swap(x, y);
                }
        }
        return d;
}

#define HILBERT true
vector<int> MO(vector<pair<int, int>> &q) {
        ans = 0;
        int m = q.size();
        vector<int> ord(m);
        iota(ord.begin(), ord.end(), 0);
#if HILBERT
        vector<ll> h(m);
        for (int i = 0; i < m; i++) h[i] = hilbert(q[i].first, q[i].second);
        sort(ord.begin(), ord.end(), [&](int l, int r) { return h[l] < h[r]; });
#else
        sort(ord.begin(), ord.end(), [&](int l, int r) {
                if (q[l].first / SQ != q[r].first / SQ) return q[l].first < q[r
                    ].first;
                if ((q[l].first / SQ) % 2) return q[l].second > q[r].second;
                return q[l].second < q[r].second;
        });
#endif
        vector<int> ret(m);
        int l = 0, r = -1;

        for (int i : ord) {
                int ql, qr;
                tie(ql, qr) = q[i];
                while (r < qr) insert(++r);
                while (l > ql) insert(--l);
                while (l < ql) erase(l++);
                while (r > qr) erase(r--);
                ret[i] = ans;
        }
        return ret;
}
```

30.30 moDsu

```
// MO - DSU
//
// Dado uma lista de arestas de um grafo, responde
// para cada query(l, r), quantos componentes conexos
// o grafo tem se soh considerar as arestas l, l+1, ..., r
// Da pra adaptar pra usar MO com qualquer estrutura rollbackavel
//
// O(m sqrt(q) log(n))

struct dsu {
        int n, ans;
        vector<int> p, sz;
        stack<int> S;

        dsu(int n_) : n(n_), ans(n), p(n), sz(n) {
                for (int i = 0; i < n; i++) p[i] = i, sz[i] = 1;
        }
        int find(int k) {
                while (p[k] != k) k = p[k];
                return k;
        }
        void add(pair<int, int> x) {
                int a = x.first, b = x.second;
                a = find(a), b = find(b);
                if (a == b) return S.push(-1);
                ans--;
                if (sz[a] > sz[b]) swap(a, b);
                S.push(a);
                sz[b] += sz[a];
                p[a] = b;
        }
        int query() { return ans; }
        void rollback() {
                int u = S.top(); S.pop();
                if (u == -1) return;
                sz[p[u]] -= sz[u];
                p[u] = u;
                ans++;
        }
};

int n;
vector<pair<int, int>> ar;

// 9d242b
vector<int> MO(vector<pair<int, int>> &q) {
        int SQ = sqrt(q.size()) + 1;
        int m = q.size();
        vector<int> ord(m);
        iota(ord.begin(), ord.end(), 0);
        sort(ord.begin(), ord.end(), [&](int l, int r) {
                        if (q[l].first / SQ != q[r].first / SQ) return q[l].
                            first < q[r].first;
                        return q[l].second < q[r].second;
                        });
        vector<int> ret(m);

        dsu small(n);
        for (int i = 0; i < m; i++) {
                auto [l, r] = q[ord[i]];
                if (l / SQ == r / SQ) {
                        for (int k = l; k <= r; k++) small.add(ar[k]);
                        ret[ord[i]] = small.query();
                        for (int k = l; k <= r; k++) small.rollback();
                }
        }

        for (int i = 0; i < m; i++) {
                dsu D(n);
                int fim = q[ord[i]].first/SQ*SQ + SQ - 1;
                int last_r = fim;
                int j = i-1;
                while (j+1 < m and q[ord[j+1]].first / SQ == q[ord[i]].first /
                     SQ) {
                        auto [l, r] = q[ord[++j]];

                        if (l / SQ == r / SQ) continue;

                        while (last_r < r) D.add(ar[++last_r]);
                        for (int k = l; k <= fim; k++) D.add(ar[k]);

                        ret[ord[j]] = D.query();

                        for (int k = l; k <= fim; k++) D.rollback();
                }
                i = j;
        }
        return ret;
}
```

30.31 moOnTrees

```
// MO em Arvores
//
// Problema que resolve: https://www.spoj.com/problems/COT2/
//
// Complexidade sendo c = O(update) e SQ = sqrt(n):
```

```cpp
// O((n + q) * sqrt(n) * c)

const int MAX = 40010, SQ = 400;

vector<int> g[MAX];

namespace LCA { ... }

int in[MAX], out[MAX], vtx[2 * MAX];
bool on[MAX];

int dif, freq[MAX];
vector<int> w;

void dfs(int v, int p, int &t) {
        vtx[t] = v, in[v] = t++;
        for (int u : g[v]) if (u != p) {
                dfs(u, v, t);
        }
        vtx[t] = v, out[v] = t++;
}

void update(int p) { // faca alteracoes aqui
        int v = vtx[p];
        if (not on[v]) { // insere vtx v
                dif += (freq[w[v]] == 0);
                freq[w[v]]++;
        }
        else { // retira o vertice v
                dif -= (freq[w[v]] == 1);
                freq[w[v]]--;
        }
        on[v] = not on[v];
}

vector<tuple<int, int, int>> build_queries(const vector<pair<int, int>>& q) {
        LCA::build(0);
        vector<tuple<int, int, int>> ret;
        for (auto [l, r] : q){
                if (in[r] < in[l]) swap(l, r);
                int p = LCA::lca(l, r);
                int init = (p == l) ? in[l] : out[l];
                ret.emplace_back(init, in[r], in[p]);
        }
        return ret;
}

vector<int> mo_tree(const vector<pair<int, int>>& vq){
        int t = 0;
        dfs(0, -1, t);

        auto q = build_queries(vq);

        vector<int> ord(q.size());
        iota(ord.begin(), ord.end(), 0);
        sort(ord.begin(), ord.end(), [&] (int l, int r) {
                int bl = get<0>(q[l]) / SQ, br = get<0>(q[r]) / SQ;
                if (bl != br) return bl < br;
                else if (bl % 2 == 1) return get<1>(q[l]) < get<1>(q[r]);
                else return get<1>(q[l]) > get<1>(q[r]);
        });

        memset(freq, 0, sizeof freq);
        dif = 0;

        vector<int> ret(q.size());
        int l = 0, r = -1;
        for (int i : ord) {
                auto [ql, qr, qp] = q[i];
                while (r < qr) update(++r);
                while (l > ql) update(--l);
                while (l < ql) update(l++);
                while (r > qr) update(r--);

                if (qp < l or qp > r) { // se LCA estah entre as pontas
                        update(qp);
                        ret[i] = dif;
                        update(qp);
                }
                else ret[i] = dif;
        }
        return ret;
}
```

30.32 palindromicFactorization

