Federal University of Itajubá
Notebook - Maratona de Programação

André Marcos Leifeld Raicoski
Bruno Said Alves de Souza
Stéfany Coura Coimbra

Itajubá - 2022

"Revelemo-nos mais por atos que por palavras..."

Programação Competitiva Básica

Funções Úteis

random_shuffle: embaralha aleatoriamente o contêiner passado como parâmetro
next_permutation: Gera as permutações dos elementos de um vetor, o vetor deve está ordenado inicialmente.
transform: Aplica uma função em todo o vetor
lower_bound: pertence a biblioteca algorithm, retorna a posição do primeiro elemento maior ou igual ao valor.
upper_bound: pertence a biblioteca algorithm, retorna a posição do primeiro elemento maior ao valor.

Otimização de Entrada e Saída:

Método 1: Se utilizar `scanf/printf` não precisa.

```
ios_base::sync_with_stdio(0); cin.tie(0);
```

Método 2: Melhora se utilizar `getchar_unlocked()`

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

using namespace std;

void fastscan (int& num) {
  register char c;
  bool neg = false;

  c = getchar();
  if (c == '-') {
    neg = true;
    c = getchar();
  }

  num = 0;

  while ('0' <= c && c <= '9') {
    num = 10*num + c - '0';
    c = getchar();
  }

  if (neg) num *= -1;
}
int main () {
  int n, k, t, cnt = 0;

  fastscan(n);
  fastscan(k);

  while (n--) {
    fastscan(t);

    if (t % k == 0) cnt++;
  }

  cout << cnt << '\n';
}
```

Entrada e Saída por Arquivo Externo:

```
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
```

Precisão de Casas Decimais:

Método 1:

```
cout.precision(3);
cout.setf(ios::fixed);
```

Método 2:

```
cout << setprecision(3) << fixed;
```

Strings

Funções úteis:

`find(string s)`: primeira ocorrência do valor passado como parâmetro, caso não encontre retorna -1
`substr(início, tamanho)`: retorna a string que inicia no início e tem o tamanho especificado
`stoi(string)`: retorna o valor int dentro da string s
`stof(string s)`: retorna o valor float dentro da string s
`stod(string s)`: retorna o valor double dentro da string s
`stold(string s)`: retorna o valor long double dentro da string s
`to_string(valor)`: converte um valor para um string

LCS - Maior Substring Comum: $O(N.M)$

Iterativo:

```
int lcs (string const& a, string const& b) {
  int n = a.size(), m = b.size();
  vector<vector<int>> dp(n + 1, vector<int> (m + 1));
  int mx = 0;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
      if (a[i - 1] == b[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
        mx = max(mx, dp[i][j]);
      }
  return mx;
}
```

Recursivo:

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

#define MAX 50

using namespace std;

string s, a, b;
int tab[MAX][MAX];

int LCS (int i, int j) {
  if (i < 0 || j < 0) return 0;

  if (tab[i][j] != -1) return tab[i][j];

  if (a[i] == b[j]) {
    return tab[i][j] = 1 + LCS(i - 1, j - 1);
  }

  return tab[i][j] = 0;
}

int main () {
  int cnt = 0;
  while (getline(cin, s)) {
    cnt++;

    if (cnt % 2 == 1) {
      a = s;
    } else {
      b = s;

      int ans = 0;

      memset(tab, -1, sizeof(tab));

      for (int i = a.size() - 1; i >= 0; i--) {
        for (int j = b.size() - 1; j >= 0; j--) {
          ans = max(ans, LCS(i, j));
        }
      }

      cout << ans << "\n";

      cnt %= 2;
    }
  }
}
```

Distância entre Strings: $O(N.M)$

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

using namespace std;

#define PB push_back

#define N 1000
#define M 100

int tab[21][21];
string x, y, a[N];
vector<string> b[M];

int dist (int i, int j) {
  if (tab[i][j] != -1) return tab[i][j];

  if (i == x.size() && j == y.size()) return 0;
  if (i == x.size() && j < y.size())
    return tab[i][j] = 1 + dist(i, j + 1);
  if (i < x.size() && j == y.size())
    return tab[i][j] = 1 + dist(i + 1, j);
  if (x[i] == y[j])
    return tab[i][j] = dist(i + 1, j + 1);
  return tab[i][j] = 1 + min({dist(i, j + 1),
    dist(i + 1, j), dist(i + 1, j + 1)});
}

int main () {
  ios::sync_with_stdio(0); cin.tie(0);

  int n, m;
  cin >> n >> m;

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

  for (int i = 0; i < m; i++) {
    string s;
    cin >> s;

    for (int j = 0; j < n; j++) {
      x = s; y = a[j];

      memset(tab, -1, sizeof(tab));

      if (dist(0, 0) <= 2) {
        b[i].PB(a[j]);
      }
    }
  }

  for (int i = 0; i < m; i++) {
    if (!b[i].empty()) {
      cout << b[i][0];
      for (int j = 1; j < b[i].size(); j++) {
        cout << ' ' << b[i][j];
      }
    }
    cout << '\n';
  }
}
```

KMP: $O(n + m)$

Exemplo 1:

```
typedef vector<int> vi;

vi preprocess (vi pat)
{
  int i = 1;
  int len = 0;
  int m = pat.size();

  vi lps(m);
  lps[0] = 0;

  while (i < m)
  {
    if (pat[i] == pat[len])
    {
      lps[i] = ++len;
      i++;
    }
    else
    {
      if (len > 0) len = lps[len - 1];
      else
      {
        lps[i] = 0;
        i++;
      }
    }
  }

  return lps;
}
```

```
vi kmp (vi txt, vi pat)
{
  vi lps = preprocess(pat);

  int i = 0, j = 0;
  int n = txt.size();
  int m = pat.size();

  vi idx;

  while (i < n)
  {
    if (txt[i] == pat[j])
    {
      i++; j++;

      if (j == m)
      {
        idx.PB(i - m);
        j = lps[j - 1];
      }
    }
    else
      if (j > 0) j = lps[j - 1];
      else i++;
  }

  return idx;
}
```

Exemplo 2: Especiais

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

using namespace std;

vector<int> buildLPS (string s) {
  int i = 1, len = 0, m = s.size();
  vector<int> lps(m); lps[0] = 0;
  while (i < m) {
    if (s[i] == s[len]) {
      lps[i] = ++len; i++;
    } else {
      if (len > 0) len = lps[len - 1];
      else { lps[i] = 0; i++; }
    }
  }
  return lps;
}
```

```
}
int main () {
  ios::sync_with_stdio(0); cin.tie(0);
  string s; cin >> s;
  int ans = -1, n = s.size();
  for (int i = 0; i < n; i++) {
    vector<int> lps = buildLPS(s.substr(i, n - i));
    for (int j = (int) lps.size() - 1; j >= 0; j--) {
      if (lps[j] > 0) {
        ans = max(ans, j + 1); break;
      }
    }
  }

  cout << ans << '\n';
}
```

Estrutura de Dados

Union-Find:

```
int pai[MAX], altura[MAX], qtd[MAX];

void join (int x, int y) {
  x = find(x);
  y = find(y);

  if (x == y) return;

  if (altura[x] > altura[y]) swap(x, y);

  pai[x] = y;
  qtd[y] += qtd[x];

  if (altura[x] == altura[y]) altura[y]++;
}
```

```
int find (int x) {
  if (pai[x] == x) return x;
  return pai[x] = find(pai[x]);
}
```

Árvore de Segmentos

Exemplo 1 - Produto

```
int x[MAXN + 1];
int arv[4*MAXN];

void atualiza (int no, int i, int j, int pos, int val) {
  if (i == j) {
    x[pos] = val;
    arvore[no] = val;
  } else {
    int esq = 2*no;
    int dir = 2*no + 1;
    int meio = (i + j)/2;

    if (pos <= meio) atualiza(esq, i, meio, pos, val);
    else atualiza(dir, meio + 1, j, pos, val);

    arv[no] = arv[esq]*arv[dir];
  }
}
```

```
int consulta (int no, int i, int j, int a, int b) {
  if (b < i || j < a) return 2;
  if (a <= i && j <= b) return arv[no];

  int esq = 2*no;
  int dir = 2*no + 1;
  int meio = (i + j)/2;

  int r_esq = consulta(esq, i, meio, a, b);
  int r_dir = consulta(dir, meio + 1, j, a, b);

  if (r_esq == 2) return r_dir;
  if (r_dir == 2) return r_esq;
  return r_esq*r_dir;
}
```

Exemplo 2: Arranha Céu - RSQ

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

#define N 100000

int tree[4*N];

void update (int no, int i, int j, int pos, int val) {
  if (i == pos && pos == j) {
    tree[no] = val;
  } else {
    int left = 2*no;
    int right = 2*no + 1;
    int mid = ((i + j) >> 1);

    if (pos <= mid) update(left, i, mid, pos, val);
    else update(right, mid + 1, j, pos, val);

    tree[no] = tree[left] + tree[right];
  }
}

int query (int no, int i, int j, int a, int b) {
  if (j < a || b < i) return -1;

  if (a <= i && j <= b) return tree[no];

  int left = 2*no;
  int right = 2*no + 1;
  int mid = ((i + j) >> 1);

  int qLeft = query(left, i, mid, a, b);
  int qRight = query(right, mid + 1, j, a, b);

  if (qLeft == -1) return qRight;
  if (qRight == -1) return qLeft;

  return qLeft + qRight;
}

int main () {
  ios::sync_with_stdio(0); cin.tie(0);

  int n, q;
  cin >> n >> q;

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

    update(1, 1, n, i, a);
  }

  while (q--) {
    int op;
    cin >> op;

    if (op == 0) {
      int k, p;
      cin >> k >> p;

      update(1, 1, n, k, p);
    } else {
      int k;
      cin >> k;

      cout << query(1, 1, n, 1, k) << '\n';
    }
  }
}
```

Exemplo 3: iterativo

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

using namespace std;

#define N 1000

int n;
int tree[2*N];

int sum (int a, int b) {
  a += n; b += n;
  int s = 0;
  while (a <= b) {
    if (a%2 == 1) s += tree[a++];
    if (b%2 == 0) s += tree[b--];
    a /= 2; b /= 2;
  }
  return s;
}

void add (int k, int x) {
  k += n;
  tree[k] += x;
  for (k /= 2; k >= 1; k /= 2) {
    tree[k] = tree[2*k] + tree[2*k + 1];
  }
}

int main () {
  cin >> n;

  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;

    add(i, x);
  }

  int q;
  cin >> q;

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

    cout << sum(a, b) << '\n';
  }
}
```

Exemplo 3: Máximo e mínimo retornando valor e Índice - Problema: Baldes

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

typedef pair<int, int> pii;

#define F first
#define S second
#define INF 0x3f3f3f3f

#define N 100000

int n;
pii treeMin[2*N];
pii treeMax[2*N];

void add (int k, int x) {
  k += n;

  treeMin[k] = {min(treeMin[k].F, x), k - n};
  treeMax[k] = {max(treeMax[k].F, x), k - n};

  for (k /= 2; k >= 1; k /= 2) {
    treeMin[k] = min(treeMin[2*k], treeMin[2*k + 1]);
    treeMax[k] = max(treeMax[2*k], treeMax[2*k + 1]);
  }
}

pii queryMin (int a, int b) {
  a += n; b += n;

  pii r = {INF, -1};
  while (a <= b) {
    if (a % 2 == 1) r = min(r, treeMin[a++]);
    if (b % 2 == 0) r = min(r, treeMin[b--]);
    a /= 2; b /= 2;
  }

  return r;
}

pii queryMax (int a, int b) {
  a += n; b += n;

  pii r = {-INF, -1};
  while (a <= b) {
    if (a % 2 == 1) r = max(r, treeMax[a++]);
    if (b % 2 == 0) r = max(r, treeMax[b--]);
    a /= 2; b /= 2;
  }

  return r;
}

int main () {
  ios::sync_with_stdio(0); cin.tie(0);

  int m; cin >> n >> m;

  for (int i = n; i < 2*n; i++) {
    treeMin[i].F = INF;
    treeMax[i].F = -INF;
  }

  for (int i = 0; i < n; i++) {
    int x; cin >> x;

    add(i, x);
  }

  while (m--) {
    int t; cin >> t;

    if (t == 1) {
      int p, i; cin >> p >> i;

      i--;

      add(i, p);
    } else {
      int a, b; cin >> a >> b;

      a--; b--;

      pii minAB = queryMin(a, b);
      pii maxTmp =
        max(queryMax(a, minAB.S - 1), queryMax(minAB.S + 1, b));

      pii maxAB = queryMax(a, b);
      pii minTmp =
        min(queryMin(a, maxAB.S - 1), queryMin(minAB.S + 1, b));

      cout << max(maxTmp.F - minAB.F, maxAB.F - minTmp.F) << '\n';
    }
  }
```

Heavy-Light Decomposition (HLD):

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

using namespace std;

#define int long long

#define PB push_back

#define N 100001

int n;
int val[N];
vector<int> adj[N];

namespace seg {
  int sz, tree[2*N];

  void update (int k, int x) {
    k += sz;
    tree[k] = x;
    for (k /= 2; k >= 1; k /= 2) {
      tree[k] = (tree[2*k]|tree[2*k + 1]);
    }
  }

  int query (int a, int b) {
    a += sz; b += sz;
    int r = 0;
    while (a <= b) {
      if (a % 2 == 1) r |= tree[a++];
      if (b % 2 == 0) r |= tree[b--];
      a /= 2; b /= 2;
    }
    return r;
  }

  void build (int t, int v[]) {
    sz = t;
    for (int i = 0; i < sz; i++) {
      update(i, v[i]);
    }
  }
}

namespace hld {
  int t, sz[N], pos[N], fr[N], v[N], h[N];

  void dfs (int s, int e = -1) {
    sz[s] = 1;
    for (auto& u : adj[s]) if (u != e) {
      dfs(u, s);
      sz[s] += sz[u] + 1;
      if (u == adj[s][0] || sz[u] > sz[adj[s][0]])
        swap(u, adj[s][0]);
    }
  }

  void build_hld (int s, int e = -1) {
    pos[s] = t++;
    v[pos[s]] = val[s];
    for (auto u : adj[s]) if (u != e) {
      fr[u] = s;
      h[u] = (u == adj[s][0] ? h[s] : u);
      build_hld(u, s);
    }
  }

  void build (int s = 1) {
    memset(sz, -1, sizeof(sz));
    t = 0; h[s] = s; fr[s] = -1;
    dfs(s);
    build_hld(s);
    seg::build(t, v);
  }

  void update (int s, int x) {
    seg::update(pos[s], x);
  }

  int query (int a, int b) {
    int r = 0;
    while (h[a] != h[b]) {
      if (pos[a] < pos[b]) swap(a, b);
      r |= seg::query(pos[h[a]], pos[a]);
      a = fr[h[a]];
    }
    if (pos[a] < pos[b]) swap(a, b);
    r |= seg::query(pos[b], pos[a]);
    return r;
  }
}

int32_t main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  cin >> n;

  for (int i = 1; i <= n; i++) {
    int x;
    cin >> x;
    val[i] = (1ll << (x - 1));
  }

  for (int i = 0; i < n - 1; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].PB(v);
    adj[v].PB(u);
  }

  hld::build();

  int q;
  cin >> q;

  while (q--) {
    int t, u, v;
    cin >> t >> u >> v;
    if (t == 1) {
      int x = (1ll << (v - 1));
      hld::update(u, x);
    } else {
      int x = hld::query(u, v);
      cout << __builtin_popcountll(x) << '\n';
    }
  }
}
```

Árvore de Indexação Binária (BIT): array indexado a partir do 1 para melhor implementação.

