Numerical (1)

1.1 Algos

AhoCor.cpp

Description: AhoCorasick

d41d8c, 122 lines

```cpp
struct AhoCorasick
{
  struct node
  {
    char c;
    vector<int> ids;
    map<char, node *> go;
    node *failLink;
    node *outputLink;
    node(char _c)
    {
      c = _c;
      failLink = NULL;
      outputLink = NULL;
    }
  };
  node *root;
  vector<string> words;
  AhoCorasick(vector<string> _words) : words(_words)
  {
    root = new node(0);
    int sz = words.size();
    for (int i = 0; i < sz; i++)
    {
      insert(words[i], i);
    }
    genLinks();
  }
  void insert(string s, int ind)
  {
    node *cur = root;
    for (char c : s)
    {
      node *&nxt = cur->go[c];
      if (nxt == NULL)
        nxt = new node(c);
      cur = nxt;
    }
    cur->ids.push_back(ind);
  }
  void genLinks()
  {
    queue<node *> q;
    root->failLink = root;
    for (auto tmp : root->go)
    {
      tmp.second->failLink = root;
      q.push(tmp.second);
    }
    while (!q.empty())
    {
      node *tp = q.front();
      q.pop();
      for (auto tmp : tp->go)
      {
        node *cur = tmp.second;
        node *prev = tp;
        node *t;
        // assign failLink
        while ((t = prev->failLink->go[tmp.fi]) == NULL)
        {
          if (prev == root)
          {
            t = root;
            break;
          }
          prev = prev->failLink;
        }
        cur->failLink = t;
        // assign ouptutLink
        if (!((cur->failLink->ids).empty()))
        {
          cur->outputLink = cur->failLink;
        }
        else
        {
          cur->outputLink = cur->failLink->outputLink;
        }
        q.push(cur);
      }
    }
  }
  vector<int> getCnt(string s, int limit)
  {
    node *curState = root;
    node *nxt;
    int sz = s.size();
    vector<int> res(words.size(), 0);
    for (int i = 0; i < sz; i++)
    {
      char c = s[i];
      nxt = curState->go[c];
      if (nxt == NULL)
      {
        if (curState != root)
        {
          i--;
        }
        curState = curState->failLink;
      }
      else
      {
        curState = nxt;
      }
      for (int cid : curState->ids)
        if (res[cid] < limit)
          res[cid]++;
      node *curOutput = curState->outputLink;
      while (curOutput != NULL)
      {
        for (int cid : curOutput->ids)
        {
          if (res[cid] == limit)
            break;
          res[cid]++;
        }
        curOutput = curOutput->outputLink;
      }
    }
    return res;
  }
};
```

AryanTemplate.cpp

Description: Aryan Template

<bits/stdc++.h> d41d8c, 13 lines

```cpp
#define all(v) v.begin(),v.end()
#define allr(v) v.rbegin(),v.rend()
#define fori(i,n) for(ll i=0;i<n;i++)
#define ford(i,n) for(ll i = n-1;i >= 0;i--) #define pb
    push_back
#define ll long long int
//#define mod 998244353
#define pi pair<int,int>
#define pll pair<ll,ll>
#define mp make_pair
#define fi first
#define se second
//#pragma GCC optimize(”unroll–loops”) //#pragma GCC optimize(”
    O3”)
//#pragma GCC target(”avx2”)
```

ConvexHullTrick.cpp

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: $\mathcal{O}(\log N)$

d41d8c, 31 lines

```cpp
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);
    //cout<<l.k<<” ”<<l.m<<”HH\n”;
    return l.k * x + l.m;
  }
};
```

Convexhull.cpp

Description: Convex Hull

d41d8c, 52 lines

```cpp
struct ConvexHull
{
    using pint = pair<int,int>;
    vector<pint> points;
    vector<pint> hull;
    int n;
    long long area(pint a,pint b,pint c)
    {
        long long area = 0;
        area = 1ll * (b.first - a.first) * (c.second - b.second
            );
        area -= 1ll * (c.first - b.first) * (b.second - a.
            second);
        return area;
    }
    ConvexHull(vector<pint> _points): points(_points),n(_points
        .size())
    {
        sort(points.begin(),points.end());
        points.erase(unique(points.begin(),points.end()),points
            .end());
        n = points.size();
        if(n < 3)
        {
          hull = points;
          return;
        }
        for(int i = 0;i < n;i++)
        {
            while(hull.size() > 1 && area(hull[hull.size()-2],
                hull.back(),points[i]) <= 0) hull.pop_back();
            hull.push_back(points[i]);
        }
        int lower_length = hull.size();
        for(int i = n-2;i >= 0;i--)
        {
            while(hull.size() > lower_length && area(hull[hull.
                size()-2],hull.back(),points[i]) <= 0) hull.
                pop_back();
            hull.push_back(points[i]);
        }
        hull.pop_back();
    }
    long long getTwiceAreaofHull()
    {
        int sz = hull.size();
        long long ans = 0;
        for(int i =0;i < sz;i++)
        {
            int j = (i+1)%sz;
            long long a1 = hull[i].first;
            long long b1 = hull[i].second;
            long long a2 = hull[j].first;
            long long b2 = hull[j].second;
            ans += a1 * b2 - a2 * b1;
        }
        return abs(ans);
    }
};
```

Dinic.cpp

Description: Dinic

d41d8c, 87 lines

```
template <typename T>
struct dinic
// Mostly copied, wrong spelling??
{
    struct Edg
    {
        int to, rev;
        T cap;
    };
    vector<vector<Edg>> adj;
    int n;
    dinic(int _n)
    {
        n = _n;
        adj.resize(n);
    }
    inline T getActualFlow(int a, int i) // i is index of edge
        in list of adj[a];
    {
        Edg e = adj[a][i];
        Edg e2 = adj[e.to][e.rev];
        return e2.cap;
    }
    void addEdge(int a, int b, T _cap, T _rcap = 0)
    {
        int sz1 = adj[a].size();
        int sz2 = adj[b].size();
        adj[a].pb({b, sz2, _cap});
        adj[b].pb({a, sz1, _rcap});
        assert(_cap >= 0 && _rcap >= 0);
    };
    vector<int> level;
    vector<int> ptr;
    bool setUpLevel(int s, int t) // returns if t reachable
        from s
    {
        level.assign(n, 0); // levels
        start from 1 !!ptr.assign(n, 0);
        queue<int> q({s});
        level[s] = 1;
        while (!q.empty())
        {
            int x = q.front();
            q.pop();
            int sz = adj[x].size();
            for (int i = 0; i < sz; i++)
            {
                int cap = adj[x][i].cap;
                int to = adj[x][i].to;
                if (cap && !level[to])
                {
                    q.push(to);
                    level[to] = level[x] + 1;
                }
            }
        }
        return bool(level[t]);
    }
    T dfs(int x, int t, T curFlow)
    {
        if (x == t)
            return curFlow;
        for (int &i = ptr[x]; i < (int)adj[x].size(); i++)
        //& very important O(E^2V) -> O(EV^2))
        {
            Edg &e = adj[x][i];
            if ((level[e.to] == level[x] + 1) && e.cap)
            {
                if (T df =
                        dfs(e.to, t, min(curFlow, e.cap)))
                {
                    e.cap -= df;
                    adj[e.to][e.rev].cap += df;
                    return df;
                }
            }
        }
        return 0;
    }
    T getMxFlow(int s, int t)
    {
        T res = 0;
        while (setUpLevel(s, t))
            while (T df =
                    dfs(s, t, numeric_limits<T>::max()))
                res += df;
        return res;
    }
};
```

DSU.cpp

Description: DSU

d41d8c, 19 lines

```
struct DSU
{ // beng implementation
    vector<int> e;
    DSU(int N) { e = vector<int>(N, -1); }
    // get representive component (uses path compression) int
        get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }
    bool same_set(int a, int b) { return get(a) == get(b); }
    int size(int x) { return -e[get(x)]; }
    bool unite(int x, int y)
    { // union by size
        x = get(x), y = get(y);
        if (x == y)
            return false;
        if (e[x] > e[y])
            swap(x, y);
        e[x] += e[y];
        e[y] = x;
        return true;
    }
};
```

EulerCircuit.cpp

Description: Euler circuit

d41d8c, 18 lines

```
multiset<int> adj[500005];
vector<int> finale;
void DFS(int u)
{
    vector<int> g;
    while (!adj[u].empty())
    {
        auto x = *(adj[u].begin());
        if (adj[u].find(x) == adj[u].end())
        {
            continue;
        }
        adj[u].erase(adj[u].find(x));
        adj[x].erase(adj[x].find(u));
        DFS(x);
    }
    finale.push_back(u);
}
```

Extendedeuclid.cpp

Description: Extended Euclid

d41d8c, 19 lines

```
int modInverse(int A, int M)
{
    int m0 = M;
    int y = 0, x = 1;
    if (M == 1)
        return 0;
    while (A > 1)
    {
        int q = A / M;
        int t = M;
        M = A % M, A = t;
        t = y;
        y = x - q * y;
        x = t;
    }
    if (x < 0)
        x += m0;
    return x;
}
```

