# GEMP - UFC Quixadá - ICPC Library

## Contents


**1 Data Structures** 1
- 1.1 BIT . . . 1
- 1.2 BIT 2D . . . 2
- 1.3 BIT In Range . . . 2
- 1.4 Dynamic Median . . . 3
- 1.5 Implicit Treap . . . 3
- 1.6 LiChao Tree . . . 4
- 1.7 MergeSort Tree . . . 4
- 1.8 Policy Based Tree . . . 5
- 1.9 Queue Query . . . 5
- 1.10 Randomized Heap . . . 5
- 1.11 Range Color . . . 6
- 1.12 Segment Tree . . . 6
- 1.13 Segment Tree 2D . . . 7
- 1.14 Segment Tree Iterative . . . 7
- 1.15 Segment Tree Lazy . . . 8
- 1.16 Segment Tree Persistent . . . 9
- 1.17 Sparse Table . . . 9
- 1.18 SQRT Decomposition . . . 10
- 1.19 SQRT Tree . . . 10
- 1.20 Stack Query . . . 11
- 1.21 Treap . . . 12
- 1.22 Union Find . . . 12
- 1.23 Wavelet Tree . . . 13

**2 Graph Algorithms** 14
- 2.1 2-SAT . . . 14
- 2.2 Arborescence . . . 14
- 2.3 Articulation Point . . . 15
- 2.4 BFS 0-1 . . . 15
- 2.5 Bridge . . . 16
- 2.6 Centroid . . . 16
- 2.7 Centroid Decomposition . . . 17
- 2.8 Checking Bipartiteness Online . . . 17
- 2.9 Dinic . . . 17
- 2.10 Flow With Demand . . . 18
- 2.11 Floyd Warshall . . . 19
- 2.12 Prim . . . 19
- 2.13 HLD . . . 19
- 2.14 Kruskal . . . 20
- 2.15 LCA . . . 20
- 2.16 Min-Cut . . . 21
- 2.17 Minimum Cost Maximum Flow . . . 22
- 2.18 Strongly Connected Component . . . 22
- 2.19 Topological Sort . . . 23
- 2.20 Tree ID . . . 23
- 2.21 Vertex Cover In Tree . . . 23

**3 Dynamic Programming** 23
- 3.1 Divide and Conquer Optimization . . . 23
- 3.2 Divide and Conquer Optimization Implementation . . . 24
- 3.3 Knuth Optimization . . . 24
- 3.4 Knuth Optimization Implementation . . . 24

**4 Math** 24
- 4.1 Basic Math . . . 24
- 4.2 BigInt . . . 25
- 4.3 Binomial Coefficients . . . 27
- 4.4 Chinese Remainder Theorem . . . 28
- 4.5 Determinant . . . 28
- 4.6 Division Trick . . . 28
- 4.7 Euler's totient . . . 28
- 4.8 Extended Euclidean . . . 29
- 4.9 Fraction . . . 29
- 4.10 FFT . . . 30
- 4.11 Floyd Cycle Finding . . . 31
- 4.12 Function Root Using Newton . . . 31
- 4.13 Gauss . . . 31
- 4.14 Gauss Xor . . . 32
- 4.15 Gray Code . . . 32
- 4.16 Matrix . . . 32
- 4.17 Modular Arithmetic . . . 33
- 4.18 Modular Integer . . . 33
- 4.19 Montgomery Multiplication . . . 34
- 4.20 NTT . . . 34
- 4.21 Prime Number . . . 35
- 4.22 Rank Matrix . . . 36
- 4.23 Simpson Integration . . . 36
- 4.24 Sieve And Primes . . . 36

**5 Geometry** 37
- 5.1 Basic Geometry . . . 37
- 5.2 Circle Area Union . . . 41
- 5.3 Circles to Tree . . . 42
- 5.4 Count Lattices . . . 42
- 5.5 Concave Polygon . . . 43
- 5.6 Convex Hull . . . 43
- 5.7 Convex Hull Trick . . . 43
- 5.8 Convex Polygon . . . 44
- 5.9 Nearest Pair Of Points . . . 44
- 5.10 Point 3D . . . 45

**6 String Algorithms** 46
- 6.1 Aho Corasick . . . 46
- 6.2 KMP . . . 47
- 6.3 Manacher . . . 47
- 6.4 Min Cyclic String . . . 48
- 6.5 String Hashing . . . 48
- 6.6 Suffix Automaton . . . 48
- 6.7 Suffix Array . . . 49
- 6.8 Suffix Tree . . . 50
- 6.9 Trie . . . 51
- 6.10 Z Function . . . 51

**7 Miscellaneous** 51
- 7.1 Counting Inversions . . . 51
- 7.2 Histogram . . . 52
- 7.3 Identify Pattern . . . 52
- 7.4 Kadane 1D and 2D . . . 52
- 7.5 Longest Increasing Subsequence . . . 52
- 7.6 Mo Algorithm . . . 53
- 7.7 Polyominoes . . . 53
- 7.8 Scheduling Jobs . . . 54
- 7.9 Sprague Grundy . . . 54

**8 Theorems and Formulas** 54
- 8.1 Binomial Coefficients . . . 54
- 8.2 Catalan Number . . . 54
- 8.3 Euler's Totient . . . 55
- 8.4 Formulas . . . 55
- 8.5 Graph . . . 55
- 8.6 Manhattan Distance . . . 55
- 8.7 Primes . . . 55


# 1 Data Structures

## 1.1 BIT

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

```
class Bit{
private:
  typedef long long t_bit;
  int nBit;
  int nLog;
  vector<t_bit> bit;
public:
  Bit(int n){
    nBit = n;
    nLog = 20;
    bit.resize(nBit + 1, 0);
  }
  //1-indexed
  t_bit get(int i){
    t_bit s = 0;
    for (; i > 0; i -= (i & -i))
      s += bit[i];
    return s;
  }
  //1-indexed [l, r]
  t_bit get(int l, int r){
    return get(r) - get(l - 1);
  }
  //1-indexed
  void add(int i, t_bit value){
    assert(i > 0);
    for (; i <= nBit; i += (i & -i))
      bit[i] += value;
  }
  t_bit lower_bound(t_bit value){
    t_bit sum = 0;
    int pos = 0;
    for (int i = nLog; i >= 0; i--){
      if ((pos + (1 << i) <= nBit) and (sum + bit[pos + (1 << i)] <
          value)){
        sum += bit[pos + (1 << i)];
        pos += (1 << i);
      }
    }
    return pos + 1;
  }
};
```

## 1.2 BIT 2D

```
#include <bits/stdc++.h>
using namespace std;
class Bit2d{
private:
  typedef long long t_bit;
  vector<vector<t_bit>> bit;
  int nBit, mBit;
public:
  Bit2d(int n, int m){
    nBit = n;
    mBit = m;
    bit.resize(nBit + 1, vector<t_bit>(mBit + 1, 0));
  }
  //1-indexed
  t_bit get(int i, int j){
    t_bit sum = 0;
    for (int a = i; a > 0; a -= (a & -a))
      for (int b = j; b > 0; b -= (b & -b))
        sum += bit[a][b];
    return sum;
  }
  //1-indexed
  t_bit get(int a1, int b1, int a2, int b2){
    return get(a2, b2) - get(a2, b1 - 1) - get(a1 - 1, b2) + get(a1 -
        1, b1 - 1);
  }
  //1-indexed [i, j]
  void add(int i, int j, t_bit value){
    for (int a = i; a <= nBit; a += (a & -a))
      for (int b = j; b <= mBit; b += (b & -b))
        bit[a][b] += value;
  }
};
```

## 1.3 BIT In Range

```
#include <bits/stdc++.h>
using namespace std;
class BitRange{
private:
  typedef long long t_bit;
  vector<t_bit> bit1, bit2;
  t_bit get(vector<t_bit> &bit, int i){
    t_bit sum = 0;
    for (; i > 0; i -= (i & -i))
      sum += bit[i];
    return sum;
  }
  void add(vector<t_bit> &bit, int i, t_bit value){
    for (; i < (int)bit.size(); i += (i & -i))
      bit[i] += value;
  }
public:
  BitRange(int n){
    bit1.assign(n + 1, 0);
    bit2.assign(n + 1, 0);
  }
  //1-indexed [i, j]
  void add(int i, int j, t_bit v){
    add(bit1, i, v);
    add(bit1, j + 1, -v);
    add(bit2, i, v * (i - 1));
    add(bit2, j + 1, -v * j);
  }
  //1-indexed
  t_bit get(int i){
    return get(bit1, i) * i - get(bit2, i);
  }
  //1-indexed [i,j]
  t_bit get(int i, int j){
    return get(j) - get(i - 1);
  }
};
```

## 1.4 Dynamic Median

```
#include <bits/stdc++.h>
using namespace std;
class DinamicMedian{
  typedef int t_median;
private:
  priority_queue<t_median> mn;
  priority_queue<t_median, vector<t_median>, greater<t_median>> mx;
public:
  double median(){
    if (mn.size() > mx.size())
      return mn.top();
    else
      return (mn.top() + mx.top()) / 2.0;
  }
  void push(t_median x){
    if (mn.size() <= mx.size())
      mn.push(x);
    else
      mx.push(x);
    if ((!mx.empty()) and (!mn.empty())){
      while (mn.top() > mx.top()){
        t_median a = mx.top();
        mx.pop();
        t_median b = mn.top();
        mn.pop();
        mx.push(b);
        mn.push(a);
      }
    }
  }
};
```

## 1.5 Implicit Treap

```
#include <bits/stdc++.h>
using namespace std;
namespace ITreap{
  const int N = 500010;
  typedef long long treap_t;
  treap_t X[N];
  int en = 1, Y[N], sz[N], L[N], R[N], root;

  const treap_t neutral = 0;
  treap_t op_val[N];
  bool rev[N];
  inline treap_t join(treap_t a, treap_t b, treap_t c){
    return a + b + c;
  }
  void calc(int u) { // update node given children info
    sz[u] = sz[L[u]] + 1 + sz[R[u]];
    // code here, no recursion
    op_val[u] = join(op_val[L[u]], X[u], op_val[R[u]]);
  }
  void unlaze(int u) {
    if(!u) return;
    // code here, no recursion
    if (rev[u]){
      if(L[u])
        rev[L[u]] ^= rev[u];
      if(R[u])
        rev[R[u]] ^= rev[u];
      swap(L[u], R[u]);
      rev[u] = false;
    }
  }
  void split(int u, int s, int &l, int &r) { // l gets first s, r gets
      remaining
    unlaze(u);
    if(!u) return (void) (l = r = 0);
    if(sz[L[u]] < s) { split(R[u], s - sz[L[u]] - 1, l, r); R[u] = l;
        l = u; }
    else { split(L[u], s, l, r); L[u] = r; r = u; }
    calc(u);
  }
  int merge(int l, int r) { // els on l <= els on r
    unlaze(l); unlaze(r);
    if(!l || !r) return l + r;
    int u;
    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
    else { L[r] = merge(l, L[r]); u = r; }
    calc(u);
    return u;
  }
  int new_node(treap_t x){
    X[en] = x;
    op_val[en] = x;
    rev[en] = false;
    return en++;
  }
  int nth(int u, int idx){
    if(!u)
      return 0;
    unlaze(u);
    if(idx <= sz[L[u]])
      return nth(L[u], idx);
    else if(idx == sz[L[u]] + 1)
      return u;
    else
      return nth(R[u], idx - sz[L[u]] - 1);
  }
//Public
  void init(int n=N-1) { // call before using other funcs
    //init position 0
    sz[0] = 0;
    op_val[0] = neutral;
    //init Treap
    root = 0;
    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i] = 1; L[i] = R[i
        ] = 0; }
    random_shuffle(Y + 1, Y + n + 1);
  }
  //0-indexed
  void insert(int idx, int val){
    int a, b;
    split(root, idx, a, b);
    root = merge(merge(a, new_node(val)), b);
  }
```

```
  //0-indexed
  void erase(int idx){
    int a, b, c, d;
    split(root, idx, a, b);
    split(b, 1, c, d);
    root = merge(a, d);
  }
  //0-indexed
  treap_t nth(int idx){
    int u = nth(root, idx+1);
    return X[u];
  }
  //0-indexed [l, r]
  treap_t query(int l, int r){
    if(l > r) swap(l, r);
    int a, b, c, d;
    split(root, l, a, d);
    split(d, r - l + 1, b, c);
    treap_t ans = op_val[b];
    root = merge(a, merge(b, c));
    return ans;
  }
  //0-indexed [l, r]
  void reverse(int l, int r){
    if (l > r) swap(l, r);
    int a, b, c, d;
    split(root, l, a, d);
    split(d, r - l + 1, b, c);
    if(b)
      rev[b] ^= 1;
    root = merge(a, merge(b, c));
  }
};
```

## 1.6 LiChao Tree

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 0x3f3f3f3f;
class LiChaoTree{
private:
  typedef int t_line;
  struct Line{
    t_line k, b;
    Line() {}
    Line(t_line k, t_line b) : k(k), b(b) {}
  };
  int n_tree, min_x, max_x;
  vector<Line> li_tree;
  t_line f(Line l, int x){
    return l.k * x + l.b;
  }
  void add(Line nw, int v, int l, int r){
    int m = (l + r) / 2;
    bool lef = f(nw, l) > f(li_tree[v], l);
    bool mid = f(nw, m) > f(li_tree[v], m);
    if (mid)
      swap(li_tree[v], nw);
    if (r - l == 1)
      return;
    else if (lef != mid)
      add(nw, 2 * v, l, m);
    else
      add(nw, 2 * v + 1, m, r);
  }
  int get(int x, int v, int l, int r){
    int m = (l + r) / 2;
    if (r - l == 1)
      return f(li_tree[v], x);
    else if (x < m)
      return max(f(li_tree[v], x), get(x, 2 * v, l, m));
    else
      return max(f(li_tree[v], x), get(x, 2 * v + 1, m, r));
  }
public:
  LiChaoTree(int mn_x, int mx_x){
    min_x = mn_x;
    max_x = mx_x;
    n_tree = max_x - min_x + 5;
    li_tree.resize(4 * n_tree, Line(0, -INF));
  }
  void add(t_line k, t_line b){
    add(Line(k, b), 1, min_x, max_x);
  }
  t_line get(int x){
    return get(x, 1, min_x, max_x);
  }
};
```

## 1.7 MergeSort Tree

```
#include <bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
class MergeSortTree{
private:
  typedef vector<int> Node;
  Node neutral;
  vector<Node> st;
  int n;
  inline void join(Node &a, Node &b, Node &ans){
    ans.resize(a.size() + b.size());
    merge(all(a), all(b), ans.begin());
  }
  inline int szEq(int node, int k){
    return upper_bound(all(st[node]), k) - lower_bound(all(st[node]),
        k);
  }
  inline int szLt(int node, int k){
    return lower_bound(all(st[node]), k) - st[node].begin();
  }
public:
  template <class MyIterator>
  MergeSortTree(MyIterator begin, MyIterator end){
    int sz = end - begin;
    for (n = 1; n < sz; n <<= 1);
    st.assign(n << 1, neutral);
    for (int i = 0; i < sz; i++, begin++)
      st[i + n].assign(1, *begin);
    for (int i = n - 1; i > 1; i--){
```

```
      int l = (i << 1);
      join(st[l], st[l+1], st[i]);
    }
  }
  // 0-indexed
  // Counts the number of elements less than k in the range [L..R]
  int lt(int l, int r, int k){
    int ans = 0;
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1){
      if (l & 1)
        ans += szLt(l++, k);
      if (r & 1)
        ans += szLt(--r, k);
    }
    return ans;
  }
  // 0-indexed
  // Counts the number of elements equal to k in the range [L..R]
  int eq(int l, int r, int k){
    int ans = 0;
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1){
      if (l & 1)
        ans += szEq(l++, k);
      if (r & 1)
        ans += szEq(--r, k);
    }
    return ans;
  }
};
```

## 1.8 Policy Based Tree

```
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> OrderedSet;
typedef tree<int, int, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> OrderedMap;
//order_of_key (k) : Number of items strictly smaller than k .
//find_by_order(k) : K-th element in a set (counting from zero).
```

## 1.9 Queue Query

```
#include <bits/stdc++.h>
using namespace std;
class QueueQuery{
private:
  typedef long long t_queue;
  stack<pair<t_queue, t_queue>> s1, s2;
  t_queue cmp(t_queue a, t_queue b){
    return min(a, b);
  }
  void move(){
    if (s2.empty()){
      while (!s1.empty()){
        t_queue element = s1.top().first;
        s1.pop();
        t_queue result = s2.empty() ? element : cmp(element, s2.top().
            second);
        s2.push({element, result});
      }
    }
  }
public:
  void push(t_queue x){
    t_queue result = s1.empty() ? x : cmp(x, s1.top().second);
    s1.push({x, result});
  }
  void pop(){
    move();
    s2.pop();
  }
  t_queue front(){
    move();
    return s2.top().first;
  }
  t_queue query(){
    if (s1.empty() || s2.empty())
      return s1.empty() ? s2.top().second : s1.top().second;
    else
      return cmp(s1.top().second, s2.top().second);
  }
  t_queue size(){
    return s1.size() + s2.size();
  }
};
```

## 1.10 Randomized Heap

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

typedef int f_type;
struct Node{
  f_type value;
  Node *l, *r;
  Node(f_type x = 0): value(x){
    l = r = nullptr;
  }
};
inline bool heapMin(f_type a, f_type b){
  return a > b;
}
inline bool heapMax(f_type a, f_type b){
  return a < b;
}
struct RandomizedHeap{
  Node *root;
  int sz;
  RandomizedHeap(){
    srand(time(NULL));
    root = nullptr;
    sz = 0;
  }
  void rdFree(Node *n){
```

```
    if(n == nullptr) return;
    rdFree(n->l); rdFree(n->r);
    delete n;
  }
  ~RandomizedHeap(){
    rdFree(root);
  }
  Node* merge(Node *t1, Node *t2) {
    if(!t1 || !t2)
      return t1 ? t1 : t2;
    if(heapMin(t1->value, t2->value))
      swap(t1, t2);
    if(rand() & 1)
      swap(t1->l, t1->r);
    t1->l = merge(t1->l, t2);
    return t1;
  }
  //Can be performed in O(logn) on average.
  void merge(RandomizedHeap &oth){
    root = merge(root, oth.root);
    sz += oth.sz;
    oth.root = nullptr;
  }
  int top(){
    return (root != nullptr) ? root->value : 0;
  }
  void pop(){
    if(root == nullptr) return;
    Node *l = root->l;
    Node *r = root->r;
    delete root;
    root = merge(l, r);
    sz--;
  }
  void push(int x){
    Node *nw = new Node(x);
    root = merge(root, nw);
    sz++;
  }
  int size(){
    return sz;
  }
};
```

## 1.11 Range Color

```
#include <bits/stdc++.h>
using namespace std;
class RangeColor{
private:
  typedef long long ll;
  struct Node{
    ll l, r;
    int color;
    Node() {}
    Node(ll l1, ll r1, int color1) : l(l1), r(r1), color(color1) {}
    bool operator<(const Node &oth) const{
      return r < oth.r;
    }
  };
  std::set<Node> st;
  vector<ll> ans;
public:
  RangeColor(ll first, ll last, int maxColor){
    ans.resize(maxColor + 1);
    ans[0] = last - first + 1LL;
    st.insert(Node(first, last, 0));
  }
  //get color in position x
  int get(ll x){
    auto p = st.upper_bound(Node(0, x - 1LL, -1));
    return p->color;
  }
  //set newColor in [a, b]
  void set(ll a, ll b, int newColor){
    auto p = st.upper_bound(Node(0, a - 1LL, -1));
    assert(p != st.end());
    ll l = p->l;
    ll r = p->r;
    int oldColor = p->color;
    ans[oldColor] -= (r - l + 1LL);
    p = st.erase(p);
    if (l < a){
      ans[oldColor] += (a - l);
      st.insert(Node(l, a - 1LL, oldColor));
    }
    if (b < r){
      ans[oldColor] += (r - b);
      st.insert(Node(b + 1LL, r, oldColor));
    }
    while ((p != st.end()) and (p->l <= b)){
      l = p->l;
      r = p->r;
      oldColor = p->color;
      ans[oldColor] -= (r - l + 1LL);
      if (b < r){
        ans[oldColor] += (r - b);
        st.erase(p);
        st.insert(Node(b + 1LL, r, oldColor));
        break;
      }else{
        p = st.erase(p);
      }
    }
    ans[newColor] += (b - a + 1LL);
    st.insert(Node(a, b, newColor));
  }
  ll countColor(int x){
    return ans[x];
  }
};
```