Exemplo 1:

```
int n;
int arv[MAXN + 1];

void atualiza (int pos, int val) {
  while (pos <= n) {
    arv[pos] += val;
    pos += (pos & -pos);
  }
}

int consulta (int pos) {
  int soma = 0;

  while (pos > 0) {
    soma += arv[pos];
    pos -= (pos & -pos);
  }

  return soma;
}
```

Exemplo 2:

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

using namespace std;

#define N 1001

int n;
int tree[N];

int sum (int k) {
  int s = 0;
  while (k >= 1) {
    s += tree[k];
    k -= k&-k;
  }
  return s;
}

int add (int k, int x) {
  while (k <= n) {
    tree[k] += x;
    k += k&-k;
  }
}

int main () {
  cin >> n;

  for (int i = 1; i <= n; i++) {
    int x;
    cin >> x;

    add(i, x);
  }

  int q;
  cin >> q;

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

    cout << sum(b) - sum(a - 1) << '\n';
  }
}
```

Exemplo 3: Problema Balé - Contagem de Inversões

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

using namespace std;

#define int long long

#define N 100001

int n;
int a[N];
int bit[N];

int sum (int k) {
  int s = 0;
  while (k > 0) {
    s += bit[k];
    k -= k & -k;
  }
  return s;
}

void add (int k, int x) {
  while (k <= n) {
    bit[k] += x;
    k += k & -k;
  }
}

int32_t main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  cin >> n;

  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }

  int ans = 0;

  for (int i = n; i > 0; i--) {
    ans += sum(a[i]);

    add(a[i], 1);
  }

  cout << ans << '\n';
}
```

Contagem de Inversões - BIT :

```
#include <iostream>

#define MAXN 60000

using namespace std;

int v[MAXN + 1];
int bit[MAXN + 1];

void atualiza (int pos) {
  for (int i = pos; i < MAXN + 1; i += (i & -i))
    bit[i] += 1;
}

int consulta (int pos) {
  int soma = 0;
  for (int i = pos; i > 0; i -= (i & -i))
    soma += bit[i];
  return soma;
}

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

  for (int i = 1; i < n + 1; i++)
    cin >> v[i];

  int inv = 0;
  for (int i = n; i > 0; i--) {
    atualiza(v[i]);
    inv += consulta(v[i] - 1);
  }

  cout << inv << "\n";

  return 0;
}
```

Lazy Propagation (Propagação Preguiçosa):

```
int arv[4*MAXN];
int lazy[4*MAXN];

void atualiza (int no, int i, int j, int a, int b, int val) {
  int esq = 2*no;
  int dir = 2*no + 1;

  if (lazy[no] != 0) {
    arv[no] += lazy[no]*(j - i + 1);

    if (i != j) {
      lazy[esq] = lazy[no];
      lazy[dir] = lazy[no];
    }

    lazy[no] = 0;
  }

  if (b < i || j < a) return;

  if (a <= i && j <= b) {
    arv[no] += val*(j - i + 1);

    if (i != j) {
      lazy[esq] = val;
      lazy[dir] = val;
    }
  } else {
    int meio = (i + j)/2;

    atualiza(esq, i, meio, a, b, val);
    atualiza(dir, meio + 1, j, a, b, val);

    arv[no] = arv[esq] + arv[dir];
  }
}

int consulta (int no, int i, int j, int a, int b) {
  int esq = 2*no;
  int dir = 2*no + 1;

  if (lazy[no] != 0) {
    arv[no] += lazy[no]*(j - i + 1);

    if (i != j) {
      lazy[esq] = lazy[no];
      lazy[dir] = lazy[no];
    }

    lazy[no] = 0;
  }

  if (b < i || j < a) return 0;
  if (a <= i && j <= b) return arv[no];

  int meio = (i + j)/2;

  int r_esq = consulta(esq, i, meio, a, b);
  int r_dir = consulta(dir, meio + 1, j, a, b);

  return r_esq + r_dir;
}
```

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

typedef pair<int, int> pii;
#define F first
#define S second
#define PB push_back
#define INF 0x3f3f3f3f

#define N 200001
#define M 200001

struct plano { int u, l, r; };

int n;
plano p[M];
vector<pii> adj[N];

pii tree[4*N];
int lazy[4*N];
bool hasLazy[4*N];

void propagation (int no, int i, int j) {
  if (hasLazy[no]) {
    tree[no].F += lazy[no];
    if (i != j) {
      lazy[2*no] += lazy[no];
      hasLazy[2*no] = 1;
      lazy[2*no + 1] += lazy[no];
      hasLazy[2*no + 1] = 1;
    }
    lazy[no] = 0;
    hasLazy[no] = 0;
  }
}

void updateRange (int no, int i, int j, int l, int r, int v)
{
  propagation(no, i, j);
  if (r < i || j < l) return;
  if (l <= i && j <= r) {
    lazy[no] = v;
    hasLazy[no] = 1;
    propagation(no, i, j);
    return;
  }
  int mid = ((i + j) >> 1);
  updateRange(2*no, i, mid, l, r, v);
  updateRange(2*no + 1, mid + 1, j, l, r, v);
  tree[no] = min(tree[2*no], tree[2*no + 1]);
}

pii query (int no, int i, int j, int l, int r) {
  propagation(no, i, j);
  if (r < i || j < l) return {INF, INF};
  if (l <= i && j <= r) return tree[no];
  int mid = ((i + j) >> 1);
   return min(query(2*no, i, mid, l, r), query(2*no + 1, mid + 1, j,
l, r));
}

void initNo (int no, int i, int j) {
  tree[no] = {0, i};
  if (i == j) return;
  int mid = ((i + j) >> 1);
  initNo(2*no, i, mid);
  initNo(2*no + 1, mid + 1, j);
}

bool solve (int x) {
  memset(lazy, 0, sizeof(lazy));
  memset(hasLazy, 0, sizeof(hasLazy));
  initNo(1, 1, n);
  for (int i = 1; i <= n; i++) adj[i].clear();
  for (int i = 1; i <= x; i++) {
    int u = p[i].u, l = p[i].l, r = p[i].r;
    adj[u].PB({l, r});
    updateRange(1, 1, n, l, r, 1);
  }

  while (1) {
    pii s = query(1, 1, n, 1, n);
    if (s.F == 0) {
      if (!adj[s.S].empty()) {
        for (auto x : adj[s.S]) {
          int l = x.F, r = x.S;
          updateRange(1, 1, n, l, r, -1);
        }
        adj[s.S].clear();
      }
      updateRange(1, 1, n, s.S, s.S, INF);
    } else if (s.F != INF) {
      return 1;
    } else if (s.F == INF) {
      return 0;
    }
  }

  return 0;
}

void fastscan (int& num) {
  register char c;
  bool neg = false;

  c = getchar();
  if (c == '-') {
    neg = true;
    c = getchar();
  }

  num = 0;

  while ('0' <= c && c <= '9') {
    num = 10*num + c - '0';
    c = getchar();
  }

  if (neg) num *= -1;
}

int main () {
  int m;
  fastscan(n);
  fastscan(m);

  for (int i = 1; i <= m; i++) {
    int u, l, r;
    fastscan(u);
    fastscan(l);
    fastscan(r);

    p[i] = {u, l, r};
  }

  int ini = 1, fim = m;

  while (ini <= fim) {
    int meio = ((ini + fim) >> 1);
    if (solve(meio)) fim = meio - 1;
    else ini = meio + 1;
  }

  fim++;

  printf("%d\n", (fim <= m ? fim : -1));
}
```

Acordes Intergaláticos

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

using namespace std;

#define N 100000

int lazy[4*N];
int tree[4*N][9];
bool hasLazy[4*N];

void build (int no, int i, int j) {
  if (i == j) {
    tree[no][1] = 1;
    return;
  }

  int mid = (i + j) >> 1;

  build(2*no, i, mid);
  build(2*no + 1, mid + 1, j);

  for (int k = 0; k < 9; k++)
    tree[no][k] = tree[2*no][k] + tree[2*no + 1][k];
}

void propagation (int no, int i, int j) {
  if (hasLazy[no]) {
    vector<int> tmp(9);

    for (int k = 0; k < 9; k++)
      tmp[(k + lazy[no]) % 9] += tree[no][k];

    for (int k = 0; k < 9; k++)
      tree[no][k] = tmp[k];

    if (i != j) {
      lazy[2*no] += lazy[no];
      lazy[2*no + 1] += lazy[no];
      hasLazy[2*no] = true;
      hasLazy[2*no + 1] = true;
    }

    lazy[no] = 0;
    hasLazy[no] = false;
  }
}

void update (int no, int i, int j, int a, int b, int v) {
  propagation(no, i, j);

  if (b < i || j < a) return;

  if (a <= i && j <= b) {
    lazy[no] = v;
    hasLazy[no] = true;
    propagation(no, i, j);
    return;
  }

  int mid = (i + j) >> 1;

  update(2*no, i, mid, a, b, v);
  update(2*no + 1, mid + 1, j, a, b, v);

  for (int k = 0; k < 9; k++)
    tree[no][k] = tree[2*no][k] + tree[2*no + 1][k];
}

vector<int> query (int no, int i, int j, int a, int b) {
  propagation(no, i, j);

  if (b < i || j < a) return vector<int> (9);

  if (a <= i && j <= b) {
    vector<int> ret(9);

    for (int k = 0; k < 9; k++)
      ret[k] = tree[no][k];

    return ret;
  }

  int mid = (i + j) >> 1;

  vector<int> l = query(2*no, i, mid, a, b);
  vector<int> r = query(2*no + 1, mid + 1, j, a, b);

  vector<int> ret(9);

  for (int k = 0; k < 9; k++) ret[k] = l[k] + r[k];

  return ret;
}

int maxFreq (vector<int> v) {
  int qtd = 0, f = -1;
  for (int i = 0; i < v.size(); i++)
    if (qtd <= v[i]) { qtd = v[i]; f = i; }
  return f;
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0); cout.tie(0);

  int n, q; cin >> n >> q;

  build(1, 1, n);

  while (q--) {
    int a, b; cin >> a >> b;
    update(1, 1, n, a, b, maxFreq(query(1, 1, n, ++a, ++b)));
  }

  for (int i = 1; i <= n; i++)
    cout << maxFreq(query(1, 1, n, i, i)) << '\n';
}
```

Árvore de Indexação Binária 2D (BIT 2D):

```
int bit[MAXN + 1][MAXN + 1];

void atualiza (int lin, int col, int val) {
  for (int i = lin; i < n + 1; i += (i & -i))
    for (int j = col; j < m + 1; j += (j & -j))
      bit[i][j] += val;
}

int consulta (int lin, int col) {
  int ret = 0;

  for (int i = lin; i > 0; i -= (i & -i))
    for (int j = col; j > 0; j -= (j & -j))
      ret += bit[i][j];

  return ret;
}
```

Suffix Array:

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

using namespace std;

struct SuffixArray
{
  string s;
  vector<int> p, c;

  SuffixArray (string const& str)
  {
    s = str; s += '$';

    int n = s.size();
    p.resize(n), c.resize(n);

    {
      vector<pair<char, int>> a(n);
      for (int i = 0; i < n; i++) a[i] = {s[i], i};
      sort(a.begin(), a.end());
      for (int i = 0; i < n; i++) p[i] = a[i].second;
      c[p[0]] = 0;
      for (int i = 1; i < n; i++)
      {
        if (a[i].first == a[i - 1].first)
          c[p[i]] = c[p[i - 1]];
        else c[p[i]] = c[p[i - 1]] + 1;
      }
    }

    int k = 0;
    while ((1 << k) < n)
    {
      vector<pair<pair<int, int>, int>> a(n);
      for (int i = 0; i < n; i++)
        a[i] = {{c[i], c[(i + (1 << k)) % n]}, i};
      sort(a.begin(), a.end());
      for (int i = 0; i < n; i++) p[i] = a[i].second;
      c[p[0]] = 0;
      for (int i = 1; i < n; i++)
      {
        if (a[i].first == a[i - 1].first)
          c[p[i]] = c[p[i - 1]];
        else c[p[i]] = c[p[i - 1]] + 1;
      }
      k++;
    }
  }

  int search (string const& str)
  {
    int n = s.size(), ini = 0, end = n - 1;

    while (ini <= end)
    {
      int mid = (ini + end) >> 1;

      string sub = s.substr(p[mid], n - p[mid]);

      if (str > sub) ini = mid + 1;
      else end = mid - 1;
    }

    string sub = "";
    int pos = end + 1;
    if (0 <= pos && pos < n)
      sub = s.substr(p[pos], n - p[pos]);

    if (sub.find(str) != -1) return p[pos];
    return -1;
  }

};

int lcs (string const& a, string const& b)
{
  int n = a.size(), m = b.size();

  vector<vector<int>> dp(n + 1, vector<int> (m + 1));

  int mx = 0;

  for (int i = 1; i <= n; i++)
  {
    for (int j = 1; j <= m; j++)
    {
      if (a[i - 1] == b[j - 1])
      {
        dp[i][j] = dp[i - 1][j - 1] + 1;
        mx = max(mx, dp[i][j]);
      }
    }
  }

  return mx;
}

int main ()
{
  ios::sync_with_stdio(0);
  cin.tie(0);

  bool firstCase = true;

  string a, b;
  while (cin >> a >> b)
  {
    int n = a.size(), m = b.size();

    int szLCS = lcs(a, b);

    set<string> gp;

    if (szLCS > 0)
    {
      SuffixArray sf(b);

      for (int i = 0; i + szLCS - 1 < n; i++)
      {
        string sub = a.substr(i, szLCS);

        if (sf.search(sub) != -1) gp.insert(sub);
      }
    }

    if (!firstCase) cout << '\n';
    else firstCase = false;

    if (gp.empty()) cout << "No common sequence." << '\n';
    else for (auto x : gp) cout << x << '\n';
  }
}
```

Trie: *O(m) inserção e consulta, m tamanho da string*

Exemplo 1:

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

using namespace std;

struct Trie {
  struct TrieNode {
    bool isEndOfWord;
    TrieNode* children[26];
  };

  TrieNode* root;

  Trie() { root = getNode(); }

  TrieNode* getNode() {
    TrieNode* node = new TrieNode();
    node->isEndOfWord = false;
    for (int i = 0; i < 26; i++) node->children[i] = NULL;
    return node;
  }

  void insert(string key) {
    TrieNode* node = root;
    for (char c : key) {
      int index = c - 'a';
      if (node->children[index] == NULL)
        node->children[index] = getNode();
      node = node->children[index];
    }
    node->isEndOfWord = true;
  }

  bool search(string key) {
    TrieNode* node = root;
    for (char c : key) {
      int index = c - 'a';
      if (node->children[index] == NULL) return false;
      node = node->children[index];
    }
    return node->isEndOfWord;
  }
};

int main() {
  ios::sync_with_stdio(0); cin.tie(0);
  Trie trie;
  trie.insert("cat");
  cout << "cat? " << trie.search("cat") << '\n';
}
```

Exemplo 2: Fake News

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

using namespace std;

struct Trie
{
  struct TrieNode
  {
    int count;
    bool isEndOfWord;
    TrieNode* childrean[26];
  };

  TrieNode* root;

  Trie () { root = new TrieNode(); }

  TrieNode* getNode ()
  {
    TrieNode* node = new TrieNode();
    node->count = 0;
    node->isEndOfWord = true;
    for (int i = 0; i < 26; i++) node->childrean[i] = NULL;
    return node;
  }

  void insert (string key)
  {
    TrieNode* node = root;
    for (char c : key)
    {
      int index = c - 'a';
      if (node->childrean[index] == NULL)
        node->childrean[index] = getNode();
      node->count++;
      node = node->childrean[index];
    }
    node->isEndOfWord = true;
  }

  int search (string key)
  {
    TrieNode* node = root;
    for (char c : key)
    {
      int index = c - 'a';
      if (node->childrean[index] == NULL)  return 0;
      node = node->childrean[index];
    }
    return node->count;
  }
};

int main ()
{
  ios::sync_with_stdio(0); cin.tie(0);