Divideandconquerfft.h

<bits/stdc++.h> d41d8c, 120 lines

```cpp
using namespace std;
const int MAXN = 300005;
const int root = 1489;
const int root_1 = 296201594;
const int root_pw = (1<<19);
const long long int MOD = 663224321;
long long int fact[MAXN], invfact[MAXN], pow2[MAXN], all_graphs
     [MAXN], poly1[MAXN], poly2[MAXN];
long long int power(long long int a, int b)
{
  if(!b)
    return 1;
  long long int ans = power(a,b/2);
  ans = (ans*ans)%MOD;
  if(b%2)
    ans = (ans*a)%MOD;
  return ans;
}
// fft code taken from e-maxx.ru/algo/fft_multiply
void fft (vector <long long int> &a, bool invert)
{
  int n = (int) 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)
  {
    long long int wlen = invert ? root_1 : root;
    for (int i = len; i < root_pw; i<<=1)
      wlen = (wlen*wlen)%MOD;
    for (int i = 0; i < n; i+=len)
    {
      long long int w = 1;
      for (int j = 0; j < len/2; ++j)
      {
        int u = a[i+j],  v = (a[i+j+len/2]*w)%MOD;
        a[i+j] = u+v < MOD ? u+v : u+v-MOD;
        a[i+j+len/2] = u-v >= 0 ? u-v : u-v+MOD;
        w = (w*wlen)%MOD;
      }
    }
  }
  if(invert)
  {
    int nrev = power(n, MOD-2);
    for (int i=0; i<n; ++i)
      a[i] = (a[i]*nrev)%MOD;
  }
}
void multiply(vector <long long int> &a, vector <long long int>
     &b, vector <long long int> &c)
{
  vector <long long int> ta(a.begin(), a.end()), tb(b.begin(),
      b.end()), tc;
  int sz = 2*a.size();
  ta.resize(sz);
  tb.resize(sz);
  fft(ta,false);
  fft(tb,false);
  for (int i = 0; i < sz; ++i)
  {
    ta[i] = (1ll*ta[i]*tb[i])%MOD;
  }
  fft(ta,true);
  c = ta;
}
void dnc(int l, int r)
{
  if(l+1 == r)
  {
    poly2[l] = (all_graphs[l]*invfact[l-1] + MOD - poly2[l])%
         MOD;
  }
  else
  {
    int m = (l + r)/2;
    dnc(l,m);
    dnc(m,r);
  }
  vector <long long int> p1, p2, ans;
  int sz = r - l;
  for (int i = sz; i < 2*sz; ++i)
  {
    p1.push_back(poly1[i]);
  }
  for (int i = l; i < r; ++i)
  {
    p2.push_back(poly2[i]);
  }
  multiply(p1,p2,ans);
  for (int i = 0; i < ans.size(); ++i)
  {
    int pos = l + sz + i;
    poly2[pos] = (poly2[pos] + ans[i])%MOD;
  }
}
int main()
{
  fact[0] = invfact[0] = pow2[0] = all_graphs[0] = 1;
  for (int i = 1; i < MAXN; ++i)
  {
    fact[i] = (fact[i-1]*i)%MOD;
    invfact[i] = power(fact[i], MOD-2);
    pow2[i] = (pow2[i-1]*2)%MOD;
    all_graphs[i] = (all_graphs[i-1]*pow2[i-1])%MOD;
    poly1[i] = (all_graphs[i]*invfact[i])%MOD;
  }
  dnc(1,(1<<17)+1);
  int t;
  cin>>t;
  while(t--)
  {
    int x;
    cin>>x;
    cout<<(poly2[x]*fact[x-1])%MOD<<"\n";
  }
  return 0;
}
```

FastHash.cpp

Description: Fast Hash

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> d41d8c, 19 lines

```cpp
using namespace std;
using namespace __gnu_pbds;
#define int long long
struct custom_hash
{
    static uint64_t splitmix64(uint64_t x)
    {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const
    {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().
                count();
        return splitmix64(x + FIXED_RANDOM);
    }
}; // use gp_hash_table<typea,typeb,custom_hash> //faster than
     unordered and policy based as well(ordered_set)
```

FFT.cpp

Description: FFT

<bits/stdc++.h> d41d8c, 76 lines

```cpp
using namespace std;

const int N = 3e5 + 9;

const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a), b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w *
          wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
}
vector<long long> multiply(vector<int> &a, vector<int> &b) {
  int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long long) round(z[i].a
       );
  while((int)ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
```

```
long long ans[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, x; cin >> n >> x;
  vector<int> a(n + 1, 0), b(n + 1, 0), c(n + 1, 0);
  int nw = 0;
  a[0]++; b[n]++;
  long long z = 0;
  for (int i = 1; i <= n; i++) {
    int k; cin >> k;
    nw += k < x;
    a[nw]++; b[-nw + n]++;
    z += c[nw] + !nw; c[nw]++;
  }
  auto res = multiply(a, b);
  for (int i = n + 1; i < res.size(); i++) {
    ans[i - n] += res[i];
  }
  ans[0] = z;
  for (int i = 0; i <= n; i++) cout << ans[i] << ' ';
  cout << '\n';
  return 0;
}
//https://codeforces.com/contest/993/problem/E
```

Gauss.cpp

Description: Gauss d41d8c, 47 lines

```
typedef vector<long double> vd;
using vi = vector<int>;
const double eps = 1e-12;
int solveLinear(vector<vd> &A, vd &b, vd &x)
{
  int n = size(A), m = size(x), rank = 0, br, bc;
  vi col(m);
  iota(begin(col), end(col), 0);
  for (int i = 0; i < n; ++i)
  {
    double v, bv = 0;
    for (int r = i; r < n; ++r)
      for (int c = i; c < m; ++c)
        if ((v = fabs(A[r][c])) > bv)
          br = r, bc = c, bv = v;
    if (bv <= eps)
    {
      for (int j = i; j < n; ++j)
        if (fabs(b[j]) > eps)
          return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    for (int j = 0; j < n; ++j)
      swap(A[j][i], A[j][bc]);
    bv = 1 / A[i][i];
    for (int j = i + 1; j < n; ++j)
    {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      for (int k = i + 1; k < m; ++k)
        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];
    for (int j = 0; j < i; ++j)
      b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

Hungarian.cpp

Description: Hungarian d41d8c, 44 lines

```
const int INF = 1e18;
int hungarianMethod(vector<vector<int>> A) //A is one indexed
    !!!!!!!!!!
{
    int n = A.size()-1;
    int m = A.size()-1;      //change this if dimensions are
        different
    vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
    for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<int> minv (m+1, INF);
    vector<char> used (m+1, false);
    do {
        used[j0] = true;
        int i0 = p[j0],  delta = INF,  j1;
        for (int j=1; j<=m; ++j)
            if (!used[j]) {
                int cur = A[i0][j]-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<=m; ++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<pair<int, int>> result;       //mapping
for (int i = 1; i <= m; ++i){
  result.push_back(make_pair(p[i], i));
}

int C = -v[0];  //total cost
return C;
}
```

KMP.cpp

Description: KMP d41d8c, 15 lines

```
vector<int> prefix_function(string s)
{
    int n = (int)s.length();
    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;
}
```

LCALOGN.cpp

Description: lca(LOGN) d41d8c, 60 lines

```
int preorder[1000005];
int postorder[1000005];
int timer = 0;
int dp[1000005][22];
vector<int> adj[1000005];
void DFS(int u, int par)
{
    preorder[u] = timer;
    timer++;
    dp[u][0] = par;
    for (int i = 1; i < 22; i++)
    {
        int z = dp[u][i - 1];
        if (z == -1)
        {
            dp[u][i] = -1;
            continue;
        }
        dp[u][i] = dp[z][i - 1];
    }
    for (auto x : adj[u])
    {
        if (x == par)
        {
            continue;
        }
        DFS(x, u);
    }
    postorder[u] = timer;
    timer++;
}
int is_ancestor(int u, int v)
{
    if (preorder[u] <= preorder[v] && postorder[u] >= postorder
        [v])
    {
        return 1;
    }
    return 0;
}
int lca(int u, int v)
{
    if (is_ancestor(u, v))
    {
        return u;
    }
    int z = u;
    for (int i = 21; i >= 0; i--)
    {
        if (dp[z][i] == -1)
        {
            continue;
        }
        if (is_ancestor(dp[z][i], v))
        {
            continue;
        }
        z = dp[z][i];
    }
    return dp[z][0];
}
```

LCAO1.cpp

Description: LCA (o(1))

d41d8c, 120 lines