```cpp
// Palindromic Factorization
//
// Precisa da eertree
// Computa o numero de formas de particionar cada
// prefixo da string em strings palindromicas
//
// O(n log n), considerando alfabeto O(1)

struct eertree { ... };

ll factorization(string s) {
        int n = s.size(), sz = 2;
        eertree PT(n);
        vector<int> diff(n+2), slink(n+2), sans(n+2), dp(n+1);
        dp[0] = 1;
        for (int i = 1; i <= n; i++) {
                PT.add(s[i-1]);
                if (PT.size()+2 > sz) {
                        diff[sz] = PT.len[sz] - PT.len[PT.link[sz]];
                        if (diff[sz] == diff[PT.link[sz]])
                                slink[sz] = slink[PT.link[sz]];
                        else slink[sz] = PT.link[sz];
                        sz++;
                }
                for (int v = PT.last; PT.len[v] > 0; v = slink[v]) {
                        sans[v] = dp[i - (PT.len[slink[v]] + diff[v])];
                        if (diff[v] == diff[PT.link[v]])
                                sans[v] = (sans[v] + sans[PT.link[v]]) % MOD;
                        dp[i] = (dp[i] + sans[v]) % MOD;
                }
        }
        return dp[n];
}
```

30.33 parsing

```cpp
// Parsing de Expressao
//
// Operacoes associativas a esquerda por default
// Para mudar isso, colocar em r_assoc
// Operacoes com maior prioridade sao feitas primeiro

bool blank(char c) {
        return c == ' ';
}

bool is_unary(char c) {
        return c == '+' or c == '-';
}

bool is_op(char c) {
        if (is_unary(c)) return true;
        return c == '*' or c == '/' or c == '+' or c == '-';
}

bool r_assoc(char op) {
        // operator unario - deve ser assoc. a direita
        return op < 0;
}

int priority(char op) {
        // operator unario - deve ter precedencia maior
        if (op < 0) return INF;

        if (op == '*' or op == '/') return 2;
```

```
        if (op == '+' or op == '-') return 1;
        return -1;
}

void process_op(stack<int>& st, stack<int>& op) {
        char o = op.top(); op.pop();
        if (o < 0) {
                o *= -1;
                int l = st.top(); st.pop();
                if (o == '+') st.push(l);
                if (o == '-') st.push(-l);
        } else {
                int r = st.top(); st.pop();
                int l = st.top(); st.pop();
                if (o == '*') st.push(l * r);
                if (o == '/') st.push(l / r);
                if (o == '+') st.push(l + r);
                if (o == '-') st.push(l - r);
        }
}

int eval(string& s) {
        stack<int> st, op;
        bool un = true;
        for (int i = 0; i < s.size(); i++) {
                if (blank(s[i])) continue;

                if (s[i] == '(') {
                        op.push('(');
                        un = true;
                } else if (s[i] == ')') {
                        while (op.top() != '(') process_op(st, op);
                        op.pop();
                        un = false;
                } else if (is_op(s[i])) {
                        char o = s[i];
                        if (un and is_unary(o)) o *= -1;
                        while (op.size() and (
                                                (!r_assoc(o) and priority(op.top
                                                        ()) >= priority(o)) or
                                                (r_assoc(o) and priority(op.top
                                                        ()) > priority(o))))
                                process_op(st, op);
                        op.push(o);
                        un = true;
                } else {
                        int val = 0;
                        while (i < s.size() and isalnum(s[i]))
                                val = val * 10 + s[i++] - '0';
                        i--;
                        st.push(val);
                        un = false;
                }
        }

        while (op.size()) process_op(st, op);
        return st.top();
}
```

30.34 rmqOffline

```
// RMQ com Divide and Conquer
//
// Responde todas as queries em
// O(n log(n))

typedef pair<pair<int, int>, int> iii;
#define f first
#define s second

int n, q, v[MAX];
iii qu[MAX];
int ans[MAX], pref[MAX], sulf[MAX];

void solve(int l=0, int r=n-1, int ql=0, int qr=q-1) {
        if (l > r or ql > qr) return;
        int m = (l+r)/2;
        int qL = partition(qu+ql, qu+qr+1, [=](iii x){return x.f.s < m;}) - qu;
        int qR = partition(qu+qL, qu+qr+1, [=](iii x){return x.f.f <=m;}) - qu;

        pref[m] = sulf[m] = v[m];
        for (int i = m-1; i >= l; i--) pref[i] = min(v[i], pref[i+1]);
        for (int i = m+1; i <= r; i++) sulf[i] = min(v[i], sulf[i-1]);

        for (int i = qL; i < qR; i++)
                ans[qu[i].s] = min(pref[qu[i].f.f], sulf[qu[i].f.s]);

        solve(l, m-1, ql, qL-1), solve(m+1, r, qR, qr);
}
```

30.35 segmentIntersection

```
// Segment Intersection
//
// Verifica, dado n segmentos, se existe algum par de segmentos
// que se intersecta
//
// O(n log n)

bool operator < (const line& a, const line& b) { // comparador pro sweepline
        if (a.p == b.p) return ccw(a.p, a.q, b.q);
        if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps < b.p.x))
                return ccw(a.p, a.q, b.p);
        return ccw(a.p, b.q, b.p);
}

bool has_intersection(vector<line> v) {
        auto intersects = [&](pair<line, int> a, pair<line, int> b) {
                return interseg(a.first, b.first);
        };
        vector<pair<pt, pair<int, int>>> w;
        for (int i = 0; i < v.size(); i++) {
                if (v[i].q < v[i].p) swap(v[i].p, v[i].q);
                w.push_back({v[i].p, {0, i}});
                w.push_back({v[i].q, {1, i}});
        }
        sort(w.begin(), w.end());
        set<pair<line, int>> se;
        for (auto i : w) {
                line at = v[i.second.second];
                if (i.second.first == 0) {
                        auto nxt = se.lower_bound({at, i.second.second});
                        if (nxt != se.end() and intersects(*nxt, {at, i.second.
                            second})) return 1;
                        if (nxt != se.begin() and intersects(*(--nxt), {at, i.
                            second.second})) return 1;
                        se.insert({at, i.second.second});
                } else {
                        auto nxt = se.upper_bound({at, i.second.second}), cur =
                            nxt, prev = --cur;
                        if (nxt != se.end() and prev != se.begin()
                                        and intersects(*nxt, *(--prev))) return 1;
                        se.erase(cur);
                }
        }
        return 0;
}
```

30.36 simplePolygon