  Trie trie;

  int n; cin >> n;

  while (n--)
  {
    string x, y; cin >> x >> y;

    if (x == "add")
    {
      trie.insert(y);
    }
    else
    {
      cout << trie.search(y) << '\n';
    }
  }
}
```

Exemplo 3: Book of the Dead's spells

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

using namespace std;

#define int long long

struct Trie
{
    struct TrieNode
    {
      int value;
      bool isEndOfWord;
      TrieNode* childrean[26];
    };

    TrieNode* root;

    Trie () { root = getNode(); };

    TrieNode* getNode ()
    {
      TrieNode* node = new TrieNode();
      node->value = 0;
      node->isEndOfWord = false;
      for (int i = 0; i < 26; i++) node->childrean[i] = NULL;
      return node;
    }

    void insert (string key, int val)
    {
      TrieNode* node = root;
      for (char c : key)
      {
        int index = c - 'a';
        if (node->childrean[index] == NULL)
          node->childrean[index] = getNode();
        node = node->childrean[index];
      }
      node->value = val;
      node->isEndOfWord = true;
    }

    int solve (TrieNode* node)
    {
      int mx = 0;
      for (int i = 0; i < 26; i++)
      {
        if (node->childrean[i] != NULL)
          mx = max(mx, solve(node->childrean[i]));
      }
      return mx + node->value;
    }
};

signed main ()
{
  ios::sync_with_stdio(0); cin.tie(0);

  int n; cin >> n;

  Trie trie;

  for (int i = 0; i < n; i++)
  {
    string s; int p; cin >> s >> p;
    trie.insert(s, p);
  }

  cout << trie.solve(trie.root) << '\n';
}
```

Programação Dinâmica:

Soma Rápida dos Elementos em Matriz: $O(1)$

```
#include <stdio.h>

#define MAXN 100

int mat[MAXN + 2][MAXN + 2];
int soma[MAXN + 2][MAXN + 2];

int main () {
  int n;
  scanf("%d", &n);

  for (int i = 1; i < n + 2; i++) {
    for (int j = 1; j < n + 2; j++) {
      scanf("%d", &mat[i][j]);

      soma[i][j] = mat[i][j] + soma[i - 1][j]
        + soma[i][j - 1] - soma[i - 1][j - 1];
    }
  }

  for (int i = 1; i < n + 1; i++) {
    for (int j = 1; j < n + 1; j++) {
      int aux = soma[i + 1][j + 1] - soma[i + 1][j - 1]
        - soma[i - 1][j + 1] + soma[i - 1][j - 1];

      if (aux > 1) {
        printf("S");
      } else {
        printf("U");
      }
    }

    printf("\n");
  }

  return 0;
}
```

Pré-processar somas em matriz (Problema da Laranja):

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

using namespace std;

#define int long long

#define L 1001
#define C 1001

int a[L][C];
int s[L][C];

int32_t main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  while (true) {
    int l, c, m, n;
    cin >> l >> c >> m >> n;

    if (l == 0 && c == 0 && m == 0 && n == 0) break;

    memset(a, 0, sizeof(a));
    memset(s, 0, sizeof(s));

    for (int i = 1; i <= l; i++) {
      for (int j = 1; j <= c; j++) {
        cin >> a[i][j];
      }
    }

    for (int i = 1; i <= l; i++) {
      for (int j = 1; j <= c; j++) {
        s[i][j] = a[i][j] + s[i - 1][j] + s[i][j - 1]
          - s[i - 1][j - 1];
      }
    }

    int ans = 0;

    for (int i = 1; i <= l - m + 1; i++) {
      for (int j = 1; j <= c - n + 1; j++) {
        int aux = s[i + m - 1][j + n - 1] - s[i + m - 1][j - 1]
          - s[i - 1][j + n - 1] + s[i - 1][j - 1];

        ans = max(ans, aux);
      }
    }

    cout << ans << '\n';
  }
}
```

Encontrar o número de subarrays com soma K: $O(N)$ *caso médio*

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

using namespace std;

#define int long long

#define N 500001

int sum[N];

int32_t main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, k;
  cin >> n >> k;

  sum[0] = 0;
  for (int i = 1; i <= n; i++) {
    int x;
    cin >> x;

    sum[i] = sum[i - 1] + x;
  }

  unordered_map<int, int> qtd;

  qtd[0] = 1;

  int ans = 0;

  for (int j = 1; j <= n; j++) {
    int aux = sum[j] - k;

    if (qtd.find(aux) != qtd.end()) {
      ans += qtd[aux];
    }

    qtd[sum[j]]++;
  }

  cout << ans << '\n';
}
```

Sweep Line: somar valores em um intervalo de forma eficiente $O(N)$ para reconstrução da soma

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

using namespace std;

#define N 100002

int c[N];
int b[N];
int sp[N];
int ans[N];

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, t;
  cin >> n >> t;

  for (int i = 1; i <= n; i++) {
    cin >> c[i];
  }

  for (int i = 1; i <= n; i++) {
    cin >> b[i];
  }

  while (t--) {
    int i, j;
    cin >> i >> j;

    sp[i]++;
    sp[j + 1]--;
  }

  int aux = 0;
  for (int i = 1; i <= n; i++) {
    aux += sp[i];

    if (aux % 2 == 0) {
      ans[i] = c[i];
    } else {
      ans[i] = b[i];
    }
  }

  cout << ans[1];
  for (int i = 2; i <= n; i++) {
    cout << ' ' << ans[i];
  }

  cout << '\n';
}
```

Divisão e Conquista - Merge Sort:

```
#define INF 1000000001

void mergesort(vector<int>& v) {
  if (v.size() == 1) return;

  vector<int> esq, dir;

  for (int i = 0; i < v.size()/2; i++)
    esq.push_back(v[i]);

  for (int i = v.size()/2; i < v.size(); i++)
    dir.push_back(v[i]);

  mergesort(esq);
  mergesort(dir);

  esq.push_back(-INF);
  dir.push_back(-INF);

  int i_esq = 0, i_dir = 0;
  for (int i = 0; i < v.size(); i++)
    if (esq[i_esq] >= dir[i_dir]) {
      v[i] = esq[i_esq];
      i_esq++;
    } else {
      v[i] = dir[i_dir];
      i_dir++;
    }

}
```

Contagem de Inversões - Merge Sort:

```
#define INF 1000000000

int mergesort (vector<int>& v) {
  if (v.size() == 1) return 0;

  int inv = 0;
  vector<int> esq, dir;

  for (int i = 0; i < v.size()/2; i++)
    esq.push_back(v[i]);

  for (int i = v.size()/2; i < v.size(); i++)
    dir.push_back(v[i]);

  inv += mergesort(esq);
  inv += mergesort(dir);

  esq.push_back(INF);
  dir.push_back(INF);

  int i_esq = 0, i_dir = 0;
  for (int i = 0; i < v.size(); i++)
    if (esq[i_esq] <= dir[i_dir]) {
      v[i] = esq[i_esq];
      i_esq++;
    } else {
      v[i] = dir[i_dir];
      i_dir++;

      inv += esq.size() - i_esq - 1;
    }

    return inv;
}
```

Problema da Mochila: $O(N.W)$

Exemplo 1: forma recursiva

```
int n;
int peso[MAXN];
int prot[MAXN];
int tab[MAXN + 1][MAXP + 1];

int resolve(int i, int p) {
  if (tab[i][p] != -1) return tab[i][p];
  if (p == 0 or i == n) return tab[i][p] = 0;

  int res = resolve(i + 1, p);

  if (p - peso[i] >= 0)
    res = max(res, prot[i] + resolve(i + 1, p - peso[i]));

  return tab[i][p] = res;
}
```

Exemplo 2: forma iterativa

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

using namespace std;

#define N 1001
#define W 1001

int w[N];
bool possible[N][W];

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;

  int sumW = 0;

  for (int i = 1; i <= n; i++) {
    cin >> w[i];
    sumW += w[i];
  }

  possible[0][0] = true;
  for (int k = 1; k <= n; k++) {
    for (int x = 0; x <= sumW; x++) {
      if (x - w[k] >= 0) possible[k][x] |= possible[k - 1][x - w[k]];
      possible[k][x] |= possible[k - 1][x];
    }
  }
}
```

Exemplo 3: forma iterativa, com $O(W)$ de espaço

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

using namespace std;

#define N 1001
#define W 1001

int w[N];
bool possible[W];

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;

  int sumW = 0;

  for (int i = 1; i <= n; i++) {
    cin >> w[i];
    sumW += w[i];
  }

  possible[0] = true;
  for (int k = 1; k <= n; k++) {
    for (int x = sumW; x >= 0; x--) {
      if (possible[x]) possible[x + w[k]] = true;
    }
  }
}
```

Maior Subsequência Comum - LCS: $O(N.M)$

```
int v[MAXN + 1];
int u[MAXN + 1];
int tab[MAXN + 1][MAXN + 1];

int LCS (int i, int j) {
  if (tab[i][j] != -1) return tab[i][j];

  if (i == 0 or j == 0) return tab[i][j] = 0;

  if (v[i] == u[j]) return tab[i][j] = 1 + LCS(i - 1, j - 1);

  return tab[i][j] = max(LCS(i - 1, j), LCS(i, j - 1));
}
```

Maior Subsequência Crescente - LIS: $O(N.log(N))$
LIS estritamente crescente: lower_bound
LIS não decrescente: upper_bound

Encontrar o tamanho:

```
int LIS (vector<int>& v) {
  vector<int> topo;

  for (auto x : v) {
      auto it = lower_bound(topo.begin(), topo.end(), x);

      if (it == topo.end())
          topo.push_back(x);
      else
          *it = x;
  }

  return topo.size();
}
```

Exemplo 2: Não decrescente

```
int LIS (vector<int>& v) {
  vector<int> pilha;

  for (auto x : v) {
    auto it = upper_bound(pilha.begin(), pilha.end(), x);

    if (it == pilha.end()) pilha.push_back(x);
    else *it = x;
  }

  return pilha.size();
}
```

Encontrar um exemplo de solução:

```
vector<int> LIS (vector<int>& v) {
  vector<int> pilha, res;
  int pos[MAXN], pai[MAXN];

  for (int i = 0; i < v.size(); i++) {
    vector<int>::iterator it =
      upper_bound(pilha.begin(), pilha.end(), v[i]);

    int p = it - pilha.begin();

    if (it == pilha.end())
      pilha.push_back(v[i]);
    else
      *it = v[i];

    pos[p] = i;

    if (p == 0)
      pai[i] = -1;
    else
      pai[i] = pos[p - 1];
  }

  int p = pos[pilha.size() - 1];

  while (p != -1) {
    res.push_back(v[p]);
    p = pai[p];
  }

  reverse(res.begin(), res.end());

  return res;
}
```

Troco / Problema das Moedas: $O(Q.N)$

```
int n;
int c[MAXN];
int tab[10][MAXM + 1];

int resolve (int q, int v) {
  if (tab[q][v] != -1) return tab[q][v];

  if (v == 0) return tab[q][v] = 1;
  if (q == 0) return tab[q][v] = 0;

  int res = 0;
  for (int i = 0; i < n; i++) {
    if (v - c[i] >= 0)
      res = max(res, resolve(q - 1, v - c[i]));

      if (res == 1) break;
  }

  return tab[q][v] = res;
}
```

Soma máxima em um Intervalo: $O(N)$

```
int maiorSoma (int w[], int n) {
  int res = 0, maior = 0;
  for (int i = 0; i < n; i++) {
    maior = max(0, w[i] + maior);
    res = max(res, maior);
  }
  return res;
}
```

Beecrowd 1798 - Cortando Canos:

DP Problemas das Moedas iterativo que conta soluções com sobra de cano:

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

#define MAXN 1000
#define MAXCV 5001

using namespace std;

struct cliente {
  int c, v;
};

cliente a[MAXN];

int c[MAXN];
int v[MAXN];
int tab[MAXCV];

bool comp (cliente a, cliente b) {
  return a.c < b.c;
}

int main () {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n, t;
  cin >> n >> t;

  for (int i = 0; i < n; i++) {
    int c, v;
    cin >> c >> v;

    a[i] = {c, v};
  }

  sort(a, a + n, comp);

  for (int i = 0; i < t + 1; i++) {
    for (int j = 0; j < n; j++) {
      if (i - a[j].c >= 0) {
        tab[i] = max(tab[i], a[j].v + tab[i - a[j].c]);
      } else {
        break;
      }
    }
  }

  cout << tab[t] << "\n";
}
```

Caminho com soma máxima em uma matriz: $O(N^2)$

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

using namespace std;

#define N 1001

int a[N][N];
int value[N][N];

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;

  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      cin >> a[i][j];
    }
  }

  for (int y = 1; y <= n; y++) {
    for (int x = 1; x <= n; x++) {
      value[y][x] = max(value[y][x - 1], value[y - 1][x]) + a[y][x];
    }
  }

  cout << value[n][n] << '\n';
}
```

Distância editável entre strings: $O(N.M)$

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

using namespace std;

#define N 1001
#define INF 0x3f3f3f3f

int dist[N][N];

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  string x, y;
  cin >> x >> y;

  memset(dist, INF, sizeof(dist));

  for (int i = 0; i <= x.size(); i++) {
    for (int j = 0; j <= y.size(); j++) {
      if (i == 0 && j == 0) {
        dist[0][0] = 0;
      } else {
        if (i - 1 >= 0)
          dist[i][j] = min(dist[i][j], dist[i - 1][j] + 1);

        if (j - 1 >= 0)
          dist[i][j] = min(dist[i][j], dist[i][j - 1] + 1);

        if (i - 1 >= 0 && j - 1 >= 0)
          dist[i][j] = min(dist[i][j], dist[i - 1][j - 1]
            + (x[i - 1] == y[j - 1] ? 0 : 1));
      }
    }
  }

  cout << dist[x.size()][y.size()] << '\n';
}
```

Two Pointers - Encontrar uma subarray de soma X (array de números positivos): $O(N)$

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

using namespace std;

#define N 1001

int a[N];
int sum[N];

int soma (int i, int j) {
  return sum[j] - sum[i - 1];
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, x;
  cin >> n >> x;

  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }

  sum[0] = 0;
  for (int i = 1; i <= n; i++) {
    sum[i] = sum[i - 1] + a[i];
  }

  int i = 1, j = 1;
  bool flag = false;

  while (1) {
    if (soma(i, j) == x) {
      flag = true;
      break;
    } else if (soma(i, j) < x) {
      j++;
      if (j == n + 1) break;
    } else {
      i++;
      if (i > j) swap(i, j);
    }
  }

  if (flag) cout << "YES";
  else cout << "NO";

  cout << '\n';
}
```

Two Pointers - Encontrar dois valores que somam X (2SUM problem): $O(N.log(N))$

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

#define N 1000

int a[N];

int main () {
  ios::sync_with_stdio(0); cin.tie(0);

  int n, x; cin >> n >> x;

  for (int i = 0; i < n; i++) cin >> a[i];

  sort(a, a + n);

  int i = 0, j = n - 1;
  bool flag = false;

  while (i < j) {
    if (a[i] + a[j] == x) {
      flag = true;
      break;
    } else if (a[i] + a[j] < x) i++;
    else j--;
  }

  if (flag) cout << "YES" << '\n';
  else cout << "NO" << '\n';
}
```

Menor elemento mais próximo a esquerda: $O(N)$

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

using namespace std;

#define N 1000

int a[N];
int nse[N];

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;