```
struct LCA
{
    int n;
    vector<vector<int>> adj;
    vector<int> level;
    vector<int> firstOcc;
    vector<int> eulerIndices;
    struct sparse_table
    {
        int n;
        std::vector<std::vector<int>> table;
        vector<int> logs;
        vector<int> logpow;
        void init(std::vector<int> &arr, vector<int> &level)
        {
            n = arr.size();
            logs.assign(n, 0);
            int t = 0, k = 1;
            logpow.push_back(1);
            for (int i = 0; i < n; i++)
            {
                if (k > ((i + 1) >> 1))
                {
                    logs[i] = t;
                }
                else
                {
                    k = (k << 1);
                    t++;
                    logpow.push_back(k);
                    logs[i] = t;
                }
            }
            table.resize(n);
            for (int i = 0; i < n; i++)
            {
                table[i].resize(logs[n - i - 1] + 1);
            }
            for (int i = 0; i < n; i++)
            {
                table[i][0] = arr[i];
            }
            for (int j = 1; j <= logs[n - 1]; j++)
            {
                for (int i = 0; (i + logpow[j] - 1) < n; i++)
                {
                    int a = table[i][j - 1];
                    int b = table[(i + logpow[j - 1])][j - 1];
                    table[i][j] = level[a] > level[b] ? b : a;
                }
            }
        }
        int RMQ(int i, int j, vector<int> &level)
        {
            if (i > j)
                swap(i, j);
            int t = logs[j - i];
            int a = table[i][t];
            ;
            int b = table[j - logpow[t] + 1][t];
            return level[a] > level[b] ? b : a;
        }
    };
    sparse_table tab;
    LCA(int N)
    {
        n = N;
        adj.resize(n);
        level.resize(n);
        firstOcc.assign(n, -1);
    }
    void addEdge(int a, int b)
    {
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    void init(int root)
    {
        vector<bool> vis(n, false);
        int d = 0;
        int eulWalkCnt = 0;
        function<void(int)> dfs = [&](int x)
        {
            vis[x] = true;
            if (firstOcc[x] == -1)
            {
                firstOcc[x] = eulWalkCnt;
            }
            level[x] = d;
            eulerIndices.push_back(x);
            eulWalkCnt++;
            d++;
            for (auto &i : adj[x])
            {
                if (!vis[i])
                {
                    dfs(i);
                    eulerIndices.push_back(x);
                    eulWalkCnt++;
                }
            }
            d--;
        };
        dfs(root);
        tab.init(eulerIndices, level);
    }
    int find_LCA(int a, int b)
    {
        if (a == b)
            return a;
        if (firstOcc[a] > firstOcc[b])
            swap(a, b);
        return tab.RMQ(firstOcc[a], firstOcc[b], level);
    }
    int find_dist(int a, int b)
    {
        int lc = find_LCA(a, b);
        return level[a] + level[b] - 2 * level[lc];
    }
};
```

MCMF.cpp

Description: MCMF

<bits/stdc++.h> d41d8c, 165 lines

```
using namespace std;

const int N = 3e5 + 9;

//Works for both directed, undirected and with negative cost
    too
//doesn't work for negative cycles
//for undirected edges just make the directed flag false
//Complexity: O(min(E^2 *V log V, E logV * flow))
using T = long long;
const T inf = 1LL << 61;
struct MCMF {
  struct edge {
    int u, v;
    T cap, cost;
    int id;
    edge(int _u, int _v, T _cap, T _cost, int _id) {
      u = _u;
      v = _v;
      cap = _cap;
      cost = _cost;
      id = _id;
    }
  };
  int n, s, t, mxid;
  T flow, cost;
  vector<vector<int>> g;
  vector<edge> e;
  vector<T> d, potential, flow_through;
  vector<int> par;
  bool neg;
  MCMF() {}
  MCMF(int _n) { // 0-based indexing
    n = _n + 10;
    g.assign(n, vector<int> ());
    neg = false;
    mxid = 0;
  }
  void add_edge(int u, int v, T cap, T cost, int id = -1, bool
      directed = true) {
    if(cost < 0) neg = true;
    g[u].push_back(e.size());
    e.push_back(edge(u, v, cap, cost, id));
    g[v].push_back(e.size());
    e.push_back(edge(v, u, 0, -cost, -1));
    mxid = max(mxid, id);
    if(!directed) add_edge(v, u, cap, cost, -1, true);
  }
  bool dijkstra() {
    par.assign(n, -1);
    d.assign(n, inf);
    priority_queue<pair<T, T>, vector<pair<T, T>>, greater<pair
        <T, T>> > q;
    d[s] = 0;
    q.push(pair<T, T>(0, s));
    while (!q.empty()) {
      int u = q.top().second;
      T nw = q.top().first;
      q.pop();
      if(nw != d[u]) continue;
      for (int i = 0; i < (int)g[u].size(); i++) {
        int id = g[u][i];
        int v = e[id].v;
        T cap = e[id].cap;
        T w = e[id].cost + potential[u] - potential[v];
        if (d[u] + w < d[v] && cap > 0) {
          d[v] = d[u] + w;
          par[v] = id;
          q.push(pair<T, T>(d[v], v));
        }
      }
    }
    for (int i = 0; i < n; i++) {
      if (d[i] < inf) d[i] += (potential[i] - potential[s]);
    }
    for (int i = 0; i < n; i++) {
      if (d[i] < inf) potential[i] = d[i];
    }
    return d[t] != inf; // for max flow min cost
    // return d[t] <= 0; // for min cost flow
  }
  T send_flow(int v, T cur) {
    if(par[v] == -1) return cur;
    int id = par[v];
    int u = e[id].u;
    T w = e[id].cost;
    T f = send_flow(u, min(cur, e[id].cap));
    cost += f * w;
    e[id].cap -= f;
    e[id ^ 1].cap += f;
    return f;
  }
  //returns {maxflow, mincost}
  pair<T, T> solve(int _s, int _t, T goal = inf) {
    s = _s;
    t = _t;
    flow = 0, cost = 0;
    potential.assign(n, 0);
    if (neg) {
      // Run Bellman-Ford to find starting potential on the
           starting graph
      // If the starting graph (before pushing flow in the
           residual graph) is a DAG,
      // then this can be calculated in O(V + E) using DP:
      // potential(v) = min({potential[u] + cost[u][v]}) for
           each u -> v and potential[s] = 0
      d.assign(n, inf);
      d[s] = 0;
      bool relax = true;
      for (int i = 0; i < n && relax; i++) {
        relax = false;
        for (int u = 0; u < n; u++) {
          for (int k = 0; k < (int)g[u].size(); k++) {
            int id = g[u][k];
            int v = e[id].v;
            T cap = e[id].cap, w = e[id].cost;
            if (d[v] > d[u] + w && cap > 0) {
              d[v] = d[u] + w;
              relax = true;
            }
          }
        }
      }
      for(int i = 0; i < n; i++) if(d[i] < inf) potential[i] =
           d[i];
    }
    while (flow < goal && dijkstra()) flow += send_flow(t, goal
          - flow);
    flow_through.assign(mxid + 10, 0);
    for (int u = 0; u < n; u++) {
      for (auto v : g[u]) {
        if (e[v].id >= 0) flow_through[e[v].id] = e[v ^ 1].cap;
      }
    }
    return make_pair(flow, cost);
  }
};
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n;
  cin >> n;
  assert(n <= 10);
  MCMF F(2 * n);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      int k;
      cin >> k;
      F.add_edge(i, j + n, 1, k, i * 20 + j);
    }
  }
  int s = 2 * n + 1, t = s + 1;
  for (int i = 0; i < n; i++) {
    F.add_edge(s, i, 1, 0);
    F.add_edge(i + n, t, 1, 0);
  }
  auto ans = F.solve(s, t).second;
  long long w = 0;
  set<int> se;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      int p = i * 20 + j;
      if (F.flow_through[p] > 0) {
        se.insert(j);
        w += F.flow_through[p];
      }
    }
  }
  assert(se.size() == n && w == n);
  cout << ans << '\n';
  return 0;
}
```

ncr.cpp

Description: ncr

d41d8c, 36 lines

```
const int mxN = 100005; // dont forget to change mxN ll power(
      ll x, ll n) //x base n exponent{
if (n == 0)
    return 1;
if (x % mod == 0)
    return 0; // For large N,%mod-> mod is prime
n = n % (mod - 1);
ll pow = 1;
while (n)
{
    if (n & 1)
        pow = (pow * x) % mod;
    n = n >> 1;
    x = (x * x) % mod;
}
return pow;
}
vector<ll> fact(mxN + 1);
vector<ll> invfact(mxN + 1);
void genfact() // dont forget to call genfact(){
    fact[0] = 1;
fori(i, mxN)
{
    fact[i + 1] = (fact[i] * (ll)(i + 1)) % mod;
}
invfact[mxN] = power(fact[mxN], mod - 2);
for (int i = mxN - 1; i >= 0; i--)
{
    invfact[i] = (invfact[i + 1] * (i + 1)) % mod;
}
}
ll ncr(ll n, ll r)
{
    if (n - r < 0 || r < 0)
        return 0;
    return ((fact[n] * invfact[n - r]) % mod * invfact[r]) %
         mod;
}
```

NTT.cpp

Description: NTT

d41d8c, 48 lines