## 1.12 Segment Tree

```
#include <bits/stdc++.h>
using namespace std;
class SegTree{
private:
  typedef long long Node;
  Node neutral = 0;
```

```
  vector<Node> st;
  vector<int> v;
  int n;
  Node join(Node a, Node b){
    return (a + b);
  }
  void build(int node, int i, int j){
    if (i == j){
      st[node] = v[i];
      return;
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    build(l, i, m);
    build(r, m + 1, j);
    st[node] = join(st[l], st[r]);
  }
  Node query(int node, int i, int j, int a, int b){
    if ((i > b) or (j < a))
      return neutral;
    if ((a <= i) and (j <= b))
      return st[node];
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    return join(query(l, i, m, a, b), query(r, m + 1, j, a, b));
  }
  void update(int node, int i, int j, int idx, Node value){
    if (i == j){
      st[node] = value;
      return;
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    if (idx <= m)
      update(l, i, m, idx, value);
    else
      update(r, m + 1, j, idx, value);
    st[node] = join(st[l], st[r]);
  }
public:
  template <class MyIterator>
  SegTree(MyIterator begin, MyIterator end){
    n = end - begin;
    v = vector<int>(begin, end);
    st.resize(4 * n + 5);
    build(1, 0, n - 1);
  }
  //0-indexed [a, b]
  Node query(int a, int b){
    return query(1, 0, n - 1, a, b);
  }
  //0-indexed
  void update(int idx, int value){
    update(1, 0, n - 1, idx, value);
  }
};
```

## 1.13 Segment Tree 2D

```
#include <bits/stdc++.h>
using namespace std;
struct SegTree2D{
private:
  int n, m;
  typedef int Node;
  Node neutral = -0x3f3f3f3f;
  vector<vector<Node>> seg;
  Node join(Node a, Node b){
    return max(a, b);
  }
public:
  SegTree2D(int n1, int m1){
    n = n1, m = m1;
    seg.assign(2 * n, vector<Node>(2 * m, 0));
  }
  void update(int x, int y, int val){
    assert(0 <= x && x < n && 0 <= y && y < m);
    x += n, y += m;
    seg[x][y] = val;
    for (int j = y / 2; j > 0; j /= 2)
      seg[x][j] = join(seg[x][2 * j], seg[x][2 * j + 1]);
    for (x /= 2; x > 0; x /= 2){
      seg[x][y] = join(seg[2 * x][y], seg[2 * x + 1][y]);
      for (int j = y / 2; j > 0; j /= 2){
        seg[x][j] = join(seg[x][2 * j], seg[x][2 * j + 1]);
      }
    }
  }
  vector<int> getCover(int l, int r, int N){
    l = std::max(0, l);
    r = std::min(N, r);
    vector<int> ans;
    for (l += N, r += N; l < r; l /= 2, r /= 2){
      if (l & 1)
        ans.push_back(l++);
      if (r & 1)
        ans.push_back(--r);
    }
    return ans;
  }
  Node query(int x1, int y1, int x2, int y2){
    auto c1 = getCover(x1, x2 + 1, n);
    auto c2 = getCover(y1, y2 + 1, m);
    Node ans = neutral;
    for (auto i : c1){
      for (auto j : c2){
        ans = join(ans, seg[i][j]);
      }
    }
    return ans;
  }
};
```

## 1.14 Segment Tree Iterative

```cpp
#include <bits/stdc++.h>
using namespace std;
class SegTreeIterative{
private:
  typedef long long Node;
  Node neutral = 0;
  vector<Node> st;
  int n;
  inline Node join(Node a, Node b){
    return a + b;
  }
public:
  template <class MyIterator>
  SegTreeIterative(MyIterator begin, MyIterator end){
    int sz = end - begin;
    for (n = 1; n < sz; n <<= 1);
    st.assign(n << 1, neutral);
    for (int i = 0; i < sz; i++, begin++)
      st[i + n] = (*begin);
    for (int i = n - 1; i > 1; i--){
      st[i] = join(st[(i << 1)], st[(i << 1) + 1]);
    }
  }
  //0-indexed
  void update(int i, Node x){
    st[i += n] = x;
    for (i >>= 1; i; i >>= 1)
      st[i] = join(st[i << 1], st[(i << 1) + 1]);
  }
  //0-indexed [l, r]
  Node query(int l, int r){
    Node ansL = neutral, ansR = neutral;
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1){
      if (l & 1)
        ansL = join(ansL, st[l++]);
      if (r & 1)
        ansR = join(st[--r], ansR);
    }
    return join(ansL, ansR);
  }
  Node lower_bound(int k){
    int no=1, l=0, r=n-1;
    while(l<r){
      int mid = (l+r)>>1;
      int lo = no<<1;
      if(st[lo] >= k){
        no = lo;
        r = mid;
      }else{
        k -= st[lo];
        no = lo + 1;
        l = mid + 1;
      }
    }
    if(st[no] >= k)
      return l;
    else
      return -1;
  }
};
```

## 1.15 Segment Tree Lazy

```cpp
#include <bits/stdc++.h>
using namespace std;
class SegTreeLazy{
private:
  typedef long long Node;
  vector<Node> st;
  vector<long long> lazy;
  vector<int> v;
  int n;
  Node neutral = 0;
  inline Node join(Node a, Node b){
    return a + b;
  }
  inline void upLazy(int &node, int &i, int &j){
    if (lazy[node] != 0){
      st[node] += lazy[node] * (j - i + 1);
      //st[node] += lazy[node];
      if (i != j){
        lazy[(node << 1)] += lazy[node];
        lazy[(node << 1) + 1] += lazy[node];
      }
      lazy[node] = 0;
    }
  }
  void build(int node, int i, int j){
    if (i == j){
      st[node] = v[i];
      return;
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    build(l, i, m);
    build(r, m + 1, j);
    st[node] = join(st[l], st[r]);
  }
  Node query(int node, int i, int j, int a, int b){
    upLazy(node, i, j);
    if ((i > b) or (j < a))
      return neutral;
    if ((a <= i) and (j <= b)){
      return st[node];
    }
    int m = (i + j) / 2;
    int l = (node << 1);
    int r = l + 1;
    return join(query(l, i, m, a, b), query(r, m + 1, j, a, b));
  }
  void update(int node, int i, int j, int a, int b, int value){
    upLazy(node, i, j);
    if ((i > j) or (i > b) or (j < a))
      return;
    if ((a <= i) and (j <= b)){
      lazy[node] = value;
      upLazy(node, i, j);
    }else{
      int m = (i + j) / 2;
      int l = (node << 1);
```

```
      int r = l + 1;
      update(l, i, m, a, b, value);
      update(r, m + 1, j, a, b, value);
      st[node] = join(st[l], st[r]);
    }
  }
public:
  template <class MyIterator>
  SegTreeLazy(MyIterator begin, MyIterator end){
    n = end - begin;
    v = vector<int>(begin, end);
    st.resize(4 * n + 5);
    lazy.assign(4 * n + 5, 0);
    build(1, 0, n - 1);
  }
  //0-indexed [a, b]
  Node query(int a, int b){
    return query(1, 0, n - 1, a, b);
  }
  //0-indexed [a, b]
  void update(int a, int b, int value){
    update(1, 0, n - 1, a, b, value);
  }
};
```

## 1.16 Segment Tree Persistent

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 3e4 + 10, UPD = 2e5 + 10, LOG = 20;
const int MAXS = 4 * MAX + UPD * LOG;
namespace PerSegTree{
  typedef long long pst_t;
  pst_t seg[MAXS];
  int T[UPD], L[MAXS], R[MAXS], cnt, t;
  int n, *v;
  pst_t neutral = 0;
  pst_t join(pst_t a, pst_t b){
    return a + b;
  }
  pst_t build(int p, int l, int r){
    if (l == r)
      return seg[p] = v[l];
    L[p] = cnt++, R[p] = cnt++;
    int m = (l + r) / 2;
    return seg[p] = join(build(L[p], l, m), build(R[p], m + 1, r));
  }
  pst_t query(int a, int b, int p, int l, int r){
    if (b < l or r < a)
      return 0;
    if (a <= l and r <= b)
      return seg[p];
    int m = (l + r) / 2;
    return join(query(a, b, L[p], l, m), query(a, b, R[p], m + 1, r));
  }
  pst_t update(int a, int x, int lp, int p, int l, int r){
    if (l == r)
      return seg[p] = x;
    int m = (l + r) / 2;
    if (a <= m)
      return seg[p] = join(update(a, x, L[lp], L[p] = cnt++, l, m),
          seg[R[p] = R[lp]]);
    return seg[p] = join(seg[L[p] = L[lp]], update(a, x, R[lp], R[p] =
        cnt++, m + 1, r));
  }
//Public:
  //O(n)
  void build(int n2, int *v2){
    n = n2, v = v2;
    T[0] = cnt++;
    build(0, 0, n - 1);
  }
  //O(log(n))
  pst_t query(int a, int b, int tt){
    return query(a, b, T[tt], 0, n - 1);
  }
  //O(log(n))
  //update: v[idx] = x;
  int update(int idx, int x, int tt = t){
    update(idx, x, T[tt], T[++t] = cnt++, 0, n - 1);
    return t;
  }
}; // namespace perseg
```

## 1.17 Sparse Table

```
#include <bits/stdc++.h>
using namespace std;
class SparseTable{
private:
  typedef int t_st;
  vector<vector<t_st>> st;
  vector<int> log2;
  t_st neutral = 0x3f3f3f3f;
  int nLog;
  t_st join(t_st a, t_st b){
    return min(a, b);
  }
public:
  template <class MyIterator>
  SparseTable(MyIterator begin, MyIterator end){
    int n = end - begin;
    nLog = 20;
    log2.resize(n + 1);
    log2[1] = 0;
    for (int i = 2; i <= n; i++)
      log2[i] = log2[i / 2] + 1;
    st.resize(n, vector<t_st>(nLog, neutral));
    for (int i = 0; i < n; i++, begin++)
      st[i][0] = (*begin);
    for (int j = 1; j < nLog; j++)
      for (int i = 0; (i + (1 << (j - 1))) < n; i++)
        st[i][j] = join(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
  }
  //0-indexed [a, b]
  t_st query(int a, int b){
    int d = b - a + 1;
    t_st ans = neutral;
    for (int j = nLog - 1; j >= 0; j--){
      if (d & (1 << j)){
```

```
        ans = join(ans, st[a][j]);
        a = a + (1 << (j));
      }
    }
    return ans;
  }
  //0-indexed [a, b]
  t_st queryRMQ(int a, int b){
    int j = log2[b - a + 1];
    return join(st[a][j], st[b - (1 << j) + 1][j]);
  }
};
```

## 1.18 SQRT Decomposition

```
#include <bits/stdc++.h>
using namespace std;
struct SqrtDecomposition{
  typedef long long t_sqrt;
  int sqrtLen;
  vector<t_sqrt> block;
  vector<t_sqrt> v;
  template <class MyIterator>
  SqrtDecomposition(MyIterator begin, MyIterator end){
    int n = end - begin;
    sqrtLen = (int)sqrt(n + .0) + 1;
    v.resize(n);
    block.resize(sqrtLen + 5);
    for (int i = 0; i < n; i++, begin++){
      v[i] = (*begin);
      block[i / sqrtLen] += v[i];
    }
  }
  //0-indexed
  void update(int idx, t_sqrt new_value){
    t_sqrt d = new_value - v[idx];
    v[idx] += d;
    block[idx / sqrtLen] += d;
  }
  //0-indexed [l, r]
  t_sqrt query(int l, int r){
    t_sqrt sum = 0;
    int c_l = l / sqrtLen, c_r = r / sqrtLen;
    if (c_l == c_r){
      for (int i = l; i <= r; i++)
        sum += v[i];
    }else{
      for (int i = l, end = (c_l + 1) * sqrtLen - 1; i <= end; i++)
        sum += v[i];
      for (int i = c_l + 1; i <= c_r - 1; i++)
        sum += block[i];
      for (int i = c_r * sqrtLen; i <= r; i++)
        sum += v[i];
    }
    return sum;
  }
};
```

## 1.19 SQRT Tree

```
#include <bits/stdc++.h>
using namespace std;
class SqrtTree{
private:
  typedef long long t_sqrt;
  t_sqrt op(const t_sqrt &a, const t_sqrt &b){
    return a | b;
  }
  inline int log2Up(int n){
    int res = 0;
    while ((1 << res) < n)
      res++;
    return res;
  }
  int n, lg, indexSz;
  vector<t_sqrt> v;
  vector<int> clz, layers, onLayer;
  vector<vector<t_sqrt>> pref, suf, between;
  inline void buildBlock(int layer, int l, int r){
    pref[layer][l] = v[l];
    for (int i = l + 1; i < r; i++)
      pref[layer][i] = op(pref[layer][i - 1], v[i]);
    suf[layer][r - 1] = v[r - 1];
    for (int i = r - 2; i >= l; i--)
      suf[layer][i] = op(v[i], suf[layer][i + 1]);
  }
  inline void buildBetween(int layer, int lBound, int rBound, int
      betweenOffs){
    int bSzLog = (layers[layer] + 1) >> 1;
    int bCntLog = layers[layer] >> 1;
    int bSz = 1 << bSzLog;
    int bCnt = (rBound - lBound + bSz - 1) >> bSzLog;
    for (int i = 0; i < bCnt; i++){
      t_sqrt ans;
      for (int j = i; j < bCnt; j++){
        t_sqrt add = suf[layer][lBound + (j << bSzLog)];
        ans = (i == j) ? add : op(ans, add);
        between[layer - 1][betweenOffs + lBound + (i << bCntLog) + j]
            = ans;
      }
    }
  }
  inline void buildBetweenZero(){
    int bSzLog = (lg + 1) >> 1;
    for (int i = 0; i < indexSz; i++){
      v[n + i] = suf[0][i << bSzLog];
    }
    build(1, n, n + indexSz, (1 << lg) - n);
  }
  inline void updateBetweenZero(int bid){
    int bSzLog = (lg + 1) >> 1;
    v[n + bid] = suf[0][bid << bSzLog];
    update(1, n, n + indexSz, (1 << lg) - n, n + bid);
  }
  void build(int layer, int lBound, int rBound, int betweenOffs){
    if (layer >= (int)layers.size())
      return;
    int bSz = 1 << ((layers[layer] + 1) >> 1);
```

```
    for (int l = lBound; l < rBound; l += bSz){
      int r = min(l + bSz, rBound);
      buildBlock(layer, l, r);
      build(layer + 1, l, r, betweenOffs);
    }
    if (layer == 0)
      buildBetweenZero();
    else
      buildBetween(layer, lBound, rBound, betweenOffs);
  }
  void update(int layer, int lBound, int rBound, int betweenOffs, int
      x){
    if (layer >= (int)layers.size())
      return;
    int bSzLog = (layers[layer] + 1) >> 1;
    int bSz = 1 << bSzLog;
    int blockIdx = (x - lBound) >> bSzLog;
    int l = lBound + (blockIdx << bSzLog);
    int r = min(l + bSz, rBound);
    buildBlock(layer, l, r);
    if (layer == 0)
      updateBetweenZero(blockIdx);
    else
      buildBetween(layer, lBound, rBound, betweenOffs);
    update(layer + 1, l, r, betweenOffs, x);
  }
  inline t_sqrt query(int l, int r, int betweenOffs, int base){
    if (l == r)
      return v[l];
    if (l + 1 == r)
      return op(v[l], v[r]);
    int layer = onLayer[clz[(l - base) ^ (r - base)]];
    int bSzLog = (layers[layer] + 1) >> 1;
    int bCntLog = layers[layer] >> 1;
    int lBound = (((l - base) >> layers[layer]) << layers[layer]) +
        base;
    int lBlock = ((l - lBound) >> bSzLog) + 1;
    int rBlock = ((r - lBound) >> bSzLog) - 1;
    t_sqrt ans = suf[layer][l];
    if (lBlock <= rBlock){
      t_sqrt add;
      if (layer == 0)
        add = query(n + lBlock, n + rBlock, (1 << lg) - n, n);
      else
        add = between[layer - 1][betweenOffs + lBound + (lBlock <<
            bCntLog) + rBlock];
      ans = op(ans, add);
    }
    ans = op(ans, pref[layer][r]);
    return ans;
  }
public:
  template <class MyIterator>
  SqrtTree(MyIterator begin, MyIterator end){
    n = end - begin;
    v.resize(n);
    for (int i = 0; i < n; i++, begin++)
      v[i] = (*begin);
    lg = log2Up(n);
    clz.resize(1 << lg);
    onLayer.resize(lg + 1);
    clz[0] = 0;
    for (int i = 1; i < (int)clz.size(); i++)
      clz[i] = clz[i >> 1] + 1;
    int tlg = lg;
    while (tlg > 1){
      onLayer[tlg] = (int)layers.size();
      layers.push_back(tlg);
      tlg = (tlg + 1) >> 1;
    }
    for (int i = lg - 1; i >= 0; i--)
      onLayer[i] = max(onLayer[i], onLayer[i + 1]);
    int betweenLayers = max(0, (int)layers.size() - 1);
    int bSzLog = (lg + 1) >> 1;
    int bSz = 1 << bSzLog;
    indexSz = (n + bSz - 1) >> bSzLog;
    v.resize(n + indexSz);
    pref.assign(layers.size(), vector<t_sqrt>(n + indexSz));
    suf.assign(layers.size(), vector<t_sqrt>(n + indexSz));
    between.assign(betweenLayers, vector<t_sqrt>((1 << lg) + bSz));
    build(0, 0, n, 0);
  }
  //0-indexed
  inline void update(int x, const t_sqrt &item){
    v[x] = item;
    update(0, 0, n, 0, x);
  }
  //0-indexed [l, r]
  inline t_sqrt query(int l, int r){
    return query(l, r, 0, 0);
  }
};
```

## 1.20 Stack Query

```
#include <bits/stdc++.h>
using namespace std;
struct StackQuery{
  typedef int t_stack;
  stack<pair<t_stack, t_stack>> st;
  t_stack cmp(t_stack a, t_stack b){
    return min(a, b);
  }
  void push(t_stack x){
    t_stack new_value = st.empty() ? x : cmp(x, st.top().second);
    st.push({x, new_value});
  }
  void pop(){
    st.pop();
  }
  t_stack top(){
    return st.top().first;
  }
  t_stack query(){
    return st.top().second;
  }
  t_stack size(){
    return st.size();
  }
};
```

## 1.21 Treap

```
#include <bits/stdc++.h>
using namespace std;
namespace Treap{
  const int N = 500010;
  typedef long long treap_t;
  treap_t X[N];
  int en = 1, Y[N], sz[N], L[N], R[N], root;

  const treap_t neutral = 0;
  treap_t op_val[N];
  inline treap_t join(treap_t a, treap_t b, treap_t c){
    return a + b + c;
  }
  void calc(int u) { // update node given children info
    sz[u] = sz[L[u]] + 1 + sz[R[u]];
    // code here, no recursion
    op_val[u] = join(op_val[L[u]], X[u], op_val[R[u]]);
  }
  void unlaze(int u) {
    if(!u) return;
    // code here, no recursion
  }
  void split(int u, treap_t x, int &l, int &r) { // l gets <= x, r
      gets > x
    unlaze(u);
    if(!u) return (void) (l = r = 0);
    if(X[u] <= x) { split(R[u], x, l, r); R[u] = l; l = u; }
    else { split(L[u], x, l, r); L[u] = r; r = u; }
    calc(u);
  }
  void split_sz(int u, int s, int &l, int &r) { // l gets first s, r
      gets remaining
    unlaze(u);
    if(!u) return (void) (l = r = 0);
    if(sz[L[u]] < s) { split_sz(R[u], s - sz[L[u]] - 1, l, r); R[u] =
        l; l = u; }
    else { split_sz(L[u], s, l, r); L[u] = r; r = u; }
    calc(u);
  }
  int merge(int l, int r) { // els on l <= els on r
    unlaze(l); unlaze(r);
    if(!l || !r) return l + r;
    int u;
    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
    else { L[r] = merge(l, L[r]); u = r; }
    calc(u);
    return u;
  }
  int new_node(treap_t x){
    X[en] = x;
    op_val[en] = x;
    return en++;
  }
  int nth(int u, int idx){
    if(!u)
      return 0;
    unlaze(u);
    if(idx <= sz[L[u]])
      return nth(L[u], idx);
    else if(idx == sz[L[u]] + 1)
      return u;
    else
      return nth(R[u], idx - sz[L[u]] - 1);
  }
//Public
  void init(int n=N-1) { // call before using other funcs
    //init position 0
    sz[0] = 0;
    op_val[0] = neutral;
    //init Treap
    root = 0;
    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i] = 1; L[i] = R[i
        ] = 0; }
    random_shuffle(Y + 1, Y + n + 1);
  }
  void insert(treap_t x){
    int a, b;
    split(root, x, a, b);
    root = merge(merge(a, new_node(x)), b);
  }
  void erase(treap_t x){
    int a, b, c, d;
    split(root, x-1, a, b);
    split(b, x, c, d);
    split_sz(c, 1, b, c);
    root = merge(a, merge(c, d));
  }
  int count(treap_t x){
    int a, b, c, d;
    split(root, x-1, a, b);
    split(b, x, c, d);
    int ans = sz[c];
    root = merge(a, merge(c, d));
    return ans;
  }
  int size(){ return sz[root];}
  //0-indexed
  treap_t nth(int idx){
    int u = nth(root, idx + 1);
    return X[u];
  }
  //Query in k smallest elements
  treap_t query(int k){
    int a, b;
    split_sz(root, k, a, b);
    treap_t ans = op_val[a];
    root = merge(a, b);
    return ans;
  }
};
```

## 1.22 Union Find

```
#include <bits/stdc++.h>
using namespace std;
class UnionFind{
private:
  vector<int> p, w, sz;
```