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

  stack<int> st;

  for (int i = 0; i < n; i++) {
    while (!st.empty() && a[st.top()] >= a[i]) st.pop();

    if (st.empty()) nse[i] = -1;
    else nse[i] = st.top();

    st.push(i);
  }
}
```

Menor elemento em uma janela deslizante de tamanho K: $O(N)$

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

using namespace std;

#define N 1000
#define INF 0x3f3f3f3f

int a[N];

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, k;
  cin >> n >> k;

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

  deque<int> dq;

  int menor = INF;

  for (int i = 0; i < k; i++) {
    while (!dq.empty() && dq.front() > a[i]) dq.pop_front();

    dq.push_front(a[i]);

    menor = min(menor, dq.back());
  }

  for (int i = 1; i <= n - k; i++) {
    if (dq.back() == a[i - 1]) dq.pop_back();

    while (!dq.empty() && dq.front() > a[i + k - 1]) dq.pop_front();

    dq.push_front(a[i + k - 1]);

    menor = min(menor, dq.back());
  }

  cout << menor << '\n';
}
```

Sparse Table - Mínimo no intervalo $[a, b]$**:** $O(N.Log(N))$ **- build,** $O(1)$ **- query**

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

using namespace std;

#define N 1000
#define LOGN 10

int a[N];

int Log[N];
int st[N][LOGN];

void build (int v[], int n) {
  Log[1] = 0;
  for (int i = 2; i <= n; i++) {
    Log[i] = Log[i/2] + 1;
  }

  for (int i = 0; i < n; i++) {
    st[i][0] = v[i];
  }

  for (int j = 1; j <= Log[n]; j++) {
    for (int i = 0; i + (1 << j) - 1 < n; i++) {
      st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
  }
}

int query (int a, int b) {
  int w = Log[b - a + 1];
  return min(st[a][w], st[b - (1 << w) + 1][w]);
}

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

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

  build(a, n);

  int q;
  cin >> q;

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

    cout << query(a, b) << '\n';
  }
}
```

Matemática

Fórmulas:

$1 + 2 + \ ... \ + n = \frac{n(n+1)}{2}$

$1^2 + 2^2 + \ ... + n^2 = \frac{n(n+1)(2n+1)}{6}$

Número de diagonais de um polígono regular de n lados: $\frac{n(n-3)}{2}$

Perímetro do triângulo: $2p = a + b + c$

Área do triângulo (Fórmula de Heron): $\sqrt{p(p-a)(p-b)(p-c)}$

Área do triângulo inscrito (triângulo dentro da circunferência): $A = \frac{abc}{4R}$

Área do triângulo circunscrito (circunferência dentro do triângulo): $A = pr$

Soma de uma linha no Triângulo de Pascal: $C_0^n + C_1^n + ... + C_n^n = 2^n$

Probabilidade e Estatística

`frequência(`n_i`)`: Número de vezes que ocorrem as realizações.

`proporção(`f_i`)`: Proporção de cada realização em relação ao total.

$1 + 3,3 \ x \ log(n)$**: Determinar número de classe para variável contínua com Regra de Sturges (logaritmo).**

$2^k \geq n$**, `com o menor número inteiro`: Determinar número de classe para variável contínua com Regra da Potência de 2.**

$k = \sqrt{n}$**: Determinar número de classe para variável contínua com Regra da Raiz Quadrada.**

$\overline{x} = f_1x_1 + f_2x_2 + ... + f_nx_n = \sum_{i=1}^{n} f_ix_i$**: Média com f_i sendo a frequência relativa e x_i o valor da variável.**

`var(`x**`)`** $= s^2 = \frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n-1}$**: Variância amostral.**

`var(`x**`)`** $= \sigma^2 = \frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n}$**: Variância populacional.**

`dp(`x**`)`** $= \sqrt{var(x)}$**: Desvio padrão.**

CV = $\frac{dp(x)}{\overline{x}} \ x \ 100\%$**: Coeficiente de variação, onde $\overline{x}$ é a média aritmética.**

Z = $\frac{x-\overline{x}}{dp(x)}$**: Escore Z, que é a diferença entre um valor e a média, dividida pelo desvio padrão.**

P = $\frac{\#eventos-favoraveis}{\#eventos-possiveis}$**, `para`** $0 \leq P(evento \ qualquer) \leq 1$**: Teoria de conjuntos.**

P(A) = $\frac{numero \ de \ vezes \ que \ A \ pode \ ocorrer}{numero \ de \ vezes \ que \ o \ evento \ e \ repetido}$**: Teoria dos grandes números.**

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$**: Regra da adição, união.**

$P(A \cup B) = P(A) + P(B)$**: Regra da adição, união, para eventos mutuamente exclusivos.**

$P(A \cap B) = P(A).P(B/A) = P(B).P(A/B)$**: Regra da multiplicação, intersecção.**

$P(A \cap B) = P(A).P(B)$**: Regra da multiplicação, intersecção, para eventos independentes.**

$P(\overline{A}) = 1 - P(A)$**: Regra do complementar.**

$n!$**: Fatorial. Um conjunto de n diferentes itens pode ser organizado em ordem de n! maneiras diferentes (o primeiro item pode ser selecionado de n diferentes maneiras, o segundo de n-1 maneiras e assim por diante).**

$nP_r = \frac{n!}{(n-r)!}$**: Permutação. Total de n diferentes itens disponíveis; selecionar r dos n itens sem reposição; considerar reorganizações dos mesmos itens como sendo seqüências diferentes.**

$\frac{n!}{n1!n2!...nk!}$**: Permutação. Quando alguns itens são repetidos.**

$nC_r = \frac{n!}{(n-r)!r!}$**: Combinação. Total de n diferentes itens disponíveis; selecionar r dos n itens sem reposição; considerar reorganizações dos mesmos itens com sendo a mesma (a combinação ABC e a mesma de CBA).**

$P(A_j/B) = \frac{P(B/A_j).P(A_j)}{\sum_{i=1}^{n} P(B/A_i).P(A_i)} = \frac{P(B/A_j).P(A_j)}{P(B)}$**: Teorema de Bayes. Suponha que os eventos formam uma partição do espaço amostral S e que suas probabilidades sejam conhecidas.**

$\mu = \sum_{i=1}^{N} x_iP(X = x_i)$**: Valor médio com probabilidades P_i para x_i.**

$\sigma^2 = \sum[x^2.P(x)] - \mu^2$**: Variância para probabilidade.**

$f(x) = \frac{1}{N} \ com \ X = \{1, \ 2, \ ..., \ N\}$**, `média =`** $E(x) = \frac{N+1}{2}$**, `var(x)` =** $\frac{N^2-1}{12}$**: Distribuição Uniforme. Todas as probabilidades são as mesmas e não muda para qualquer i.**

$f(x) = p^xq^{1-x} com \ X = \{0, 1\}$**, `média =`** $E(x) = p$**, `var(x)` =** pq**: Distribuição de Bernoulli. Dizemos que uma variável X segue o modelo Bernoulli se atribui 0 ou 1 à ocorrência de fracasso ou sucesso, respectivamente com p representando a probabilidade de sucesso.**

$f(x) = \binom{n}{x} p^xq^{n-x} com \ X = \{0, 1, \ ..., \ n\}$**, `média =`** $E(x) = np$**, `var(x)` =** npq**: Distribuição Binomial. Repetição de ensaios de Bernoulli independentes. O experimento tem um número fixo de tentativas; tentativas independentes; tentativa deve ter todos os resultados classificados em duas categorias; probabilidades constantes.**

$f(x) = pq^x com \ X = \{0, 1, \ 2, \ ...\}$**, `média =`** $E(x) = \frac{q}{p}$**, `var(x)` =** $\frac{q}{p^2}$**: Distribuição Geométrica. Conta sucessos até que ocorra um fracasso.**

$f(x) = \binom{x+r-1}{x} p^r q^x com \ X = \{0, 1, \ 2, \ ...\}$**, `média =`** $E(x) = \frac{rq}{p}$**, `var(x)` =** $\frac{rq}{p^2}$**: Distribuição Binomial Negativa. Conta sucessos até que seja atingido k fracassos com posições determinadas.**

$f(x) = \frac{e^{-\mu}\mu^x}{x!} \ com \ X = \{0, 1, \ 2, \ ...\}$**, `média =`** $E(x) = \mu$**, `var(x)` =** μ**: Distribuição de Poisson. A variável aleatória x é o número de ocorrências de um evento ao longo de algum interval; devem ser aleatórias; independentes umas das outras; uniformemente distribuídas sobre o intervalo em uso.**

BigInt

add(BigInteger val): Retorna um BigInteger cujo valor é (this + val).
abs(): Retorna um BigInteger cujo valor é o valor absoluto desse BigInteger.
and(BigInteger val): Retorna um BigInteger cujo valor é (this & val).
andNot(BigInteger val): Retorna um BigInteger cujo valor é (this & ~val).
bitCount(): Retorna o número de bits na representação do complemento de dois deste BigInteger que difere do seu bit de sinal.
bitLength(): Retorna o número de bits na representação mínima em complemento de dois deste BigInteger, excluindo um bit de sinal.
byteValueExact(): Converte esse BigInteger em um byte, verificando se há informações perdidas.
clearBit(int n): Retorna um BigInteger cujo valor é equivalente a esse BigInteger com o bit designado limpo.
compareTo(BigInteger val): Compara este BigInteger com o BigInteger especificado.
divide(BigInteger val): Retorna um BigInteger cujo valor é (this/val).
divideAndRemainder(BigInteger val): Retorna um array de dois BigIntegers contendo (this / val) seguido por (this % val).
doubleValue(): Converte este BigInteger em um double.
equals(Object x): Compara este BigInteger com o Object especificado para igualdade.
flipBit(int n): Retorna um BigInteger cujo valor é equivalente a esse BigInteger com o bit designado invertido.
floatValue(): Converte este BigInteger em um float.
gcd(BigInteger val): Retorna um BigInteger cujo valor é o máximo divisor comum de abs(this) e abs(val).
getLowestSetBit(): Retorna o índice do bit mais à direita (ordem mais baixa) neste BigInteger (o número de zero bits à direita do bit mais à direita).
hashCode(): Retorna o código hash para este BigInteger.
intValue(): Converte este BigInteger em um int.
intValueExact(): Converte esse BigInteger em um int, verificando informações perdidas.
isProbablePrime(int certainty): Retorna true se este BigInteger for provavelmente primo, false se for definitivamente composto.
longValue(): Converte este BigInteger em um long.
longValueExact(): Converte este BigInteger em um longo, verificando informações perdidas.
max(BigInteger val): Retorna o máximo deste BigInteger e val.
min(BigInteger val): Retorna o mínimo deste BigInteger e val.
mod(BigInteger m): Retorna um BigInteger cujo valor é (este mod m).
modInverse(BigInteger m): Retorna um BigInteger cujo valor é (this-1 mod m).
modPow(BigInteger exponent, BigInteger m): Retorna um BigInteger cujo valor é (esse expoente mod m).
multiply(BigInteger val): Retorna um BigInteger cujo valor é (este * val).
negate(): Retorna um BigInteger cujo valor é (-this).
nextProbablePrime(): Retorna o primeiro inteiro maior que este BigInteger que provavelmente é primo.
not(): Retorna um BigInteger cujo valor é (~this).
or(BigInteger val): Retorna um BigInteger cujo valor é (this | val).
pow(int exponent): Retorna um BigInteger cujo valor é (o expoente).
probablePrime(int bitLength, Random rnd): Retorna um BigInteger positivo que provavelmente é primo, com o bitLength especificado.
remainder(BigInteger val): Retorna um BigInteger cujo valor é (this % val).
setBit(int n): Retorna um BigInteger cujo valor é equivalente a este BigInteger com o conjunto de bits designado.
shiftLeft(int n): Retorna um BigInteger cujo valor é (this << n).
shiftRight(int n): Retorna um BigInteger cujo valor é (this >> n).
shortValueExact(): Converte este BigInteger em um short, verificando informações perdidas.
signum(): Retorna a função signum deste BigInteger.
sqrt(): Retorna a raiz quadrada inteira deste BigInteger.
sqrtAndRemainder(): Retorna uma matriz de dois BigIntegers contendo a raiz quadrada inteira s de this e seu restante this − s*s, respectivamente.
subtract(BigInteger val): Retorna um BigInteger cujo valor é (this − val).
testBit(int n): Retorna verdadeiro se e somente se o bit designado estiver definido.
toByteArray(): Retorna uma matriz de bytes contendo a representação em complemento de dois desse BigInteger.
toString(): Retorna a representação String decimal deste BigInteger.
toString(int radix): Retorna a representação de string desse BigInteger na base fornecida.
valueOf(long val): Retorna um BigInteger cujo valor é igual ao do long especificado.
xor(BigInteger val): Retorna um BigInteger cujo valor é (this ^ val).

Funções:

ceil(double x): arredonda o valor de x para cima
floor(double x): arredonda o valor de x para baixo
log(double x): logaritmo de x na base de euler
log2(double x): logaritmo de x na base 2
log10(double x): logaritmo de x na base 10

Quantidade de dias nos meses do ano:

```
int mes[] {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Converter de decimal para binário:

```
string toBin (int x) {
  if (x == 0) return "0";
  string aux = "";
  while (x > 0) {
    aux += x % 2 + '0';
    x /= 2;
  }
  reverse(aux.begin(), aux.end());
  return aux;
}
```

Gerar números binários de 1 até N: TIME: $O(N)$, SPACE: $O(N)$

```
vector<string> generate(int N) {
  string x;
  vector<string> v;
  queue<string> fila;

  fila.push("1");

  while (N--) {
    x = fila.front();
    fila.pop();

    v.push_back(x);

    if (N > fila.size()) fila.push(x + "0");
    if (N > fila.size()) fila.push(x + "1");
  }
  return v;
}
```

Inverter os dígitos de um número:

```
int inverte (int v) {
  int aux = 0;
  while (v > 0) {
    aux *= 10;
    aux += v % 10;
    v /= 10;
  }
  return aux;
}
```

Josephus:

```
#include <vector>
#include <iostream>

using namespace std;

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

  int nc;
  cin >> nc;

  int t = 0;
  while (nc--) {
    int n, k;
    cin >> n >> k;

    vector<int> v;

    for (int i = 1; i < n + 1; i++) {
      v.push_back(i);
    }

    int ini = 0;
    while (v.size() > 1) {
      ini = (ini + k - 1) % v.size();

      v.erase(v.begin() + ini);
    }

    cout << "Case " << ++t << ": " << v[0] << "\n";
  }

  return 0;
}
```

Gerar Todas as Permutações Ordenadas:

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

using namespace std;

void permuta (set<string>& cj, string& s, int inf, int sup) {
  if (inf == sup) {
    cj.insert(s);
  } else {
    for (int i = inf; i <= sup; i++) {
      swap(s[inf], s[i]);
      permuta(cj, s, inf + 1, sup);
      swap(s[inf], s[i]);
    }
  }
}

int main () {
  ios_base::sync_with_stdio(0); cin.tie(0);

  int n;
  cin >> n;

  while (n--) {
    string s;
    cin >> s;

    set<string> cj;

    permuta(cj, s, 0, s.size() - 1);

    for (auto x : cj) {
      cout << x << "\n";
    }

    cout << "\n";
  }
}
```

Critério de Divisibilidade por N:

Para descobrir o critério de divisibilidade de um número por uma valor n, temos que verificar se o somatório dos dígitos do número com a suas respectivas potências de 10 módulo n é uma valor divisível por n, exemplo:

Divisibilidade por 11:

$x = a_0 + a_1 \cdot 10 + a_2 \cdot 10^2 + a_3 \cdot 10^3 + \ldots + a_n \cdot 10^n$

Modulando as potências por $n = 11$, temos:

$x = a_0 + a_1 \cdot 10 + a_2 \cdot 1 + a_3 \cdot 10 + \ldots + a_n \cdot (10^n \,\%\, 11)$

Portanto, para um número inteiro ser divisível por 11, esse somatório terá que ser divisível por 11.

Verificar se um Número é Primo: $O(\sqrt{N})$