```
const int N = 1 << 20;
const int mod = 998244353;
const int root = 3;
int lim, rev[N], w[N], wn[N], inv_lim;
void reduce(int &x) { x = (x + mod) % mod; }
int POW(int x, int y, int ans = 1) {
  for (; y; y >>= 1, x = (long long) x * x % mod) if (y & 1)
       ans = (long long) ans * x % mod;
  return ans;
}
//There should be an or in //rev[i] = rev[i>>1]>>1(i&1)<<s;
void precompute(int len) {
  lim = wn[0] = 1; int s = -1;
  while (lim < len) lim <<= 1, ++s;
  for (int i = 0; i < lim; ++i) rev[i] = rev[i >> 1] >> 1 | (i
       & 1) << s;
  const int g = POW(root, (mod - 1) / lim);
  inv_lim = POW(lim, mod - 2);
  for (int i = 1; i < lim; ++i) wn[i] = (long long) wn[i - 1] *
        g % mod;
}
void ntt(vector<int> &a, int typ) {
  for (int i = 0; i < lim; ++i) if (i < rev[i]) swap(a[i], a[
       rev[i]]);
  for (int i = 1; i < lim; i <<= 1) {
    for (int j = 0, t = lim / i / 2; j < i; ++j) w[j] = wn[j *
         t];
    for (int j = 0; j < lim; j += i << 1) {
      for (int k = 0; k < i; ++k) {
        const int x = a[k + j], y = (long long) a[k + j + i] *
             w[k] % mod;
        reduce(a[k + j] += y - mod), reduce(a[k + j + i] = x -
             y);
      }
    }
  }
  if (!typ) {
    reverse(a.begin() + 1, a.begin() + lim);
    for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] *
         inv_lim % mod;
  }
}

//Issues of tle can arise if you do not pop back the
     uneccesaary 0 due to rounding to nearest 2
//Already done in this code but keep in mind
vector<int> multiply(vector<int> &f, vector<int> &g) {
  int n=(int)f.size() + (int)g.size() - 1;
  precompute(n);
  vector<int> a = f, b = g;
  a.resize(lim); b.resize(lim);
  ntt(a, 1), ntt(b, 1);
  for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] * b[i]
       % mod;
  ntt(a, 0);
  while((int)a.size() && a.back() == 0) a.pop_back();
  return a;
}
```

orderedset.cpp

Description: orderedset

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp>   d41d8c, 4 lines

```
using namespace __gnu_pbds;
#define int long long
```

```
typedef
tree<pair<int,int>,null_type,less<pair<int,int>>,rb_tree_tag,
     tree_order_statistics_node_update> ordered_set;
```

persistentsegtree.cpp

Description: Persistent Segtree

d41d8c, 41 lines

```
struct Node{
    int val;
    Node* l;
    Node* r;
    Node(){
        val=0;
        l=nullptr;
        r=nullptr;
    }
};
Node* build(int lx,int rx){
    if(rx-lx==1){return new Node();}
    Node* ret = new Node();
    int m=(lx+rx)/2;
    ret->l=build(lx,m);
    ret->r=build(m,rx);
    return ret;
}
Node* modify(int lx,int rx,Node* a,int i,int z){
    if(rx-lx==1){
            Node* ret=new Node();
            ret->val=(a->val)^z;
            return ret;
        }
    Node* ret = new Node();
    ret->val=a->val;
    ret->l=a->l;
    ret->r=a->r;
    ret->val=((ret->val)^z);
    int m=(lx+rx)/2;
    if(i<m){ret->l = modify(lx,m,ret->l,i,z);}
    else{ret->r = modify(m,rx,ret->r,i,z);}
    return ret;
}
int getans(int lx,int rx,Node* a,int li,int ri)
{
    if(lx>=li && rx<=ri){return a->val;}
    if(lx>=ri || li>=rx){return 0;}
    int m=(lx+rx)/2;
    return getans(lx,m,a->l,li,ri)^getans(m,rx,a->r,li,ri);
}
```

PersistentTrie.cpp

Description: Persistent Trie

<iostream>, <cstdio> d41d8c, 56 lines

```
using namespace std;
int a[10000000][2],ww[20],sz[10000000],type,root[600000],m,ind
    [10000000];
int ans,n,v,l,r,x;
void add(int prev_root, int now_root, int x) {
    for (int i=18;i>=0;i--) {
        int edge=(x & (1 << i));
        if (edge!=0) edge=1;
        v++;
        ww[i]=now_root;
        ind[v]=n;
        a[now_root][edge]=v;
        a[now_root][1-edge]=a[prev_root][1-edge];
        sz[now_root]=sz[a[now_root][1-edge]];
        now_root=v;
        prev_root=a[prev_root][edge];
    }
    sz[now_root]+=sz[prev_root]+1;
    for (int i=0;i<=18;i++) sz[ww[i]]=sz[a[ww[i]][0]]+sz[a[ww[i
        ]][1]];
 }
void findxor(int v, int l, int x) {
    for (int i=18;i>=0;i--) {
        int edge=(x & (1 << i));
        if (edge!=0) edge=1;
        if (ind[a[v][1-edge]]>=l) {
            v=a[v][1-edge];
            ans+=(1-edge)*(1 << i);
        }
        else {
            v=a[v][edge];
            ans+=edge*(1 << i);
        }
     }
}
int findless(int v, int x) {
    int ans=0;
    for (int i=18;i>=0;i--) {
        int edge=(x & (1 << i));
        if (edge!=0) edge=1;
        for (int j=0;j<edge;j++) ans+=sz[a[v][j]];
        v=a[v][edge];
    }
    ans+=sz[v];
    return ans;
}
void findkth(int vr, int vl, int x) {
    for (int i=18;i>=0;i--) {
        for (int j=0;j<=1;j++)
            if (sz[a[vr][j]]-sz[a[vl][j]]<x) x-=sz[a[vr][j]]-sz
                [a[vl][j]];
            else {
                vr=a[vr][j];
                vl=a[vl][j];
                ans+=j*(1 << i);
                break;
            }
    }
}
```

rng.cpp

Description: rng

d41d8c, 2 lines

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

SCCwithtopo.cpp

Description: scc with topo

d41d8c, 95 lines

```
npconst int N = 1e5;

vector<int> adj[N];
int num_vertex;

//KDVinit's Implementation for Tarjan's Algorithm (GFG)
//Global Variables for SCC,
//This will additionally use N = max, adj[N] for edges,
//and Vertexes are defined from 1 -> num_vertex.

//Outputs of the Tarjan's Algorithm
vector<int> SCC_Nodes;           //Nodes from main graph that
      are heads of SCC.
vector<int> SCC_Component[N];    //Vertexes in the SCC Component
      of a head node.
int SCC_par[N];                  //SCC Head of a node from
      original graph.
int SCC_Topo[N];                    //Vertex of SCC, Topologically
      Sorted.
int SCC_Topo_Index[N];               //Index of vertex v in the
      topological sort.

//Note - Adj List may store multiple edges between 2 vertex
vector<int> SCC_adj[N];          //Adj list in SCC compressed,
      outgoing edges.
vector<int> SCC_incoming[N];     //Adj list in SCC compressed,
      incoming edges.

//Helpers
int disc[N], low[N], In_stk[N], SCC_time;
stack<int> stk;

//Helper DFS for SCC
void SCC_DFS(int chi)
{
    //Assigns the low and disc values
    SCC_time++; disc[chi]=SCC_time; low[chi]=SCC_time;
    stk.push(chi); In_stk[chi]=1;        //Put them in Stack

    //Runs DFS
    for(auto x: adj[chi])
    {
        if(disc[x]==-1) { SCC_DFS(x); low[chi] = min(low[chi],
            low[x]); }
        else if(In_stk[x]==1) low[chi] = min(low[chi], disc[x])
            ;
    }

    //if low = disc, it is the head node, and thus extracts the
        subtree
    if(low[chi]==disc[chi])
    {
        int w; SCC_Nodes.push_back(chi); SCC_Component[chi].
            push_back(chi);
        while(stk.top()!=chi)
        {
            w=stk.top(); stk.pop(); In_stk[w]=0;
            SCC_par[w]=chi; SCC_Component[chi].push_back(w);
        }
        stk.pop(); In_stk[chi]=0; SCC_par[chi]=chi;
    }
}

void SCC_Toposort()          //Topologically sorts the newly for
     DAG of SCC
{
    int in_deg[num_vertex+1] = {0};
    for(int i=1; i<=num_vertex; i++) in_deg[i] = SCC_incoming[i
        ].size();

    queue<int> topo_qu; int topo_cnt=0;
    for(int i=1; i<=num_vertex; i++) if(in_deg[i]==0) topo_qu.
        push(i);

    while(!topo_qu.empty())
    {
        int node = topo_qu.front(); topo_qu.pop();
        SCC_Topo[++topo_cnt] = (node);
        for(auto x: SCC_adj[node]) { in_deg[x]--; if(in_deg[x
            ]==0) topo_qu.push(x); }
    }

    for(int i=1; i<=num_vertex; i++) SCC_Topo_Index[SCC_Topo[i
        ]]=i;
}
```

```
//Main Function for SCC.
void SCC_Compress()
{
    //Initializes Containers (for multiple testcases)
    while(!stk.empty()) stk.pop(); SCC_Nodes.clear(); SCC_time
        =0;
    for(int i=1; i<=num_vertex; i++) { disc[i]=-1; In_stk[i]=0;
         SCC_adj[i].clear(); }