```
public:
  UnionFind(int n){
    w.resize(n + 1, 1);
    sz.resize(n + 1, 1);
    p.resize(n + 1);
    for (int i = 0; i <= n; i++)
      p[i] = i;
  }
  int find(int x){
    if (p[x] == x)
      return x;
    return p[x] = find(p[x]);
  }
  bool join(int x, int y){
    x = find(x);
    y = find(y);
    if (x == y)
      return false;
    if (w[x] > w[y])
      swap(x, y);
    p[x] = y;
    sz[y] += sz[x];
    if (w[x] == w[y])
      w[y]++;
    return true;
  }
  bool isSame(int x, int y){
    return find(x) == find(y);
  }
  int size(int x){
    return sz[find(x)];
  }
};
```

## 1.23 Wavelet Tree

```
#include <bits/stdc++.h>
using namespace std;
namespace WaveletTree{
  const int MAXN = 100010, MAXW = MAXN*30; // MAXN * LOG(maxX-MinX)
  typedef int t_wavelet;
  int last;
  int v[MAXN], aux[MAXN];
  int lo[MAXW], hi[MAXW], l[MAXW], r[MAXW];
  vector<t_wavelet> a[MAXW];
  int stable_partition(int i, int j, t_wavelet mid){
    int pivot=0;
    for(int k=i; k<j; k++)
      aux[k] = v[k], pivot += (v[k]<=mid);
    int i1=i, i2=i+pivot;
    for(int k=i; k<j; k++){
      if(aux[k]<=mid) v[i1++] = aux[k];
      else v[i2++] = aux[k];
    }
    return i1;
  }
  void build(int u, int i, int j, t_wavelet minX, t_wavelet maxX){
    lo[u] = minX, hi[u] = maxX;
    if (lo[u] == hi[u] or i >= j)
      return;
    t_wavelet mid = (minX + maxX - 1)/2;
    a[u].resize(j - i + 1);
    a[u][0] = 0;
    for(int k=i; k<j; k++)
      a[u][k-i+1] = a[u][k-i] + (v[k] <= mid);
    int pivot = stable_partition(i, j, mid);
    l[u] = last++, r[u] = last++;
    build(l[u], i, pivot, minX, mid);
    build(r[u], pivot, j, mid + 1, maxX);
  }
  inline int b(int u, int i){
    return i - a[u][i];
  }
//Public
  template <class MyIterator>
  void init(MyIterator begin, MyIterator end, t_wavelet minX,
      t_wavelet maxX){
    last = 1;
    int n = end-begin;
    for(int i=0; i<n; i++, begin++)
      v[i] = *begin;
    build(last++, 0, n, minX, maxX);
  }
  //kth smallest element in range [i, j]
  //1-indexed
  int kth(int i, int j, int k, int u=1){
    if (i > j)
      return 0;
    if (lo[u] == hi[u])
      return lo[u];
    int inLeft = a[u][j] - a[u][i - 1];
    int i1 = a[u][i - 1] + 1, j1 = a[u][j];
    int i2 = b(u, i - 1) + 1, j2 = b(u, j);
    if (k <= inLeft)
      return kth(i1, j1, k, l[u]);
    return kth(i2, j2, k - inLeft, r[u]);
  }
  //Amount of numbers in the range [i, j] Less than or equal to k
  //1-indexed
  int lte(int i, int j, int k, int u=1){
    if (i > j or k < lo[u])
      return 0;
    if (hi[u] <= k)
      return j - i + 1;
    int i1 = a[u][i - 1] + 1, j1 = a[u][j];
    int i2 = b(u, i - 1) + 1, j2 = b(u, j);
    return lte(i1, j1, k, l[u]) + lte(i2, j2, k, r[u]);
  }
  //Amount of numbers in the range [i, j] equal to k
  //1-indexed
  int count(int i, int j, int k, int u=1){
    if (i > j or k < lo[u] or k > hi[u])
      return 0;
    if (lo[u] == hi[u])
      return j - i + 1;
    t_wavelet mid = (lo[u] + hi[u] - 1) / 2;
    int i1 = a[u][i - 1] + 1, j1 = a[u][j];
    int i2 = b(u, i - 1) + 1, j2 = b(u, j);
    if (k <= mid)
      return count(i1, j1, k, l[u]);
    return count(i2, j2, k, r[u]);
```

```
  }
  //swap v[i] with v[i+1]
  //1-indexed
  void swp(int i, int u=1){
    if (lo[u] == hi[u] or a[u].size() <= 2)
      return;
    if (a[u][i - 1] + 1 == a[u][i] and a[u][i] + 1 == a[u][i + 1])
      swp(a[u][i], l[u]);
    else if (b(u, i - 1) + 1 == b(u, i) and b(u, i) + 1 == b(u, i + 1)
        )
      swp(b(u, i), r[u]);
    else if (a[u][i - 1] + 1 == a[u][i])
      a[u][i]--;
    else
      a[u][i]++;
  }
};
```

# 2 Graph Algorithms

## 2.1 2-SAT

```
#include "strongly_connected_component.h"
using namespace std;
struct SAT{
  typedef pair<int, int> pii;
  vector<pii> edges;
  int n;
  SAT(int size){
    n = 2 * size;
  }
  vector<bool> solve2SAT(){
    vector<bool> vAns(n / 2, false);
    vector<int> comp = SCC::scc(n, edges);
    for (int i = 0; i < n; i += 2){
      if (comp[i] == comp[i + 1])
        return vector<bool>();
      vAns[i / 2] = (comp[i] > comp[i + 1]);
    }
    return vAns;
  }
  int v(int x){
    if (x >= 0)
      return (x << 1);
    x = ~x;
    return (x << 1) ^ 1;
  }
  void add(int a, int b){
    edges.push_back(pii(a, b));
  }
  void addOr(int a, int b){
    add(v(~a), v(b));
    add(v(~b), v(a));
  }
  void addImp(int a, int b){
    addOr(~a, b);
  }
  void addEqual(int a, int b){
    addOr(a, ~b);
    addOr(~a, b);
  }
  void addDiff(int a, int b){
    addEqual(a, ~b);
  }
};
```

## 2.2 Arborescence

```
#include <bits/stdc++.h>
#include "../data_structures/union_find.h"

using namespace std;
/**
 * Source: https://github.com/spaghetti-source/algorithm/blob/master/
     graph/arborescence.cc
 * Description: Edmonds' algorithm for finding the weight of the
     minimum spanning
 * tree/arborescence of a directed graph, given a root node. If no MST
      exists, returns -1.
 * Time: O(E \log V)
 */
typedef long long ll;
namespace Arborescence{
  struct Edge { int a, b; ll w; };
  struct Heap {
    Edge key;
    Heap *l, *r;
    ll delta;
    void prop() {
      key.w += delta;
      if (l) l->delta += delta;
      if (r) r->delta += delta;
      delta = 0;
    }
    Edge top() { prop(); return key; }
  };
  Heap *merge(Heap *a, Heap *b) {
    if (!a || !b) return a ?: b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
  }
  void pop(Heap*& a) { a->prop(); a = merge(a->l, a->r); }

//public
  vector<Edge> g;
  void init(){
    g.clear();
  }
  void addEdge(int u, int to, ll w) {
    Edge e;
    e.a = u, e.b = to, e.w = w;
    g.push_back(e);
  }
  ll dmst(int n, int root) {
    UnionFind uf(n);
    vector<Heap*> heap(n);
```

```
    vector<Heap*> vp;
    for (Edge e : g){
      Heap* h = new Heap{e};
      vp.push_back(h);
      heap[e.b] = merge(heap[e.b], h);
    }
    ll res = 0;
    vector<int> seen(n, -1), path(n);
    seen[root] = root;
    for(int s=0; s<n; s++) {
      int u = s, qi = 0, w;
      while (seen[u] < 0) {
        path[qi++] = u, seen[u] = s;
        if (!heap[u]){
          for(Heap *h: vp)
            delete h;
          return -1;
        }
        Edge e = heap[u]->top();
        heap[u]->delta -= e.w, pop(heap[u]);
        res += e.w; u = uf.find(e.a);
        if (seen[u] == s) {
          Heap* cyc = 0;
          do cyc = merge(cyc, heap[w = path[--qi]]);
          while (uf.join(u, w));
          u = uf.find(u);
          heap[u] = cyc, seen[u] = -1;
        }
      }
    }
    for(Heap *h: vp)
      delete h;
    return res;
  }
  //Careful with overflow
  ll dmstAnyRoot(int n) {
    ll maxEdge = 1000000010;
    ll INF = n*maxEdge;
    for(int i=0; i<n; i++)
      addEdge(n, i, INF);
    ll ans = dmst(n+1, n);
    if(ans >= 0 and ans < 2*INF)
      return ans - INF;
    else
      return -1;
  }
};
```

## 2.3 Articulation Point

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

using namespace std;

const int MAXN = 500010;
typedef pair<int, int> pii;
//Articulation Point
namespace AP{
  vector<int> adj[MAXN];
  vector<bool> visited, isAP;
  vector<int> tin, low;
  int timer, n;
  vector<pii> bridges;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++) adj[i].clear();
  }
  void addEdge(int a, int b){
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  void dfs(int u, int p = -1) {
    visited[u] = true;
    tin[u] = low[u] = timer++;
    int children=0;
    for (int to : adj[u]) {
      if (to == p) continue;
      if (visited[to]) {
        low[u] = min(low[u], tin[to]);
      } else {
        dfs(to, u);
        low[u] = min(low[u], low[to]);
        if (low[to] >= tin[u] && p!=-1)
          isAP[u] = true;
        ++children;
      }
    }
    if(p == -1 && children > 1)
      isAP[u] = true;
  }
  vector<bool> findArticulationPoint() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    isAP.assign(n, false);
    for (int i = 0; i < n; i++) {
      if (!visited[i])
        dfs(i);
    }
    return isAP;
  }
};
```

## 2.4 BFS 0-1

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int N = 500010;
const int INF = 0x3f3f3f3f;
namespace BFS01{
  vector<pii> adj[N];
  int n;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++) adj[i].clear();
  }
  //0-indexed
  void addEdge(int u, int to, int w){
```

```
    adj[u].emplace_back(to, w);
  }
  vector<int> solve(int s){
    vector<int> d(n, INF);
    d[s] = 0;
    deque<int> q;
    q.push_front(s);
    while (!q.empty()) {
      int u = q.front();
      q.pop_front();
      for (auto edge : adj[u]) {
        int to = edge.first;
        int w = edge.second;
        if (d[u] + w < d[to]) {
          d[to] = d[u] + w;
          if (w == 1)
            q.push_back(to);
          else
            q.push_front(to);
        }
      }
    }
    return d;
  }
};
```

## 2.5 Bridge

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 500010;
typedef pair<int, int> pii;
namespace Bridge{
  vector<int> adj[MAXN];
  vector<bool> visited;
  vector<int> tin, low;
  int timer, n;
  vector<pii> bridges;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++) adj[i].clear();
  }
  void addEdge(int a, int b){
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  void dfs(int u, int p = -1) {
    visited[u] = true;
    tin[u] = low[u] = timer++;
    for (int to : adj[u]) {
      if (to == p) continue;
      if (visited[to]) {
        low[u] = min(low[u], tin[to]);
      } else {
        dfs(to, u);
        low[u] = min(low[u], low[to]);
        if (low[to] > tin[u])
          bridges.push_back({u, to});
      }
    }
  }
  vector<pii> findBridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    bridges.clear();
    for (int i = 0; i < n; i++) {
      if (!visited[i])
        dfs(i);
    }
    return bridges;
  }
};
```

## 2.6 Centroid

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 500010;
typedef pair<int, int> pii;
namespace Centroid{
  vector<int> adj[MAXN];
  int sub[MAXN];
  int n;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++) adj[i].clear();
  }
  void addEdge(int a, int b){
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  int dfsS(int u, int p){
    sub[u] = 1;
    for(int to: adj[u]){
      if(to != p)
        sub[u] += dfsS(to, u);
    }
    return sub[u];
  }
  pii dfsC(int u, int p){
    for(int to : adj[u]){
      if(to != p and sub[to] > n/2)
        return dfsC(to, u);
    }
    for(int to : adj[u]){
      if(to != p and (sub[to]*2) == n)
        return pii(u, to);
    }
    return pii(u, u);
  }
  pii findCentroid(){
    dfsS(0, -1);
    return dfsC(0, -1);
  }
}
```

## 2.7 Centroid Decomposition

```
#include <bits/stdc++.h>
using namespace std;
// O(N*log(N))
struct CentroidDecomposition{
  vector<vector<int>> adj;
  vector<int> dad, sub;
  vector<bool> rem;
  int centroidRoot, n;
  void init(int _n){
    n = _n;
    adj.resize(n);
    dad.resize(n);
    sub.resize(n);
    rem.assign(n, false);
  }
  // Return Centroid Decomposition Tree
  vector<vector<int>> build(){
    assert(n > 0);
    centroidRoot = decomp(0, -1);
    vector<vector<int>> ret(n);
    for (int u = 0; u < n; u++){
      if (dad[u] != u)
        ret[dad[u]].push_back(u);
    }
    return ret;
  }
  void addEdge(int a, int b){
    adj[a].push_back(b);
    adj[b].push_back(a);
  }
  int decomp(int u, int p){
    int sz = dfs(u, p);
    int c = centroid(u, p, sz);
    if (p == -1)
      p = c;
    dad[c] = p;
    rem[c] = true;
    for (auto to : adj[c]){
      if (!rem[to])
        decomp(to, c);
    }
    return c;
  }
  int dfs(int u, int p){
    sub[u] = 1;
    for (int to : adj[u]){
      if (!rem[to] and to != p)
        sub[u] += dfs(to, u);
    }
    return sub[u];
  }
  int centroid(int u, int p, int sz){
    for (auto to : adj[u])
      if (!rem[to] and to != p and sub[to] > sz / 2)
        return centroid(to, u, sz);
    return u;
  }
  int operator[](int i){
    return dad[i];
  }
};
```

## 2.8 Checking Bipartiteness Online

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
const int N = 500010;
pii parent[N];
int rk[N];
int bipartite[N];
void make_set(int v) {
  parent[v] = pii(v, 0);
  rk[v] = 0;
  bipartite[v] = true;
}
pii find_set(int v) {
  if (v != parent[v].first) {
    int parity = parent[v].second;
    parent[v] = find_set(parent[v].first);
    parent[v].second ^= parity;
  }
  return parent[v];
}
void add_edge(int a, int b) {
  pii pa = find_set(a);
  a = pa.first;
  int x = pa.second;
  pair<int, int> pb = find_set(b);
  b = pb.first;
  int y = pb.second;
  if (a == b) {
    if (x == y)
      bipartite[a] = false;
  }else{
    if (rk[a] < rk[b])
      swap (a, b);
    parent[b] = pii(a, x^y^1);
    bipartite[a] &= bipartite[b];
    if (rk[a] == rk[b])
      ++rk[a];
  }
}
bool is_bipartite(int v) {
  return bipartite[find_set(v).first];
}
```

## 2.9 Dinic

```
#include <bits/stdc++.h>
using namespace std;
//O((V^2)*E): for generic graph.
//O(sqrt(V)*E): on unit networks. A unit network is a network in which
     all the edges have unit capacity, and for any vertex except s and
     t either incoming or outgoing edge is unique. That's exactly the
```

```
    case with the network we build to solve the maximum matching
    problem with flows.
template <typename flow_t>
struct Dinic{
  struct FlowEdge{
    int from, to;
    flow_t cap, flow = 0;
    FlowEdge(int f, int t, flow_t c) : from(f), to(t), cap(c) {}
  };
  const flow_t flow_inf = numeric_limits<flow_t>::max();
  vector<FlowEdge> edges;
  vector<vector<int>> adj;
  int n, m = 0;
  int s, t;
  vector<int> level, ptr;
  queue<int> q;
  bool bfs(){
    while (!q.empty()){
      int u = q.front();
      q.pop();
      for (int id : adj[u]){
        if (edges[id].cap - edges[id].flow < 1)
          continue;
        if (level[edges[id].to] != -1)
          continue;
        level[edges[id].to] = level[u] + 1;
        q.push(edges[id].to);
      }
    }
    return level[t] != -1;
  }
  flow_t dfs(int u, flow_t pushed){
    if (pushed == 0)
      return 0;
    if (u == t)
      return pushed;
    for (int &cid = ptr[u]; cid < (int)adj[u].size(); cid++){
      int id = adj[u][cid];
      int to = edges[id].to;
      if (level[u] + 1 != level[to] || edges[id].cap - edges[id].flow
          < 1)
        continue;
      flow_t tr = dfs(to, min(pushed, edges[id].cap - edges[id].flow))
          ;
      if (tr == 0)
        continue;
      edges[id].flow += tr;
      edges[id ^ 1].flow -= tr;
      return tr;
    }
    return 0;
  }
//Public:
  Dinic(){}
  void init(int _n){
    n = _n;
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
  }
  void addEdge(int from, int to, flow_t cap){
    assert(n>0);
    edges.push_back(FlowEdge(from, to, cap));
    edges.push_back(FlowEdge(to, from, 0));
    adj[from].push_back(m);
    adj[to].push_back(m + 1);
    m += 2;
  }
  flow_t maxFlow(int s1, int t1){
    s = s1, t = t1;
    flow_t f = 0;
    //Reset Flow
    for(int i=0; i<m; i++)
      edges[i].flow = 0;
    while (true){
      level.assign(n, -1);
      level[s] = 0;
      q.push(s);
      if (!bfs())
        break;
      ptr.assign(n, 0);
      while (flow_t pushed = dfs(s, flow_inf))
        f += pushed;
    }
    return f;
  }
};
typedef pair<int, int> pii;
vector<pii> recoverCut(Dinic<int> &d){
  vector<int> level(d.n, 0);
  vector<pii> rc;
  queue<int> q;
  q.push(d.s);
  level[d.s] = 1;
  while (!q.empty()){
    int u = q.front();
    q.pop();
    for (int id : d.adj[u]){
      if ((id & 1) == 1)
        continue;
      if (d.edges[id].cap == d.edges[id].flow){
        rc.push_back(pii(d.edges[id].from, d.edges[id].to));
      }else{
        if (level[d.edges[id].to] == 0){
          q.push(d.edges[id].to);
          level[d.edges[id].to] = 1;
        }
      }
    }
  }
  vector<pii> ans;
  for (pii p : rc)
    if ((level[p.first] == 0) or (level[p.second] == 0))
      ans.push_back(p);
  return ans;
}
```

## 2.10 Flow With Demand

```
#include "dinic.h"
using namespace std;
```

```cpp
template <typename flow_t>
struct MaxFlowEdgeDemands{
  Dinic<flow_t> mf;
  vector<flow_t> ind, outd;
  flow_t D;
  int n;
  MaxFlowEdgeDemands(int n) : n(n){
    D = 0;
    mf.init(n + 2);
    ind.assign(n, 0);
    outd.assign(n, 0);
  }
  void addEdge(int a, int b, flow_t cap, flow_t demands){
    mf.addEdge(a, b, cap - demands);
    D += demands;
    ind[b] += demands;
    outd[a] += demands;
  }
  bool solve(int s, int t){
    mf.addEdge(t, s, numeric_limits<flow_t>::max());
    for (int i = 0; i < n; i++){
      if (ind[i]) mf.addEdge(n, i, ind[i]);
      if (outd[i]) mf.addEdge(i, n + 1, outd[i]);
    }
    return mf.maxFlow(n, n + 1) == D;
  }
};
```

## 2.11 Floyd Warshall

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INFLL = 0x3f3f3f3f3f3f3f3f;
namespace FloydWarshall{
  vector<vector<ll>> dist;
  int n;
  void init(int n1){
    n = n1;
    dist.assign(n, vector<ll>(n, INFLL));
    for(int i=0; i<n; i++)
      dist[i][i] = 0LL;
  }
  void addEdge(int a, int b, ll w){
    dist[a][b] = min(dist[a][b], w);
  }
  vector<vector<ll>> solve(){
    for(int k=0; k<n; k++){
      for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
          dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
      }
    }
    return dist;
  }
};
```

## 2.12 Prim

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int MAXN = 500010;
namespace Prim{
  vector<pii> adj[MAXN];
  int weight[MAXN];
  bool seen[MAXN];
  int n;
  void init(int n1){
    n = n1;
    for(int i=0; i<n; i++) adj[i].clear();
  }
  void addEdge(int a, int b, int w){
    adj[a].emplace_back(w, b);
    adj[b].emplace_back(w, a);
  }
  ll solve(){
    for(int i=0; i<n; i++){
      weight[i] = 0x3f3f3f3f;
      seen[i] = 0;
    }
    weight[0] = 0;
    priority_queue<pii, vector<pii>, greater<pii> > st;
    st.push(pii(weight[0], 0));
    ll ans = 0;
    while(!st.empty()){
      int u = st.top().second;
      st.pop();
      if(seen[u])
        continue;
      seen[u] = true;
      ans += weight[u];
      for(auto [edge, to]: adj[u]){
        if(!seen[to] and (edge < weight[to])){
          weight[to] = edge;
          st.emplace(weight[to], to);
        }
      }
    }
    return ans;
  }
};
```