```
// Simple Polygon
//
// Verifica se um poligono com n pontos eh simples
//
// O(n log n)

bool operator < (const line& a, const line& b) { // comparador pro sweepline
```

```
        if (a.p == b.p) return ccw(a.p, a.q, b.q);
        if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps < b.p.x))
                return ccw(a.p, a.q, b.p);
        return ccw(a.p, b.q, b.p);
}

bool simple(vector<pt> v) {
        auto intersects = [&](pair<line, int> a, pair<line, int> b) {
                if ((a.second+1)%v.size() == b.second or
                        (b.second+1)%v.size() == a.second) return false;
                return interseg(a.first, b.first);
        };
        vector<line> seg;
        vector<pair<pt, pair<int, int>>> w;
        for (int i = 0; i < v.size(); i++) {
                pt at = v[i], nxt = v[(i+1)%v.size()];
                if (nxt < at) swap(at, nxt);
                seg.push_back(line(at, nxt));
                w.push_back({at, {0, i}});
                w.push_back({nxt, {1, i}});
                // casos degenerados estranhos
                if (isinseg(v[(i+2)%v.size()], line(at, nxt))) return 0;
                if (isinseg(v[(i+v.size()-1)%v.size()], line(at, nxt))) return
                    0;
        }
        sort(w.begin(), w.end());
        set<pair<line, int>> se;
        for (auto i : w) {
                line at = seg[i.second.second];
                if (i.second.first == 0) {
                        auto nxt = se.lower_bound({at, i.second.second});
                        if (nxt != se.end() and intersects(*nxt, {at, i.second.
                            second})) return 0;
                        if (nxt != se.begin() and intersects(*(--nxt), {at, i.
                            second.second})) return 0;
                        se.insert({at, i.second.second});
                } else {
                        auto nxt = se.upper_bound({at, i.second.second}), cur =
                            nxt, prev = --cur;
                        if (nxt != se.end() and prev != se.begin()
                                and intersects(*nxt, *(--prev))) return 0;
                        se.erase(cur);
                }
        }
        return 1;
}
```

30.37 steinerTree

```
// Steiner Tree
//
// steiner: retorna o peso da menor arvore que cobre os vertices S
// get_steiner: retorna o valor minimo e as arestas de uma solucao
// se nao tiver solucao retorna LINF
//
// grafo nao pode ter pesos negativos
// se so tiver peso nas arestas/vertices pode deletar os vw/w no codigo
//
// k = |S|
// O(3^k * n + 2^k * m log m)

// otimizacao: joga um vertice x do S fora e pegue a resposta em dp[...][x] e
    reconstrua a arvore a partir dele
// ta comentado no codigo as mudancas necessarias

int n; // numero de vertices
vector<pair<int, int>> g[MAX]; // {vizinho, peso}
ll d[1 << K][MAX]; // dp[mask][v] = arvore minima com o subconjunto mask de S e
    o vertice v
ll vw[MAX]; // peso do vertice

ll steiner(const vector<int> &S) {
        int k = S.size(); // k--;
        for (int mask = 0; mask < (1 << k); mask++) for(int v = 0; v < n; v++) d
            [mask][v] = LINF;
        for (int v = 0; v < n; v++) d[0][v] = vw[v];
        for (int i = 0; i < k; ++i) d[1 << i][S[i]] = vw[S[i]];
        for (int mask = 1; mask < (1 << k); mask++) {
                for (int a = (mask - 1) & mask; a; a = (a - 1) & mask) {
                        int b = mask ^ a;
                        if (b > a) break;
                        for (int v = 0; v < n; v++)
                                d[mask][v] = min(d[mask][v], d[a][v] + d[b][v] -
                                        vw[v]);
                }
                priority_queue<pair<ll, int>> pq;
                for (int v = 0; v < n; v++) {
                        if (d[mask][v] == LINF) continue;
                        pq.emplace(-d[mask][v], v);
                }
                while (pq.size()) {
                        auto [ndist, u] = pq.top(); pq.pop();
                        if (-ndist > d[mask][u]) continue;
                        for (auto [idx, w] : g[u]) if (d[mask][idx] > d[mask][u]
                              + w + vw[idx]) {
                                d[mask][idx] = d[mask][u] + w + vw[idx];
                                pq.emplace(-d[mask][idx], idx);
                        }
                }
        }
        return d[(1 << k) - 1][S[0]]; // S[k]
}

#warning se k=1 a solucao eh a folha isolada e a funcao retorna edg = {}
#warning se k=0 crasha
pair<ll,vector<pair<int,int>>> get_steiner(const vector<int> &S) {
        int k = S.size(); // k--;
        ll ans = steiner(S);
        vector<pair<int,int>> edg;
        stack<pair<int,int>> stk;
        stk.emplace((1 << k) - 1, S[0]); // S[k]
        while (!stk.empty()) {
                bool cont = 0;
                auto [mask,u] = stk.top();stk.pop();
                if ((__builtin_popcount(mask) == 1 and u == S[__bit_width(mask)
                    - 1])) continue;
                for (auto [idx, w] : g[u]){
                        if (d[mask][u] == d[mask][idx] + w + vw[u]) {
                                edg.emplace_back(u, idx);
                                stk.emplace(mask, idx);
                                cont = true;
                                break;
                        }
                }
                if (cont) continue;
                for (int a = (mask - 1) & mask; a; a = (a - 1) & mask) {
                        int b = mask ^ a;
                        if (d[mask][u] == d[a][u] + d[b][u] - vw[u]) {
                                stk.emplace(a, u);
                                stk.emplace(b, u);
                                cont = true;
                                break;
                        }
                }
                assert(!mask || cont);
        }
        return {ans, edg};
}
```

30.38 sweepDirection

```
// Sweep Direction
//
// Passa por todas as ordenacoes dos pontos definitas por "direcoes"
// Assume que nao existem pontos coincidentes
//
// O(n^2 log n)

void sweep_direction(vector<pt> v) {
        int n = v.size();
```

```
        sort(v.begin(), v.end(), [](pt a, pt b) {
                if (a.x != b.x) return a.x < b.x;
                return a.y > b.y;
        });
        vector<int> at(n);
        iota(at.begin(), at.end(), 0);
        vector<pair<int, int>> swapp;
        for (int i = 0; i < n; i++) for (int j = i+1; j < n; j++)
                swapp.push_back({i, j}), swapp.push_back({j, i});

        sort(swapp.begin(), swapp.end(), [&](auto a, auto b) {
                pt A = rotate90(v[a.first] - v[a.second]);
                pt B = rotate90(v[b.first] - v[b.second]);
                if (quad(A) == quad(B) and !sarea2(pt(0, 0), A, B)) return a < b
                        ;
                return compare_angle(A, B);
        });
        for (auto par : swapp) {
                assert(abs(at[par.first] - at[par.second]) == 1);
                int l = min(at[par.first], at[par.second]),
                        r = n-1 - max(at[par.first], at[par.second]);
                // l e r sao quantos caras tem de cada lado do par de pontos
                // (cada par eh visitado duas vezes)
                swap(v[at[par.first]], v[at[par.second]]);
                swap(at[par.first], at[par.second]);
        }
}
```

31 ufmg forked/Strings

31.1 ahocorasick