```
bool e_primo (int x) {
  if (x == 1) return false;

  for (int i = 2; i*i <= x; i++)
    if (x % i == 0)
      return false;

  return true;
}
```

Algoritmo de Euclides:
Máximo Divisor Comum (MDC):

```
int mdc (int x, int y) {
  if (y == 0) return x;
  return mdc(y, x % y);
}
```

Mínimo Múltiplo Comum (MMC): Utilizar a relação entre MDC e MMC:

$MMC(x, y) = \frac{x \cdot y}{MDC(x, y)}$

Exemplo 1: Órbitas

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

using namespace std;

#define int long long

int mdc (int x, int y) {
  if (y == 0) return x;
  return mdc(y, x % y);
}

int mmc (int x, int y) {
  return x*(y/mdc(x, y));
}

int32_t main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int a, b;
  cin >> a >> b;

  cout << mmc(a, b) << '\n';
}
```

Crivo de Eratóstenes: $O(N.log(log(N)))$
Exemplo 1

```
bool e_composto[MAXN + 1];

void crivo (int n) {
  e_composto[1] = true;

  for (int i = 2; i*i < n + 1; i++)
    if (e_composto[i] == false)
      for (int j = 2; j*i < n + 1; j++)
        e_composto[j*i] = true;
}
```

Exemplo 2: monta o vetor com o menor divisor do número

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

using namespace std;

#define N 10001

int not_prime[N];

void sieve (int n) {
  not_prime[1] = 1;

  for (int i = 2; 2*i <= n; i++) {
    not_prime[2*i] = 2;
  }

  for (int i = 3; i*i <= n; i += 2) {
    if (not_prime[i]) continue;

    for (int j = 2; j*i <= n; j++) {
      if (not_prime[j*i] == 0) not_prime[j*i] = i;
    }
  }
}
```

```
int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;

  sieve(n);

  for (int i = 1; i <= n; i++) {
    if (not_prime[i] == 0) {
      cout << i << '\n';
    }
  }
}
```

Fatorar um número: $O(\sqrt{N})$

Exemplo 1:

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

using namespace std;

#define PB push_back

void factors (int n, vector<int>& v) {
  while (n % 2 == 0) {
    v.PB(2);
    n /= 2;
  }

  for (int i = 3; i*i <= n; i += 2) {
    while (n % i == 0) {
      v.PB(i);
      n /= i;
    }
  }

  if (n > 1) v.PB(n);
}
```

```cpp
int main () {
  ios::sync_with_stdio(0); cin.tie(0);

  int n;
  cin >> n;

  vector<int> v;

  factors(n, v);

  for (auto x : v) cout << x << ' ';
  cout << '\n';
}
```

Exemplo 2: Problema Despojados

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

using namespace std;

#define int long long

void factors (int n, unordered_set<int>& cj) {
  while (n % 2 == 0) {
    cj.insert(2);
    n /= 2;
  }

  for (int i = 3; i*i <= n; i += 2) {
    while (n % i == 0) {
      cj.insert(i);
      n /= i;
    }
  }

  if (n > 1) cj.insert(n);
}

int32_t main () {
  ios::sync_with_stdio(0); cin.tie(0);

  int n;
  cin >> n;

  unordered_set<int> cj;

  factors(n, cj);

  int k = cj.size();

  cout << ((1LL << k) - k - 1) << '\n';
}
```

Encontrar todos os divisores de um número: $O(2\sqrt{N})$ *ou* $O(\sqrt[3]{N})$

Exemplo 1: Divisores

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

using namespace std;

#define F first
#define S second
#define PB push_back

int a, b, c, d;
vector<int> f, expo, prime, ans, dv;

void factors () {
  while (c % 2 == 0) {
    f.PB(2);
    c /= 2;
  }

  for (int i = 3; i*i <= c; i += 2) {
    while (c % i == 0) {
      f.PB(i);
      c /= i;
    }
  }

  if (c > 1) f.PB(c);
}

int value () {
  int aux = 1;
  for (int i = 0; i < (int) ans.size(); i++) {
    for (int j = 1; j <= ans[i]; j++) {
      aux *= prime[i];
    }
  }
  return aux;
}

void build (int i) {
  if (i == (int) expo.size()) {
    dv.PB(value());
  } else {
    for (int j = 0; j <= expo[i]; j++) {
      ans[i] = j;
      build(i + 1);
    }
  }
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  cin >> a >> b >> c >> d;

  factors();

  map<int, int> mp;

  for (auto x : f) {
    mp[x]++;
  }

  ans.resize((int) mp.size());

  for (auto x : mp) {
    prime.PB(x.F);
    expo.PB(x.S);
  }

  build(0);

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

  int n = -1;

  for (auto x : dv) {
    if (x % a == 0 && x % b != 0 && d % x != 0) {
      n = x;
      break;
    }
  }

  cout << n << '\n';
}
```

Gerar as combinações de N escolhe k por backtracking:

Exemplo 1:

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

using namespace std;

#define int long long

int n, ans;

void combinacao (int i, int k, int p, vector<int>& a, bool flag) {
  if (k == 0) {
    if (flag) {
      ans -= n/p;
    } else {
      ans += n/p;
    }
  } else {
    for (int j = i; j <= (int) a.size() - k; j++) {
      if (p*a[j] <= n) {
        p *= a[j];
        combinacao(j + 1, k - 1, p, a, flag);
        p /= a[j];
      } else {
        break;
      }
    }
  }
}

int32_t main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int k;
  cin >> n >> k;

  vector<int> a(k);

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

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

  ans = n;
  vector<int> v;

  for (int i = 1; i <= min(k, 9LL); i++) {
    combinacao(0, i, 1, a, i % 2);
  }

  cout << ans << '\n';
}
```

Meet in the Middle:

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

using namespace std;

#define int long long

#define N 21

int n;
int a[N][N];
vector<int> p[N], q[N];

void up (int l, int c, int sum)
{
  if (l + c == n + 1)
    p[l].push_back((sum ^ a[l][c]));
  else
  {
    up(l, c + 1, (sum ^ a[l][c]));
    up(l + 1, c, (sum ^ a[l][c]));
  }
}

void down (int l, int c, int sum)
{
  if (l + c == n + 1)
    q[l].push_back((sum ^ a[l][c]));
  else
  {
    down(l - 1, c, (sum ^ a[l][c]));
    down(l, c - 1, (sum ^ a[l][c]));
  }
}
```

```
signed main ()
{
  ios::sync_with_stdio(0); cin.tie(0);

  cin >> n;

  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      cin >> a[i][j];

  up(1, 1, 0);
  down(n, n, 0);

  int ans = 0;

  for (int i = 1; i <= n; i++)
  {
    sort(q[i].begin(), q[i].end());

    for (int j = 0; j < (int) p[i].size(); j++)
    {
      int v = (p[i][j] ^ a[i][n + 1 - i]);

      auto pos = equal_range(q[i].begin(), q[i].end(), v);

            ans += (pos.second - q[i].begin()) - (pos.first -
q[i].begin());
    }
  }

  cout << ans << '\n';
}
```

Manipulação de Bits (Bitwise):

Verificar se um bit está ligado:

```
bool is_set (int x, int i) {
  return (x & (1 << i));
}
```

Bit menos significativo (LSB):

```
int lsb (int x) {
  return (x & -x);
}
```

Bit menos significativo para zero:

```
void LSB_zero (int& x) {
  x &= x - 1;
}
```

Contar o número de bits iguais a 1:

```
int count_bit (int x) {
  int cont = 0;
  while (x > 0) {
    cont++;
    x -= (x & -x);
  }
  return cont;
}
```

Verificar se um número é potência de dois:

```
bool is_pow_of_two (int x) {
  if (x == 0) return false;
  return !(x & (x - 1));
}
```

Ligar um determinado bit:

```
void set_bit (int& x, int i) {
  x |= (1 << i);
}
```

Desligar um determinado bit:

Exemplo 1:

```
void set_off (int& x, int i) {
  x |= (1 << i);
  x ^= (1 << i);
}
```

Exemplo 2:

```
void set_off (int& x, int i) {
  x &= ~(1 << i);
}
```

Trocar estado de um bit:

```
void swap_bit (int& x, int i) {
  x ^= (1 << i);
}
```

Trocar todos os bits depois do LSB:

```
void swap_after (int& x) {
  x |= x - 1;
}
```

Funções adicionais:

Contar o número de zeros no início do número: `__builtin_clz(x)`
Contar o número de zeros no final do número: `__builtin_ctz(x)`
Contar quantos bit 1 existem no número: `__builtin_popcount(x)`
Paridade de quantos bit 1 existem no número: `__builtin_parity(x)`

Percorrer todos os subconjuntos de um conjunto x:

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

using namespace std;

int main () {
  int x = 14;

  int b = 0;
  do {
    for (int i = 0; i < 32; i++) {
      if (b & (1 << i)) cout << '[' << i << ']';
    }
    cout << '\n';
  } while (b = (b - x)&x);
}
```

Exponenciação Rápida

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

using namespace std;

#define int long long

int b, m;
unordered_map<int, int> tab;

int solve (int e)
{
  if (e == 0) return 1;
  if (e == 1) return b % m;

  if (tab.find(e) != tab.end()) return tab[e];

  return tab[e] = ((solve((e >> 1)) % m) * (solve(((e + 1) >> 1)) %
m)) % m;
}

void fastscan (int& num)
{
  register char c;
  bool neg = false;

  c = getchar();
  if (c == '-')
  {
    neg = true;
    c = getchar();
  }

  num = 0;

  while ('0' <= c && c <= '9')
  {
    num = 10*num + c - '0';
    c = getchar();
  }

  if (neg) num *= -1;
}

int32_t main ()
{
  ios::sync_with_stdio(0);
  cin.tie(0);

  int e;

  fastscan(b);
  fastscan(e);
  fastscan(m);

  printf("%lld\n", solve(e));
}
```

Distância de Ponto a Reta:

$$d = \frac{|A.x_1 + B.x_1 + C|}{\sqrt{A^2 + B^2}}, P(x_1, x_2)$$

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

#define SQ(x) (x)*(x)
#define int long long

struct Ponto
{
  int x, y;

  Ponto () {}

  Ponto (int xx, int yy) : x{xx}, y{yy} {}
};

struct Reta
{
  Ponto a, b;
  int dy, dx;

  Reta (Ponto aa, Ponto bb) : a{aa}, b{bb}
  {
    dy = b.y - a.y;
    dx = a.x - b.x;
  }

  int valorNoPonto (Ponto c)
  {
    int dy = b.y - a.y, dx = b.x - a.x;
    return dy*c.x - dx*c.y + a.y*dx - a.x*dy;
  }
};

int32_t main ()
{
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, xc, yc, r;
  cin >> n >> xc >> yc >> r;

  int cnt = 0;

  while (n--)
  {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;

    Reta reta = Reta(Ponto(x1, y1), Ponto(x2, y2));

    if (SQ(reta.valorNoPonto(Ponto(xc, yc))) <=
      SQ(r)*(reta.dy*reta.dy + reta.dx*reta.dx)) cnt++;
  }

  cout << cnt << '\n';
}
```

PF - Exponenciação de Matriz - Exponenciação Binária:

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

#define int long long

typedef vector<int> vi;
typedef vector<vi> vii;

#define PB push_back
#define debug(arg...) printf(arg)

#define MOD 1000000007

vii g;
vi aux;
unordered_map<int, vii> tab;

void gerar (int i)
{
  if (i == 3)
  {
    g.PB(aux);
  }
  else
  {
    for (int j = 0; j < 3; j++)
    {
      if (i == 0 || aux[i - 1] != j)
      {
        aux.PB(j);
        gerar(i + 1);
        aux.pop_back();
      }
    }
  }
}

vii multi (vii a, vii b)
{
  int ll = a.size();
  int cc = b[0].size();

  vii c(ll);

  for (int i = 0; i < ll; i++)
  {
    c[i].resize(cc);

    for (int j = 0; j < cc; j++)
    {
      for (int k = 0; k < b.size(); k++)
      {
        c[i][j] += ((a[i][k] % MOD) * (b[k][j] % MOD)) % MOD;
        c[i][j] %= MOD;
      }
    }
  }
  return c;
}

vii expMat (vii& mat, int e)
{
  if (tab.find(e) != tab.end()) return tab[e];

  if (e == 1) return mat;

   return tab[e] = multi(expMat(mat, ((e + 1) >> 1)), expMat(mat, (e
>> 1)));
}

int32_t main ()
{
  ios::sync_with_stdio(0); cin.tie(0);

  gerar(0);

  vii mat(12);

  for (int i = 0; i < 12; i++)
  {
    mat[i].resize(12);

    for (int j = 0; j < 12; j++)
    {
      bool ok = true;

      for (int k = 0; k < 3; k++)
      {
        if (g[i][k] == g[j][k])
        {
          ok = false;
          break;
        }
      }

      if (ok) mat[i][j] = 1;
    }
  }

  int n;
  cin >> n;

  n--;

  if (n == 0)
  {
    cout << 12;
  }
  else
  {
    vii res = expMat(mat, n);

    vii s(12);

    for (int i = 0; i < 12; i++)
    {
      s[i].resize(1);
      s[i][0] = 1;
    }

    vii matFinal = multi(res, s);

    int resp = 0;

    for (int i = 0; i < matFinal.size(); i++)
    {
      for (int j = 0; j < matFinal[i].size(); j++)
      {
        resp += matFinal[i][j] % MOD;
        resp %= MOD;
      }
    }

    cout << resp << '\n';
  }
}
```

Fibonacci

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

#define int long long
#define MAX 60

int tab[MAX + 1];

int fib (int x) {
  if (tab[x] != -1) return tab[x];
  if (x == 0) return tab[x] = 0;
  if (x == 1) return tab[x] = 1;
  return tab[x] = fib(x - 1) + fib(x - 2);
}

int32_t main () {
  ios_base::sync_with_stdio(false); cin.tie(NULL);

  int t; cin >> t;
  memset(tab, -1, sizeof(tab));

  for (int i = 0; i < t; i++) {
    int x; cin >> x;
    cout << "Fib(" << x << ") = " << fib(x) << "\n";
  }
}
```

Grafos

Percorrer uma Matriz - Cruz:
(Cima, Direita, Baixo, Esquerda)

```
int dl[] {-1, 0, 1,  0};
int dc[] { 0, 1, 0, -1};
```

Percorrer uma Matriz - Vizinhos:
(Cima-Esquerda, Cima, Cima-Direita, Direita, Baixo-Direita, Baixo, Baixo-Esquerda, Esquerda)

```
int dl[] {-1, -1, -1, 0, 1, 1,  1,  0};
int dc[] {-1,  0,  1, 1, 1, 0, -1, -1};
```

Percorrer uma Matriz - Movimento do Cavalo - Sentido Horário:

```
int dl[] {-2, -1, 1, 2,  2,  1, -1, -2};
int dc[] { 1,  2, 2, 1, -1, -2, -2, -1};
```

Flood Fill - Componentes Conexas - Grafo não direcionado: $O(V + E)$

DFS:

```
int grupo[MAXN + 1];
vector<int> g[MAXN + 1];

void DFS (int v) {
  for (int i = 0; i < g[v].size(); i++) {
    int u = g[v][i];

    if (grupo[u] == -1) {
      grupo[u] = grupo[v];
      DFS(u);
    }
  }
}
```

BFS:

```
int grupo[MAXN + 1];
vector<int> g[MAXN + 1];

void BFS (int v) {
  queue<int> fila;
  fila.push(v);

  while (fila.empty() == false) {
    int u = fila.front();
    fila.pop();

    for (int i = 0; i < g[u].size(); i++) {
      int w = g[u][i];

      if (grupo[w] == -1) {
        grupo[w] = grupo[u];
        fila.push(w);
      }
    }
  }
}
```

Verificar se um grafo não direcionado contém ciclo:

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

using namespace std;

#define PB push_back

#define N 1001

bool visited[N];
vector<int> adj[N];

bool findCycle (int x) {
  memset(visited, false, sizeof(visited));

  queue<int> q;

  visited[x] = true;
  q.push(x);

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

    for (auto u : adj[s]) {
      if (!visited[u]) {
        visited[u] = true;
        q.push(u);
      } else if (u != s) {
        return true;
      }
    }
  }

  return false;
}

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

  while (m--) {
    int a, b;
    cin >> a >> b;

    adj[a].PB(b);
    adj[b].PB(a);
  }

  bool flag = false;

  for (int i = 1; i <= n; i++) {
    if (!visited[i] && findCycle(i)) {
      flag = true;
      break;
    }
  }

  if (flag) cout << "CICLICO";
  else cout << "ACICLICO";

  cout << '\n';
}
```

Bellman-Ford - Menor caminho em grafo com pesos negativos: $O(V.E)$

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

using namespace std;

#define PB push_back
#define INF 0x3f3f3f3f

#define N 1001

int n;
int dist[N];
vector<tuple<int, int, int>> edges;

bool bellmanFord (int x) {
  memset(dist, INF, sizeof(dist));
  dist[x] = 0;

  for (int i = 1; i <= n; i++) {
    bool update = false;

    for (auto e : edges) {
      int u, v, w;
      tie(u, v, w) = e;

      if (dist[u] != INF && dist[v] > dist[u] + w) {
        update = true;
        dist[v] = dist[u] + w;
      }
    }

    if (!update) break;
    if (i == n) return false;
  }

  return true;
}

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

  for (int i = 0; i < m; i++) {
    int u, v, w;
    cin >> u >> v >> w;

    edges.PB({u, v, w});
    edges.PB({v, u, w});
  }

  if (bellmanFord(1)) {
    cout << dist[n] << '\n';
  } else {
    cout << "Possui ciclo negativo\n";
  }
}
```

Dijkstra - Menor Caminho: $O(V + E.log(E))$

Exemplo 1:

```
int d[MAXN + 2];
bool v[MAXN + 2];
vector< tuple<int, int> > g[MAXN + 2];

void menorCaminho (int x, int y) {
  for (int i = 0; i < n + 2; i++)
    d[i] = INF;

    d[x] = 0;

  priority_queue< tuple<int, int> > fila;
  fila.push(make_tuple(-d[x], x));

  while (!fila.empty()) {
    int u = get<1>(fila.top());
    fila.pop();

    if (u == y) return;

    v[u] = true;

    for (int i = 0; i < g[u].size(); i++) {
      int w = get<1>(g[u][i]);
      int dist = get<0>(g[u][i]);

      if (!v[w] && d[w] > dist + d[u]) {
        d[w] = dist + d[u];
        fila.push(make_tuple(-d[w], w));
      }
    }
  }
}
```

Exemplo 2: Problema Frete

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

typedef pair<int, int> pii;
#define F first
#define S second
#define PB push_back

#define N 101
#define INF 0x3f3f3f3f

int n;
int dist[N];
bool visited[N];
vector<pii> adj[N];

void dijkstra () {
  memset(dist, INF, sizeof(dist));
  dist[1] = 0;
  priority_queue<pii> pq;
  pq.push({-dist[1], 1});
  while (!pq.empty()) {
    int u = pq.top().S;
    while (visited[u]) {
      pq.pop();
      if (pq.empty()) break;
      u = pq.top().S;
    }
    if (u == n) break;
    visited[u] = 1;
    for (auto x : adj[u]) {
      int v = x.F, d = x.S;
      if (!visited[v] && dist[v] > dist[u] + d) {
        dist[v] = dist[u] + d;
        pq.push({-dist[v], v});
      }
    }
  }
}

int main () {
  ios::sync_with_stdio(0); cin.tie(0);
  int m; cin >> n >> m;
  while (m--) {
    int a, b, c; cin >> a >> b >> c;
    adj[a].PB({b, c});
    adj[b].PB({a, c});
  }
  dijkstra();
  cout << dist[n] << '\n';
}
```

Kruskal - Árvore Geradora Mínima - MST: $O(E.log(V))$

```
#include <tuple>
#include <iostream>
#include <algorithm>

#define MAXN 500
#define MAXM 124750

using namespace std;

int pai[MAXN + 1];
int nivel[MAXN + 1];
tuple<int, int, int> a[MAXM];

int find (int x) {
  if (pai[x] == x)
    return x;

  return pai[x] = find(pai[x]);
}

void join (int x, int y) {
  x = find(x);
  y = find(y);

  if (x == y) return;

  if (nivel[x] > nivel[y])
    pai[y] = x;
  else if (nivel[x] < nivel[y])
    pai[x] = y;
  else {
    pai[y] = x;

    nivel[x]++;
  }
}
```

```
int main () {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  int n, m;
  cin >> n >> m;

  for (int i = 0; i < m; i++)
    cin >> get<1>(a[i]) >> get<2>(a[i]) >> get<0>(a[i]);

  sort(a, a + m);

  for (int i = 1; i < n + 1; i++) {
    pai[i] = i;
    nivel[i] = 1;
  }

  int res = 0;
  for (int i = 0; i < m; i++) {
    int p = get<0>(a[i]);
    int u = get<1>(a[i]);
    int v = get<2>(a[i]);

    int cont = 0;
    if (find(u) != find(v)) {
      join(u, v);

      res += p;

      if (cont == n - 1) break;
    }
  }

  cout << res << "\n";

  return 0;
}
```

Prim - Árvore Geradora Mínima - MST: $O(E.log(V))$

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

#define MAXN 500
#define INF 1000000000

using namespace std;

int n;
int d[MAXN + 1];
bool v[MAXN + 1];
vector< tuple<int, int> > g[MAXN + 1];

void Prim () {
  for (int i = 2; i < n + 1; i++)
    d[i] = INF;

  d[1] = 0;

  priority_queue< tuple<int, int> > fila;
  fila.push(make_tuple(-d[1], 1));

  while (!fila.empty()) {
    int u = get<1>(fila.top());
    fila.pop();

    v[u] = true;

    for (int i = 0; i < g[u].size(); i++) {
      int w = get<0>(g[u][i]);
      int dist = get<1>(g[u][i]);

      if (!v[w] && d[w] > dist) {
        d[w] = dist;
        fila.push(make_tuple(-d[w], w));
      }
    }
  }
```

```
}

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

  while (true) {
    int m;
    cin >> n >> m;

    if (n == 0 && m == 0) return 0;

    memset(v, 0, sizeof(v));
    memset(d, 0, sizeof(d));

    for (int i = 1; i < n + 1; i++)
      g[i].clear();

    for (int i = 0; i < m; i++) {
      int u, v, c;
      cin >> u >> v >> c;

      g[u].push_back(make_tuple(v, c));
      g[v].push_back(make_tuple(u, c));
    }

    Prim();

    int menor = 0;
    for (int i = 1; i < n + 1; i++)
      menor += d[i];

    cout << menor << "\n";
  }

  return 0;
}
```

Ordenação Topológica: $O(E)$

```
#include <bits/stdc++.h>
using namespace std;
#define MAXN 1000
int grau[MAXN + 1];
vector<int> g[MAXN + 1];
int main () {
  int n, m; cin >> n >> m;

  for (int i = 0; i < m; i++) {
    int u, v; cin >> u >> v;
    grau[v]++;
    g[u].push_back(v);
  }

  vector<int> lista;

  for (int i = 1; i < n + 1; i++)
    if (grau[i] == 0)
      lista.push_back(i);

  int ini = 0;
  while (ini < lista.size()) {
    int u = lista[ini];

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

      grau[v]--;

      if (grau[v] == 0)
        lista.push_back(v);
    }

    ini++;
  }

  if (lista.size() < n)
    cout << "Impossivel";
  else {
    cout << lista[0];

    for (int i = 1; i < lista.size(); i++)
      cout << " -> " << lista[i];
  }

  cout << "\n";
}
```

Floyd-Warshall: menor caminho de todos para todos. $O(n^3)$

```
#include <iostream>
using namespace std;
#define MAXN 100
#define INF 1000000000
int d[MAXN][MAXN];
int main () {
  int n, m; cin >> n >> m;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      if (i != j)
        d[i][j] = INF;
  for (int i = 0; i < m; i++) {
    int u, v, w; cin >> u >> v >> w;
    d[u][v] = w;
    d[v][u] = w;
  }
  for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
  int menor = INF;
  for (int i = 0; i < n; i++) {
    int maior = 0;
    for (int j = 0; j < n; j++)
      maior = max(maior, d[i][j]);
    menor = min(menor, maior);
  }
  cout << menor << "\n";
}
```

Menor Ancestral Comum - LCA: $O(N\ .\ log(N))$

```
#include <bits/stdc++.h>
using namespace std;
#define MAXL 16
#define MAXN 50000
int pai[MAXN + 1];
int nivel[MAXN + 1];
int ances[MAXN + 1][MAXL + 1];
tuple<int, int> c[MAXN/2 + 1];
vector<int> g[MAXN + 1];

void DFS (int x) {
  for (int i = 0; i < g[x].size(); i++) {
    int u = g[x][i];
    if (nivel[u] == -1) {
      pai[u] = x;
      nivel[u] = nivel[x] + 1;
      DFS(u);
    }
  }
}

int LCA (int u, int v) {
  if (nivel[u] < nivel[v]) swap(u, v);
  for (int i = MAXL; i >= 0 ; i--)
    if (nivel[u] - (1 << i) >= nivel[v])
      u = ances[u][i];
  if (u == v) return u;
  for (int i = MAXL; i >= 0; i--)
    if (ances[u][i] != ances[v][i]) {
      u = ances[u][i];
      v = ances[v][i];
  }
  return pai[u];
}
int main () {
  ios_base::sync_with_stdio(false); cin.tie(NULL);
  int n; cin >> n;

  for (int i = 1; i < n + 1; i++) {
    int x; cin >> x;

    if (get<0>(c[x]) == 0) get<0>(c[x]) = i;
    else get<1>(c[x]) = i;
  }

  for (int i = 0; i < n - 1; i++) {
    int a, b; cin >> a >> b;
    g[a].push_back(b);
    g[b].push_back(a);
  }
  memset(pai, -1, sizeof(pai));
  memset(nivel, -1, sizeof(nivel));
  memset(ances, -1, sizeof(ances));
  nivel[1] = 0; DFS(1);
  for (int i = 1; i < n + 1; i++) ances[i][0] = pai[i];

  for (int j = 1; j < MAXL + 1; j++)
    for (int i = 1; i < n + 1; i++)
      if (ances[i][j - 1] != -1)
        ances[i][j] = ances[ ances[i][j - 1] ][j - 1];

  int soma = 0;
  for (int i = 1; i < n/2 + 1; i++) {
    int u = get<0>(c[i]);
    int v = get<1>(c[i]);

    int w = LCA(u, v);

    soma += nivel[u] + nivel[v] - 2*nivel[w];
  }

  cout << soma << "\n";
}
```

Exemplo 2: Jogo da Memória

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

using namespace std;

#define PB push_back

#define N 50001
#define MAXL 16

int lv[N];
int fr[N][MAXL];

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

void preprocess (int s, int e)
{
  fr[s][0] = e;
  for (int i = 1; i < MAXL; i++)
    if (fr[s][i - 1] != -1)
      fr[s][i] = fr[fr[s][i - 1]][i - 1];

  for (auto u : adj[s]) if (u != e)
  {
    lv[u] = lv[s] + 1;
    preprocess(u, s);
  }
}

int lca (int u, int v)
{
  if (lv[u] < lv[v]) swap(u, v);

  for (int i = MAXL - 1; i >= 0; i--)
    if (lv[u] - (1 << i) >= lv[v]) u = fr[u][i];

  if (u == v) return u;

  for (int i = MAXL - 1; i >= 0; i--)
    if (fr[u][i] != fr[v][i])
    {
      u = fr[u][i];
      v = fr[v][i];
    }

  return fr[u][0];
}
int main ()
{
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;

  for (int i = 1; i <= n; i++)
  {
    int c;
    cin >> c;
    a[c].PB(i);
  }

  for (int i = 1; i <= n - 1; i++)
  {
    int u, v;
    cin >> u >> v;
    adj[u].PB(v);
    adj[v].PB(u);
  }

  memset(fr, -1, sizeof(fr));

  lv[1] = 0;
  preprocess(1, -1);

  int ans = 0;

  for (int i = 1; i <= n/2; i++)
  {
    int u = a[i][0], v = a[i][1];
    ans += lv[u] + lv[v] - 2*lv[lca(u, v)];
  }

  cout << ans << '\n';
}
```

Caminho Euleriano: Para existir o caminho Euleriano o número de vértices de grau ímpar deve ser zero ou dois. Funcia para multigrafos e grafos com *self-loops*.

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

#define MAXN 1000

using namespace std;

vector<int> lista;
int grau[MAXN + 1];
vector<int> g[MAXN + 1];
map< tuple<int, int>, bool > mapa;

void caminhoEuleriano (int x) {
  for (int i = 0; i < g[x].size(); i++) {
    int u = g[x][i];

    if (mapa.find(make_tuple(x, u)) == mapa.end()) {
      mapa[make_tuple(x, u)] = true;
      mapa[make_tuple(u, x)] = true;

      caminhoEuleriano(u);
    }
  }

  lista.push_back(x);
}

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

  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;

    grau[u]++;
    grau[v]++;
    g[u].push_back(v);
    g[v].push_back(u);
  }

  int ini = 1, impar = 0;
  for (int i = 1; i < n + 1; i++)
    if (grau[i] % 2 == 1) {
      ini = i;
      impar++;
    }

  if (impar > 2)
    cout << "impossivel";
  else {
    cout << "possivel\n";

    caminhoEuleriano(ini);

    cout << lista[0];

    for (int i = 1; i < lista.size(); i++)
      cout << " " << lista[i];
  }

  cout << "\n";

  return 0;
}
```

Grafos Bipartidos:

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

#define MAXN 1000

using namespace std;

int n;
int cor[MAXN + 1];
vector<int> g[MAXN + 1];

void colore (int x) {
  cor[x] = 0;

  queue<int> fila;
  fila.push(x);

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

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

      if (cor[v] == -1) {
        cor[v] = 1 - cor[u];

        fila.push(v);
      }
    }
  }
}

bool bipartido () {
  memset(cor, -1, sizeof(cor));

  for (int i = 1; i < n + 1; i++)
    if (cor[i] == -1)
      colore(i);

  for (int i = 1; i < n + 1; i++)
    for (int j = 0; j < g[i].size(); j++) {
      int v = g[i][j];

      if (cor[i] == cor[v])
        return false;
    }

  return true;
}

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

  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;

    g[u].push_back(v);
    g[v].push_back(u);
  }

  cout << "Grafo Bipartido? : ";

  if (bipartido())
    cout << "Sim";
  else
    cout << "Nao";

  cout << "\n";
}
```

Exemplo 2:

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

using namespace std;

#define PB push_back

#define N 1001

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

bool isBipartite (int x) {
  memset(color, -1, sizeof(color));

  queue<int> q;

  color[x] = 0;
  q.push(x);