## 2.13 HLD

```cpp
#include <bits/stdc++.h>
#include "../data_structures/bit_range.h"
using namespace std;
#define F first
#define S second
using hld_t = long long;
using pii = pair<int, hld_t>;
struct HLD{
  vector<vector<pii>> adj;
```

```
vector<int> sz, h, dad, pos;
vector<hld_t> val, v;
int t;
bool edge;
//Begin Internal Data Structure
BitRange *bit;
hld_t neutral = 0;
inline hld_t join(hld_t a, hld_t b){
  return a+b;
}
inline void update(int a, int b, hld_t x){
  bit->add(a+1, b+1, x);
}
inline hld_t query(int a, int b){
  return bit->get(a+1, b+1);
}
//End Internal Data Structure
void init(int n){
  dad.resize(n); pos.resize(n); val.resize(n); v.resize(n);
  adj.resize(n); sz.resize(n); h.resize(n);
  bit = new BitRange(n);
}
void dfs(int u, int p = -1){
  sz[u] = 1;
  for(pii &to: adj[u]) if(to.F != p){
    if(edge) val[to.F] = to.S;
    dfs(to.F, u);
    sz[u] += sz[to.F];
    if(sz[to.F] > sz[adj[u][0].F] or adj[u][0].F == p)
      swap(to, adj[u][0]);
  }
}
void build_hld(int u, int p=-1){
  dad[u] = p;
  pos[u] = t++;
  v[pos[u]] = val[u];
  for(pii to: adj[u]) if(to.F != p){
    h[to.F] = (to == adj[u][0]) ? h[u] : to.F;
    build_hld(to.F, u);
  }
}
void addEdge(int a, int b, hld_t w = 0){
  adj[a].emplace_back(b, w);
  adj[b].emplace_back(a, w);
}
void build(int root, bool is_edge){
  assert(!adj.empty());
  edge = is_edge;
  t = 0;
  h[root] = 0;
  dfs(root);
  build_hld(root);
  //Init Internal Data Structure
  for(int i=0; i<t; i++)
    update(i, i, v[i]);
}
hld_t query_path(int a, int b) {
  if (edge and a == b) return neutral;
  if (pos[a] < pos[b]) swap(a, b);
  if (h[a] == h[b]) return query(pos[b]+edge, pos[a]);
  return join(query(pos[h[a]], pos[a]), query_path(dad[h[a]], b));
}
void update_path(int a, int b, hld_t x) {
  if (edge and a == b) return;
  if (pos[a] < pos[b]) swap(a, b);
  if (h[a] == h[b]) return (void)update(pos[b]+edge, pos[a], x);
  update(pos[h[a]], pos[a], x); update_path(dad[h[a]], b, x);
}
hld_t query_subtree(int a) {
  if (edge and sz[a] == 1) return neutral;
  return query(pos[a]+edge, pos[a]+sz[a]-1);
}
void update_subtree(int a, hld_t x) {
  if (edge and sz[a] == 1) return;
  update(pos[a] + edge, pos[a]+sz[a]-1, x);
}
int lca(int a, int b) {
  if (pos[a] < pos[b]) swap(a, b);
  return h[a] == h[b] ? b : lca(dad[h[a]], b);
}
};
```

## 2.14 Kruskal

```
#include "../data_structures/union_find.h"
typedef long long ll;
struct Edge{
  int u, v; ll w;
  Edge(){}
  Edge(int u, int v, ll w):u(u), v(v), w(w){}
};
ll kruskal(vector<Edge> v, int nVet){
  ll cost = 0;
  UnionFind uf(nVet);
  sort(v.begin(), v.end(), [&](Edge a, Edge b){
    return a.w < b.w;
  });
  for(Edge &e: v){
    if(!uf.isSame(e.u, e.v)){
      cost += e.w;
      uf.join(e.u, e.v);
    }
  }
  return cost;
}
```

## 2.15 LCA

```
#include <bits/stdc++.h>
using namespace std;
namespace LCA{
  typedef long long lca_t;
  typedef pair<int, lca_t> lca_p;
  const lca_t neutral = 0;
  vector<vector<lca_p>> adj;
  vector<int> level;
  vector<vector<lca_t>> D;
  vector<vector<int>> P;
  int n, mxLogN;
```

```
void init(int _n, int _mxLogN = 20){
  n = _n;
  mxLogN = _mxLogN;
  adj.assign(n, vector<lca_p>());
  D.resize(n, vector<lca_t>(mxLogN));
  level.resize(n);
}
lca_t join(lca_t a, lca_t b){
  return a + b;
}
void addEdge(int a, int b, lca_t w = 1){
  adj[a].emplace_back(b, w);
  adj[b].emplace_back(a, w);
}
void dfs(int u){
  for (auto to : adj[u]){
    int v = to.first;
    lca_t w = to.second;
    if (v == P[u][0])
      continue;
    P[v][0] = u;
    D[v][0] = w;
    level[v] = level[u] + 1;
    dfs(v);
  }
}
void build(int root = 0){
  P.assign(n, vector<int>(mxLogN, -1));
  level[root] = 0;
  P[root][0] = root;
  D[root][0] = neutral;
  dfs(root);
  for (int j = 1; j < mxLogN; j++)
    for (int i = 0; i < n; i++){
      P[i][j] = P[P[i][j - 1]][j - 1];
      D[i][j] = join(D[P[i][j - 1]][j - 1], D[i][j - 1]);
    }
}
lca_p lca(int u, int v){
  if (level[u] > level[v])
    swap(u, v);
  int d = level[v] - level[u];
  lca_t ans = neutral;
  for (int i = 0; i < mxLogN; i++){
    if (d & (1 << i)){
      ans = join(ans, D[v][i]);
      v = P[v][i];
    }
  }
  if (u == v)
    return lca_p(u, ans);
  for (int i = mxLogN - 1; i >= 0; i--){
    while (P[u][i] != P[v][i]){
      ans = join(ans, D[v][i]);
      ans = join(ans, D[u][i]);
      u = P[u][i];
      v = P[v][i];
    }
  }
  ans = join(ans, D[v][0]);
  ans = join(ans, D[u][0]);
    return lca_p(P[u][0], ans);
  }
};
```

## 2.16 Min-Cut

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
//This algorithm finds the Global Min-Cut in O(|V|^3)
namespace MinCut{
  const int MAXN = 510;
  bool exist[MAXN], in_a[MAXN];
  ll g[MAXN][MAXN], w[MAXN];
  vector<int> v[MAXN];
  int n;
  void init(int n1){
    n = n1;
    memset(g, 0, sizeof(g));
  }
  void addEdge(int a, int b, int w1){
    if(a == b) return;
    g[a][b] += w1;
    g[b][a] += w1;
  }
  pair<ll, vector<int>> mincut() {
    ll best_cost = 0x3f3f3f3f3f3f3f3fLL;
    vector<int> best_cut;
    for (int i=0; i<n; ++i)
      v[i].assign (1, i);
    memset (exist, true, sizeof(exist));
    for(int ph=0; ph<n-1; ++ph) {
      memset (in_a, false, sizeof in_a);
      memset (w, 0, sizeof w);
      for(int it=0, prev=0; it<n-ph; ++it){
        int sel = -1;
        for(int i=0; i<n; ++i)
          if(exist[i] && !in_a[i] && (sel == -1 || w[i] > w[sel]))
            sel = i;
        if(it == n-ph-1){
          if(w[sel] < best_cost)
            best_cost = w[sel],  best_cut = v[sel];
          v[prev].insert (v[prev].end(), v[sel].begin(), v[sel].end())
              ;
          for(int i=0; i<n; ++i)
            g[prev][i] = g[i][prev] += g[sel][i];
          exist[sel] = false;
        }else{
          in_a[sel] = true;
          for(int i=0; i<n; ++i)
            w[i] += g[sel][i];
          prev = sel;
        }
      }
    }
    return {best_cost, best_cut};
  }
};
```

## 2.17 Minimum Cost Maximum Flow

```
#include <bits/stdc++.h>
using namespace std;
template <class T = int>
class MCMF{
private:
  struct Edge{
    int to;
    T cap, cost;
    Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
  };
  int n;
  vector<vector<int>> edges;
  vector<Edge> list;
  vector<int> from;
  vector<T> dist, pot;
  vector<bool> visit;
  pair<T, T> augment(int src, int sink){
    pair<T, T> flow = {list[from[sink]].cap, 0};
    for (int v = sink; v != src; v = list[from[v] ^ 1].to){
      flow.first = std::min(flow.first, list[from[v]].cap);
      flow.second += list[from[v]].cost;
    }
    for (int v = sink; v != src; v = list[from[v] ^ 1].to){
      list[from[v]].cap -= flow.first;
      list[from[v] ^ 1].cap += flow.first;
    }
    return flow;
  }
  queue<int> q;
  bool SPFA(int src, int sink){
    T INF = numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    q.push(src);
    dist[src] = 0;
    while (!q.empty()){
      int on = q.front();
      q.pop();
      visit[on] = false;
      for (auto e : edges[on]){
        auto ed = list[e];
        if (ed.cap == 0)
          continue;
        T toDist = dist[on] + ed.cost + pot[on] - pot[ed.to];
        if (toDist < dist[ed.to]){
          dist[ed.to] = toDist;
          from[ed.to] = e;
          if (!visit[ed.to]){
            visit[ed.to] = true;
            q.push(ed.to);
          }
        }
      }
    }
    return dist[sink] < INF;
  }
  void fixPot(){
    T INF = numeric_limits<T>::max();
    for (int i = 0; i < n; i++){
      if (dist[i] < INF)
        pot[i] += dist[i];
    }
  }
public:
  MCMF(int size){
    n = size;
    edges.resize(n);
    pot.assign(n, 0);
    dist.resize(n);
    visit.assign(n, false);
  }
  pair<T, T> solve(int src, int sink){
    pair<T, T> ans(0, 0);
    // Can use dijkstra to speed up depending on the graph
    if (!SPFA(src, sink))
      return ans;
    fixPot();
    // Can use dijkstra to speed up depending on the graph
    while (SPFA(src, sink)){
      auto flow = augment(src, sink);
      ans.first += flow.first;
      ans.second += flow.first * flow.second;
      fixPot();
    }
    return ans;
  }
  void addEdge(int from, int to, T cap, T cost){
    edges[from].push_back(list.size());
    list.push_back(Edge(to, cap, cost));
    edges[to].push_back(list.size());
    list.push_back(Edge(from, 0, -cost));
  }
};
```

## 2.18 Strongly Connected Component

```
#include "topological_sort.h"
using namespace std;
namespace SCC{
  typedef pair<int, int> pii;
  vector<vector<int>> revAdj;
  vector<int> component;
  void dfs(int u, int c){
    component[u] = c;
    for (int to : revAdj[u]){
      if (component[to] == -1)
        dfs(to, c);
    }
  }
  vector<int> scc(int n, vector<pii> &edges){
    revAdj.assign(n, vector<int>());
    for (pii p : edges)
      revAdj[p.second].push_back(p.first);
    vector<int> tp = TopologicalSort::order(n, edges);
    component.assign(n, -1);
    int comp = 0;
    for (int u : tp){
      if (component[u] == -1)
```

```
        dfs(u, comp++);
    }
    return component;
  }
} // namespace SCC
```

## 2.19 Topological Sort

```
#include <bits/stdc++.h>
using namespace std;
namespace TopologicalSort{
  typedef pair<int, int> pii;
  vector<vector<int>> adj;
  vector<bool> visited;
  vector<int> vAns;
  void dfs(int u){
    visited[u] = true;
    for (int to : adj[u]){
      if (!visited[to])
        dfs(to);
    }
    vAns.push_back(u);
  }
  vector<int> order(int n, vector<pii> &edges){
    adj.assign(n, vector<int>());
    for (pii p : edges)
      adj[p.first].push_back(p.second);
    visited.assign(n, false);
    vAns.clear();
    for (int i = 0; i < n; i++){
      if (!visited[i])
        dfs(i);
    }
    reverse(vAns.begin(), vAns.end());
    return vAns;
  }
}; // namespace TopologicalSort
```

## 2.20 Tree ID

```
#include "centroid.h"
#define F first
#define S second
namespace TreeID{
  int id=0;
  map<map<int, int>, int> mpId;
  vector<int> adj[MAXN];
  int treeID(int u, int p){
    map<int, int> mp;
    for(int to: adj[u]){
      if(to != p)
        mp[treeID(to, u)]++;
    }
    if(!mpId.count(mp))
      mpId[mp] = ++id;
    return mpId[mp];
  }
  //Returns a pair of values that represents a tree only. O((N+M)*log(
      M))
  //0-indexed
  pii getTreeID(vector<pii> &edges, int n){
    for(int i=0; i<n; i++)
      adj[i].clear();
    Centroid::init(n);
    for(pii e: edges){
      adj[e.F].push_back(e.S);
      adj[e.S].push_back(e.F);
      Centroid::addEdge(e.F, e.S);
    }
    pii c = Centroid::findCentroid();
    pii ans(treeID(c.F, -1), treeID(c.S, -1));
    if(ans.F > ans.S)
      swap(ans.F, ans.S);
    return ans;
  }
  bool isomorphic(vector<pii> &tree1, vector<pii> &tree2, int n){
    return getTreeID(tree1, n) == getTreeID(tree2, n);
  }
};
```

## 2.21 Vertex Cover In Tree

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 200010;
int dp[MAXN][2];
vector<int> adj[MAXN];
// vertexCover(node current, free to choose, dad)
int vertexCover(int u, bool color=true, int p=-1){
  if(dp[u][color] != -1)
    return dp[u][color];
  int case1 = 1, case2 = 0;
  for(int to: adj[u]){
    if(to == p) continue;
    case1 += vertexCover(to, true, u);
    case2 += vertexCover(to, false, u);
  }
  if(color)
    return dp[u][color] = min(case1, case2);
  else
    return dp[u][color] = case1;
}
```

# 3 Dynamic Programming

## 3.1 Divide and Conquer Optimization

Reduces the complexity from $O(n^2k)$ to $O(nk \log n)$ of PD's in the following ways (and other variants):

$$dp[n][k] = \max_{0 \le i < n} (dp[i][k-1] + C[i+1][n]),\ base\ case:\ dp[0][j], dp[i][0] \quad (1)$$

- $C[i][j]$ = the cost only depends on $i$ and $j$.

- $opt[n][k] = i$ is the optimal value that maximizes $dp[n][k]$.

It is necessary that $opt$ is increasing along each column: $opt[j][k] \leq opt[j+1][k]$.

### 3.2 Divide and Conquer Optimization Implementation

```
#include <bits/stdc++.h>
using namespace std;
int C(int i, int j);
const int MAXN = 100010;
const int MAXK = 110;
const int INF = 0x3f3f3f3f;
int dp[MAXN][MAXK];
void calculateDP(int l, int r, int k, int opt_l, int opt_r){
  if (l > r)
    return;
  int mid = (l + r) >> 1;
  int ans = -INF, opt = mid;
//  int ans = dp[mid][k-1], opt=mid; //If you accept empty subsegment
  for (int i = opt_l; i <= min(opt_r, mid - 1); i++){
    if (ans < dp[i][k - 1] + C(i + 1, mid)){
      opt = i;
      ans = dp[i][k - 1] + C(i + 1, mid);
    }
  }
  dp[mid][k] = ans;
  calculateDP(l, mid - 1, k, opt_l, opt);
  calculateDP(mid + 1, r, k, opt, opt_r);
}
int solve(int n, int k){
  for (int i = 0; i <= n; i++)
    dp[i][0] = -INF;
  for (int j = 0; j <= k; j++)
    dp[0][j] = -INF;
  dp[0][0] = 0;
  for (int j = 1; j <= k; j++)
    calculateDP(1, n, j, 0, n - 1);
  return dp[n][k];
}
```

### 3.3 Knuth Optimization

Reduces the complexity from $O(n^3)$ to $O(n^2)$ of PD's in the following ways (and other variants):

$$dp[i][j] = C[i][j] + \min_{i<k<j}(dp[i][k] + dp[k][j]),\ caso\ base:\ dp[i][i] \tag{2}$$

$$dp[i][j] = \min_{i<k<j}(dp[i][k] + C[i][k]),\ caso\ base:\ dp[i][i] \tag{3}$$

- $C[i][j]$ = the cost only depends on $i$ and $j$.
- $opt[i][j] = k$ is the optimal value that maximizes $dp[i][j]$.

The following conditions must be met:

- Foursquare inequality on $C$: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$, $a \leq b \leq c \leq d$.
- Monotonicity on $C$: $C[b][c] \leq C[a][d]$, $a \leq b \leq c \leq d$.

Or the following condition:

- $opt$ increasing in rows and columns: $opt[i][j-1] \leq opt[i][j] \leq opt[i+1][j]$.

### 3.4 Knuth Optimization Implementation

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1009;
const ll INFLL = 0x3f3f3f3f3f3f3f3f;
ll C(int a, int b);
ll dp[MAXN][MAXN];
int opt[MAXN][MAXN];
ll knuth(int n){
  for (int i = 0; i < n; i++){
    dp[i][i] = 0;
    opt[i][i] = i;
  }
  for (int s = 1; s < n; s++){
    for (int i = 0, j; (i + s) < n; i++){
      j = i + s;
      dp[i][j] = INFLL;
      for (int k = opt[i][j - 1]; k < min(j, opt[i + 1][j] + 1); k++){
        ll cur = dp[i][k] + dp[k + 1][j] + C(i, j);
        if (dp[i][j] > cur){
          dp[i][j] = cur;
          opt[i][j] = k;
        }
      }
    }
  }
  return dp[0][n - 1];
}
```

## 4 Math

### 4.1 Basic Math

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;

ull fastPow(ull base, ull exp, ull mod){
  base %= mod;
  //exp %= phi(mod) if base and mod are relatively prime
  ull ans = 1LL;
  while (exp > 0){
```

```
    if (exp & 1LL)
      ans = (ans * (__int128_t)base) % mod;
    base = (base * (__int128_t)base) % mod;
    exp >>= 1;
  }
  return ans;
}
int fastPow(int base, string bigExp, int mod){
  int ans = 1;
  for(char c: bigExp){
    ans = fastPow(ans, 10, mod);
    ans = (ans*1LL*fastPow(base, c-'0', mod))%mod;
  }
  return ans;
}
ll gcd(ll a, ll b){ return __gcd(a, b); }
ll lcm(ll a, ll b){ return (a / gcd(a, b)) * b; }
void enumeratingAllSubmasks(int mask){
  for (int s = mask; s; s = (s - 1) & mask)
    cout << s << endl;
}
//MOD to Hash
namespace ModHash{
  const uint64_t MOD = (1ll<<61) - 1;
  uint64_t modmul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (uint32_t)b, h2 = b
        >>32;
    uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
    uint64_t ret = (l&MOD) + (l>>61) + (h << 3) + (m >> 29) + ((m <<
        35) >> 3) + 1;
    ret = (ret & MOD) + (ret>>61);
    ret = (ret & MOD) + (ret>>61);
    return ret-1;
  }
};
```