```
// Aho-corasick
//
// query retorna o somatorio do numero de matches de
// todas as stringuinhas na stringona
//
// insert - O(|s| log(SIGMA))
// build - O(N), onde N = somatorio dos tamanhos das strings
// query - O(|s|)

namespace aho {
        map<char, int> to[MAX];
        int link[MAX], idx, term[MAX], exit[MAX], sobe[MAX];

        void insert(string& s) {
                int at = 0;
                for (char c : s) {
                        auto it = to[at].find(c);
                        if (it == to[at].end()) at = to[at][c] = ++idx;
                        else at = it->second;
                }
                term[at]++, sobe[at]++;
        }
#warning nao esquece de chamar build() depois de inserir
        void build() {
                queue<int> q;
                q.push(0);
                link[0] = exit[0] = -1;
                while (q.size()) {
                        int i = q.front(); q.pop();
                        for (auto [c, j] : to[i]) {
                                int l = link[i];
                                while (l != -1 and !to[l].count(c)) l = link[l];
                                link[j] = l == -1 ? 0 : to[l][c];
                                exit[j] = term[link[j]] ? link[j] : exit[link[j
                                    ]];
                                if (exit[j]+1) sobe[j] += sobe[exit[j]];
                                q.push(j);
                        }
                }
        }
        int query(string& s) {
                int at = 0, ans = 0;
                for (char c : s){
                        while (at != -1 and !to[at].count(c)) at = link[at];
                        at = at == -1 ? 0 : to[at][c];
                        ans += sobe[at];
                }
                return ans;
        }
}
```

31.2 dynamicSuffixArray

```
// Suffix Array Dinamico
//
// Mantem o suffix array, lcp e rank de uma string,
// premitindo push_front e pop_front
// O operador [i] return um par com sa[i] e lcp[i]
// lcp[i] tem o lcp entre sa[i] e sa[i-1] (lcp[0] = 0)
//
// Complexidades:
// Construir sobre uma string de tamanho n: O(n log n)
// push_front e pop_front: O(log n) amortizado

struct dyn_sa {
        struct node {
                int sa, lcp;
                node *l, *r, *p;
                int sz, mi;
                node(int sa_, int lcp_, node* p_) : sa(sa_), lcp(lcp_),
                        l(NULL), r(NULL), p(p_), sz(1), mi(lcp) {}
                void update() {
                        sz = 1, mi = lcp;
                        if (l) sz += l->sz, mi = min(mi, l->mi);
                        if (r) sz += r->sz, mi = min(mi, r->mi);
                }
        };

        node* root;
        vector<ll> tag; // tag of a suffix (reversed id)
        string s; // reversed

        dyn_sa() : root(NULL) {}
        dyn_sa(string s_) : dyn_sa() {
                reverse(s_.begin(), s_.end());
                for (char c : s_) push_front(c);
        }
        ~dyn_sa() {
                vector<node*> q = {root};
                while (q.size()) {
                        node* x = q.back(); q.pop_back();
                        if (!x) continue;
                        q.push_back(x->l), q.push_back(x->r);
                        delete x;
                }
        }

        int size(node* x) { return x ? x->sz : 0; }
        int mirror(int i) { return s.size()-1 - i; }
        bool cmp(int i, int j) {
                if (s[i] != s[j]) return s[i] < s[j];
                if (i == 0 or j == 0) return i < j;
                return tag[i-1] < tag[j-1];
        }
        void fix_path(node* x) { while (x) x->update(), x = x->p; }
        void flatten(vector<node*>& v, node* x) {
                if (!x) return;
                flatten(v, x->l);
                v.push_back(x);
                flatten(v, x->r);
        }
        void build(vector<node*>& v, node*& x, node* p, int L, int R, ll l, ll r
             ) {
                if (L > R) return void(x = NULL);
                int M = (L+R)/2;
                ll m = (l+r)/2;
```

```
        x = v[M];
        x->p = p;
        tag[x->sa] = m;
        build(v, x->l, x, L, M-1, l, m-1), build(v, x->r, x, M+1, R, m
             +1, r);
        x->update();
}
void fix(node*& x, node* p, ll l, ll r) {
        if (3*max(size(x->l), size(x->r)) <= 2*size(x)) return x->update
             ();
        vector<node*> v;
        flatten(v, x);
        build(v, x, p, 0, v.size()-1, l, r);
}
node* next(node* x) {
        if (x->r) {
                x = x->r;
                while (x->l) x = x->l;
                return x;
        }
        while (x->p and x->p->r == x) x = x->p;
        return x->p;
}
node* prev(node* x) {
        if (x->l) {
                x = x->l;
                while (x->r) x = x->r;
                return x;
        }
        while (x->p and x->p->l == x) x = x->p;
        return x->p;
}

int get_lcp(node* x, node* y) {
        if (!x or !y) return 0; // change defaut value here
        if (s[x->sa] != s[y->sa]) return 0;
        if (x->sa == 0 or y->sa == 0) return 1;
        return 1 + query(mirror(x->sa-1), mirror(y->sa-1));
}
void add_suf(node*& x, node* p, int id, ll l, ll r) {
        if (!x) {
                x = new node(id, 0, p);
                node *prv = prev(x), *nxt = next(x);
                int lcp_cur = get_lcp(prv, x), lcp_nxt = get_lcp(x, nxt)
                     ;
                if (nxt) nxt->lcp = lcp_nxt, fix_path(nxt);
                x->lcp = lcp_cur;
                tag[id] = (l+r)/2;
                x->update();
                return;
        }
        if (cmp(id, x->sa)) add_suf(x->l, x, id, l, tag[x->sa]-1);
        else add_suf(x->r, x, id, tag[x->sa]+1, r);
        fix(x, p, l, r);
}
void push_front(char c) {
        s += c;
        tag.push_back(-1);
        add_suf(root, NULL, s.size() - 1, 0, 1e18);
}

void rem_suf(node*& x, int id) {
        if (x->sa != id) {
                if (tag[id] < tag[x->sa]) return rem_suf(x->l, id);
                return rem_suf(x->r, id);
        }
        node* nxt = next(x);
        if (nxt) nxt->lcp = min(nxt->lcp, x->lcp), fix_path(nxt);

        node *p = x->p, *tmp = x;
        if (!x->l or !x->r) {
                x = x->l ? x->l : x->r;
                if (x) x->p = p;
        } else {
                for (tmp = x->l, p = x; tmp->r; tmp = tmp->r) p = tmp;
                x->sa = tmp->sa, x->lcp = tmp->lcp;
                if (tmp->l) tmp->l->p = p;
                if (p->l == tmp) p->l = tmp->l;
                else p->r = tmp->l;
                }
                fix_path(p);
                delete tmp;
        }
        void pop_front() {
                if (!s.size()) return;
                s.pop_back();
                rem_suf(root, s.size());
                tag.pop_back();
        }

        int query(node* x, ll l, ll r, ll a, ll b) {
                if (!x or tag[x->sa] == -1 or r < a or b < l) return s.size();
                if (a <= l and r <= b) return x->mi;
                int ans = s.size();
                if (a <= tag[x->sa] and tag[x->sa] <= b) ans = min(ans, x->lcp);
                ans = min(ans, query(x->l, l, tag[x->sa]-1, a, b));
                ans = min(ans, query(x->r, tag[x->sa]+1, r, a, b));
                return ans;
        }
        int query(int i, int j) { // lcp(s[i..], s[j..])
                if (i == j) return s.size() - i;
                ll a = tag[mirror(i)], b = tag[mirror(j)];
                int ret = query(root, 0, 1e18, min(a, b)+1, max(a, b));
                return ret;
        }
        // optional: get rank[i], sa[i] and lcp[i]
        int rank(int i) {
                i = mirror(i);
                node* x = root;
                int ret = 0;
                while (x) {
                        if (tag[x->sa] < tag[i]) {
                                ret += size(x->l)+1;
                                x = x->r;
                        } else x = x->l;
                }
                return ret;
        }
        pair<int, int> operator[](int i) {
                node* x = root;
                while (1) {
                        if (i < size(x->l)) x = x->l;
                        else {
                                i -= size(x->l);
                                if (!i) return {mirror(x->sa), x->lcp};
                                i--, x = x->r;
                        }
                }
        }
};
```

31.3 eertree