    //SCC_DFS from vertex unvisited
    for(int i=1; i<=num_vertex; i++) if(disc[i]==-1) SCC_DFS(i)
        ;

    //To add edges to SCC Compressed
    for(int i=1; i<=num_vertex; i++)
    {
        for(auto x: adj[i])
        {
            int su = SCC_par[i], sv = SCC_par[x];   //Head of
                SCC comp
            if(su==sv) continue;                    //Belong to
                 same comp
            SCC_adj[su].push_back(sv);              //Can
                 possible add multiple edges between 2 vertex
            SCC_incoming[sv].push_back(su);
        }
    }

    //Topologically sorts the newly for DAG of SCC
    SCC_Toposort();
}
```

SegtreeKaustubhAddinarangeMax.cpp

Description: Segtree kaustubh (add in a range, max in a range, cnt) d41d8c, 78 lines

```
struct segtree
{
    vector<pair<int,int>> maxa;    //Max, cnt of max??
    vector<int> ops;

    void init(int n)
    {
        int curr=1;
        while(curr<n){curr*=2;}

        for(int i=0;i<2*curr;i++)
            {
                maxa.push_back({0,0});
                ops.push_back(0);
            }
    }

    void init_sp(int lx,int rx,int curr)
    {
        if(rx-lx==1){maxa[curr]={0,1};return;}
        maxa[curr]={0,rx-lx};
        int m=(lx+rx)/2;
        init_sp(lx,m,2*curr+1);
        init_sp(m,rx,2*curr+2);
    }

    pair<int,int> comb(pair<int,int> a,pair<int,int> b)
    {
        int maxs=max(a.first,b.first);
        int cnt=0;
        if(a.first==maxs){cnt+=a.second;}
        if(b.first==maxs){cnt+=b.second;}

        return {maxs,cnt};
    }

    void propogate(int lx,int rx,int curr)
    {
        if(rx-lx==1){return;}
        maxa[2*curr+1].first+=ops[curr];
        maxa[2*curr+2].first+=ops[curr];
        ops[2*curr+1]+=ops[curr];
        ops[2*curr+2]+=ops[curr];
        ops[curr]=0;
        return;
    }

    void update(int lx,int rx,int curr,int l,int r,int val)
    {
        if(lx>=l && rx<=r)
            {
                maxa[curr].first+=val;
                ops[curr]+=val;
                return;
            }

        if(l>=rx || lx>=r){return;}
        propogate(lx,rx,curr);
        int m=(lx+rx)/2;
        update(lx,m,2*curr+1,l,r,val);
        update(m,rx,2*curr+2,l,r,val);
        maxa[curr]=comb(maxa[2*curr+1],maxa[2*curr+2]);

    }

    pair<int,int> getans(int lx,int rx,int curr,int l,int r)
    {
        if(lx>=l && rx<=r){return maxa[curr];}
        if(lx>=r || l>=rx){return {-1,0};}

        propogate(lx,rx,curr);
        int m=(lx+rx)/2;
        return comb(getans(lx,m,2*curr+1,l,r),getans(m,rx,2*
             curr+2,l,r));
    }
};
```

segtree.cpp

Description: segtree kaustubh d41d8c, 47 lines

```
struct segtree
{
    vector<int> v;
    void init(int n)
    {
        int curr = 1;
        while (curr < n)
        {
            curr = curr * 2;
        }
        for (int i = 0; i < 2 * curr; i++)
        {
            v.push_back(0);
        }
    }
    void update(int lx, int rx, int curr, int i, int z)
    {
        if (rx - lx == 1)
        {
            v[curr] = z;
            return;
        }
        int m = (lx + rx) / 2;
        if (i < m)
        {
            update(lx, m, 2 * curr + 1, i, z);
        }
        else
        {
            update(m, rx, 2 * curr + 2, i, z);
        }
        v[curr] = min(v[2 * curr + 1], v[2 * curr + 2]);
    }
    int getans(int lx, int rx, int curr, int l, int r)
    {
        if (lx >= l && rx <= r)
        {
            return v[curr];
        }
        if (l >= rx || lx >= r)
        {
            return 1e12;
        }
        int m = (lx + rx) / 2;
        return min(getans(lx, m, 2 * curr + 1, l, r), getans(m,
             rx, 2 * curr + 2, l, r));
    }
};
```

segtree2d.cpp

Description: Segtree 2D d41d8c, 168 lines

```
vector<vector<pair<int, int>>> zz;
vector<pair<int, int>> merger(vector<pair<int, int>> &a, vector
    <pair<int, int>> &b)
{
    int n = a.size();
    int m = b.size();
    int curra = 0;
    int currb = 0;
    vector<pair<int, int>> v;
    while (curra < n && currb < m)
    {
        if (a[curra] < b[currb])
        {
            v.push_back(a[curra]);
            curra++;
        }
        else
        {
            v.push_back(b[currb]);
            currb++;
        }
    }
    while (curra < n)
    {
        v.push_back(a[curra]);
        curra++;
    }
    while (currb < m)
    {
        v.push_back(b[currb]);
        currb++;
    }
    return v;
}
struct segtree
{
    vector<pair<int, int>> present;
    vector<int> v;
    int sizes;
    int ind_to_upd(int y, int i)
    {
        return lower_bound(present.begin(), present.end(),
             make_pair(y, i)) - pre sent.begin();
    }
    int ind_just_g(int y, int i)
    {
        return lower_bound(present.begin(), present.end(),
             make_pair(y, i)) - pre sent.begin();
    }
    int ind_just_l(int y, int i)
    {
        // returns 1 more as required by segtree
        return (lower_bound(present.begin(), present.end(),
             make_pair(y + 1, i)) - present.begin());
    }
    void init(int n)
    {
        int curr = 1;
        while (curr < n)
        {
            curr = curr * 2;
        }
        for (int i = 0; i < 2 * curr; i++)
        {
            v.push_back(0);
        }
        sizes = n;
    }
    void update(int lx, int rx, int curr, int i, int z)
    {
        if (rx - lx == 1)
        {
            v[curr] = z;
            return;
        }
        int m = (lx + rx) / 2;
        if (i < m)
        {
            update(lx, m, 2 * curr + 1, i, z);
        }
        else
        {
            update(m, rx, 2 * curr + 2, i, z);
        }
        v[curr] = min(v[2 * curr + 1], v[2 * curr + 2]);
    }
    int getans(int lx, int rx, int curr, int l, int r)
    {
        if (lx >= l && rx <= r)
        {
            return v[curr];
        }
        if (l >= rx || lx >= r)
        {
            return 1e12;
        }
        int m = (lx + rx) / 2;
        return min(getans(lx, m, 2 * curr + 1, l, r), getans(m,
              rx, 2 * curr + 2, l, r));
    }
};
struct segtree_2d
{
    vector<struct segtree> v;
    void build(int curr, int lx, int rx)
    {
        // cout<<curr<<" "<<lx<<" "<<rx<<"\n";
        if (rx - lx == 1)
        {
            v[curr].present = zz[lx];
            v[curr].init(zz[lx].size());
            re turn;
        }
        int m = (lx + rx) / 2;
        build(2 * curr + 1, lx, m);
        build(2 * curr + 2, m, rx);
        v[curr].present = merger(v[2 * curr + 1].present, v[2 *
              curr + 2].prese nt);
        v[curr].init(v[curr].present.size());
    }
    void real_init(int n)
    {
        int curr = 1;
        while (curr < n)
        {
            curr = curr * 2;
        }
        for (int i = 0; i < 2 * curr; i++)
        {
            struct segtree x;
            x.init(1);
            v.push_back(x);
        }
    }
    void real_update(int lx, int rx, int ix, int i, int y, int
         z, int curr)
    {
        int zz = v[curr].ind_to_upd(y, i);
        v[curr].update(0, v[curr].sizes, 0, zz, z);
        if (rx - lx == 1)
        {
            return;
        }
        int m = (lx + rx) / 2;
        if (ix < m)
        {
            real_update(lx, m, ix, i, y, z, 2 * curr + 1);
        }
        else
        {
            real_update(m, rx, ix, i, y, z, 2 * curr + 2);
        }
    }
    int real_getans(int lx, int rx, int ixl, int ixr, int i,
         int yl, int yr, int curr)
    {
        if (lx >= ixl && rx <= ixr)
        {
            // cout<<"FFG\n";
            int lly = v[curr].ind_just_g(yl, -1e12);
            int rry = v[curr].ind_just_l(yr, -1e12);
            // cout<<lly<<" "<<rry<<"\n";
            // for(auto x:v[curr].present){cout<<x.first<<"
            "<<x.second<<" HH\n ";}
                return v[curr]
                    .getans(0, v[curr].sizes, 0, lly, rry);
        }
        // cout<<"HIH\n";
        if (ixl >= rx || lx >= ixr)
        {
            return 1e12;
        }
        int m = (lx + rx) / 2;
        return min(real_getans(lx, m, ixl, ixr, i, yl, yr, 2 *
             curr + 1), real_getans(m, rx, ixl, ixr, i, yl, yr,
              2 * curr + 2));
    }
};
```

segtreelazyaryan.cpp

Description: Segtree Lazy Aryan

d41d8c, 141 lines