## 4.2 BigInt

```
#include <bits/stdc++.h>
using namespace std;
typedef int32_t intB;
typedef int64_t longB;
typedef vector<intB> vib;
class BigInt{
private:
  vib vb;
  bool neg;
  const int BASE_DIGIT = 9;
  const intB base = 1000000LL*1000;//000LL*1000000LL;
  void fromString(string &s){
    if(s[0] == '-'){
      neg = true;
      s = s.substr(1);
    }else{
      neg = false;
    }
    vb.clear();
    vb.reserve((s.size()+BASE_DIGIT-1)/BASE_DIGIT);
    for(int i=(int)s.length(); i>0; i-=BASE_DIGIT){
      if(i < BASE_DIGIT)
        vb.push_back(stol(s.substr(0, i)));
      else
        vb.push_back(stol(s.substr(i-BASE_DIGIT, BASE_DIGIT)));
    }
    fix(vb);
  }
  void fix(vib &v){
    while(v.size()>1 && v.back()==0)
      v.pop_back();
    if(v.size() == 0)
      neg = false;
  }
  bool comp(vib &a, vib &b){
    fix(a); fix(b);
    if(a.size() != b.size()) return a.size() < b.size();
    for(int i=(int)a.size()-1; i>=0; i--) {
      if(a[i] != b[i]) return a[i] < b[i];
    }
    return false;
  }
  vib sum(vib a, vib b){
    int carry = 0;
    for(size_t i=0; i<max(a.size(), b.size()) or carry; i++){
      if(i == a.size())
        a.push_back(0);
      a[i] += carry + (i<b.size() ? b[i] : 0);
      carry = (a[i] >= base);
      if(carry) a[i] -= base;
    }
    fix(a);
    return a;
  }
  vib sub(vib a, vib b){
    int carry = 0;
    for(size_t i=0; i<b.size() or carry; i++){
      a[i] -= carry + (i<b.size() ? b[i] : 0);
      carry = a[i] < 0;
      if(carry) a[i] += base;
    }
    fix(a);
    return a;
  }

public:
  BigInt(){}
  BigInt(intB n){
    neg = (n<0);
    vb.push_back(abs(n));
    fix(vb);
  }
  BigInt(string s){
    fromString(s);
  }
  BigInt operator =(BigInt oth){
    this->neg = oth.neg;
    this->vb = oth.vb;
    return *this;
  }
  BigInt operator +(BigInt &oth){
    vib &a = vb, &b = oth.vb;
    BigInt ans;
```

```
  if(neg == oth.neg){
    ans.vb = sum(vb, oth.vb);
    ans.neg = neg;
  }else{
    if(comp(a, b)){
      ans.vb = sub(b, a);
      ans.neg = oth.neg;
    }else{
      ans.vb = sub(a, b);
      ans.neg = neg;
    }
  }
  return ans;
}
BigInt operator -(BigInt oth){
  oth.neg ^= true;
  return (*this) + oth;
}
BigInt operator *(intB b){
  bool negB = false;
  if(b < 0){
    negB = true;
    b = -b;
  }
  BigInt ans = *this;
  auto &a = ans.vb;
  intB carry = 0;
  for(size_t i=0; i<a.size() or carry; i++){
    if(i == a.size()) a.push_back(0);
    longB cur = carry + a[i] *(longB) b;
    a[i] = intB(cur%base);
    carry = intB(cur/base);
  }
  ans.neg ^= negB;
  fix(ans.vb);
  return ans;
}
BigInt operator *(BigInt &oth){
  BigInt ans;
  auto a = vb, &b = oth.vb, &c = ans.vb;
  c.assign(a.size() + b.size(), 0);
  for(size_t i=0; i<a.size(); i++){
    intB carry=0;
    for(size_t j=0; j<b.size() or carry; j++){
      longB cur = c[i+j] + a[i]*(longB)(j<b.size() ? b[j] : 0);
      cur += carry;
      c[i+j] = intB(cur%base);
      carry = intB(cur/base);
    }
  }
  ans.neg = neg^oth.neg;
  fix(ans.vb);
  return ans;
}
BigInt operator /(intB b){
  bool negB = false;
  if(b < 0){
    negB = true;
    b = -b;
  }
  BigInt ans = *this;
  auto &a = ans.vb;
  intB carry = 0;
  for(int i=(int)a.size()-1; i>=0; i--){
    longB cur = a[i] + (longB)carry * base;
    a[i] = intB(cur/b);
    carry = intB(cur%b);
  }
  ans.neg ^= negB;
  fix(ans.vb);
  return ans;
}
void shiftL(int b){
  vb.resize(vb.size() + b);
  for(int i=(int)vb.size()-1; i>=0; i--) {
    if(i>=b) vb[i] = vb[i-b];
    else vb[i] = 0;
  }
  fix(vb);
}
void shiftR(int b) {
  if((int)vb.size() <= b){
    vb.clear();
    vb.push_back(0);
    return;
  }
  for(int i=0; i<((int)vb.size() - b); i++)
    vb[i] = vb[i+b];
  vb.resize((int)vb.size() - b);
  fix(vb);
}
void divide(BigInt a, BigInt b, BigInt &q, BigInt &r){
  BigInt z(0), p(1);
  while(b < a) {
    p.shiftL(max(1, int(a.vb.size()-b.vb.size())));
    b.shiftL(max(1, int(a.vb.size()-b.vb.size())));
  }
  while(true) {
    while ((a < b) && (z < p)) {
      p = p/10;
      b = b/10;
    }
    if(!(z < p)) break;
    a = a - b;
    q = q + p;
  }
  r = a;
}
BigInt operator /(BigInt &oth){
  BigInt q, r;
  divide(*this, oth, q, r);
  return q;
}
BigInt operator %(BigInt &oth){
  BigInt q, r;
  divide(*this, oth, q, r);
  return r;
}
bool operator <(BigInt &oth){
  BigInt ans = (*this) - oth;
  return ans.neg;
}
```

```
  bool operator ==(BigInt &oth){
    BigInt ans = (*this) - oth;
    return (ans.vb.size()==1) and (ans.vb.back()==0);
  }
  friend ostream &operator<<(ostream &out, const BigInt &D){
    if(D.neg)
      out << '-';
    out << (D.vb.empty() ? 0 : D.vb.back());
    for(int i=(int)D.vb.size()-2; i>=0; i--)
      out << setfill('0') << setw(D.BASE_DIGIT) << D.vb[i];
    return out;
  }
  string to_string(){
    std::stringstream ss;
    ss << (*this);
    return ss.str();
  }
  friend istream &operator>>(istream  &input, BigInt &D) {
    string s;
    input >> s;
    D.fromString(s);
    return input;
  }
};
```

## 4.3 Binomial Coefficients

```
#include <bits/stdc++.h>
#include "./basic_math.h"
#include "./modular.h"
using namespace std;
typedef long long ll;
//O(k)
ll C1(int n, int k){
  ll res = 1LL;
  for (int i = 1; i <= k; ++i)
    res = (res * (n - k + i)) / i;
  return res;
}
//O(n^2)
vector<vector<ll>> C2(int maxn, int mod){
  vector<vector<ll>> mat(maxn + 1, vector<ll>(maxn + 1, 0));
  mat[0][0] = 1;
  for (int n = 1; n <= maxn; n++){
    mat[n][0] = mat[n][n] = 1;
    for (int k = 1; k < n; k++)
      mat[n][k] = (mat[n - 1][k - 1] + mat[n - 1][k]) % mod;
  }
  return mat;
}
//O(N)
vector<int> factorial, inv_factorial;
void prevC3(int maxn, int mod){
  factorial.resize(maxn + 1);
  factorial[0] = 1;
  for (int i = 1; i <= maxn; i++)
    factorial[i] = (factorial[i - 1] * 1LL * i) % mod;
  inv_factorial.resize(maxn + 1);
  inv_factorial[maxn] = fastPow(factorial[maxn], mod - 2, mod);
  for (int i = maxn - 1; i >= 0; i--)
    inv_factorial[i] = (inv_factorial[i + 1] * 1LL * (i + 1)) % mod;
}
int C3(int n, int k, int mod){
  if (n < k)
    return 0;
  return (((factorial[n] * 1LL * inv_factorial[k]) % mod) * 1LL *
      inv_factorial[n - k]) % mod;
}
//O(P*log(P))
//C4(n, k, p) = Comb(n, k)%p
vector<int> changeBase(int n, int p){
  vector<int> v;
  while (n > 0){
    v.push_back(n % p);
    n /= p;
  }
  return v;
}
int C4(int n, int k, int p){
  auto vn = changeBase(n, p);
  auto vk = changeBase(k, p);
  int mx = max(vn.size(), vk.size());
  vn.resize(mx, 0);
  vk.resize(mx, 0);
  prevC3(p - 1, p);
  int ans = 1;
  for (int i = 0; i < mx; i++)
    ans = (ans * 1LL * C3(vn[i], vk[i], p)) % p;
  return ans;
}
//O(P^k)
//C5(n, k, p, pk) = Comb(n, k)%(p^k)
int fat_p(ll n, int p, int pk){
  vector<int> fat1(pk, 1);
    int res = 1;
    for(int i=1; i<pk; i++){
    if(i%p == 0)
      fat1[i] = fat1[i-1];
    else
      fat1[i] = (fat1[i-1]*1LL*i)%pk;
  }
  while(n > 1){
    res = (res*1LL*fastPow(fat1[pk-1], n/pk, pk))%pk;
    res = (res*1LL*fat1[n%pk])%pk;
    n /= p;
  }
  return res;
}
ll cnt(ll n, int p){
  ll ans = 0;
  while(n > 1){
    ans += n/p;
    n/=p;
  }
  return ans;
}
int C5(ll n, ll k, int p, int pk){
  ll exp = cnt(n, p) - cnt(n-k, p) - cnt(k, p);
  int d = (fat_p(n-k, p, pk)*1LL*fat_p(k, p, pk))%pk;
  int ans = (fat_p(n, p, pk)*1LL*inv(d, pk))%pk;
  return (ans*1LL*fastPow(p, exp, pk))%pk;
```

```
}
```

## 4.4 Chinese Remainder Theorem

```
#include <bits/stdc++.h>
#include "extended_euclidean.h"
using namespace std;
typedef long long ll;
namespace CRT{
  inline ll normalize(ll x, ll mod){
    x %= mod;
    if (x < 0)
      x += mod;
    return x;
  }
  ll solve(vector<ll> a, vector<ll> m){
    int n = a.size();
    for (int i = 0; i < n; i++)
      normalize(a[i], m[i]);
    ll ans = a[0];
    ll lcm1 = m[0];
    for (int i = 1; i < n; i++){
      ll x, y;
      ll g = extGcd(lcm1, m[i], x, y);
      if ((a[i] - ans) % g != 0)
        return -1;
      ans = normalize(ans + ((((a[i] - ans) / g) * x) % (m[i] / g)) *
          lcm1, (lcm1 / g) * m[i]);
      lcm1 = (lcm1 / g) * m[i]; //lcm(lcm1, m[i]);
    }
    return ans;
  }
} // namespace CRT
```

## 4.5 Determinant

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
const ld EPS = 1e-9;
ld determinant(vector<vector<ld>> a) {
  int n = a.size();
  ld det = 1;
  for(int i=0; i<n; i++) {
    int b = i;
    for(int j=i+1; j<n; j++)
      if(abs (a[j][i]) > abs (a[b][i]))
        b = j;
    if(abs(a[b][i]) < EPS)
      return 0;
    swap(a[i], a[b]);
    if(i != b)
      det = -det;
    det *= a[i][i];
    for(int j=i+1; j<n; ++j)
      a[i][j] /= a[i][i];
    for(int j=0; j<n; ++j)
      if(j != i && abs (a[j][i]) > EPS)
        for(int k=i+1; k<n; k++)
          a[j][k] -= a[i][k] * a[j][i];
  }
  return det;
}
```

## 4.6 Division Trick

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using pll = pair<ll, ll>;
// O(N)
pll bruteForce(ll n){
  ll ans1 = 0, ans2 = 0;
  for(ll i = 1; i <= n; i++){
    ans1 += n/i;
    ans2 += (n/i)*i; // n - (n mod i);
  }
  return pll(ans1, ans2);
}
ll AP(ll a1, ll an){
  ll n = (an-a1+1);
  return ((a1+an)*n)/2LL;
}
// O(sqrt(N))
pll divisionTrick(ll n){
  ll ans1 = 0, ans2 = 0;
  for(ll l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / i has the same value for l <= i <= r
    ans1 += (n/l)*(r-l+1);
    ans2 += (n/l)*AP(l, r);
  }
  return pll(ans1, ans2);
}
```

## 4.7 Euler's totient

```
#include <bits/stdc++.h>
using namespace std;
int nthPhi(int n){
  int result = n;
  for (int i = 2; i <= n / i; i++){
    if (n % i == 0){
      while (n % i == 0)
        n /= i;
      result -= result / i;
    }
  }
  if (n > 1)
    result -= result / n;
  return result;
}
vector<int> phiFrom1toN(int n){
  vector<int> vPhi(n + 1);
  vPhi[0] = 0;
  vPhi[1] = 1;
```

```
  for (int i = 2; i <= n; i++)
    vPhi[i] = i;
  for (int i = 2; i <= n; i++){
    if (vPhi[i] == i){
      for (int j = i; j <= n; j += i)
        vPhi[j] -= vPhi[j] / i;
    }
  }
  return vPhi;
}
```

## 4.8 Extended Euclidean

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll extGcd(ll a, ll b, ll &x, ll &y){
  if (b == 0){
    x = 1, y = 0;
    return a;
  }else{
    ll g = extGcd(b, a % b, y, x);
    y -= (a / b) * x;
    return g;
  }
}
//a*x + b*y = g
//a*(x-(b/g)*k) + b*(y+(a/g)*k) = g
bool dioEq(ll a, ll b, ll c, ll &x0, ll &y0, ll &g){
  g = extGcd(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
inline void shift_solution(ll &x, ll &y, ll a, ll b, ll cnt){
  x += cnt * b;
  y -= cnt * a;
}
ll findAllSolutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll
    maxy){
  ll x, y, g;
  if(a==0 or b==0){
    if(a==0 and b==0)
      return (c==0)*(maxx-minx+1)*(maxy-miny+1);
    if(a == 0)
      return (c%b == 0)*(maxx-minx+1)*(miny<=c/b and c/b<=maxy);
    return (c%a == 0)*(minx<=c/a and c/a<=maxx)*(maxy-miny+1);
  }
  if (!dioEq(a, b, c, x, y, g))
    return 0;
  a /= g;
  b /= g;
  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx)
    shift_solution(x, y, a, b, sign_b);
  if (x > maxx)
    return 0;
  ll lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx)
    shift_solution(x, y, a, b, -sign_b);
  ll rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny)
    shift_solution(x, y, a, b, -sign_a);
  if (y > maxy)
    return 0;
  ll lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy)
    shift_solution(x, y, a, b, sign_a);
  ll rx2 = x;
  if (lx2 > rx2)
    swap(lx2, rx2);
  ll lx = max(lx1, lx2);
  ll rx = min(rx1, rx2);
  if (lx > rx)
    return 0;
  return (rx - lx) / abs(b) + 1;
}
```

## 4.9 Fraction

```
#include <bits/stdc++.h>
using namespace std;
typedef long long f_type;
//Representation of the a/b
struct Fraction {
  f_type a, b;
  Fraction(f_type _a = 0): a(_a), b(1){}
  Fraction(f_type _a, f_type _b) {
    f_type g = __gcd(_a, _b);
    a = _a/g;
    b = _b/g;
    if(b < 0){
      a = -a;
      b = -b;
    }
  }
  Fraction operator+(Fraction oth) {
    return Fraction(a*oth.b + oth.a*b, b*oth.b);
  }
  Fraction operator-(Fraction oth) {
    return Fraction(a*oth.b - oth.a*b, b*oth.b);
  }
  Fraction operator*(Fraction oth) {
    return Fraction(a*oth.a, b*oth.b);
  }
  Fraction operator/(Fraction oth) {
    return Fraction(a*oth.b, b*oth.a);
  }
  bool operator>=(Fraction oth){
    return ((*this) - oth).a >= 0;
  }
  bool operator==(Fraction oth){
```

```
    return a == oth.a and b == oth.b;
  }
  operator f_type() {return a/b;}
  operator double() {return double(a)/b;}
};
```

## 4.10 FFT

```
#include <bits/stdc++.h>
using namespace std;
struct complex_t {
  double a {0.0}, b {0.0};
  complex_t(){}
  complex_t(double na) : a{na}{}
  complex_t(double na, double nb) : a{na}, b{nb} {}
  const complex_t operator+(const complex_t &c) const {
    return complex_t(a + c.a, b + c.b);
  }
  const complex_t operator-(const complex_t &c) const {
    return complex_t(a - c.a, b - c.b);
  }
  const complex_t operator*(const complex_t &c) const {
    return complex_t(a*c.a - b*c.b, a*c.b + b*c.a);
  }
  const complex_t operator/(const int &c) const {
    return complex_t(a/c, b/c);
  }
};
//using cd = complex<double>;
using cd = complex_t;
const double PI = acos(-1);
void fft(vector<cd> &a, bool invert) {
  int n = a.size();
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;
    if (i < j)
      swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++) {
        cd u = a[i+j], v = a[i+j+len/2] * w;
        a[i+j] = u + v;
        a[i+j+len/2] = u - v;
        w = w * wlen;
      }
    }
  }
  if (invert){
    for (cd &x : a)
      x = x / n;
  }
}
typedef long long ll;
vector<ll> multiply(vector<int> &a, vector<int> &b) {
  vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  int n = 1;
  while(n < int(a.size() + b.size()) )
    n <<= 1;
  fa.resize(n);
  fb.resize(n);
  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < n; i++)
    fa[i] = fa[i]*fb[i];
  fft(fa, true);
  vector<ll> result(n);
  for (int i = 0; i < n; i++)
    result[i] = ll(fa[i].a + 0.5);
  return result;
}
vector<ll> scalarProdut(vector<int> t, vector<int> p, bool isCyclic=
    false) {
  int nt = t.size();
  int np = p.size();
  t.resize(nt+np, 0);
  reverse(p.begin(), p.end());
  if(isCyclic)
    for(int i=nt; i<nt+np; i++)
      t[i] = t[i%nt];
  vector<ll> ans = multiply(t, p);
  for(int i=0; i<nt; i++)
    ans[i] = ans[np-1+i];
  ans.resize(nt);
  return ans;
}
inline int getID(char c){
  return c - 'a';
}
// Find p in text t. Wildcard character *
vector<bool> stringMatchingWithWildcards(string t, string p){
  int nt = t.size();
  int np = p.size();
  vector<cd> fa(nt), fb(np);
  for(int i=0; i<nt; i++){
    double apha = (2*PI*getID(t[i]))/26;
    fa[i] = cd(cos(apha), sin(apha));
  }
  reverse(p.begin(), p.end());
  int k = 0;
  for(int i=0; i<np; i++){
    if(p[i] != '*'){
      double apha = (2*PI*getID(p[i]))/26;
      fb[i] = cd(cos(apha), -sin(apha));
      k++;
    }else{
      fb[i] = cd(0, 0);
    }
  }
  int n = 1;
  while(n < int(nt + np) )
    n <<= 1;
  fa.resize(n);
  fb.resize(n);
  fft(fa, false);
```

```
  fft(fb, false);
  for (int i = 0; i < n; i++)
    fa[i] = fa[i]*fb[i];
  fft(fa, true);
  vector<bool> result(nt - np+1);
  for (int i = 0; i < (nt - np+1); i++)
    result[i] = (int(fa[np-1+i].a + 1e-9) == k);
  return result;
}
```

## 4.11 Floyd Cycle Finding

```
#include <bits/stdc++.h>
using namespace std;
int f(int x);
typedef pair<int, int> pii;
pii floydCycleFinding(int x0){
  int tortoise = f(x0), hare = f(f(x0));
  while(tortoise != hare){
    tortoise = f(tortoise);
    hare = f(f(hare));
  }
  int mu = 0;
  hare = x0;
  while(tortoise != hare){
    tortoise = f(tortoise);
    hare = f(hare);
    mu++;
  }
  int lambda = 1;
  hare = f(tortoise);
  while(tortoise != hare){
    hare = f(hare);
    lambda++;
  }
  return pii(mu, lambda);
}
```

## 4.12 Function Root Using Newton

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
struct Poly{
  vector<ld> v;
  Poly(vector<ld> &v1):v(v1){}
  //return f(x)
  ld f(ld x){
    ld ans = 0;
    ld e = 1;
    int n = v.size();
    for(int i=0; i<n; i++){
      ans += v[i] * e;
      e *= x;
    }
    return ans;
  }
  //return f'(x)
  ld df(ld x){
    ld ans = 0;
    ld e = 1;
    int n = v.size();
    for(int i=1; i<n; i++){
      ans += i * v[i] * e;
      e *= x;
    }
    return ans;
  }
  // takes some root of the polynomial
  ld root(ld x0=1){
    const ld eps = 1E-10;
    ld x = x0;
    for (;;) {
      ld nx = x - (f(x)/df(x));
      if (abs(x - nx) < eps)
        break;
      x = nx;
    }
    return x;
  }
  //div f(x) by (x-a)
  void div(ld a){
    int g = (int)v.size() - 1;
    vector<ld> aux(g);
    for(int i=g; i>=1; i--){
      aux[i-1] = v[i];
      v[i-1] += a*aux[i-1];
    }
    v = aux;
  }
};
```

## 4.13 Gauss

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 0x3f3f3f3f;
typedef long double ld;
const ld EPS = 1e-9;
int gauss(vector<vector<ld>> a, vector<ld> &ans) {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;
  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; col++) {
    int sel = row;
    for (int i=row; i<n; i++)
      if (abs(a[i][col]) > abs(a[sel][col]))
        sel = i;
    if (abs(a[sel][col]) < EPS)
      continue;
    for (int i=col; i<=m; i++)
      swap(a[sel][i], a[row][i]);
    where[col] = row;
    for (int i=0; i<n; i++){
      if (i != row) {
        ld c = a[i][col] / a[row][col];
        for (int j=col; j<=m; j++)
          a[i][j] -= a[row][j] * c;
```

```
      }
    }
    row++;
  }
  ans.assign(m, 0);
  for (int i=0; i<m; i++)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; i++) {
    ld sum = 0;
    for (int j=0; j<m; j++)
      sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
      return 0;
  }
  for (int i=0; i<m; i++)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

## 4.14 Gauss Xor

```
#include <bits/stdc++.h>
using namespace std;
const int MAXB = 30;
struct GaussXOR {
  int table[MAXB];
  GaussXOR() {
    for(int i = 0; i < MAXB; i++) {
      table[i] = 0;
    }
  }
  int size() {
    int ans = 0;
    for(int i = 0; i < MAXB; i++) {
      if(table[i]) ans++;
    }
    return ans;
  }
  bool isComb(int x) {
    for(int i = MAXB-1; i >= 0; i--) {
      x = std::min(x, x ^ table[i]);
    }
    return x == 0;
  }
  void add(int x) {
    for(int i = MAXB-1; i >= 0; i--) {
      if( (table[i] == 0) and (x & (1LL<<i)) ){
        table[i] = x;
        x = 0;
      } else {
        x = std::min(x, x ^ table[i]);
      }
    }
  }
  int max(){
    int ans = 0;
    for(int i = MAXB-1; i >= 0; i--) {
      ans = std::max(ans, ans ^ table[i]);
    }
    return ans;
  }
};
```

## 4.15 Gray Code

```
int grayCode(int nth){
  return nth ^ (nth >> 1);
}
int revGrayCode(int g){
  int nth = 0;
  for (; g > 0; g >>= 1)
    nth ^= g;
  return nth;
}
```