```
// eertree
//
// Constroi a eertree, caractere a caractere
// Inicializar com a quantidade de caracteres maxima
// size() retorna a quantidade de substrings pal. distintas
// depois de chamar propagate(), cada substring palindromica
// ocorre qt[i] vezes. O propagate() retorna o numero de
// substrings pal. com repeticao
//
// O(n) amortizado, considerando alfabeto O(1)

struct eertree {
        vector<vector<int>> t;
        int n, last, sz;
        vector<int> s, len, link, qt;

        eertree(int N) {
                t = vector(N+2, vector(26, int()));
                s = len = link = qt = vector<int>(N+2);
                s[0] = -1;
                link[0] = 1, len[0] = 0, link[1] = 1, len[1] = -1;
                sz = 2, last = 0, n = 1;
```

```
        }

        void add(char c) {
                s[n++] = c -= 'a';
                while (s[n-len[last]-2] != c) last = link[last];
                if (!t[last][c]) {
                        int prev = link[last];
                        while (s[n-len[prev]-2] != c) prev = link[prev];
                        link[sz] = t[prev][c];
                        len[sz] = len[last]+2;
                        t[last][c] = sz++;
                }
                qt[last = t[last][c]]++;
        }
        int size() { return sz-2; }
        ll propagate() {
                ll ret = 0;
                for (int i = n; i > 1; i--) {
                        qt[link[i]] += qt[i];
                        ret += qt[i];
                }
                return ret;
        }
};
```

31.4 hashing

```
// String Hashing
//
// Complexidades:
// construtor - O(|s|)
// operator() - O(1)

mt19937 rng((int) chrono::steady_clock::now().time_since_epoch().count());

int uniform(int l, int r) {
        uniform_int_distribution<int> uid(l, r);
        return uid(rng);
}

template<int MOD> struct str_hash { // 116fcb
        static int P;
        vector<ll> h, p;
        str_hash(string s) : h(s.size()), p(s.size()) {
                p[0] = 1, h[0] = s[0];
                for (int i = 1; i < s.size(); i++)
                        p[i] = p[i - 1]*P%MOD, h[i] = (h[i - 1]*P + s[i])%MOD;
        }
        ll operator()(int l, int r) { // retorna hash s[l...r]
                ll hash = h[r] - (l ? h[l - 1]*p[r - l + 1]%MOD : 0);
                return hash < 0 ? hash + MOD : hash;
        }
};
template<int MOD> int str_hash<MOD>::P = uniform(256, MOD - 1); // l > |sigma|
```

31.5 hashingLargeMod

```
// String Hashing - modulo 2^61 - 1
//
// Quase duas vezes mais lento
//
// Complexidades:
// build - O(|s|)
// operator() - O(1)

const ll MOD = (1ll<<61) - 1;
ll mulmod(ll a, ll b) {
        const static ll LOWER = (1ll<<30) - 1, GET31 = (1ll<<31) - 1;
        ll l1 = a&LOWER, h1 = a>>30, l2 = b&LOWER, h2 = b>>30;
        ll m = l1*h2 + l2*h1, h = h1*h2;
        ll ans = l1*l2 + (h>>1) + ((h&1)<<60) + (m>>31) + ((m&GET31)<<30) + 1;
        ans = (ans&MOD) + (ans>>61), ans = (ans&MOD) + (ans>>61);
        return ans - 1;
}

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

ll uniform(ll l, ll r) {
        uniform_int_distribution<ll> uid(l, r);
        return uid(rng);
}

struct str_hash {
        static ll P;
        vector<ll> h, p;
        str_hash(string s) : h(s.size()), p(s.size()) {
                p[0] = 1, h[0] = s[0];
                for (int i = 1; i < s.size(); i++)
                        p[i] = mulmod(p[i - 1], P), h[i] = (mulmod(h[i - 1], P)
                             + s[i])%MOD;
        }
        ll operator()(int l, int r) { // retorna hash s[l...r]
                ll hash = h[r] - (l ? mulmod(h[l - 1], p[r - l + 1]) : 0);
                return hash < 0 ? hash + MOD : hash;
        }
};
ll str_hash::P = uniform(256, MOD - 1); // l > |sigma|
```

31.6 kmp

```
// KMP
//
// matching(s, t) retorna os indices das ocorrencias
// de s em t
// autKMP constroi o automato do KMP
//
// Complexidades:
// pi - O(n)
// match - O(n + m)
// construir o automato - O(|sigma|*n)
// n = |padrao| e m = |texto|

template<typename T> vector<int> pi(T s) {
        vector<int> p(s.size());
        for (int i = 1, j = 0; i < s.size(); i++) {
                while (j and s[j] != s[i]) j = p[j-1];
                if (s[j] == s[i]) j++;
                p[i] = j;
        }
        return p;
}

template<typename T> vector<int> matching(T& s, T& t) {
        vector<int> p = pi(s), match;
        for (int i = 0, j = 0; i < t.size(); i++) {
                while (j and s[j] != t[i]) j = p[j-1];
                if (s[j] == t[i]) j++;
                if (j == s.size()) match.push_back(i-j+1), j = p[j-1];
        }
        return match;
}

struct KMPaut : vector<vector<int>> {
        KMPaut(){}
        KMPaut (string& s) : vector<vector<int>>(26, vector<int>(s.size()+1)) {
                vector<int> p = pi(s);
                auto& aut = *this;
                aut[s[0]-'a'][0] = 1;
                for (char c = 0; c < 26; c++)
                        for (int i = 1; i <= s.size(); i++)
                                aut[c][i] = s[i]-'a' == c ? i+1 : aut[c][p[i
                                    -1]];
        }
};
```

31.7 manacher