```
struct Node
{
    ll val;
    ll len;
};
template <typename ValType, typename OpType>
struct segtree
{
    int treesize;
    vector<ValType> values;
    vector<OpType> operations;
    ValType IDENTITY_ELEMENT;
    OpType NO_OPERATION;      // No operation for modify op
    ValType (*calc_op)(ValType, ValType);
    OpType (*composite)(OpType, OpType);
    ValType (*modify_op)(ValType, OpType);
    void build(int x, int l, int r) // build is not called,
         call it manually
    {
        if (l == r)
            // be careful l can be out of bound of
            vector you are assigning values from
            {
                ValType res;
                res.val = 0;
                res.len = 1;
                values[x] = res; // change to required
                values in build and dont forget to build if
                     required, its not included in init return;
            }
        int mid = (l + r) / 2;
        build(2 * x + 1, l, mid);
        build(2 * x + 2, mid + 1, r);
        values[x] =
            calc_op(values[2 * x + 1], values[2 * x + 2]);
    }
    segtree(int n, ValType identity, OpType noop, ValType (*cop
         )(ValType, ValType), OpType (*comp)(OpType, OpType),
         ValType (*modiop)(ValType, OpType))
    {
        IDENTITY_ELEMENT = identity;
        NO_OPERATION = noop;
        calc_op = cop;
        composite = comp;
        modify_op = modiop;
        init(n);
        build(0, 0, treesize - 1);
    }
    void init(int n) // Allocate space and initialize values
         for seg tree
    {
        // included in build function
        treesize = 1;
        while (treesize < n)
        {
            treesize *= 2;
        }
        operations.assign(2 * treesize, NO_OPERATION); //
             Initial state values dont forget to change
        values.assign(2 * treesize, IDENTITY_ELEMENT);
    }
    OPERATIONS
        inline void
        apply_modify_op(ValType &a, OpType b)
    {
        a = modify_op(a, b);
    }
    inline void propagate(int x, int l, int r)
    {
        if (x >= treesize - 1)
            return;
        operations[2 * x + 1] =
            composite(operations[x], operations[2 * x + 1]);
        operations[2 * x + 2] =
            composite(operations[x], operations[2 * x + 2]);
        apply_modify_op(values[2 * x + 1], operations[x]);
        apply_modify_op(values[2 * x + 2], operations[x]);
        operations[x] = NO_OPERATION;
    }
    void modify(OpType val, int ql, int qr, int curr, int l,
         int r)
    {
        propagate(curr, l, r);
        if (l > qr || ql > r)
        {
            return;
        }
        if (l >= ql && r <= qr)

        {
            operations[curr] =
                composite(val, operations[curr]);
            apply_modify_op(values[curr], val);
            return;
        }
        int mid = (l + r) / 2;
        modify(val, ql, qr, 2 * curr + 1, l, mid);
        modify(val, ql, qr, 2 * curr + 2, mid + 1, r);
        values[curr] =
            calc_op(values[2 * curr + 1], values[2 * curr + 2])
                ;
    }
    void modify(OpType val, int ql, int qr)
    {
        if (ql > qr)
            return;
        modify(val, ql, qr, 0, 0, treesize - 1);
    }
    ValType calc(int ql, int qr, int curr, int l, int r)
    {
        propagate(curr, l, r);
        if (l > qr || ql > r)
            return IDENTITY_ELEMENT;
        if (l >= ql && r <= qr)
        {
            return values[curr];
        }
        int mid = (l + r) / 2;
        ValType a = calc(ql, qr, 2 * curr + 1, l, mid);
        ValType b = calc(ql, qr, 2 * curr + 2, mid + 1, r);
        return calc_op(a, b);
    }
    ValType calc(int ql, int qr)
    {
        return calc(ql, qr, 0, 0, treesize - 1);
    }
};
Node calc_op(Node a, Node b)
{
    Node res;
    res.val = a.val + b.val;
    res.len = a.len + b.len;
    return res;
}
Node modify_op(Node a, ll b)
{
    if (b == 2)
        return a;
    Node res;
    res.val = b * a.len;
    res.len = a.len;
    return res;
}
ll composite(ll a, ll b)
{
    if (a == 2)
        return b;
    return a;
}
```

Seive.cpp

Description: Seive

d41d8c, 27 lines

```
vector<ll> primes;
vector<ll> leastPrime;
void sieveofErath(int mxSieve)
{
    leastPrime.assign(mxSieve + 1, 0);
    leastPrime[1] = -1;
    ll p;
    for (p = 2; p <= mxSieve; p++)
    {
        if (!leastPrime[p])
        {
            leastPrime[p] = p;
            primes.pb(p);
        }
        for (int j = 0; j < primes.size() &&
                        (primes[j] * p) <= mxSieve;
             j++)
        {
            leastPrime[primes[j] * p] =
                primes[j];
            if (!(p % primes[j]))
            {
                break;
            }
        }
    }
}
```

SuffixArray.cpp

Description: SuffixArray

d41d8c, 118 lines

```
struct suffixArray
{
    int n;
    int lgN;
    string str;
    vector<vector<int>> eqclasses;
    vector<int> arraySuffix;
    vector<int> lcparray;
    void counting_sort(vector<pair<int, int>> &C, vector<pair<
         int, int>> &temp)
    {
        int n = C.size();
        vector<int> pos(n);
        int arr[n];
        fori(i, n) { arr[i] = 0; }
        fori(i, n) { (arr[C[i].first])++; }
        pos[0] = 0;
        fori(i, n - 1) { pos[i + 1] = pos[i] + arr[i]; }
        fori(i, n)
        {
            temp[pos[C[i].first]] = C[i];
            pos[C[i].first]++;
        }
        return;
    }
    void suffixarr(vector<int> &j)
    {
        str += " ";
        n++;
        vector<pair<char, int>> init(n);
        fori(i, n) { init[i] = {str[i], i}; }
        sort(init.begin(), init.end());
        vector<int> value(n), j_new(n), value_new(n);
        int k = 0;
        fori(i, n) { j[i] = init[i].second; }
        value[j[0]] = 0;
        fori(i, n - 1)
        {
            if (init[i].first == init[i + 1].first)
            {
                value[j[i + 1]] = value[j[i]];
                eqclasses[k][j[i + 1]] = value[j[i]];
            }
            else
            {
                value[j[i + 1]] = value[j[i]] + 1;
                eqclasses[k][j[i + 1]] = value[j[i]] + 1;
            }
        }
        int t;
        vector<pair<int, int>> C(n), C_new(n);
        while ((1 << k) < n)
        {
            t = 1 << k;
            fori(i, n) { C[i] = {value[(j[i] - t + n) % n], (j[
                 i] - t + n) % n}; }
            counting_sort(C, C_new);
            value_new[C_new[0].second] = 0;
            fori(i, n - 1)
            {
                if (C_new[i].first == C_new[i + 1].first &&
                    value[(C_new[i].second + t) % n] == value[(
                        C_new[i + 1].second + t) % n])
                {
                    value_new[(C_new[i + 1]).second] =
                        value_new[(C_new[i]).seco nd];
                    eqclasses[k + 1][(C_new[i + 1]).second] =
                        value_new[(C_new[i]).s econd];
                }
                else
                {
                    value_new[(C_new[i + 1]).second] =
                        value_new[(C_new[i]).seco nd] + 1;
                    eqclasses[k + 1][(C_new[i + 1]).second] =
                        value_new[(C_new[i]).s econd] + 1;
                }
            }
            fori(i, n) { j[i] = C_new[i].second; }
            swap(value, value_new);
            k++;
        }
        str.pop_back();
        n--;
        return;
    }
    void lcp()
    {
        str += ' ';
        n++;
        vector<int> invsuff(n);
        fori(i, n)
        {
            invsuff[arraySuffix[i]] = i;
        }
        int k = 0;
        fori(i, n - 1)
        {
            while ((arraySuffix[invsuff[i] - 1] + k < n) && (i
                 + k < n) && (str[i + k] == str[arraySuffix[
                 invsuff[i] - 1] + k]))
            {
                k++;
            }
            lcparray[invsuff[i] - 1] = k;
            k = max(k - 1, 0);
        }
        str.pop_back();
        n--;
    }
    suffixArray(string input)
    {
        n = input.size();
        str = input;
        int lgN = 0;
        int cur = 1;
        while (cur < n)
        {
            cur = (cur << 1);
            lgN++;
        }
        eqclasses.assign(lgN + 2, vector<int>(n + 1, 0));
        arraySuffix.assign(n + 1, 0);
        lcparray.assign(n, 0);
        suffixarr(this->arraySuffix);
        lcp();
    }
};
```

Trie.cpp

Description: Trie

d41d8c, 84 lines