## 4.16 Matrix

```
#include <bits/stdc++.h>
#include "modular.h"
using namespace std;
const int D = 3;
struct Matrix{
  int m[D][D];
  Matrix(bool identify = false){
    memset(m, 0, sizeof(m));
    for (int i = 0; i < D; i++)
      m[i][i] = identify;
  }
  Matrix(vector<vector<int>> mat){
    for(int i=0; i<D; i++)
      for(int j=0; j<D; j++)
        m[i][j] = mat[i][j];
  }
  int * operator[](int pos){
    return m[pos];
  }
  Matrix operator*(Matrix oth){
    Matrix ans;
    for (int i = 0; i < D; i++){
      for (int j = 0; j < D; j++){
        int &sum = ans[i][j];
        for (int k = 0; k < D; k++)
          sum = modSum(sum, modMul(m[i][k], oth[k][j]));
      }
    }
    return ans;
  }
};
Matrix fastPow(Matrix base, ll exp){
  Matrix ans(true);
  while(exp){
    if(exp&1LL)
      ans = ans * base;
    base = base*base;
    exp>>=1;
  }
```

```
  return ans;
}
```

## 4.17 Modular Arithmetic

```
#include <bits/stdc++.h>
#include "extended_euclidean.h"
using namespace std;
const int MOD = 1000000007;
inline int modSum(int a, int b, int mod = MOD){
  int ans = a+b;
  if(ans > mod) ans -= mod;
  return ans;
}
inline int modSub(int a, int b, int mod = MOD){
  int ans = a-b;
  if(ans < 0) ans += mod;
  return ans;
}
inline int modMul(int a, int b, int mod = MOD){
  return (a*1LL*b)%mod;
}
int inv(int a, int mod=MOD){
  ll inv_x, y;
  extGcd(a, mod, inv_x, y);
  return (inv_x%mod + mod)%mod;
}
int modDiv(int a, int b, int mod = MOD){
  return modMul(a, inv(b, mod));
}
```

## 4.18 Modular Integer

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MOD = 1e9 + 7;
struct modInt{
  int val;
  modInt(long long v = 0) {
    if (v < 0)
      v = (v % MOD) + MOD;
    if (v >= MOD)
      v %= MOD;
    val = v;
  }
  static int modInv(int a, int m = MOD) {
    int g = m, r = a, x = 0, y = 1;
    while (r != 0) {
      int q = g / r;
      g %= r; swap(g, r);
      x -= q * y; swap(x, y);
    }
    return x < 0 ? x + m : x;
  }
  explicit operator int() const {
    return val;
  }
  modInt& operator+=(const modInt &other) {
    val += other.val;
    if (val >= MOD)
      val -= MOD;
    return *this;
  }
  modInt& operator-=(const modInt &other) {
    val -= other.val;
    if (val < 0) val += MOD;
    return *this;
  }
  modInt& operator*=(const modInt &other) {
    val = ((uint64_t) val * other.val) % MOD;
    return *this;
  }
  modInt& operator/=(const modInt &other) {
    return *this *= other.inv();
  }
  friend modInt operator+(const modInt &a, const modInt &b) {
    return modInt(a) += b;
  }
  friend modInt operator-(const modInt &a, const modInt &b) {
    return modInt(a) -= b;
  }
  friend modInt operator*(const modInt &a, const modInt &b) {
    return modInt(a) *= b;
  }
  friend modInt operator/(const modInt &a, const modInt &b) {
    return modInt(a) /= b;
  }
  modInt& operator++() {
    val = (val == (MOD - 1)) ? 0 : val + 1;
    return *this;
  }
  modInt operator++(int) {
    modInt before = *this;
    ++*this;
    return before;
  }
  modInt& operator--() {
    val = val == 0 ? MOD - 1 : val - 1;
    return *this;
  }
  modInt operator--(int) {
    modInt before = *this;
    --*this;
    return before;
  }
  modInt operator-() const {
    return (val == 0) ? 0 : MOD - val;
  }
  bool operator==(const modInt &other) const {
    return val == other.val;
  }
  bool operator!=(const modInt &other) const {
    return val != other.val;
  }
  modInt inv() const {
    return modInv(val);
  }
  modInt pow(long long p) const {
```

```
    assert(p >= 0);
    modInt a = *this, result = 1;
    while (p > 0) {
      if (p & 1)
        result *= a;
      a *= a;
      p >>= 1;
    }
    return result;
  }
};
```

## 4.19 Montgomery Multiplication

```
#include <bits/stdc++.h>
using namespace std;
using u64 = uint64_t;
using u128 = __uint128_t;
using i128 = __int128_t;
struct u256{
  u128 high, low;
  static u256 mult(u128 x, u128 y){
    u64 a = x >> 64, b = x;
    u64 c = y >> 64, d = y;
    u128 ac = (u128)a * c;
    u128 ad = (u128)a * d;
    u128 bc = (u128)b * c;
    u128 bd = (u128)b * d;
    u128 carry = (u128)(u64)ad + (u128)(u64)bc + (bd >> 64u);
    u128 high = ac + (ad >> 64u) + (bc >> 64u) + (carry >> 64u);
    u128 low = (ad << 64u) + (bc << 64u) + bd;
    return {high, low};
  }
};
//x_m := x*r mod n
struct Montgomery{
  u128 mod, inv, r2;
  //the N will be an odd number
  Montgomery(u128 n) : mod(n), inv(1), r2(-n % n){
    for (int i = 0; i < 7; i++)
      inv *= 2 - n * inv;
    for (int i = 0; i < 4; i++){
      r2 <<= 1;
      if (r2 >= mod)
        r2 -= mod;
    }
    for (int i = 0; i < 5; i++)
      r2 = mult(r2, r2);
  }
  u128 init(u128 x){
    return mult(x, r2);
  }
  u128 reduce(u256 x){
    u128 q = x.low * inv;
    i128 a = x.high - u256::mult(q, mod).high;
    if (a < 0)
      a += mod;
    return a;
  }
  u128 mult(u128 a, u128 b){
    return reduce(u256::mult(a, b));
  }
};
```

## 4.20 NTT

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MOD = 998244353;
inline int modMul(int a, int b) {
  return (int) ((a*(ll)b) % MOD);
}
namespace ntt {
  int base = 1;
  vector<int> roots = {0, 1};
  vector<int> rev = {0, 1};
  int max_base = -1;
  int root = -1;
  inline int power(int a, long long b) {
    int res = 1;
    while (b > 0) {
      if (b & 1)
        res = modMul(res, a);
      a = modMul(a, a);
      b >>= 1;
    }
    return res;
  }
  inline int inv(int a) {
    a %= MOD;
    if (a < 0) a += MOD;
    int b = MOD, u = 0, v = 1;
    while(a){
      int t = b / a;
      b -= t * a; swap(a, b);
      u -= t * v; swap(u, v);
    }
    assert(b == 1);
    if (u < 0) u += MOD;
    return u;
  }
  void init() {
    int tmp = MOD - 1;
    max_base = 0;
    while (tmp % 2 == 0) {
      tmp /= 2;
      max_base++;
    }
    root = 2;
    while (true) {
      if (power(root, 1 << max_base) == 1) {
        if (power(root, 1 << (max_base - 1)) != 1) {
          break;
        }
      }
      root++;
    }
  }
  void ensure_base(int nbase) {
```

```
  if (max_base == -1)
    init();
  if (nbase <= base)
    return;
  assert(nbase <= max_base);
  rev.resize(1 << nbase);
  for (int i = 0; i < (1 << nbase); i++)
    rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
  roots.resize(1 << nbase);
  while (base < nbase) {
    int z = power(root, 1 << (max_base - 1 - base));
    for (int i = 1 << (base - 1); i < (1 << base); i++) {
      roots[i << 1] = roots[i];
      roots[(i << 1) + 1] = modMul(roots[i], z);
    }
    base++;
  }
}
void fft(vector<int> &a) {
  int n = (int) a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = base - zeros;
  for (int i = 0; i < n; i++) {
    if (i < (rev[i] >> shift)) {
      swap(a[i], a[rev[i] >> shift]);
    }
  }
  for (int k = 1; k < n; k <<= 1) {
    for (int i = 0; i < n; i += 2 * k) {
      for (int j = 0; j < k; j++) {
        int x = a[i + j];
        int y = modMul(a[i + j + k], roots[j + k]);
        a[i + j] = x + y - MOD;
        if (a[i + j] < 0) a[i + j] += MOD;
        a[i + j + k] = x - y + MOD;
        if (a[i + j + k] >= MOD) a[i + j + k] -= MOD;
      }
    }
  }
}
vector<int> multiply(vector<int> a, vector<int> b, int eq = 0) {
  int need = (int) (a.size() + b.size() - 1);
  int nbase = 0;
  while ((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  int sz = 1 << nbase;
  a.resize(sz);
  b.resize(sz);
  fft(a);
  if (eq)
    b = a;
  else
    fft(b);
  int inv_sz = inv(sz);
  for (int i = 0; i < sz; i++)
    a[i] = modMul(modMul(a[i], b[i]), inv_sz);
  reverse(a.begin() + 1, a.end());
  fft(a);
  a.resize(need);
  return a;
}
vector<int> square(vector<int> a) {
  return multiply(a, a, 1);
}
vector<int> pow(vector<int> a, ll e){
  int need = (int) ( (a.size()-1)*e + 1);
  int nbase = 0;
  while ((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  int sz = 1 << nbase;
  a.resize(sz);
  fft(a);
  int inv_sz = ntt::inv(sz);
  for (int i = 0; i < sz; i++)
    a[i] = modMul(power(a[i], e), inv_sz);
  reverse(a.begin() + 1, a.end());
  fft(a);
  a.resize(need);
  return a;
}
};
```

## 4.21 Prime Number

```
#include <bits/stdc++.h>
#include "basic_math.h"
using namespace std;
typedef unsigned long long ull;
ull modMul(ull a, ull b, ull mod){
  return (a * (__uint128_t)b) % mod;
}
bool checkComposite(ull n, ull a, ull d, int s){
  ull x = fastPow(a, d, n);
  if (x == 1 or x == n - 1)
    return false;
  for (int r = 1; r < s; r++){
    x = modMul(x, x, n);
    if (x == n - 1LL)
      return false;
  }
  return true;
};
bool millerRabin(ull n){
  if (n < 2)
    return false;
  int r = 0;
  ull d = n - 1LL;
  while ((d & 1LL) == 0){
    d >>= 1;
    r++;
  }
  for (ull a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
    if (n == a)
      return true;
    if (checkComposite(n, a, d, r))
      return false;
  }
  return true;
}
```

```
ull pollard(ull n){
  auto f = [n](ull x) { return modMul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 0, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1){
    if (x == y)
      x = ++i, y = f(x);
    if ((q = modMul(prd, max(x, y) - min(x, y), n)))
      prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n){
  if (n == 1)
    return {};
  if (millerRabin(n))
    return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
```

## 4.22 Rank Matrix

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
const ld EPS = 1e-9;
int compute_rank(vector<vector<ld>> A) {
  int n = A.size();
  int m = A[0].size();
  int rank = max(n, m);
  vector<bool> row_selected(n, false);
  for (int i = 0; i < m; ++i) {
    int j;
    for (j = 0; j < n; ++j) {
      if (!row_selected[j] && abs(A[j][i]) > EPS)
        break;
    }
    if (j == n) {
      rank--;
    } else {
      row_selected[j] = true;
      for (int p = i + 1; p < m; p++)
        A[j][p] /= A[j][i];
      for (int k = 0; k < n; k++) {
        if (k != j && abs(A[k][i]) > EPS) {
          for (int p = i + 1; p < m; p++)
            A[k][p] -= A[j][p] * A[k][i];
        }
      }
    }
  }
  return rank;
}
```

## 4.23 Simpson Integration

```
#include <bits/stdc++.h>
using namespace std;
double f(double x);
const int N = 1000000;
double simpson_integration(double a, double b){
  double h = (b - a) / N;
  double s = f(a) + f(b); // a = x_0 and b = x_2n
  for (int i = 1; i <= N - 1; ++i) { // Refer to final Simpson's
      formula
    double x = a + h * i;
    s += f(x) * ((i & 1) ? 4 : 2);
  }
  s *= h / 3;
  return s;
}
```

## 4.24 Sieve And Primes

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll ns;
int np;
bitset<10000010> bs;
vector<ll> primes;
void sieve(ll l) {
  ns = l+1;
  bs.set();
  primes.clear();
  bs[0] = bs[1] = 0;
  for (ll i = 2; i < ns; i++) if (bs[i]) {
    for(ll j = i*i; j < ns; j += i)
      bs[j] = 0;
    primes.push_back(i);
  }
  np = primes.size();
}
bool isPrime(ll n) {
  if(n < ns)
    return bs[n];
  for(ll p: primes){
    if(p*p > n) break;
    if(n%p == 0)
      return false;
  }
  return true;
}
vector<ll> primeFactors(ll n) {
  vector<ll> factors;
  for(ll p: primes){
    if(p*p > n) break;
    while(n%p == 0LL) {
      n /= p;
      factors.push_back(p);
    }
  }
```

```
  if(n != 1LL) factors.push_back(n);
  return factors;
}
ll numDiv(ll n) {
  ll ans = 1;
  for(ll p: primes){
    if(p*p > n) break;
    ll f = 0;
    while(n%p == 0LL) {
      n /= p;
      f++;
    }
    ans *= (f+1LL);
  }
  return (n != 1LL) ? 2LL*ans : ans;
}
ll sumDiv(ll n) {
  ll ans = 1;
  for(ll p: primes){
    if(p*p > n) break;
    ll power = p;
    while(n%p == 0LL) {
      n /= p;
      power *= p;
    }
    ans *= (power - 1LL)/(p - 1LL);
  }
  if(n != 1LL)
    ans *= (n*n - 1LL)/(n - 1LL);
  return ans;
}
int mobius[1000010];
void sieveMobius(ll l) {
  sieve(l);
  mobius[1] = 1;
  for(int i=2; i<=l; i++)
    mobius[i] = 0;
  for(ll p: primes){
    if(p > l) break;
    for(ll j = p; j <= l; j += p){
      if(mobius[j] != -1){
        mobius[j]++;
        if(j%(p*p) == 0)
          mobius[j] = -1;
      }
    }
  }
  for(int i=2; i<=l; i++){
    if(mobius[i] == -1)
      mobius[i] = 0;
    else if(mobius[i]%2 == 0)
      mobius[i] = 1;
    else
      mobius[i] = -1;
  }
}
```