```
// Manacher
//
// manacher recebe um vetor de T e retorna o vetor com tamanho dos palindromos
// ret[2*i] = tamanho do maior palindromo centrado em i
// ret[2*i+1] = tamanho maior palindromo centrado em i e i+1
//
// Complexidades:
// manacher - O(n)
// palindrome - <O(n), O(1)>
// pal_end - O(n)

template<typename T> vector<int> manacher(const T& s) {
        int l = 0, r = -1, n = s.size();
        vector<int> d1(n), d2(n);
        for (int i = 0; i < n; i++) {
                int k = i > r ? 1 : min(d1[l+r-i], r-i);
                while (i+k < n && i-k >= 0 && s[i+k] == s[i-k]) k++;
                d1[i] = k--;
                if (i+k > r) l = i-k, r = i+k;
        }
        l = 0, r = -1;
        for (int i = 0; i < n; i++) {
                int k = i > r ? 0 : min(d2[l+r-i+1], r-i+1); k++;
                while (i+k <= n && i-k >= 0 && s[i+k-1] == s[i-k]) k++;
                d2[i] = --k;
                if (i+k-1 > r) l = i-k, r = i+k-1;
        }
        vector<int> ret(2*n-1);
        for (int i = 0; i < n; i++) ret[2*i] = 2*d1[i]-1;
        for (int i = 0; i < n-1; i++) ret[2*i+1] = 2*d2[i+1];
        return ret;
}

// verifica se a string s[i..j] eh palindromo
template<typename T> struct palindrome {
        vector<int> man;

        palindrome(const T& s) : man(manacher(s)) {}
        bool query(int i, int j) {
                return man[i+j] >= j-i+1;
        }
};

// tamanho do maior palindromo que termina em cada posicao
template<typename T> vector<int> pal_end(const T& s) {
        vector<int> ret(s.size());
        palindrome<T> p(s);
        ret[0] = 1;
        for (int i = 1; i < s.size(); i++) {
                ret[i] = min(ret[i-1]+2, i+1);
                while (!p.query(i-ret[i]+1, i)) ret[i]--;
        }
        return ret;
}
```

31.8 minMaxSuffixCyclic

```
// Min/max suffix/cyclic shift
//
// Computa o indice do menor/maior sufixo/cyclic shift
// da string, lexicograficamente
//
// O(n)

template<typename T> int max_suffix(T s, bool mi = false) {
        s.push_back(*min_element(s.begin(), s.end())-1);
        int ans = 0;
        for (int i = 1; i < s.size(); i++) {
                int j = 0;
                while (ans+j < i and s[i+j] == s[ans+j]) j++;
                if (s[i+j] > s[ans+j]) {
                        if (!mi or i != s.size()-2) ans = i;
                } else if (j) i += j-1;
        }
        return ans;
}

template<typename T> int min_suffix(T s) {
        for (auto& i : s) i *= -1;
        s.push_back(*max_element(s.begin(), s.end())+1);
        return max_suffix(s, true);
}

template<typename T> int max_cyclic_shift(T s) {
        int n = s.size();
        for (int i = 0; i < n; i++) s.push_back(s[i]);
        return max_suffix(s);
}

template<typename T> int min_cyclic_shift(T s) {
        for (auto& i : s) i *= -1;
        return max_cyclic_shift(s);
}
```

31.9 suffixArray

```
// Suffix Array - O(n)
//
// Rapidao
// Computa o suffix array em 'sa', o rank em 'rnk'
// e o lcp em 'lcp'
// query(i, j) retorna o LCP entre s[i..n-1] e s[j..n-1]
//
// Complexidades
// O(n) para construir
// query - O(1)

template<typename T> struct rmq {
        vector<T> v;
        int n; static const int b = 30;
        vector<int> mask, t;

        int op(int x, int y) { return v[x] <= v[y] ? x : y; }
        int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
        int small(int r, int sz = b) { return r-msb(mask[r]&((1<<sz)-1)); }
        rmq() {}
        rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
                for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
                        at = (at<<1)&((1<<b)-1);
                        while (at and op(i-msb(at&-at), i) == i) at ^= at&-at;
                }
                for (int i = 0; i < n/b; i++) t[i] = small(b*i+b-1);
                for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0; i+(1<<j) <=
                     n/b; i++)
                        t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j-1)+i+(1<<(j-1))
                             ]);
        }
        int index_query(int l, int r) {
                if (r-l+1 <= b) return small(r, r-l+1);
                int x = l/b+1, y = r/b-1;
                if (x > y) return op(small(l+b-1), small(r));
                int j = msb(y-x+1);
                int ans = op(small(l+b-1), op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
                return op(ans, small(r));
        }
        T query(int l, int r) { return v[index_query(l, r)]; }
};

struct suffix_array {
        string s;
        int n;
        vector<int> sa, cnt, rnk, lcp;
        rmq<int> RMQ;

        bool cmp(int a1, int b1, int a2, int b2, int a3=0, int b3=0) {
                return a1 != b1 ? a1 < b1 : (a2 != b2 ? a2 < b2 : a3 < b3);
```

```
}
template<typename T> void radix(int* fr, int* to, T* r, int N, int k) {
        cnt = vector<int>(k+1, 0);
        for (int i = 0; i < N; i++) cnt[r[fr[i]]]++;
        for (int i = 1; i <= k; i++) cnt[i] += cnt[i-1];
        for (int i = N-1; i+1; i--) to[--cnt[r[fr[i]]]] = fr[i];
}
void rec(vector<int>& v, int k) {
        auto &tmp = rnk, &m0 = lcp;
        int N = v.size()-3, sz = (N+2)/3, sz2 = sz+N/3;
        vector<int> R(sz2+3);
        for (int i = 1, j = 0; j < sz2; i += i%3) R[j++] = i;

        radix(&R[0], &tmp[0], &v[0]+2, sz2, k);
        radix(&tmp[0], &R[0], &v[0]+1, sz2, k);
        radix(&R[0], &tmp[0], &v[0]+0, sz2, k);

        int dif = 0;
        int l0 = -1, l1 = -1, l2 = -1;
        for (int i = 0; i < sz2; i++) {
                if (v[tmp[i]] != l0 or v[tmp[i]+1] != l1 or v[tmp[i]+2]
                    != l2)
                        l0 = v[tmp[i]], l1 = v[tmp[i]+1], l2 = v[tmp[i
                            ]+2], dif++;
                if (tmp[i]%3 == 1) R[tmp[i]/3] = dif;
                else R[tmp[i]/3+sz] = dif;
        }

        if (dif < sz2) {
                rec(R, dif);
                for (int i = 0; i < sz2; i++) R[sa[i]] = i+1;
        } else for (int i = 0; i < sz2; i++) sa[R[i]-1] = i;

        for (int i = 0, j = 0; j < sz2; i++) if (sa[i] < sz) tmp[j++] =
             3*sa[i];
        radix(&tmp[0], &m0[0], &v[0], sz, k);
        for (int i = 0; i < sz2; i++)
                sa[i] = sa[i] < sz ? 3*sa[i]+1 : 3*(sa[i]-sz)+2;