```
struct Trie
{
    static const int B = 31;
    struct node
    {
        node *nxt[2];

        int sz;
        node()
        {
            nxt[0] = nxt[1] = NULL;
            sz = 0;
        }
    } *root;
    Trie()
    {
        root = new node();
    }
    void insert(int val)
    {
        node *cur = root;
        cur->sz++;
        for (int i = B - 1; i >= 0; i--)
        {
            int b = val >> i & 1;
            if (cur->nxt[b] == NULL)
                cur->nxt[b] = new node();
            cur = cur->nxt[b];
            cur->sz++;
        }
    }
    int query(int x, int k)
    { // number of values s.t. val ^ x < k node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--)
        {
            if (cur == NULL)
                break;
            int b1 = x >> i & 1, b2 = k >> i & 1;
            if (b2 == 1)
            {
                if (cur->nxt[b1])
                    ans += cur->nxt[b1]->sz;
                cur = cur->nxt[!b1];
            }
            else
                cur = cur->nxt[b1];
        }
        return ans;
    }
    int get_max(int x)
    { // returns maximum of val ^ x node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--)
        {
            int k = x >> i & 1;
            if (cur->nxt[!k])
                cur = cur->nxt[!k], ans <<= 1, ans++;
            else
                cur = cur->nxt[k], ans <<= 1;
        }
        return ans;
    }
    int get_min(int x)
    { // returns minimum of val ^ x node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--)
        {
            int k = x >> i & 1;
            if (cur->nxt[k])
                cur = cur->nxt[k], ans <<= 1;
            else
                cur = cur->nxt[!k], ans <<= 1, ans++;
        }
        return ans;
    }
    void del(node *cur)
    {
        for (int i = 0; i < 2; i++)
            if (cur->nxt[i])
                del(cur->nxt[i]);
        delete (cur);
    }
} t;
```

ZAlgo.cpp

Description: Z Algo

d41d8c, 35 lines

```
template <typename T>

struct ZAlgo
{
    vector<int> z;
    // z[i] -> maximum length of a substring starting at i, and
          also a prefix
    ZAlgo(T a)
    {
        int n = a.size();
        z.assign(n, 0);
        z[0] = n;
        for (int i = 1, l = 0, r = 0; i < n; i++)
        {
            if (i >= r)
            {
                int c = i, d = 0;
                while (c < n && a[c] == a[d])
                    c++, d++;
                l = i, r = c;
                z[i] = d;
            }
            else
            {
                // l..i..r i..r-> i-l..r-l -> 0..r-i
                int offset = min(z[i - l], r - i);
                int c = i + offset, d = offset;
                while (c < n && a[c] == a[d])
                    c++, d++;
                if (c > r)
                    l = i, r = c;
                z[i] = d;
            }
        }
    }
};
```

hld.h

d41d8c, 54 lines

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;

int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
}

void decompose(int v, int h, vector<vector<int>> const& adj) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
    }
}

void init(vector<vector<int>> const& adj) {
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;

    dfs(0, adj);
    decompose(0, 0, adj);
}
int query(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);
        int cur_heavy_path_max = segment_tree_query(pos[head[b
            ]], pos[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max = segment_tree_query(pos[a], pos[b
        ]);
    res = max(res, last_heavy_path_max);
    return res;
}
```

VirtualTree.h

<bits/stdc++.h> d41d8c, 114 lines

```
using namespace std;

const int N = 3e5 + 9;

vector<int> g[N];
int par[N][20], dep[N], sz[N], st[N], en[N], T;
void dfs(int u, int pre) {
  par[u][0] = pre;
  dep[u] = dep[pre] + 1;
  sz[u] = 1;
  st[u] = ++T;
  for (int i = 1; i <= 18; i++) par[u][i] = par[par[u][i - 1]][
      i - 1];
  for (auto v : g[u]) {
    if (v == pre) continue;
    dfs(v, u);
    sz[u] += sz[v];
  }
  en[u] = T;
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = 18; k >= 0; k--) if (dep[par[u][k]] >= dep[v]) u
      = par[u][k];
  if (u == v) return u;
  for (int k = 18; k >= 0; k--) if (par[u][k] != par[v][k]) u =
       par[u][k], v = par[v][k];
  return par[u][0];
}
int kth(int u, int k) {
  for (int i = 0; i <= 18; i++) if (k & (1 << i)) u = par[u][i
      ];
  return u;
}
int dist(int u, int v) {
  int lc = lca(u, v);
  return dep[u] + dep[v] - 2 * dep[lc];
}
int isanc(int u, int v) {
  return (st[u] <= st[v]) && (en[v] <= en[u]);
}
vector<int> t[N];
// given specific nodes, construct a compressed directed tree
     with these vertices(if needed some other nodes included)
// returns the nodes of the tree
// nodes.front() is the root
// t[] is the specific tree
vector<int> buildtree(vector<int> v) {
  // sort by entry time
  sort(v.begin(), v.end(), [](int x, int y) {
    return st[x] < st[y];
  });
  // finding all the ancestors, there are few of them
  int s = v.size();
  for (int i = 0; i < s - 1; i++) {
    int lc = lca(v[i], v[i + 1]);
    v.push_back(lc);
  }
  // removing duplicated nodes
  sort(v.begin(), v.end());
  v.erase(unique(v.begin(), v.end()), v.end());
  // again sort by entry time
  sort(v.begin(), v.end(), [](int x, int y) {
    return st[x] < st[y];
  });
  stack<int> st;
  st.push(v[0]);
  for (int i = 1; i < v.size(); i++) {
    while (!isanc(st.top(), v[i])) st.pop();
    t[st.top()].push_back(v[i]);
    st.push(v[i]);
  }
  return v;
}
int ans;
int imp[N];
int yo(int u) {
  vector<int> nw;
  for (auto v : t[u]) nw.push_back(yo(v));
  if (imp[u]) {
    for (auto x : nw) if (x) ans++;
    return 1;
  } else {
    int cnt = 0;
    for (auto x : nw) cnt += x > 0;
    if (cnt > 1) {
      ans++;
      return 0;
    }
    return cnt;
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int i, j, k, n, m, q, u, v;
  cin >> n;
  for (i = 1; i < n; i++) cin >> u >> v, g[u].push_back(v), g[v
      ].push_back(u);
  dfs(1, 0);
  cin >> q;
  while (q--) {
    cin >> k;
    vector<int> v;
    for (i = 0; i < k; i++) cin >> m, v.push_back(m), imp[m] =
        1;
    int fl = 1;
    for (auto x : v) if (imp[par[x][0]]) fl = 0;
    ans = 0;
    vector<int> nodes;
    if (fl) nodes = buildtree(v);
    if (fl) yo(nodes.front());
    if (!fl) ans = -1;
    cout << ans << '\n';
    // clear the tree
    for (auto x : nodes) t[x].clear();
    for (auto x : v) imp[x] = 0;
  }
  return 0;
}
// https://codeforces.com/contest/613/problem/D
```