# 5 Geometry

## 5.1 Basic Geometry

```
#include <bits/stdc++.h>
using namespace std;
#define POINT_DOUBLE
#ifdef POINT_DOUBLE
  // Se necessario, apelar para __float128
  typedef double ftype;
  typedef long double ftLong;
  const double EPS = 1e-9;
  #define eq(a, b) (abs(a - b) < EPS)
  #define lt(a, b) ((a + EPS) < b)
  #define gt(a, b) (a > (b + EPS))
  #define le(a, b) (a < (b + EPS))
  #define ge(a, b) ((a + EPS) > b)
#else
  typedef int32_t ftype;
  typedef int64_t ftLong;
  #define eq(a, b) (a == b)
  #define lt(a, b) (a < b)
  #define gt(a, b) (a > b)
  #define le(a, b) (a <= b)
  #define ge(a, b) (a >= b)
#endif
//Begin Point 2D
struct Point2d{
  ftype x, y;
  Point2d() {}
  Point2d(ftype x1, ftype y1) : x(x1), y(y1) {}
  Point2d operator+(const Point2d &t){
    return Point2d(x + t.x, y + t.y);
  }
  Point2d operator-(const Point2d &t){
    return Point2d(x - t.x, y - t.y);
  }
  Point2d operator*(ftype t){
    return Point2d(x * t, y * t);
  }
  Point2d operator/(ftype t){
    return Point2d(x / t, y / t);
  }
  bool operator<(const Point2d &o) const{
    return lt(x, o.x) or (eq(x, o.x) and lt(y, o.y));
  }
  bool operator==(const Point2d &o) const{
    return eq(x, o.x) and eq(y, o.y);
  }
  friend std::istream& operator >> (std::istream &is, Point2d &p) {
    return is >> p.x >> p.y;
  }
  friend std::ostream& operator << (std::ostream &os, const Point2d &p
      ) {
    return os << p.x << ' ' << p.y;
  }
};
ftLong pw2(ftype a){
```

```
  return a * (ftLong)a;
}
//Scalar product
ftLong dot(Point2d a, Point2d b){
  return a.x*(ftLong)b.x + a.y*(ftLong)b.y;
}
ftLong norm(Point2d a){
  return dot(a, a);
}
double len(Point2d a){
  return sqrtl(dot(a, a));
}
double dist(Point2d a, Point2d b){
  return len(a - b);
}
//Vector product
ftLong cross(Point2d a, Point2d b){
  return a.x * (ftLong)b.y - a.y * (ftLong)b.x;
}
//Projection size from A to B
double proj(Point2d a, Point2d b){
  return dot(a, b) / len(b);
}
//The angle between A and B
double angle(Point2d a, Point2d b){
  return acos(dot(a, b) / len(a) / len(b));
}
//Left rotation. Angle in radian
Point2d rotateL(Point2d p, double ang){
  return Point2d(p.x * cos(ang) - p.y * sin(ang), p.x * sin(ang) + p.y
       * cos(ang));
}
//90 degree left rotation
Point2d perpL(Point2d a){
  return Point2d(-a.y, a.x);
}
//0-> 1o,2o quadrant, 1-> 3o,4o
int half(Point2d &p){
  if (gt(p.y, 0) or (eq(p.y, 0) and ge(p.x, 0)))
    return 0;
  else
    return 1;
}
//angle(a) < angle(b)
bool cmpByAngle(Point2d a, Point2d b){
  int ha = half(a), hb = half(b);
  if (ha != hb){
    return ha < hb;
  }else{
    ftLong c = cross(a, b);
    if(eq(c, 0))
      return lt(norm(a), norm(b));
    else
      return gt(c, 0);
  }
}
inline int sgn(ftLong x){
  return ge(x, 0) ? (eq(x, 0) ? 0 : 1) : -1;
}
//-1: angle(a, b) < angle(b, c)
// 0: angle(a, b) = angle(b, c)
//+1: angle(a, b) > angle(b, c)
int cmpAngleBetweenVectors(Point2d a, Point2d b, Point2d c){
  ftLong dotAB = dot(a, b), dotBC = dot(b, c);
  int sgnAB = sgn(dotAB), sgnBC = sgn(dotBC);
  if(sgnAB == sgnBC){
    //Careful with overflow
    ftLong l = pw2(dotAB)*dot(c, c), r = pw2(dotBC)*dot(a, a);
    if(l == r)
      return 0;
    if(sgnAB == 1)
      return gt(l, r)? -1 : +1;
    return lt(l, r)? -1 : +1;
  }else{
    return (sgnAB > sgnBC)? -1 : +1;
  }
}
//Line parameterized: r1 = a1 + d1*t
//This function can be generalized to 3D
Point2d intersect(Point2d a1, Point2d d1, Point2d a2, Point2d d2){
  return a1 + d1 * (cross(a2 - a1, d2) / cross(d1, d2));
}
//Distance between the point(a) and segment(ps1, ps2)
//This function can be generalized to 3D
ftLong distance_point_to_segment(Point2d a, Point2d ps1, Point2d ps2)
    {
  if(ps1 == ps2)
    return dist(ps1, a);
  Point2d d = ps2 - ps1;
  ftLong t = max(ftLong(0), min(ftLong(1), dot(a-ps1, d)/len(d)));
  Point2d proj = ps1 + Point2d(d.x*t, d.y*t);
  return dist(a, proj);
}
//Distance between the point(a) and line(pl1, pl2)
//This function can be generalized to 3D
double dist(Point2d a, Point2d pl1, Point2d pl2){
  //crs = parallelogram area
  double crs = cross(Point2d(a - pl1), Point2d(pl2 - pl1));
  //h = area/base
  return abs(crs / dist(pl1, pl2));
}
long double area(vector<Point2d> p){
  long double ret = 0;
  for (int i = 2; i < (int)p.size(); i++)
    ret += cross(p[i] - p[0], p[i - 1] - p[0]) / 2.0;
  return abs(ret);
}
ftLong signed_area_parallelogram(Point2d p1, Point2d p2, Point2d p3){
  return cross(p2 - p1, p3 - p2);
}
long double triangle_area(Point2d p1, Point2d p2, Point2d p3){
  return abs(signed_area_parallelogram(p1, p2, p3)) / 2.0;
}
bool pointInTriangle(Point2d a, Point2d b, Point2d c, Point2d p){
  ftLong s1 = abs(cross(b - a, c - a));
  ftLong s2 = abs(cross(a - p, b - p)) + abs(cross(b - p, c - p)) +
      abs(cross(c - p, a - p));
  return eq(s1, s2);
}
bool clockwise(Point2d p1, Point2d p2, Point2d p3){
  return lt(signed_area_parallelogram(p1, p2, p3), 0);
}
```

```
bool counter_clockwise(Point2d p1, Point2d p2, Point2d p3){
  return gt(signed_area_parallelogram(p1, p2, p3), 0);
}
//End Point 2D

//Begin Line
ftLong det(ftype a, ftype b, ftype c, ftype d){
  return a * (ftLong)d - b * (ftLong)c;
}
struct Line{
  ftype a, b, c;
  Line() {}
  Line(ftype a1, ftype b1, ftype c1) : a(a1), b(b1), c(c1){
    normalize();
  }
  Line(Point2d p1, Point2d p2){
    a = p1.y - p2.y;
    b = p2.x - p1.x;
    c = -a * p1.x - b * p1.y;
    normalize();
  }
  void normalize(){
#ifdef POINT_DOUBLE
    ftype z = sqrt(pw2(a) + pw2(b));
#else
    ftype z = __gcd(abs(a), __gcd(abs(b), abs(c)));
#endif
    if(eq(z, 0)) return;
    a /= z;
    b /= z;
    c /= z;
    if (lt(a, 0) or (eq(a, 0) and lt(b, 0))){
      a = -a;
      b = -b;
      c = -c;
    }
  }
};
bool intersect(Line m, Line n, Point2d &res){
  ftype zn = det(m.a, m.b, n.a, n.b);
  if (eq(zn, 0))
    return false;
  res.x = -det(m.c, m.b, n.c, n.b) / zn;
  res.y = -det(m.a, m.c, n.a, n.c) / zn;
  return true;
}
bool parallel(Line m, Line n){
  return eq(det(m.a, m.b, n.a, n.b), 0);
}
bool equivalent(Line m, Line n){
  return eq(det(m.a, m.b, n.a, n.b), 0) &&
         eq(det(m.a, m.c, n.a, n.c), 0) &&
         eq(det(m.b, m.c, n.b, n.c), 0);
}
//Distance from a point(x, y) to a line m
double dist(Line m, ftype x, ftype y){
  return abs(m.a * (ftLong)x + m.b * (ftLong)y + m.c) /
         sqrt(m.a * (ftLong)m.a + m.b * (ftLong)m.b);
}
//End Line

//Begin Segment
struct Segment{
  Point2d a, b;
  Segment() {}
  Segment(Point2d a1, Point2d b1) : a(a1), b(b1) {}
};
bool inter1d(ftype a, ftype b, ftype c, ftype d){
  if (gt(a, b)) swap(a, b);
  if (gt(c, d)) swap(c, d);
  return le(max(a, c), min(b, d));
}
bool check_intersection(Segment s1, Segment s2){
  Point2d a = s1.a, b = s1.b, c = s2.a, d = s2.b;
  if (eq(cross(a - c, d - c), 0) && eq(cross(b - c, d - c), 0))
    return inter1d(a.x, b.x, c.x, d.x) && inter1d(a.y, b.y, c.y, d.y);
  return sgn(cross(b - a, c - a)) != sgn(cross(b - a, d - a)) &&
         sgn(cross(d - c, a - c)) != sgn(cross(d - c, b - c));
}
inline bool betw(ftype l, ftype r, ftype x){
  return le(min(l, r), x) and le(x, max(l, r));
}
bool intersect(Segment s1, Segment s2, Segment &ans){
  Point2d a = s1.a, b = s1.b, c = s2.a, d = s2.b;
  if (!inter1d(a.x, b.x, c.x, d.x) || !inter1d(a.y, b.y, c.y, d.y))
    return false;
  Line m(a, b);
  Line n(c, d);
  if (parallel(m, n)){
    if (!equivalent(m, n))
      return false;
    if (b < a)
      swap(a, b);
    if (d < c)
      swap(c, d);
    ans = Segment(max(a, c), min(b, d));
    return true;
  }else{
    Point2d p(0, 0);
    intersect(m, n, p);
    ans = Segment(p, p);
    return betw(a.x, b.x, p.x) && betw(a.y, b.y, p.y) &&
           betw(c.x, d.x, p.x) && betw(c.y, d.y, p.y);
  }
}
//End Segment

//Begin Circle
struct Circle{
  ftype x, y, r;
  Circle() {}
  Circle(ftype x1, ftype y1, ftype r1) : x(x1), y(y1), r(r1){};
};
bool pointInCircle(Circle c, Point2d p){
  return ge(c.r, dist(Point2d(c.x, c.y), p));
}
//CircumCircle of a triangle is a circle that passes through all the
    vertices
Circle circumCircle(Point2d a, Point2d b, Point2d c){
  Point2d u((b - a).y, -((b - a).x));
  Point2d v((c - a).y, -((c - a).x));
  Point2d n = (c - b) * 0.5;
```

```
  double t = cross(u, n) / cross(v, u);
  Point2d ct = (((a + c) * 0.5) + (v * t));
  double r = dist(ct, a);
  return Circle(ct.x, ct.y, r);
}
//InCircle is the largest circle contained in the triangle
Circle inCircle(Point2d a, Point2d b, Point2d c){
  double m1 = dist(a, b);
  double m2 = dist(a, c);
  double m3 = dist(b, c);
  Point2d ct = ((c * m1) + (b * m2) + a * (m3)) / (m1 + m2 + m3);
  double sp = 0.5 * (m1 + m2 + m3);
  double r = sqrt(sp * (sp - m1) * (sp - m2) * (sp - m3)) / sp;
  return Circle(ct.x, ct.y, r);
}
//Minimum enclosing circle, O(n)
Circle minimumCircle(vector<Point2d> p){
  random_shuffle(p.begin(), p.end());
  Circle c = Circle(p[0].x, p[0].y, 0.0);
  for (int i = 0; i < (int)p.size(); i++){
    if (pointInCircle(c, p[i]))
      continue;
    c = Circle(p[i].x, p[i].y, 0.0);
    for (int j = 0; j < i; j++){
      if (pointInCircle(c, p[j]))
        continue;
      c = Circle((p[j].x + p[i].x) * 0.5, (p[j].y + p[i].y) * 0.5, 0.5
          * dist(p[j], p[i]));
      for (int k = 0; k < j; k++){
        if (pointInCircle(c, p[k]))
          continue;
        c = circumCircle(p[j], p[i], p[k]);
      }
    }
  }
  return c;
}
//Return the number of the intersection
int circle_line_intersection(Circle circ, Line line, Point2d &p1,
    Point2d &p2){
  ftLong r = circ.r;
  ftLong a = line.a, b = line.b, c = line.c + line.a * circ.x + line.b
      * circ.y; //take a circle to the (0, 0)
  ftLong x0 = -a * c / (pw2(a) + pw2(b)), y0 = -b * c / (pw2(a) + pw2(
      b));       //(x0, y0) is the shortest distance point of the line
       for (0, 0)
  if (gt(pw2(c), pw2(r) * (pw2(a) + pw2(b)))){
    return 0;
  }
  else if (eq(pw2(c), pw2(r) * (pw2(a) + pw2(b)))){
    p1.x = p2.x = x0 + circ.x;
    p1.y = p2.y = y0 + circ.y;
    return 1;
  }else{
    ftLong d_2 = pw2(r) - pw2(c) / (pw2(a) + pw2(b));
    ftLong mult = sqrt(d_2 / (pw2(a) + pw2(b)));
    p1.x = x0 + b * mult + circ.x;
    p2.x = x0 - b * mult + circ.x;
    p1.y = y0 - a * mult + circ.y;
    p2.y = y0 + a * mult + circ.y;
    return 2;
  }
}
//Return the number of the intersection
int circle_intersection(Circle c1, Circle c2, Point2d &p1, Point2d &p2
    ){
  if (eq(c1.x, c2.x) and eq(c1.y, c2.y)){
    if (eq(c1.r, c2.r))
      return -1; //INF
    else
      return 0;
  }else{
    Circle circ(0, 0, c1.r);
    Line line;
    line.a = -2 * (c2.x - c1.x);
    line.b = -2 * (c2.y - c1.y);
    line.c = pw2(c2.x - c1.x) + pw2(c2.y - c1.y) + pw2(c1.r) - pw2(c2.
        r);
    int sz = circle_line_intersection(circ, line, p1, p2);
    p1.x += c1.x;
    p2.x += c1.x;
    p1.y += c1.y;
    p2.y += c1.y;
    return sz;
  }
}

bool checkIfTheSegmentIsCompletelyCoveredByCircles(vector<Circle> &vc,
     Segment s){
  vector<Point2d> v = {s.a, s.b};
  Line l(s.a, s.b);
  for (Circle c : vc){
    Point2d p1, p2;
    int inter = circle_line_intersection(c, l, p1, p2);
    if (inter >= 1 and betw(s.a.x, s.b.x, p1.x) and betw(s.a.y, s.b.y,
         p1.y))
      v.push_back(p1);
    if (inter == 2 and betw(s.a.x, s.b.x, p2.x) and betw(s.a.y, s.b.y,
         p2.y))
      v.push_back(p2);
  }
  sort(v.begin(), v.end());
  bool ans = true;
  for (int i = 1; i < (int)v.size(); i++){
    bool has = false;
    for (Circle c : vc){
      if (pointInCircle(c, v[i - 1]) and pointInCircle(c, v[i])){
        has = true;
        break;
      }
    }
    ans = ans && has;
  }
  return ans;
}

void tangents(Point2d c, double r1, double r2, vector<Line> &ans){
  double r = r2 - r1;
  double z = pw2(c.x) + pw2(c.y);
  double d = z - pw2(r);
  if (lt(d, 0))
    return;
```

```
  d = sqrt(abs(d));
  Line l;
  l.a = (c.x * r + c.y * d) / z;
  l.b = (c.y * r - c.x * d) / z;
  l.c = r1;
  ans.push_back(l);
}
vector<Line> tangents(Circle a, Circle b){
  vector<Line> ans;
  for (int i = -1; i <= 1; i += 2)
    for (int j = -1; j <= 1; j += 2)
      tangents(Point2d(b.x - a.x, b.y - a.y), a.r * i, b.r * j, ans);
  for (size_t i = 0; i < ans.size(); ++i){
    ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
    ans[i].normalize();
  }
  return ans;
}
//End Circle
```

## 5.2 Circle Area Union

```
#include "basic_geometry.h"
using namespace std;

const double PI = acos(-1);
pair<double, double> isCC(Circle circ1, Circle circ2){
  Point2d c1(circ1.x, circ1.y), c2(circ2.x, circ2.y);
  double r1 = circ1.r, r2 = circ2.r;
  double d = dist(c1, c2);
  double x1 = c1.x, x2 = c2.x, y1 = c1.y, y2 = c2.y;
  double mid = atan2(y2 - y1, x2 - x1);
  double a = r1, c = r2;
  double t = acos((a * a + d * d - c * c) / (2 * a * d));
  return make_pair(mid - t, mid + t);
}
int testCC(Circle circ1, Circle circ2){
  Point2d c1(circ1.x, circ1.y), c2(circ2.x, circ2.y);
  double r1 = circ1.r, r2 = circ2.r;
  double d = dist(c1, c2);
  if (le(r1 + r2, d))
    return 1; // not intersected or tged
  if (le(r1 + d, r2))
    return 2; // C1 inside C2
  if (le(r2 + d, r1))
    return 3; // C2 inside C1
  return 0;   // intersected
}
struct event_t{
  double theta;
  int delta;
  event_t(double t, int d) : theta(t), delta(d) {}
  bool operator<(const event_t &r) const{
    if (fabs(theta - r.theta) < EPS)
      return delta > r.delta;
    return theta < r.theta;
  }
};
vector<event_t> e;
void add(double begin, double end){
  if (begin <= -PI)
    begin += 2 * PI, end += 2 * PI;
  if (end > PI){
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(PI, -1));
    e.push_back(event_t(-PI, 1));
    e.push_back(event_t(end - 2 * PI, -1));
  }else{
    e.push_back(event_t(begin, 1));
    e.push_back(event_t(end, -1));
  }
}
double calc(Point2d c, double r, double a1, double a2){
  double da = a2 - a1;
  double aa = r * r * (da - sin(da)) / 2;
  Point2d p1 = Point2d(cos(a1), sin(a1)) * r + c;
  Point2d p2 = Point2d(cos(a2), sin(a2)) * r + c;
  return cross(p1, p2) / 2 + aa;
}
/* O(n^2logn), please remove coincided circles first. */
double circle_union(vector<Circle> &vc){
  int n = vc.size();
  for (int i = n - 1; i >= 0; i--){
    if (eq(vc[i].r, 0)){
      swap(vc[i], vc[n - 1]);
      n--;
      continue;
    }
    for (int j = 0; j < i; j++){
      if (eq(vc[i].x, vc[j].x) and eq(vc[i].y, vc[j].y) and eq(vc[i].r
          , vc[j].r)){
        swap(vc[i], vc[n - 1]);
        n--;
      }
    }
  }
  if (n == 0)
    return 0;
  vc.resize(n);
  vector<double> cntarea(2 * n, 0);
  for (int c = 0; c < n; c++){
    int cvrcnt = 0;
    e.clear();
    for (int i = 0; i < n; i++){
      if (i != c){
        int r = testCC(vc[c], vc[i]);
        if (r == 2){
          cvrcnt++;
        } else if (r == 0){
          auto paa = isCC(vc[c], vc[i]);
          add(paa.first, paa.second);
        }
      }
    }
    if (e.size() == 0){
      double a = PI * vc[c].r * vc[c].r;
      cntarea[cvrcnt] -= a;
      cntarea[cvrcnt + 1] += a;
    } else {
      e.push_back(event_t(-PI, 1));
      e.push_back(event_t(PI, -2));
```

```
      sort(e.begin(), e.end());
      for (int i = 0; i < int(e.size()) - 1; i++){
        cvrcnt += e[i].delta;
        double a = calc(Point2d(vc[c].x, vc[c].y), vc[c].r, e[i].theta
            , e[i + 1].theta);
        cntarea[cvrcnt - 1] -= a;
        cntarea[cvrcnt] += a;
      }
    }
  }
  double ans = 0;
  for(int i=1; i<=n; i++)
    ans += cntarea[i];
  return ans;
}
```

## 5.3 Circles to Tree

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
struct Circle{
  int x, y, r, id;
  Circle(){}
  Circle(int x1, int y1, int r1, int id1): x(x1), y(y1), r(r1), id(id1
      ){}
};
// a^2 + b^2 == c^2
double findB(double a, double c){
  return sqrt(c*c - a*a);
}
//- There is no intersection between the circles
//- The parent of circle i will be the smallest circle that includes i
namespace CirclesToTree{
  int X = 0;
  int n;
  vector<Circle> vc;
  vector<int> p;
  struct SetElement{
    int id;
    int side; //Up:1, Down:-1
    SetElement(int id1, int side1): id(id1), side(side1){};
    double getY(int x = X) const{
      return vc[id].y + side*findB(vc[id].x - x, vc[id].r);
    }
    bool operator <(const SetElement &o) const{
      auto l = getY(), r = o.getY();
      if(abs(l-r)<1e-9)
        return vc[id].r*side < vc[o.id].r*o.side;
      else
        return l < r;
    }
  };
  long long pw2(int a){
    return a*1LL*a;
  }
  bool contains(int big, int small){
    if(big == -1 or small == -1) return false;
    Circle &s = vc[small], &b = vc[big];
    if(s.r > b.r) return false;
    return pw2(s.x-b.x) + pw2(s.y-b.y) <= pw2(b.r-s.r);
  }
  void updateParent(int id, int par){
    if(par != -1 and p[id] == -1) p[id] = par;
  }
//Public
  vector<vector<int>> solve(vector<Circle> circles){
    vc = circles; n = vc.size();
    p.assign(n, -1);
    vector<vector<int>> adj(n, vector<int>());
    vector<pii> events;
    for(auto c: vc){
      events.emplace_back(c.x-c.r, ~c.id);
      events.emplace_back(c.x+c.r, c.id);
    }
    sort(events.begin(), events.end());
    set<SetElement> st;
    for(auto e: events){
      X = e.first;
      int id = e.second;
      if(id < 0){
        id = ~id;
        auto it = st.lower_bound(SetElement(id, -2));
        if(it != st.end()){
          int id2 = it->id;
          if(contains(id2, id)) updateParent(id, id2);
          if(contains(p[id2], id)) updateParent(id, p[id2]);
        }
        if(it != st.begin()){
          it--;
          int id2 = it->id;
          if(contains(id2, id)) updateParent(id, id2);
          if(contains(p[id2], id)) updateParent(id, p[id2]);
        }
        st.emplace(id, 1);
        st.emplace(id, -1);
        if(p[id] != -1){
          adj[p[id]].push_back(id);
        }
      }else{
        st.erase(SetElement(id, 1));
        st.erase(SetElement(id, -1));
      }
    }
    return adj;
  }
};
```

## 5.4 Count Lattices

```
#include "../../code/math/fraction.h"
Fraction f_1 = 1;
//Calculates number of integer points (x,y) such for 0<=x<n and 0<y<=
    floor(k*x+b)
//O(log(N)*log(MAXV))
f_type count_lattices(Fraction k, Fraction b, f_type n) {
  auto fk = (f_type)k;
  auto fb = (f_type)b;
  auto cnt = 0LL;
```

```
  if (k >= f_1 || b >= f_1) {
    cnt += (fk * (n - 1) + 2 * fb) * n / 2;
    k = k - Fraction(fk, 1);
    b = b - Fraction(fb, 1);
  }
  auto t = k * Fraction(n, 1) + b;
  auto ft = (f_type)t;
  if (ft >= 1) {
    cnt += count_lattices(f_1 / k, (t - Fraction((f_type)t, 1)) / k, (
        f_type)t);
  }
  return cnt;
}
```

## 5.5 Concave Polygon

```
#include "basic_geometry.h"
const int INSIDE=-1, BOUNDARY=0, OUTSIDE=1;
struct ConcavePolygon{
  vector<Point2d> vp;
  ConcavePolygon(vector<Point2d> aux){
    vp = aux;
  }
  // -1 inside, 0 boundary, 1 outside
  int pointInPolygon(Point2d pt) {
    int n = vp.size(), w = 0;
    for(int i=0; i<n; i++){
      if(pt == vp[i])
        return 0;
      int j = (i+1==n?0:i+1);
      if(vp[i].y == pt.y and vp[j].y == pt.y) {
        if (min(vp[i].x, vp[j].x) <= pt.x and pt.x <= max(vp[i].x, vp[
            j].x))
          return 0;
      }else{
        bool below = vp[i].y < pt.y;
        if (below != (vp[j].y < pt.y)) {
          auto orientation = cross(pt-vp[i], vp[j]-vp[i]);
          if (orientation == 0) return 0;
          if (below == (orientation > 0))
            w += below ? 1 : -1;
        }
      }
    }
    return (w==0?1:-1);
  }
};
```

## 5.6 Convex Hull

```
#include "basic_geometry.h"
using namespace std;
//If accept collinear points then change for <=
bool cw(Point2d a, Point2d b, Point2d c) {
  return lt(cross(b - a, c - b), 0);
}
//If accept collinear points then change for >=
bool ccw(Point2d a, Point2d b, Point2d c) {
  return gt(cross(b - a, c - b), 0);
}
// Returns the points clockwise
vector<Point2d> convex_hull(vector<Point2d> a){
  if (a.size() == 1)
    return a;
  sort(a.begin(), a.end());
  a.erase(unique(a.begin(), a.end()), a.end());
  vector<Point2d> up, down;
  Point2d p1 = a[0], p2 = a.back();
  up.push_back(p1);
  down.push_back(p1);
  for (int i = 1; i < (int)a.size(); i++){
    if ((i == int(a.size() - 1)) || cw(p1, a[i], p2)){
      while (up.size() >= 2 && !cw(up[up.size() - 2], up[up.size() -
          1], a[i]))
        up.pop_back();
      up.push_back(a[i]);
    }
    if ((i == int(a.size() - 1)) || ccw(p1, a[i], p2)){
      while (down.size() >= 2 && !ccw(down[down.size() - 2], down[down
          .size() - 1], a[i]))
        down.pop_back();
      down.push_back(a[i]);
    }
  }
  a.clear();
  for (int i = 0; i < (int)up.size(); i++)
    a.push_back(up[i]);
  for (int i = down.size() - 2; i > 0; i--)
    a.push_back(down[i]);
  return a;
}
```

## 5.7 Convex Hull Trick

```
#include "basic_geometry.h"
using namespace std;
struct LineCHT{
    ftype k, b;
    int id;
    LineCHT() {}
    LineCHT(ftype k, ftype b, int id=-1): k(k), b(b), id(id) {}
};
struct ConvexHullTrick{
  vector<Point2d> hull, vecs;
  ConvexHullTrick(){}
  ConvexHullTrick(vector<LineCHT> v){
    sort(v.begin(), v.end(), [&](LineCHT a, LineCHT b){
      return a.k < a.k;
    });
    for(auto l: v)
      add_line(l.k, l.b);
  }
  //Here we will assume that when linear functions are added, their k
      only increases and we want to find minimum values.
  void add_line(ftype k, ftype b) {
    Point2d nw(k, b);
    while(!vecs.empty() && lt(dot(vecs.back(), nw - hull.back()), 0))
        {
```

```
      hull.pop_back();
      vecs.pop_back();
    }
    if(!hull.empty())
      vecs.push_back(perpL(nw - hull.back()));
    hull.push_back(nw);
  }
  //Find minimum value
  ftLong get(ftype x) {
    Point2d query(x, 1);
    auto it = lower_bound(vecs.begin(), vecs.end(), query, [](Point2d
        a, Point2d b) {
      return gt(cross(a, b), 0);
    });
    return dot(query, hull[it - vecs.begin()]);
  }
};
```

## 5.8 Convex Polygon