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

    for (auto v : adj[s]) {
      if (color[v] == -1) {
        color[v] = 1 - color[s];
        q.push(v);
      } else if (color[v] == color[s]) {
        return false;
      }
    }
  }

  return true;
}

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

  while (m--) {
    int u, v;
    cin >> u >> v;

    adj[u].PB(v);
    adj[v].PB(u);
  }

  if (isBipartite(1)) cout << "YES";
  else cout << "NO";

  cout << '\n';
}
```

Reconstrução de Grafos (Pré-Ordem e In-Ordem):

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

using namespace std;

pair<char, char> g[26];

void build (string pre, string in) {
  char r = pre[0];

  g[r - 'A'] = {'*', '*'};

  int pos = in.find(r);

  string esq_in = "", dir_in = "", esq_pre = "", dir_pre = "";

  esq_in = in.substr(0, pos);
  dir_in = in.substr(pos + 1, in.size() - pos - 1);

  pos = pre.size();

  for (int i = 1; i < pre.size(); i++) {
    if (dir_in.find(pre[i]) != -1) {
      pos = i;
      break;
    }
  }

  esq_pre = pre.substr(1, pos - 1);
  dir_pre = pre.substr(pos, pre.size() - pos);

  if (esq_pre.size() >= 1) {
    g[esq_pre[0] - 'A'] = {'*', '*'};
    g[r - 'A'].first = esq_pre[0];

    if (esq_pre.size() > 1) {
      build(esq_pre, esq_in);
    }
  }

  if (dir_pre.size() >= 1) {
    g[dir_pre[0] - 'A'] = {'*', '*'};
    g[r - 'A'].second = dir_pre[0];

    if (dir_pre.size() > 1) {
      build(dir_pre, dir_in);
    }
  }
}

void posOrder (char no) {
  if (no == '*') return;

  posOrder(g[no - 'A'].first);
  posOrder(g[no - 'A'].second);

  cout << no;
}

int main () {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  string pre, in;
  while (cin >> pre >> in) {
    for (int i = 0; i < 26; i++) {
      g[i] = {'*', '*'};
    }

    char r = pre[0];

    g[r - 'A'] = {'*', '*'};

    build(pre, in);

    posOrder(r);

    cout << "\n";
  }
}
```

Fluxo Máximo - Ford-Fulkerson:
Baunilha e Chocolate

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

#define N 201
#define INF 1000000000

int n, t;
int fr[N];
bool visit[N];
int adj[N][N];
int adjCopy[N][N];

bool bfs (int u) {
  memset(visit, false, sizeof(visit));
  queue<int> q; q.push(u);
  fr[u] = -1; visit[u] = true;
  while (!q.empty()) {
    int v = q.front(); q.pop();
    if (v == t) return true;
    for (int i = 1; i < n + 1; i++) {
      if (!visit[i] && adjCopy[v][i] > 0) {
        q.push(i); fr[i] = v; visit[i] = true;
      }
    }
  }
  return false;
}

int fluxoMaximo (int s) {
  for (int i = 0; i < n + 1; i++)
    for (int j = 0; j < n + 1; j++) adjCopy[i][j] = adj[i][j];

  int fluxo = 0;

  while (bfs(s)) {
    int minAresta = INF;

    for (int u = t; u != s; u = fr[u]) {
      minAresta = min(minAresta, adjCopy[fr[u]][u]);
    }

    for (int u = t; u != s; u = fr[u]) {
      adjCopy[fr[u]][u] -= minAresta;
      adjCopy[u][fr[u]] += minAresta;
    }

    fluxo += minAresta;
  }

  return fluxo;
}

int main () {
  ios::sync_with_stdio(0); cin.tie(0);

  while (true) {
    int m; cin >> n >> m;

    if (n == 0 && m == 0) break;

    int c, b; cin >> t >> c >> b;

    memset(adj, 0, sizeof(adj));

    adj[0][c] = adj[0][b] = INF;

    while (m--) {
      int u, v, x; cin >> u >> v >> x;
      adj[u][v] += x;
      adj[v][u] += x;
    }

    int fluxoTotal = fluxoMaximo(0);

    int minFluxo = min(fluxoMaximo(c), fluxoMaximo(b));

    for (int i = minFluxo; i >= 0; i--)
      if (2*i <= fluxoTotal) { cout << 2*i << '\n'; break; }
  }
```

Potholers:

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

#define INF 1e9
#define MAXN 201

using namespace std;

int n;
int pai[MAXN];
int g[MAXN][MAXN];

bool visit[MAXN];

bool BFS () {
  memset(visit, 0, sizeof(visit));

  queue<int> fila;

  pai[1] = -1; fila.push(1);

  while (!fila.empty()) {
    int x = fila.front(); fila.pop();

    if (x == n) return 1;

    visit[x] = 1;

    for (int i = 1; i < n + 1; i++)
      if (!visit[i] && g[x][i] > 0)
        pai[i] = x, fila.push(i);
  }

  return 0;
}

int fluxoMaximo () {
  int fluxo = 0;

  while (BFS()) {
    int menor = INF;

    for (int u = n; pai[u] != -1; u = pai[u])
      menor = min(menor, g[pai[u]][u]);

    for (int u = n; pai[u] != -1; u = pai[u]) {
      g[pai[u]][u] -= menor;
      g[u][pai[u]] += menor;
    }

    fluxo += menor;
  }

  return fluxo;
}

int main () {
  int t;
  scanf("%d", &t);

  while (t--) {
    memset(g, -1, sizeof(g));

    scanf("%d", &n);

    for (int i = 1; i < n; i++) {
      int m;
      scanf("%d", &m);

      for (int j = 0; j < m; j++) {
        int x;
        scanf("%d", &x);

        if (i == 1 || x == n)
          g[i][x] = 1;
        else
          g[i][x] = INF;

        g[x][i] = 0;
      }
    }

    printf("%d\n", fluxoMaximo());
  }
}
```

Contar o número de nós em uma sub-árvore: $O(N)$

Exemplo 1:

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

using namespace std;

#define PB push_back

#define N 1001

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

void dfs (int s, int e) {
  cnt[s] = 1;
  for (auto u : adj[s]) {
    if (u == e) continue;
    dfs(u, s);
    cnt[s] += cnt[u];
  }
}
int main () {
  int n; cin >> n;

  for (int i = 1; i <= n - 1; i++) {
    int a, b; cin >> a >> b;

    adj[a].PB(b);
    adj[b].PB(a);
  }

  dfs(1, 0);

  cout << cnt[1] << '\n';
}
```

Exemplo 2: Dividindo o Império

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

#define PB push_back
#define INF 0x3f3f3f3f
#define N 100001

int cnt[N];
int menor = INF;
vector<int> adj[N];

void dfsCount (int s, int e) {
  cnt[s] = 1;
  for (auto u : adj[s]) {
    if (u == e) continue;
    dfsCount(u, s);
    cnt[s] += cnt[u];
  }
}

void dfsCut (int s, int e) {
  menor = min(menor, abs(2*cnt[s] - cnt[1]));
  for (auto u : adj[s]) {
    if (u == e) continue;
    dfsCut(u, s);
  }
}

int main () {
  ios::sync_with_stdio(0); cin.tie(0);
  int n; cin >> n;
  for (int i = 0; i < n - 1; i++) {
    int a, b; cin >> a >> b;
    adj[a].PB(b); adj[b].PB(a);
  }
  dfsCount(1, 0); dfsCut(1, 0);
  cout << menor << '\n';
}
```

Encontrar o diâmetro de uma árvore:

Exemplo 1 : Programação dinâmica

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

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back

#define N 1001

int toLeaf[N];
int maxLength[N];
vector<int> adj[N];

void dfs (int s, int e) {
  toLeaf[s] = 0;
  pii aux = {-1, -1};

  for (auto v : adj[s]) {
    if (v == e) continue;

    dfs(v, s);

    aux.S = max(aux.S, toLeaf[v]);
    if (aux.F < aux.S) swap(aux.F, aux.S);

    toLeaf[s] = max(toLeaf[s], 1 + toLeaf[v]);
  }

  if (aux.F == -1) maxLength[s] = 0;
  else if (aux.S == -1) maxLength[s] = aux.F + 1;
  else maxLength[s] = aux.F + aux.S + 2;
}

int diameter (int root, int n) {
  dfs(root, 0);

  int d = 0;
  for (int i = 1; i <= n; i++) {
    d = max(d, maxLength[i]);
  }

  return d;
}

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

  for (int i = 0; i < n - 1; i++) {
    int a, b;
    cin >> a >> b;

    adj[a].PB(b);
    adj[b].PB(a);
  }

  cout << diameter(4, n) << '\n';
}
```

Exemplo 2: Dupla BFS

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

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back
#define INF 0x3f3f3f3f

#define N 1001

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

pii bfs (int s) {
  memset(dist, INF, sizeof(dist));

  queue<int> q;

  dist[s] = 0;
  q.push(s);

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

    for (auto v : adj[u]) {
      if (dist[v] != INF) continue;

      dist[v] = dist[u] + 1;
      q.push(v);
    }
  }

  return {u, dist[u]};
}

int diameter (int root) {
  return bfs(bfs(root).F).S;
}

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

  for (int i = 0; i < n - 1; i++) {
    int a, b;
    cin >> a >> b;

    adj[a].PB(b);
    adj[b].PB(a);
  }

  cout << diameter(1) << '\n';
}
```

Exemplo 3: Dupla BFS

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

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back

#define N 1000001

bool visited[N];
vector<int> adj[N];

pii bfs (int x) {
  memset(visited, 0, sizeof(visited));

  queue<pii> q;

  visited[x] = 1;
  q.push({x, 0});

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

    for (auto v : adj[u.F]) {
      if (!visited[v]) {
        visited[v] = 1;
        q.push({v, 1 + u.S});
      }
    }
  }

  return u;
}

int diameter (int x) {
  return bfs(bfs(x).F).S;
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;

  for (int i = 0; i < n - 1; i++) {
    int a, b;
    cin >> a >> b;

    adj[a].PB(b);
    adj[b].PB(a);
  }

  cout << diameter(1) << '\n';
}
```

Todos os maiores caminhos em um árvore:
Exemplo 1: cuidado, passível de erro kkkkk

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

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back

#define N 1001

pii maxLength[N];
vector<int> adj[N];

void dfsLeaf (int s, int e) {
  maxLength[s] = {0, 0};

  for (auto u : adj[s]) {
    if (u == e) continue;

    dfsLeaf(u, s);

    maxLength[s].S = max(maxLength[s].S,
                         1 + maxLength[u].F);

    if (maxLength[s].F < maxLength[s].S)
      swap(maxLength[s].F, maxLength[s].S);
  }
}

void dfsLength (int s, int e) {
  if (e != 0) {
    int aux = 0;

    if (maxLength[e].F == maxLength[s].F + 1) {
      aux = 1 + maxLength[e].S;
    } else {
      aux = 1 + maxLength[e].F;
    }

    maxLength[s].F = max(maxLength[s].F, aux);
  }

  for (auto u : adj[s]) {
    if (u == e) continue;
    dfsLength(u, s);
  }
}

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

  for (int i = 0; i < n - 1; i++) {
    int a, b;
    cin >> a >> b;

    adj[a].PB(b);
    adj[b].PB(a);
  }

  dfsLeaf(1, 0);
  dfsLength(1, 0);

  for (int i = 1; i <= n; i++)
    cout << maxLength[i].F << ' ';
  cout << '\n';
}
```

Exemplo 2: Problema Metrô da NLogônia

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

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back
#define INF 0x3f3f3f3f

#define MAX 100001

int lv[2][MAX];
pii mx[2][MAX];
vector<int> adj[2][MAX];

void update (pii& p, int x) {
  p.S = max(p.S, x);
  if (p.F < p.S) swap(p.F, p.S);
}

void leaf (int s, int e, int g) {
  mx[g][s] = {0, 0};

  for (auto u : adj[g][s])
    if (u != e) {
      lv[g][u] = 1 + lv[g][s];
      leaf(u, s, g);
      update(mx[g][s], 1 + mx[g][u].F);
    }
}

void max_path (int s, int e, int g) {
  if (e != 0)
    if (mx[g][e].F == 1 + mx[g][s].F)
      update(mx[g][s], 1 + mx[g][e].S);
    else
      update(mx[g][s], 1 + mx[g][e].F);

  for (auto u : adj[g][s])
    if (u != e) max_path(u, s, g);
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;

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

    adj[0][a].PB(b);
    adj[0][b].PB(a);
  }

  for (int i = 1; i < m; i++) {
    int x, y;
    cin >> x >> y;

    adj[1][x].PB(y);
    adj[1][y].PB(x);
  }

  leaf(1, 0, 0);
  max_path(1, 0, 0);

  leaf(1, 0, 1);
  max_path(1, 0, 1);

  int idc = -1, idq = -1, mnc = INF, mnq = INF;

  for (int i = 1; i <= n; i++) {
    if (mx[0][i].F < mnc) {
      idc = i;
      mnc = mx[0][i].F;
    } else if (mx[0][i].F == mnc && lv[0][i] > lv[0][idc]) {
      idc = i;
    }
  }

  for (int i = 1; i <= m; i++) {
    if (mx[1][i].F < mnq) {
      idq = i;
      mnq = mx[1][i].F;
    } else if (mx[1][i].F == mnq && lv[1][i] > lv[1][idq]) {
      idq = i;
    }
  }

  cout << idc << ' ' << idq << '\n';
}
```

Encontrar as pontes em um grafo não direcionado: $O(V + E)$

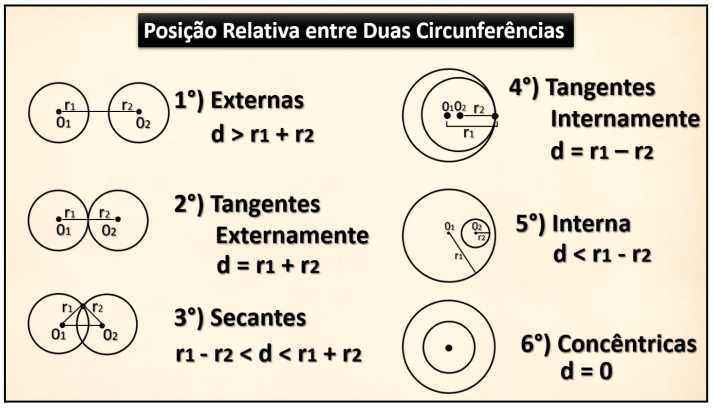
```
#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back
#define debug(arg...) printf(arg)

#define N 1000

int order;
int pre[N];
int low[N];
bool visited[N];
vector<int> adj[N];
vector<pii> bridges;

void dfs (int s, int e) {
  visited[s] = 1;
  pre[s] = low[s] = order++;
  for (auto u : adj[s]) if (u != e) {
    if (visited[u]) {
      low[s] = min(low[s], pre[u]);
    } else {
      dfs(u, s);
      low[s] = min(low[s], low[u]);
      if (low[u] > pre[s]) {
        bridges.PB({s, u});
      }
    }
  }
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;

  while (m--) {
    int u, v;
    cin >> u >> v;
    adj[u].PB(v);
    adj[v].PB(u);
  }

  order = 0;
  memset(visited, 0, sizeof(visited));

  dfs(0, -1);

  for (auto x : bridges) {
    debug("(%d, %d)\n", x.F, x.S);
  }
  debug("\n");
}
```

Compressão de grafo com Union-Find, problema das pontes e diâmetro:

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

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back

#define N 100000
#define M 100000

int t;
pii e[M];
int fr[N];
int lv[N];
int pre[N];
int low[N];
set<pii> cj;
bool visited[N];
vector<int> adj[N];

void findBridges (int s, int f) {
  visited[s] = 1;
  pre[s] = low[s] = t++;
  for (auto u : adj[s]) if (u != f) {
    if (visited[u]) {
      low[s] = min(low[s], pre[u]);
    } else {
      findBridges(u, s);
      low[s] = min(low[s], low[u]);
      if (low[u] > pre[s]) {
        cj.insert({min(s, u), max(s, u)});
      }
    }
  }
}

int find (int x) {
  if (fr[x] == x) return x;
  return fr[x] = find(fr[x]);
}

void join (int x, int y) {
  x = find(x);
  y = find(y);

  if (x == y) return;

  if (lv[x] > lv[y]) swap(x, y);

  fr[x] = y;

  if (lv[x] == lv[y]) lv[y]++;
}

pii bfs (int s, int f) {
  memset(visited, 0, sizeof(visited));
  queue<pii> q;

  visited[s] = 1;
  q.push({s, 0});

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

    for (auto v : adj[u.F]) if (!visited[v]) {
      visited[v] = 1;
      q.push({v, u.S + 1});
    }
  }

  return u;
}

int findDiameter (int s) {
  return bfs(bfs(s, -1).F, -1).S;
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;

  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;

    e[i] = {min(u, v), max(u, v)};

    adj[u].PB(v);
    adj[v].PB(u);
  }

  findBridges(0, -1);

  for (int i = 0; i < n; i++) {
    fr[i] = i;
    lv[i] = 0;
    adj[i].clear();
  }

  for (int i = 0; i < m; i++) {
    if (cj.find(e[i]) == cj.end()) {
      join(e[i].F, e[i].S);
    }
  }

  for (auto x : cj) {
    int u = find(x.F), v = find(x.S);
    adj[u].PB(v);
    adj[v].PB(u);
  }

  cout << findDiameter(find(0)) << '\n';
}
```

Geometria:

Posição relativa entre circunferências: ($R1 \geq R2$, d é a distância entre os centro das circunferências)



Orientação de três pontos no espaço 2D: 0 - Colinear, 1 - Anti Horário, 2 - Horário

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

using namespace std;

#define E 1e-9

struct Point {
  double x, y;
};

struct Vector {
  double x, y;

  Vector(Point a, Point b) {
    x = b.x - a.x;
    y = b.y - a.y;
  }
};

double PV (Vector v1, Vector v2) {
  return v1.x*v2.y - v2.x*v1.y;
}

int orientation (Point a, Point b, Point c) {
  Vector v1(a, b), v2(a, c);

  double aux = PV(v1, v2);

  if (abs(aux) < E) return 0;
  else if (aux > 0) return 1;
  else return 2;
}

int main () {
  Point p1 = {1, 2}, p2 = {4, 4}, p3 = {0, 0};

  int o = orientation(p1, p2, p3);

  cout << "orientation: " << o << '\n';
}
```

Interseção entre segmentos de retas:

P1-Q1 (Segmento 1), P2-Q2 (Segmento 2). Retorna 1 se há interseção e 0 caso contrário.