persistantCentroidDecomposition.h

<bits/stdc++.h> d41d8c, 165 lines

```
using namespace std;

const int N = 2e5 + 9, M = N * 2 + N * 19 * 2;

vector<pair<int, int>> g[N * 2], G[N];
inline void add_edge(int u, int v, int w) {
  g[u].push_back({v, w});
}
int T;
void binarize(int u, int p = 0) {
  int last = 0, tmp = 0;
  for (auto e : G[u]) {
    int v = e.first, w = e.second;
    if (v == p) continue;
    ++tmp;
    if (tmp == 1) {
      add_edge(u, v, w);
      add_edge(v, u, w);
      last = u;
    } else if (tmp == ((int) G[u].size()) - (u != 1)) {
      add_edge(last, v, w);
      add_edge(v, last, w);
    } else {
      T++;
      add_edge(last, T, 0);
      add_edge(T, last, 0);
      last = T;
      add_edge(T, v, w);
      add_edge(v, T, w);
    }
  }
  for (auto e : G[u]) {
    int v = e.first;
    if (v == p) continue;
    binarize(v, u);
  }
}
int sz[N * 2];
int tot, done[N * 2], cenpar[N * 2];
void calc_sz(int u, int p) {
  tot ++;
  sz[u] = 1;
  for (auto e : g[u]) {
    int v = e.first;
    if(v == p || done[v]) continue;
    calc_sz(v, u);
    sz[u] += sz[v];
  }
}
int find_cen(int u, int p) {
  for (auto e : g[u]) {
    int v = e.first;
    if(v == p || done[v]) continue;
    else if(sz[v] > tot / 2) return find_cen(v, u);
  }
  return u;
}
long long d[20][N * 2];
void yo(int u, int p, long long nw, int l) {
  d[l][u] = nw;
  for(auto e : g[u]) {
    int v = e.first;
    long long w = e.second;
    if (v == p || done[v]) continue;
    yo(v, u, nw + w, l);
  }
}
int st[N * 2], en[N * 2], DT;
struct node {
  vector<int> ct; //adjacent edges in centroid tree
  int level = 0, id = 0, cnt = 0;
  long long sum = 0, parsum = 0;
} t[M];
int decompose(int u, int p = 0, int l = 0) {
  tot = 0;
  calc_sz(u, p);
  int cen = find_cen(u, p);
  cenpar[cen] = p;
  done[cen] = 1;
  u = cen;
  st[u] = ++DT;
  t[u].id = u;
  t[u].level = l;
  yo(u, p, 0, l);
  for (auto e : g[u]) {
    int v = e.first;
    if(v == p || done[v]) continue;
    int x = decompose(v, u, l + 1);
    t[u].ct.push_back(x);
  }
  en[u] = DT;
  return u;
}
int insub(int r, int c) {
  r = t[r].id, c = t[c].id;
  return st[r] <= st[c] && en[c] <= en[r];
}
int upd(int cur, int u) { //update node u in cur tree
  T++;
  assert(T < M);
  t[T] = t[cur];
  cur = T;
  t[cur].cnt++;
  t[cur].sum += d[t[cur].level][u];
  for(auto &v : t[cur].ct) if(insub(v, u)) {
      v = upd(v, u);
      t[v].parsum += d[t[cur].level][u];
    }
  return cur;
}
long long query(int cur, int u) { //query on cur tree
  long long ans = 0;
  while (t[cur].id != t[u].id) {
    int v = 0;
    for (auto x : t[cur].ct) if(insub(x, u)) v = x;
    assert(v);
    ans += d[t[cur].level][t[u].id] * (t[cur].cnt - t[v].cnt);
    ans += t[cur].sum - t[v].parsum;
    cur = v;
  }
  ans += t[cur].sum;
  return ans;
}
int a[N], root[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, q;
  cin >> n >> q;
  for(int i = 1; i <= n; i++) cin >> a[i];
  for (int i = 1; i < n; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    G[u].push_back({v, w});
    G[v].push_back({u, w});
  }
  T = n;
  binarize(1);
  root[0] = decompose(1);
  for(int i = 1; i <= n; i++) root[i] = upd(root[i - 1], a[i]);
  long long ans = 0;
  const int mod = 1 << 30;
  while(q--) {
    int ty;
    cin >> ty;
    if(ty == 1) {
      int l, r, u;
      cin >> l >> r >> u;
      l ^= ans;
      r ^= ans;
      u ^= ans;
      ans = query(root[r], u) - query(root[l - 1], u);
      cout << ans << '\n';
      ans %= mod;
    } else {
      int x;
      cin >> x;
      x ^= ans;
      root[x] = upd(root[x - 1], a[x + 1]);
      swap(a[x], a[x + 1]);
    }
  }
  return 0;
}
//https://codeforces.com/contest/757/problem/G
```

MillerRabin.h

<bits/stdc++.h> d41d8c, 92 lines

```
// C++ program Miller-Rabin primality test
using namespace std;

// Utility function to do modular exponentiation.
// It returns (x^y) % p
int power(int x, unsigned int y, int p)
{
    int res = 1;      // Initialize result
    x = x % p;  // Update x if it is more than or
                // equal to p
    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
            res = (res*x) % p;

        // y must be even now
        y = y>>1; // y = y/2
        x = (x*x) % p;
    }
    return res;
}

// This function is called for all k trials. It returns
// false if n is composite and returns true if n is
// probably prime.
// d is an odd number such that  d*2 = n-1
// for some r >= 1
bool miillerTest(int d, int n)
{
    // Pick a random number in [2..n-2]
    // Corner cases make sure that n > 4
    int a = 2 + rand() % (n - 4);

    // Compute a^d % n
    int x = power(a, d, n);

    if (x == 1  || x == n-1)
       return true;

    // Keep squaring x while one of the following doesn't
    // happen
    // (i)   d does not reach n-1
    // (ii)  (x^2) % n is not 1
    // (iii) (x^2) % n is not n-1
    while (d != n-1)
    {
        x = (x * x) % n;
        d *= 2;

        if (x == 1)      return false;
        if (x == n-1)    return true;
    }

    // Return composite
    return false;
}

// It returns false if n is composite and returns true if n
// is probably prime.  k is an input parameter that determines
// accuracy level. Higher value of k indicates more accuracy.
bool isPrime(int n, int k)
{
    // Corner cases
    if (n <= 1 || n == 4)  return false;
    if (n <= 3) return true;

    // Find r such that n = 2^d * r + 1 for some r >= 1
    int d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    // Iterate given number of 'k' times
    for (int i = 0; i < k; i++)
         if (!miillerTest(d, n))
              return false;

    return true;
}

// Driver program
int main()
{
    int k = 4;  // Number of iterations

    cout << "All primes smaller than 100: \n";
    for (int n = 1; n < 100; n++)
       if (isPrime(n, k))
          cout << n << " ";

    return 0;
}
```

```
  }
};
```

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 $\mathcal{O}(\log N)$.

0fb462, 90 lines

```
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() {
    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() {
    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) {
    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) {
    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;
  }
};
```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

"../data-structures/UnionFindRollback.h" 39e620, 60 lines

```
struct Edge { int a, b; ll w; };
struct 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) {
        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};
}
```

7.8 Math

7.8.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--`, `mat[b][b]++` (and `mat[b][a]--`, `mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

7.8.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k).$$

Geometry (8)

8.1 Geometric primitives

Point.h

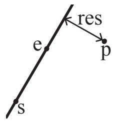
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
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 << ")"; }
};
```

lineDistance.h

Description:
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.

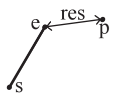


"Point.h" f6bf6b, 4 lines

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

SegmentDistance.h

Description:
Returns the shortest distance between point p and the line segment from point s to e.



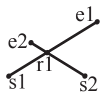
Usage: Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;

"Point.h" 5c88f4, 6 lines

```
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;
}
```

SegmentIntersection.h

Description:
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.



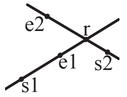
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;

"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
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)};
}
```

lineIntersection.h

Description:
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.



Usage: auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;

"Point.h" a01f81, 8 lines

```
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};
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 ⇔ 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;

"Point.h" 3af81c, 9 lines

```
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);
}
```

OnSegment.h

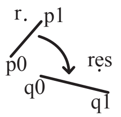
Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.

"Point.h" c597e8, 3 lines

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

linearTransformation.h

Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.



"Point.h" 03a306, 6 lines

```
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();
}
```

Angle.h

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

0f0602, 35 lines

```
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)};
}
```

8.2 Circles

CircleIntersection.h

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 84d6d3, 11 lines

```
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;
}
```

CircleTangents.h

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.

"Point.h" b0153d, 13 lines

```
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;
}
```

CirclePolygonIntersection.h

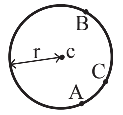
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

```
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 = p + d * t;
    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;
}
```

circumcircle.h

Description:
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.



"Point.h" 1caa3a, 9 lines

```
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;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

```
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};
}
```

8.3 Polygons

InsidePolygon.h

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.
Usage: vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);
Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
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;
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

```
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;
}
```

PolygonCenter.h

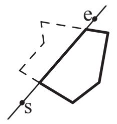
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
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;
}
```

PolygonCut.h

Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.



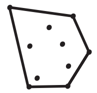
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
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();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

ConvexHull.h

Description:
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.
Time: $\mathcal{O}(n \log n)$



"Point.h" 310954, 13 lines

```
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])};
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h" c571b8, 12 lines

```
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;
}
```

PointInsideHull.h

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.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
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;
}
```

LineHullIntersection.h

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: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. 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. `extrVertex` returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(\log n)$

"Point.h" 7cf45b, 39 lines

```
#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;
}
```

8.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

```
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;
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

"Point.h" bac5b0, 63 lines

```
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);
  }
};
```

FastDelaunay.h

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], . . . }, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h" eefdf5, 88 lines

```
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;
}
```

8.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards.

3058c3, 6 lines

```
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;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

8058ae, 32 lines

```
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;
  }
};
```

3dHull.h

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: $\mathcal{O}(n^2)$

"Point3D.h" 5b45fc, 49 lines

```
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;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_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.

611f07, 8 lines

```
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);
}
```

Strings (9)

KMP.h

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: $\mathcal{O}(n)$

d4375c, 16 lines

```
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;
}
```

Zfunc.h

Description: z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

Time: $\mathcal{O}(n)$

ee09e2, 12 lines

```
vi Z(const string& S) {
  vi z(sz(S));
```