        int at = sz2+sz-1, p = sz-1, p2 = sz2-1;
        while (p >= 0 and p2 >= 0) {
                if ((sa[p2]%3==1 and cmp(v[m0[p]], v[sa[p2]], R[m0[p
                    ]/3],
                        R[sa[p2]/3+sz])) or (sa[p2]%3==2 and cmp(v[m0[p
                            ]], v[sa[p2]],
                        v[m0[p]+1], v[sa[p2]+1], R[m0[p]/3+sz], R[sa[p2
                            ]/3+1])))
                        sa[at--] = sa[p2--];
                else sa[at--] = m0[p--];
        }
        while (p >= 0) sa[at--] = m0[p--];
        if (N%3==1) for (int i = 0; i < N; i++) sa[i] = sa[i+1];
}

suffix_array(const string& s_) : s(s_), n(s.size()), sa(n+3),
                cnt(n+1), rnk(n), lcp(n-1) {
        vector<int> v(n+3);
        for (int i = 0; i < n; i++) v[i] = i;
        radix(&v[0], &rnk[0], &s[0], n, 256);
        int dif = 1;
        for (int i = 0; i < n; i++)
                v[rnk[i]] = dif += (i and s[rnk[i]] != s[rnk[i-1]]);
        if (n >= 2) rec(v, dif);
        sa.resize(n);

        for (int i = 0; i < n; i++) rnk[sa[i]] = i;
        for (int i = 0, k = 0; i < n; i++, k -= !!k) {
                if (rnk[i] == n-1) {
                        k = 0;
                        continue;
                }
                int j = sa[rnk[i]+1];
                while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
                lcp[rnk[i]] = k;
        }
        RMQ = rmq<int>(lcp);
}

int query(int i, int j) {
        if (i == j) return n-i;
        i = rnk[i], j = rnk[j];
        return RMQ.query(min(i, j), max(i, j)-1);
}
pair<int, int> next(int L, int R, int i, char c) {
        int l = L, r = R+1;
        while (l < r) {
                int m = (l+r)/2;
                if (i+sa[m] >= n or s[i+sa[m]] < c) l = m+1;
                else r = m;
        }
        if (l == R+1 or s[i+sa[l]] > c) return {-1, -1};
        L = l;

        l = L, r = R+1;
        while (l < r) {
                int m = (l+r)/2;
                if (i+sa[m] >= n or s[i+sa[m]] <= c) l = m+1;
                else r = m;
        }
        R = l-1;
        return {L, R};
}
// quantas vezes 't' ocorre em 's' - O(|t| log n)
int count_substr(string& t) {
        int L = 0, R = n-1;
        for (int i = 0; i < t.size(); i++) {
                tie(L, R) = next(L, R, i, t[i]);
                if (L == -1) return 0;
        }
        return R-L+1;
}

// exemplo de f que resolve o problema
// https://codeforces.com/edu/course/2/lesson/2/5/practice/contest
    /269656/problem/D
ll f(ll k) { return k*(k+1)/2; }

ll dfs(int L, int R, int p) { // dfs na suffix tree chamado em pre ordem
        int ext = L != R ? RMQ.query(L, R-1) : n - sa[L];

        // Tem 'ext - p' substrings diferentes que ocorrem 'R-L+1' vezes
        // O LCP de todas elas eh 'ext'
        ll ans = (ext-p)*f(R-L+1);

        // L eh terminal, e folha sse L == R
        if (sa[L]+ext == n) L++;

        // se for um SA de varias strings separadas como s#t$u&, usar no
            lugar do if de cima
        //      (separadores < 'a', diferentes e inclusive no final)
        // while (L <= R && (sa[L]+ext == n || s[sa[L]+ext] < 'a')) {
        //   L++;
        // }

        while (L <= R) {
                int idx = L != R ? RMQ.index_query(L, R-1) : -1;
                if (idx == -1 or lcp[idx] != ext) idx = R;

                ans += dfs(L, idx, ext);
                L = idx+1;
        }
        return ans;
}

// sum over substrings: computa, para toda substring t distinta de s,
// \sum f(# ocorrencias de t em s) - O (n)
ll sos() { return dfs(0, n-1, 0); }
};
```

31.10 suffixArray2

```
// Suffix Array - O(n log n)
//
// kasai recebe o suffix array e calcula lcp[i],
```

```
// o lcp entre s[sa[i],...,n-1] e s[sa[i+1],..,n-1]
//
// Complexidades:
// suffix_array - O(n log(n))
// kasai - O(n)

vector<int> suffix_array(string s) {
        s += "$";
        int n = s.size(), N = max(n, 260);
        vector<int> sa(n), ra(n);
        for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

        for(int k = 0; k < n; k ? k *= 2 : k++) {
                vector<int> nsa(sa), nra(n), cnt(N);

                for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n, cnt[ra[i
                    ]]++;
                for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
                for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] = nsa[i];

                for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r += ra[sa[i]] !=
                        ra[sa[i-1]] or ra[(sa[i]+k)%n] != ra[(sa[i-1]+k)%n];
                ra = nra;
                if (ra[sa[n-1]] == n-1) break;
        }
        return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
        int n = s.size(), k = 0;
        vector<int> ra(n), lcp(n);
        for (int i = 0; i < n; i++) ra[sa[i]] = i;

        for (int i = 0; i < n; i++, k -= !!k) {
                if (ra[i] == n-1) { k = 0; continue; }
                int j = sa[ra[i]+1];
                while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
                lcp[ra[i]] = k;
        }
        return lcp;
}
```

31.11 suffixAutomaton

```
// Suffix Automaton
//
// Automato que aceita os sufixos de uma string
// Todas as funcoes sao lineares

namespace sam {
        int cur, sz, len[2*MAX], link[2*MAX], acc[2*MAX];
        int nxt[2*MAX][26];

        void add(int c) {
                int at = cur;
                len[sz] = len[cur]+1, cur = sz++;
                while (at != -1 and !nxt[at][c]) nxt[at][c] = cur, at = link[at
                    ];
                if (at == -1) { link[cur] = 0; return; }
                int q = nxt[at][c];
                if (len[q] == len[at]+1) { link[cur] = q; return; }
                int qq = sz++;
                len[qq] = len[at]+1, link[qq] = link[q];
                for (int i = 0; i < 26; i++) nxt[qq][i] = nxt[q][i];
                while (at != -1 and nxt[at][c] == q) nxt[at][c] = qq, at = link[
                    at];
                link[cur] = link[q] = qq;
        }
        void build(string& s) {
                cur = 0, sz = 0, len[0] = 0, link[0] = -1, sz++;
                for (auto i : s) add(i-'a');
                int at = cur;
                while (at) acc[at] = 1, at = link[at];
        }