```
bool intersect (Point p1, Point q1, Point p2, Point q2) {
  if (p1.x > q1.x) swap(p1, q1);
  if (p2.x > q2.x) swap(p2, q2);

  int o1 = orientation(p1, q1, p2);
  int o2 = orientation(p1, q1, q2);
  int o3 = orientation(p2, q2, p1);
  int o4 = orientation(p2, q2, q1);

  if (o1 == 0 && o2 == 0 && o3 == 0 && o4 == 0) {
    if ((p2.x <= p1.x && p1.x <= q2.x) ||
        (p2.x <= q1.x && q1.x <= q2.x) ||
        (p1.x <= p2.x && p2.x <= q1.x) ||
        (p1.x <= q2.x && q2.x <= q1.x))
    {
      return 1;
    }

    return 0;
  }

  if (o1 != o2 && o3 != o4) return 1;

  return 0;
}
```

Interseção entre duas circunferências:

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

using namespace std;

#define MAXN 3001

struct circ { long double x, y, r; };

bool conected (circ a, circ b) {
  long double d = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
  if (abs(a.r - b.r) <= d && d <= a.r + b.r) return true;
  return false;
}
```

Interseção entre Retângulo e Circunferência:

```cpp
#include <cmath>
#include <iostream>

using namespace std;

struct Ponto2D {
  double x, y;

  Ponto2D (double _x = 0., double _y = 0.) : x{_x}, y{_y} {}

  double dist (Ponto2D p) {
    return sqrt((x - p.x)*(x - p.x) + (y - p.y)*(y - p.y));
  }
};

struct Retangulo2D {
  Ponto2D a, b, c, d;

  Retangulo2D (Ponto2D _a, double l = 0., double h = 0.) {
    a = _a;
    b = Ponto2D {_a.x + l, _a.y};
    c = Ponto2D {_a.x + l, _a.y - h};
    d = Ponto2D {_a.x, _a.y - h};
  }
};

struct Circunferencia2D {
  double r;
  Ponto2D c;

  Circunferencia2D (Ponto2D _c, double _r = 0.) : c{_c}, r{_r} {}

  bool intersecao (Retangulo2D rt) {
    double minDist = 0;

    if (c.x < rt.a.x && c.y > rt.a.y) {
      // 1
      minDist = c.dist(rt.a);
    } else if (rt.a.x <= c.x && c.x <= rt.b.x && c.y > rt.a.y) {
      // 2
      minDist = c.y - rt.a.y;
    } else if (c.x > rt.b.x && c.y > rt.b.y) {
      // 3
      minDist = c.dist(rt.b);
    } else if (rt.c.y <= c.y && c.y <= rt.b.y && c.x > rt.b.x) {
      // 4
      minDist = c.x - rt.b.x;
    } else if (c.x > rt.c.x && c.y < rt.c.y) {
      // 5
      minDist = c.dist(rt.c);
    } else if (rt.d.x <= c.x && c.x <= rt.c.x && c.y < rt.c.y) {
      // 6
      minDist = rt.c.y - c.y;
    } else if (c.x < rt.d.x && c.y < rt.d.y) {
      // 7
      minDist = c.dist(rt.d);
    } else if (rt.d.y <= c.y && c.y <= rt.a.y && c.x < rt.d.x) {
      // 8
      minDist = rt.d.x - c.x;
    }

    if (minDist > r)
      return false;

    return true;
  }
};

double raio (string magia, int nivel) {
  if (magia == "fire") {
    switch(nivel) {
      case 1:
        return 20.;
      case 2:
        return 30.;
      case 3:
        return 50.;
    }
  } else if (magia == "water") {
    switch(nivel) {
      case 1:
        return 10.;
      case 2:
        return 25.;
      case 3:
        return 40.;
    }
  } else if (magia == "earth") {
    switch(nivel) {
      case 1:
        return 25.;
      case 2:
        return 55.;
      case 3:
        return 70.;
    }
  } else {
    switch(nivel) {
      case 1:
        return 18.;
      case 2:
        return 38.;
      case 3:
        return 60.;
    }
  }

  return -1.;
}

int dano (string magia) {
  if (magia == "fire") {
    return 200;
  } else if (magia == "water") {
    return 300;
  } else if (magia == "earth") {
    return 400;
  }

  return 100;
}

int main () {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL); cout.tie(NULL);

  freopen("input.txt", "r", stdin);
  freopen("output.txt", "w", stdout);

  int t;
  cin >> t;

  while (t--) {
    double w, h, x0, y0;
    cin >> w >> h >> x0 >> y0;

    string magia; int nivel; double cx, cy;
    cin >> magia >> nivel >> cx >> cy;

    Retangulo2D rt {Ponto2D {x0, y0 + h}, w, h};

    Circunferencia2D cf {Ponto2D {cx, cy}, raio(magia, nivel)};

    if (!cf.intersecao(rt)) {
      cout << 0;
    } else {
      cout << dano(magia);
    }

    cout << "\n";
  }

  return 0;
}
```

Convex Hull: Algoritmo para encontrar a casca de vértices mais externa.

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

using namespace std;

struct Ponto { int x, y; }

Ponto Aux, P, Q, A;
vector<Ponto> Casca;
vector<Ponto> Conjunto;
int N, X, Y, Virada, Contador;

bool Compara(Ponto a, Ponto b){
  if(a.x < b.x){
    return true;
  }else{
    return false;
  }
}

void ConvexHull(){
  P.x = Conjunto[0].x;
  P.y = Conjunto[0].y;

  Casca.push_back(P);

  Conjunto.push_back(P);
  Conjunto.erase(Conjunto.begin() + 0);

  while(true){
    Q.x = Conjunto[0].x;
    Q.y = Conjunto[0].y;

    vector<Ponto>::iterator Indice = Conjunto.begin();

    vector<Ponto>::iterator it;
    for(it = Conjunto.begin(); it != Conjunto.end(); it++){
      A.x = it->x;
      A.y = it->y;

      if(A.x != Q.x || A.y != Q.y){
        Virada = (A.x - P.x)*(Q.y - P.y) - (Q.x - P.x)*(A.y - P.y);

        if(Virada < 0){
          Q.x = A.x;
          Q.y = A.y;
          Indice = it;
        }
      }
    }

    if(Q.x == Casca[0].x && Q.y == Casca[0].y){
      Conjunto.erase(Indice); break;
    }

    Casca.push_back(Q);
    Conjunto.erase(Indice);

    P.x = Q.x;
    P.y = Q.y;
  }
}

int main(){
  while(true){
    scanf("%d", &N);

    if(N == 0){
      break;
    }

    for(int i = 0; i < N; i++){
      scanf("%d%d", &X, &Y);

      Aux.x = X;
      Aux.y = Y;

      Conjunto.push_back(Aux);
    }

    sort(Conjunto.begin(), Conjunto.end(), Compara);

    Contador = 0;

    while(!Conjunto.empty()){
      Contador++;
      ConvexHull();

      Casca.clear();
    }

    if(Contador % 2 == 0){
      printf("menina");
    }else{
      printf("menino");
    }
    printf("\n");
  }
  return 0;
}
```

Exemplo 2 : Macacos me Mordam!

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

#define int long long
#define N 300001

struct ponto { int x, y; };

struct vetor {
  int x, y;
  vetor (ponto a, ponto b) {
    x = b.x - a.x;
    y = b.y - a.y;
  }
};

ponto arr[N];

int PV (vetor a, vetor b) {
  return a.x*b.y - b.x*a.y;
}

int direc (ponto a, ponto b, ponto c) {
  vetor v1 = {a, b}, v2 = {a, c};

  int pv = PV(v1, v2);

  if (pv > 0) return 1; // AH
  if (pv < 0) return -1; // H
  return 0; // COLINEAR
}

bool comp (ponto a, ponto b) {
  return a.x < b.x;
}


int32_t main () {
  ios::sync_with_stdio(0); cin.tie(0);
  int n; cin >> n;
  for (int i = 1; i <= n; i++) cin >> arr[i].x >> arr[i].y;
  sort(arr + 1, arr + n + 1, comp);
  stack<ponto> q;
  int i = 3;
  ponto a = arr[1], b = arr[2];
  while (i <= n) {
    ponto c = arr[i];
    if (direc(a, b, c) < 0) {
      q.push(a); a = b; b = c; i++;
    } else {
      if (!q.empty()) {
        b = a; a = q.top(); q.pop();
      } else {
        b = c; i++;
      }
    }
  }

  q.push(a);
  q.push(b);

  cout << ((int) q.size() - 1)  << '\n';
}
```

Circle Sweep:

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

using namespace std;

#define int long long

#define F first
#define S second
#define SQ(x) ((x)*(x))

#define N 500000

struct Ponto {
  int x, y;
  Ponto () {}
  Ponto (int xx, int yy) : x{xx}, y{yy} {}
};

struct Vetor {
  int x, y;
  Vetor () {}
  Vetor (Ponto a, Ponto b) {
    x = b.x - a.x;
    y = b.y - a.y;
  }
};

Ponto p[N];
pair<Vetor, int> v[N];

bool top (Vetor a) {
  return a.y < 0 || a.y == 0 && a.x < 0;
}

int cross (Vetor a, Vetor b) {
  return a.x*b.y - a.y*b.x;
}

bool polarLess (Vetor a, Vetor b) {
  if (top(a) != top(b)) return top(a);
  return cross(a, b) > 0;
}

bool comp (pair<Vetor, int> a, pair<Vetor, int> b) {
  return polarLess(a.F, b.F);
}

int32_t main () {
  ios::sync_with_stdio(0); cin.tie(0);
  int n; cin >> n;
  for (int i = 0; i < n; i++) {
    int x, y; cin >> x >> y;
    p[i] = Ponto(x, y);
  }
  Ponto c; cin >> c.x >> c.y;

  for (int i = 0; i < n; i++) {
    Ponto a = p[i], b = p[(i + 1) % n];

    if (a.x == b.x && a.x == c.x) {
      cout << 'N' << '\n';
      return 0;
    } else if (a.y == b.y && a.y == c.y) {
      cout << 'N' << '\n';
      return 0;
    }
  }

  for (int i = 0; i < n; i++)
    v[i] = {Vetor(c, p[i]), i};

  sort(v, v + n, comp);

  int ini;

  for (int i = 0; i < n; i++)
    if (v[i].S == 0) {
      ini = i;
      break;
    }

  bool ok1 = true, ok2 = true;

  for (int i = 0; i < n; i++)
    if (v[(ini + i) % n].S != i) {
      ok1 = false;
      break;
    }

  for (int i = 0; i < n; i++) {
    int aux = (ini - i) % n;
    if (aux < 0) aux += n;

    if (v[aux].S != i) {
      ok2 = false;
      break;
    }
  }

  if (ok1 || ok2) cout << 'S';
  else cout << 'N';

  cout << '\n';
}
```

Operações Gerais em Geometria: Jardim de Infância

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

using namespace std;

struct Ponto
{
  double x, y;
  Ponto () { x = 0.0; y = 0.0; }
  Ponto (double xx, double yy) : x(xx), y(yy) {}
  bool operator == (Ponto const& p) const
  {
    if (x == p.x && y == p.y) return true;
    return false;
  }
};

struct Vetor {
  double x, y, modulo;
  Vetor (Ponto pi, Ponto pf)
  {
    x = pf.x - pi.x;
    y = pf.y - pi.y;
    modulo = sqrt(x*x + y*y);
  }
};

double produtoEscalar (Vetor v1, Vetor v2) {
  return v1.x*v2.x + v1.y*v2.y;
}

double produtoVetorial (Vetor v1, Vetor v2) {
  return v1.x*v2.y - v2.x*v1.y;
}

bool colinear (Ponto a, Ponto b, Ponto c) {
  if ((b.x - a.x)*(c.y - a.y) == (c.x - a.x)*(b.y - a.y))
   return true;
  return false;
}

Ponto pontoMedio (Ponto p1, Ponto p2) {
  Ponto p ((p1.x + p2.x)/2.0, (p1.y + p2.y)/2.0);
  return p;
}

int main () {
  ios_base::sync_with_stdio(0); cin.tie(0);

  Ponto p[8];
  for (int i = 1; i < 8; i++) cin >> p[i].x >> p[i].y;

  Vetor v12 (p[1], p[2]), v13 (p[1], p[3]);

  if (produtoEscalar(v12, v13) > 0.0) {
    if (v12.modulo == v13.modulo) {
      if (colinear(p[2], p[3], p[4]) == true &&
        colinear(p[2], p[3], p[5]) == true) {
        if (pontoMedio(p[2], p[3]) == pontoMedio(p[4], p[5])) {
          Vetor v23 (p[2], p[3]), v45 (p[4], p[5]);

          if (v23.modulo > v45.modulo) {
            Vetor v46 (p[4], p[6]), v57 (p[5], p[7]);

            if (produtoEscalar(v23, v46) == 0.0 &&
              produtoEscalar(v23, v57) == 0.0) {
              if (v46.modulo == v57.modulo) {
                Vetor v21 (p[2], p[1]), v26 (p[2], p[6]);

                if ((produtoVetorial(v23, v21) > 0.0 &&
                  produtoVetorial(v23, v26) < 0.0) ||
                  produtoVetorial(v23, v21) < 0.0 &&
                  produtoVetorial(v23, v26) > 0.0) {
                  cout << "S";
                } else {
                  cout << "N";
                }
              } else {
                cout << "N";
              }
            } else {
              cout << "N";
            }
          } else {
            cout << "N";
          }
        } else {
          cout << "N";
        }
      } else {
        cout << "N";
      }
    } else {
      cout << "N";
    }
  } else {
    cout << "N";
  }

  cout << "\n";
}
```