        // coisas que da pra fazer:
        ll distinct_substrings() {
                ll ans = 0;
                for (int i = 1; i < sz; i++) ans += len[i] - len[link[i]];
                return ans;
        }
        string longest_common_substring(string& S, string& T) {
                build(S);
                int at = 0, l = 0, ans = 0, pos = -1;
                for (int i = 0; i < T.size(); i++) {
                        while (at and !nxt[at][T[i]-'a']) at = link[at], l = len
                            [at];
                        if (nxt[at][T[i]-'a']) at = nxt[at][T[i]-'a'], l++;
                        else at = 0, l = 0;
                        if (l > ans) ans = l, pos = i;
                }
                return T.substr(pos-ans+1, ans);
        }
        ll dp[2*MAX];
        ll paths(int i) {
                auto& x = dp[i];
                if (x) return x;
                x = 1;
                for (int j = 0; j < 26; j++) if (nxt[i][j]) x += paths(nxt[i][j
                    ]);
                return x;
        }
        void kth_substring(int k, int at=0) { // k=1 : menor substring lexicog.
                for (int i = 0; i < 26; i++) if (k and nxt[at][i]) {
                        if (paths(nxt[at][i]) >= k) {
                                cout << char('a'+i);
                                kth_substring(k-1, nxt[at][i]);
                                return;
                        }
                        k -= paths(nxt[at][i]);
                }
        }
};
```

31.12 trie

```
// Trie
//
// trie T() constroi uma trie para o alfabeto das letras minusculas
// trie T(tamanho do alfabeto, menor caracter) tambem pode ser usado
//
// T.insert(s) - O(|s|*sigma)
// T.erase(s) - O(|s|)
// T.find(s) retorna a posicao, -1 se nao achar - O(|s|)
// T.count_pref(s) numero de strings que possuem s como prefixo - O(|s|)

struct trie {
        vector<vector<int>> to;
        vector<int> end, pref;
        int sigma; char norm;
        trie(int sigma_=26, char norm_='a') : sigma(sigma_), norm(norm_) {
                to = {vector<int>(sigma)};
                end = {0}, pref = {0};
        }
        void insert(string s) {
                int x = 0;
                for (auto c : s) {
                        int &nxt = to[x][c-norm];
                        if (!nxt) {
                                nxt = to.size();
                                to.push_back(vector<int>(sigma));
                                end.push_back(0), pref.push_back(0);
                        }
                        x = nxt, pref[x]++;
                }
                end[x]++, pref[0]++;
        }
        void erase(string s) {
                int x = 0;
                for (char c : s) {
                        int &nxt = to[x][c-norm];
```

```
                x = nxt, pref[x]--;
                if (!pref[x]) nxt = 0;
            }
            end[x]--, pref[0]--;
        }
        int find(string s) {
            int x = 0;
            for (auto c : s) {
                x = to[x][c-norm];
                if (!x) return -1;
            }
            return x;
        }
        int count_pref(string s) {
            int id = find(s);
            return id >= 0 ? pref[id] : 0;
        }
};
```

31.13 z

```
// Z
//
// z[i] = lcp(s, s[i..n))
//
// Complexidades:
// z - O(|s|)
// match - O(|s| + |p|)

vector<int> get_z(string s) {
    int n = s.size();
    vector<int> z(n, 0);

    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n and s[z[i]] == s[i + z[i]]) z[i]++;
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }

    return z;
}
```

32 Utils

32.1 execution time

```
// https://codeforces.com/blog/entry/57647
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 1000000007

signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  // just call clock in the beginning
  clock_t time = clock();

  // ...
  // ...
  // some code here ...
  // ...
  // ...

  // execution time:
  cout << setprecision(3) << fixed << (double)(clock() - time) / CLOCKS_PER_SEC
       << endl;
  return 0;
}
```

32.2 rand

```
// https://codeforces.com/blog/entry/61587
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<int, pi>
#define fir first
#define sec second
#define MAXN 300005
#define mod 1000000007

signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

    int n = 10;
    vector<int> v(n);
    for (int i = 0; i < n; i++)
        v[i] = i;
    shuffle(v.begin(), v.end(), rng); // random shuffle

    int x = rng() % 10; // better than rand()
    cout << x << endl;

    // random integer values on the closed interval [a, b]
    int y = uniform_int_distribution<int>(3, 77)(rng);
    cout << y << endl;

    return 0;
}
```

32.3 runner

```
# This script does the following:
# 1 - Generate a random testcase
# 2 - Run some "naive" code with this input
# 3 - Run your code with this input
# 4 - Compare the outputs
import os
import subprocess

naive = "brute.cpp" # path to naive code
code = "d.cpp" # path to your code
```

```
generator = "g.cpp" # path to test generator

def compile_codes():
    os.system('g++ ' + generator + ' -o generator -O2')
    os.system('g++ ' + naive + ' -o naive -O2')
    os.system('g++ ' + code + ' -o code -O2')

def generate_case():
    os.system('./generator > in');

def get_naive_output():
    output = os.popen('./naive <in').read()
    return output

def get_code_output():
    # se tiver um runtime error, vai parar
    xalala = subprocess.run('./code <in', shell=True, text=True, capture_output=
        True, check=True)
    return xalala.stdout

def main():
    compile_codes()

    while True:
        generate_case()
        naive_output = get_naive_output()
        code_output = get_code_output()

        if naive_output == code_output:
            print('ACCEPTED')
        else :
            print('FAILED\n')
            print('ANSWER:')
            print(naive_output)
            print('\nCODE OUTPUT:')
            print(code_output)
            break

if __name__ == '__main__':
    main()
```

32.4 runner2

```
# This script does the following:
# 1 - Run a code with all inputs files from a folder
# 2 - Compare the output for each test case with the answer
import os

code = "a.cpp" # Path to your code
input_folder = "input" # Path to folder which the input files are
output_folder = "output" # Path to folder which the output files are
input_prefix = "L_" # prefix of all input files names
output_prefix = "L_" # prefix of all input files names
tests = 56 # Number of test cases

def compile_code():
    os.system('g++ ' + code + ' -o code -O2')

def get_ans(output):
    out = open(output, "r")
    ret = out.read()
    out.close()
    return ret

def get_code_output(input):
    output = os.popen('./code <' + input).read()
    return output

def main():
    compile_code()
    # tests indexed from 1
    for i in range (1, tests + 1):
        ans = get_ans(output_folder + '/' + output_prefix + str(i))
        code_output = get_code_output(input_folder + '/' + input_prefix + str(i)
            )
        print('Case' + str(i) + ': ')
        if ans == code_output:
            print('ACCEPTED')
        else :
            print('FAILED\n')
            print('ANSWER:')
            print(ans)
            print('\nCODE OUTPUT:')
            print(code_output)
        print()

if __name__ == '__main__':
    main()
```

32.5 int128

```
// https://codeforces.com/blog/entry/75044
// functions to print and read a __int128 in c++
__int128 read()
{
  __int128 x = 0, f = 1;
  char ch = getchar();
  while (ch < '0' || ch > '9')
  {
    if (ch == '-')
      f = -1;
    ch = getchar();
  }
  while (ch >= '0' && ch <= '9')
  {
    x = x * 10 + ch - '0';
    ch = getchar();
  }
  return x * f;
}
void print(__int128 x)
{
  if (x < 0)
  {
    cout << "-";
    x = -x;
  }
  if (x > 9)
    print(x / 10);
  char at = (x % 10) + '0';
  cout << at;
}
```