```
#include "convex_hull.h"
using namespace std;
//Checks if the point P belongs to the segment AB
bool pointInSegment(Point2d &a, Point2d &b, Point2d &p) {
  if(!eq(cross(a-p, b-p), 0))
    return false;
  return betw(a.x, b.x, p.x) && betw(a.y, b.y, p.y);
}
struct ConvexPolygon{
  vector<Point2d> vp;
  ConvexPolygon(vector<Point2d> aux){
    //The points have to be clockwise
    vp = convex_hull(aux);
  }
  //O(log(N))
  //Accepts points on the edge
  bool pointInPolygon(Point2d point){
    if(vp.size() < 3)
      return pointInSegment(vp[0], vp[1], point);
    if(!eq(cross(vp[1]-vp[0], point-vp[0]), 0) and sgn(cross(vp[1]-vp
        [0], point-vp[0])) != sgn(cross(vp[1]-vp[0], vp.back()-vp[0]))
         )
      return false;
    if(!eq(cross(vp.back()-vp[0], point-vp[0]), 0) and sgn(cross(vp.
        back()-vp[0], point-vp[0])) != sgn(cross(vp.back() - vp[0], vp
        [1]-vp[0])) )
      return false;
    if(eq(cross(vp[1]-vp[0], point-vp[0]), 0))
      return ge(norm(vp[1]-vp[0]), norm(point-vp[0]));
    int pos = 1, l = 1, r = vp.size() - 2;
    while(l <= r){
      int mid = (l + r)/2;
      if(le(cross(vp[mid] - vp[0], point - vp[0]), 0)){
        pos = mid;
        l = mid+1;
      }else{
        r = mid-1;
      }
    }
    return pointInTriangle(vp[0], vp[pos], vp[pos+1], point);
  }
};
```

## 5.9 Nearest Pair Of Points

```
#include <bits/stdc++.h>
using namespace std;
struct pt {
  long long x, y, id;
  pt(){}
  pt(int _x, int _y, int _id=-1):x(_x), y(_y), id(_id){}
};
namespace NearestPairOfPoints{
  struct cmp_x {
    bool operator()(const pt & a, const pt & b) const {
      return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
  };
  struct cmp_y {
    bool operator()(const pt & a, const pt & b) const {
      return a.y < b.y;
    }
  };
  int n;
  vector<pt> v;
  vector<pt> t;
  double mindist;
  pair<int, int> best_pair;
  void upd_ans(const pt & a, const pt & b) {
    double dist = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.
        y));
    if (dist < mindist) {
      mindist = dist;
      best_pair = {a.id, b.id};
    }
  }
  void rec(int l, int r) {
    if (r - l <= 3) {
      for (int i = l; i < r; ++i) {
        for (int j = i + 1; j < r; ++j) {
          upd_ans(v[i], v[j]);
        }
      }
      sort(v.begin() + l, v.begin() + r, cmp_y());
      return;
    }
    int m = (l + r) >> 1;
    int midx = v[m].x;
    rec(l, m);
    rec(m, r);
    merge(v.begin() + l, v.begin() + m, v.begin() + m, v.begin() + r,
        t.begin(), cmp_y());
    copy(t.begin(), t.begin() + r - l, v.begin() + l);
    int tsz = 0;
    for (int i = l; i < r; ++i) {
      if (abs(v[i].x - midx) < mindist) {
        for (int j = tsz - 1; j >= 0 && v[i].y - t[j].y < mindist; --j
            )
          upd_ans(v[i], t[j]);
        t[tsz++] = v[i];
```

```
      }
    }
  }
  pair<int, int> solve(vector<pt> _v){
    v = _v;
    n = v.size();
    t.resize(n);
    sort(v.begin(), v.end(), cmp_x());
    mindist = 1E20;
    rec(0, n);
    return best_pair;
  }
};
```

## 5.10 Point 3D

```
#include <bits/stdc++.h>
using namespace std;
//#define POINT_DOUBLE
#ifdef POINT_DOUBLE
  typedef double ftype;
  typedef long double ftLong;
  const double EPS = 1e-9;
  #define eq(a, b) (abs(a-b)<EPS)
  #define lt(a, b) ((a+EPS)<b)
  #define gt(a, b) (a>(b+EPS))
  #define le(a, b) (a<(b+EPS))
  #define ge(a, b) ((a+EPS)>b)
#else
  typedef int32_t ftype;
  typedef int64_t ftLong;
  #define eq(a, b) (a==b)
  #define lt(a, b) (a<b)
  #define gt(a, b) (a>b)
  #define le(a, b) (a<=b)
  #define ge(a, b) (a>=b)
#endif
//Point3D
struct Point3d{
  ftype x, y, z;
  Point3d() {}
  Point3d(ftype x, ftype y, ftype z) : x(x), y(y), z(z) {}
  Point3d operator+(Point3d t){
    return Point3d(x + t.x, y + t.y, z + t.z);
  }
  Point3d operator-(Point3d t){
    return Point3d(x - t.x, y - t.y, z - t.z);
  }
  Point3d operator*(ftype t){
    return Point3d(x * t, y * t, z * t);
  }
  Point3d operator/(ftype t){
    return Point3d(x / t, y / t, z / t);
  }
};
ftLong dot(Point3d a, Point3d b){
  return a.x * (ftLong)b.x + a.y * (ftLong)b.y + a.z * (ftLong)b.z;
}
double len(Point3d a){
  return sqrt(dot(a, a));
}
double dist(Point3d a, Point3d b){
  return len(a-b);
}
double proj(Point3d a, Point3d b){
  return dot(a, b) / len(b);
}
//theta -> XY; phi -> ZY;
Point3d toVetor(double theta, double phi, double r){
  return Point3d(r*cos(theta)*sin(phi), r*sin(theta)*sin(phi), r*cos(
      phi));
}
double getAngleTheta(Point3d p){
  return atan2(p.y, p.x);
}
double getAnglePhi(Point3d p){
  return acos(p.z/len(p));
}
Point3d rotateX(Point3d p, double ang){
  return Point3d(p.x, p.y*cos(ang)-p.z*sin(ang), p.y*sin(ang)+p.z*cos(
      ang));
}
Point3d rotateY(Point3d p, double ang){
  return Point3d(p.x*cos(ang)+p.z*sin(ang), p.y, -p.x*sin(ang)+p.z*cos
      (ang));
}
Point3d rotateZ(Point3d p, double ang){
  return Point3d(p.x*cos(ang)-p.y*sin(ang), p.x*sin(ang)+p.y*cos(ang),
       p.z);
}
//Rotation in relation to the normal axis
Point3d rotateNormal(Point3d v, Point3d n, double ang){
  double theta = getAngleTheta(n);
  double phi = getAnglePhi(n);
  v = rotateZ(v, -theta);
  v = rotateY(v, -phi);
  v = rotateZ(v, ang);
  v = rotateY(v, phi);
  v = rotateZ(v, theta);
  return v;
}
Point3d cross(Point3d a, Point3d b){
  return Point3d(a.y * b.z - a.z * b.y,
                 a.z * b.x - a.x * b.z,
                 a.x * b.y - a.y * b.x);
}
ftLong triple(Point3d a, Point3d b, Point3d c){
  return dot(a, cross(b, c));
}
Point3d planeIntersect(Point3d a1, Point3d n1, Point3d a2, Point3d n2,
     Point3d a3, Point3d n3){
  Point3d x(n1.x, n2.x, n3.x);
  Point3d y(n1.y, n2.y, n3.y);
  Point3d z(n1.z, n2.z, n3.z);
  Point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
  return Point3d(triple(d, y, z),
                 triple(x, d, z),
                 triple(x, y, d)) / triple(n1, n2, n3);
}
struct Sphere{
  ftype x, y, z, r;
```

```
  Sphere(){}
  Sphere(ftype x, ftype y, ftype z,  ftype r):x(x), y(y), z(z), r(r){}
};
//Minimum enclosing Sphere, O(n*70000)
//It is also possible to do with ternary search in the 3 dimensions
Sphere minimumSphere(vector<Point3d> vp){
  Point3d ans(0, 0, 0);
  int n = vp.size();
  for(Point3d p: vp)
    ans = ans + p;
  ans = ans/n;
  double P = 0.1;
  double d = 0, e = 0;
  for(int i = 0; i < 70000; i++){
    int f = 0;
    d = dist(ans, vp[0]);
    for (int j = 1; j < n; j++) {
      e = dist(ans, vp[j]);
      if (d < e) {
        d = e;
        f = j;
      }
    }
    ans = ans + (vp[f]-ans)*P;
    P *= 0.998;
  }
  return Sphere(ans.x, ans.y, ans.z, d);
}
```

# 6 String Algorithms

## 6.1 Aho Corasick

```
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
const int K = 26;
inline int getID(char c){
  return c-'a';
}
namespace Aho{
  struct Vertex {
    int next[K], go[K];
    int leaf = -1; // CAUTION with repeated strings!
    int p = -1, sz, match=-1;
    char pch;
    int suff_link = -1;
    int end_link = -1;
    Vertex(int p1=-1, char ch1='$', int sz1=0) : p(p1), pch(ch1){
      fill(begin(next), end(next), -1);
      fill(begin(go), end(go), -1);
      sz = sz1;
    }
  };
  vector<Vertex> trie;
  void init(){
    trie.clear();
    trie.emplace_back();
  }
  int add_string(string const& s, int id=1) {
    int v = 0;
    for (char ch : s) {
      int c = getID(ch);
      if (trie[v].next[c] == -1) {
        trie[v].next[c] = trie.size();
        trie.emplace_back(v, ch, trie[v].sz+1);
      }
      v = trie[v].next[c];
    }
    trie[v].leaf = id;
    return v;
  }
  int go(int v, char ch);
  int get_suff_link(int v) {
    if (trie[v].suff_link == -1) {
      if (v == 0 || trie[v].p == 0)
        trie[v].suff_link = 0;
      else
        trie[v].suff_link = go(get_suff_link(trie[v].p), trie[v].pch);
    }
    return trie[v].suff_link;
  }
  int get_end_link(int v) {
    if (trie[v].end_link == -1) {
      if (v == 0 || trie[v].p == 0){
        trie[v].end_link = 0;
      }else{
        int suff_link = get_suff_link(v);
        if(trie[suff_link].leaf != -1)
          trie[v].end_link = suff_link;
        else
          trie[v].end_link = get_end_link(suff_link);
      }
    }
    return trie[v].end_link;
  }
  int go(int v, char ch) {
    int c = getID(ch);
    if (trie[v].go[c] == -1) {
      if (trie[v].next[c] != -1)
        trie[v].go[c] = trie[v].next[c];
      else
        trie[v].go[c] = (v == 0) ? 0 : go(get_suff_link(v), ch);
    }
    return trie[v].go[c];
  }
};
//Aplication:
typedef pair<int, int> pii;
void addMatch(vector<pii> &ans, int v, int i){
  // This runs at most sqrt(N) times:1+2+3+4+..+sqrt(N)=N
  while(v != 0){
    // The string id is Aho::trie[v].leaf
    ans.emplace_back(i - Aho::trie[v].sz + 1, i);
    v = Aho::get_end_link(v);
  }
}
//Get match positions: O(answer) = O(N * sqrt(N))
```

```
vector<pii> whatMatch(string t){
  int state = 0;
  int i=0;
  vector<pii> ans;
  for(char c : t){
    state = Aho::go(state, c);
    if(Aho::trie[state].leaf != -1)
      addMatch(ans, state, i);
    else
      addMatch(ans, Aho::get_end_link(state), i);
    i++;
  }
  sort(ans.begin(), ans.end());
  return ans;
}

int countMatch(int v){
  if(Aho::trie[v].match == -1) {
    if (v == 0 || Aho::trie[v].p == 0){
      if(Aho::trie[v].leaf != -1)
        Aho::trie[v].match = 1;
      else
        Aho::trie[v].match = 0;
    }else{
      if(Aho::trie[v].leaf != -1)
        Aho::trie[v].match = 1 + countMatch(Aho::get_end_link(v));
      else
        Aho::trie[v].match = countMatch(Aho::get_end_link(v));
    }
  }
  return Aho::trie[v].match;
}
//Get match amount: O(t)
long long matchAmount(string t){
  int state = 0;
  long long ans = 0;
  for(char c : t){
    state = Aho::go(state, c);
    ans += countMatch(state);
  }
  return ans;
}
```

## 6.2 KMP

```
#include <bits/stdc++.h>
using namespace std;
// "abcabcd" is [0,0,0,1,2,3,0]
// "aabaaab" is [0,1,0,1,2,2,3]
vector<int> kmp(string s) {
  int n = (int)s.length();
  // pi[i] is the length of the longest proper prefix of the substring
  // s[0..i] which is also a suffix of this substring.
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i-1];
    while (j > 0 and s[i] != s[j])
      j = pi[j-1];
    if (s[i] == s[j])
      j++;
    pi[i] = j;
  }
  return pi;
}
//The ans[i] count the amount of occurrence of the prefix s[0..i] in s
vector<int> prefixOccurrences(string &s){
  auto pi = kmp(s);
  int n = pi.size();
  vector<int> ans(n + 1);
  for (int i = 0; i < n; i++)
    ans[pi[i]]++;
  for (int i = n-1; i > 0; i--)
    ans[pi[i-1]] += ans[i];
  for (int i = 1; i <= n; i++)
    ans[i-1] = ans[i] + 1;
  ans.pop_back();
  return ans;
}
int K = 26;
inline int getID(char c){
  return c-'a';
}
vector<vector<int>> computeAutomaton(string s) {
  s += '#';
  int n = s.size();
  vector<int> pi = kmp(s);
  vector<vector<int>> aut(n, vector<int>(26));
  for(int i = 0; i < n; i++){
    for(int c = 0; c < K; c++){
      if(i > 0 and c != getID(s[i]))
        aut[i][c] = aut[pi[i-1]][c];
      else
        aut[i][c] = i + (c == getID(s[i]));
    }
  }
  return aut;
}
```

## 6.3 Manacher

```
#include <bits/stdc++.h>
using namespace std;
// source: https://github.com/brunomaletta/Biblioteca/blob/master/
    Codigo/Strings/manacher.cpp
// ret[2*i] = larger size palindrome centered on i
// ret[2*i+1] = larger size palindrome centered on i and i + 1
vector<int> manacher(const string &s) {
  int l = 0, r = -1, n = s.size();
  vector<int> d1(n), d2(n);
  for (int i = 0; i < n; i++) {
    int k = i > r ? 1 : min(d1[l+r-i], r-i);
    while (i+k < n && i-k >= 0 && s[i+k] == s[i-k]) k++;
    d1[i] = k--;
    if (i+k > r) l = i-k, r = i+k;
  }
  l = 0, r = -1;
  for (int i = 0; i < n; i++) {
    int k = i > r ? 0 : min(d2[l+r-i+1], r-i+1); k++;
    while (i+k <= n && i-k >= 0 && s[i+k-1] == s[i-k]) k++;
    d2[i] = --k;
```

```
    if (i+k-1 > r) l = i-k, r = i+k-1;
  }
  vector<int> ret(2*n-1);
  for (int i = 0; i < n; i++) ret[2*i] = 2*d1[i]-1;
  for (int i = 0; i < n-1; i++) ret[2*i+1] = 2*d2[i+1];
  return ret;
}
struct Palindrome {
  vector<int> man;
  Palindrome(const string &s) : man(manacher(s)) {}
  bool isPalindrome(int i, int j) {
    return man[i+j] >= j-i+1;
  }
};
```

## 6.4 Min Cyclic String

```
#include <bits/stdc++.h>
using namespace std;
string min_cyclic_string(string s){
  s += s;
  int n = s.size();
  int i = 0, ans = 0;
  while (i < n / 2){
    ans = i;
    int j = i + 1, k = i;
    while (j < n && s[k] <= s[j]){
      if (s[k] < s[j])
        k = i;
      else
        k++;
      j++;
    }
    while (i <= k)
      i += j - k;
  }
  return s.substr(ans, n / 2);
}
```

## 6.5 String Hashing

```
#include <bits/stdc++.h>
using namespace std;
struct StringHashing{
  const uint64_t MOD = (1ll<<61) - 1;
  const int base = 31;
  const uint64_t invBase = 2231460976658413501uLL; //31^-1 % MOD
  uint64_t modMul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (uint32_t)b, h2 = b
        >>32;
    uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
    uint64_t ret = (l&MOD) + (l>>61) + (h << 3) + (m >> 29) + ((m <<
        35) >> 3) + 1;
    ret = (ret & MOD) + (ret>>61);
    ret = (ret & MOD) + (ret>>61);
    return ret-1;
  }
  int getInt(char c){
    return c-'a'+1;
  }
  vector<uint64_t> pre, p, inv;
//Public:
  StringHashing(string s){
    int n = s.size();
    pre.resize(n);
    p.resize(n);
    inv.resize(n);
    p[0] = 1;
    inv[0] = 1;
    pre[0] = getInt(s[0]);
    for(int i=1; i<n; i++){
      p[i] = modMul(p[i-1], base);
      inv[i] = modMul(inv[i-1], invBase);
      pre[i] = (pre[i-1] + modMul(p[i], getInt(s[i])))%MOD;
    }
  }
  uint64_t getValue(int i, int j){
    return modMul(inv[i], (pre[j] - ((i>0)?pre[i-1]:0) + MOD)%MOD);
  }
};
```

## 6.6 Suffix Automaton

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct SuffixAutomaton{
  struct state{
    int len, link, first_pos;
    bool is_clone = false;
    map<char, int> next;
  };
  vector<state> st;
  int sz, last;
  SuffixAutomaton(string s){
    st.resize(2 * s.size() + 10);
    st[0].len = 0;
    st[0].link = -1;
    st[0].is_clone = false;
    sz = 1;
    last = 0;
    for (char c : s)
      insert(c);
    preCompute();
  }
  void insert(char c){
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    st[cur].first_pos = st[cur].len - 1;
    st[cur].is_clone = false;
    int p = last;
    while (p != -1 && !st[p].next.count(c)){
      st[p].next[c] = cur;
      p = st[p].link;
    }
    if (p == -1){
      st[cur].link = 0;
    }else{
```

```
    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len){
      st[cur].link = q;
    }else{
      int clone = sz++;
      st[clone].len = st[p].len + 1;
      st[clone].next = st[q].next;
      st[clone].link = st[q].link;
      st[clone].first_pos = st[q].first_pos;
      st[clone].is_clone = true;
      while (p != -1 && st[p].next[c] == q){
        st[p].next[c] = clone;
        p = st[p].link;
      }
      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}
string lcs(string s){
  int v = 0, l = 0, best = 0, bestpos = 0;
  for (int i = 0; i < (int)s.size(); i++){
    while (v and !st[v].next.count(s[i])){
      v = st[v].link;
      l = st[v].len;
    }
    if (st[v].next.count(s[i])){
      v = st[v].next[s[i]];
      l++;
    }
    if (l > best){
      best = l;
      bestpos = i;
    }
  }
  return s.substr(bestpos - best + 1, best);
}
vector<ll> dp;
vector<int> cnt;
ll dfsPre(int s){
  if (dp[s] != -1)
    return dp[s];
  dp[s] = cnt[s]; //Accepts repeated substrings
  //dp[s] = 1; //Does not accept repeated substrings
  for (auto p : st[s].next)
    dp[s] += dfsPre(p.second);
  return dp[s];
}
void preCompute(){
  cnt.assign(sz, 0);
  vector<pair<int, int>> v(sz);
  for (int i = 0; i < sz; i++){
    cnt[i] = !st[i].is_clone;
    v[i] = make_pair(st[i].len, i);
  }
  sort(v.begin(), v.end(), greater<pair<int, int>>());
  for (int i = 0; i < sz - 1; i++)
    cnt[st[v[i].second].link] += cnt[v[i].second];
  dp.assign(sz, -1);
  dfsPre(0);
}
```

## 6.7 Suffix Array

```
};
```

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
vector<int> sort_cyclic_shifts(string const& s) {
  int n = s.size();
  const int alphabet = 256;
  vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
  for(int i = 0; i < n; i++)
    cnt[s[i]]++;
  for(int i = 1; i < alphabet; i++)
    cnt[i] += cnt[i-1];
  for(int i = 0; i < n; i++)
    p[--cnt[s[i]]] = i;
  c[p[0]] = 0;
  int classes = 1;
  for(int i = 1; i < n; i++) {
    if(s[p[i]] != s[p[i-1]])
      classes++;
    c[p[i]] = classes - 1;
  }
  vector<int> pn(n), cn(n);
  for(int h = 0; (1 << h) < n; ++h) {
    //Ordenando pelo second no RadixSort
    int h2 = (1 << h);
    for(int i = 0; i < n; i++){
      pn[i] = p[i] - h2;
      if(pn[i] < 0) pn[i] += n;
    }
    fill(cnt.begin(), cnt.begin() + classes, 0);
    for(int i = 0; i < n; i++)
      cnt[c[p[i]]]++;
    for(int i = 1; i < classes; i++)
      cnt[i] += cnt[i-1];
    for(int i = n-1; i >= 0; i--)
      p[--cnt[c[pn[i]]]] = pn[i];
    cn[p[0]] = 0;
    classes = 1;
    for(int i = 1; i < n; i++){
      pii cur(c[p[i]], c[(p[i] + h2) % n]);
      pii prev(c[p[i-1]], c[(p[i-1] + h2) % n]);
      if(cur != prev)
        ++classes;
      cn[p[i]] = classes - 1;
    }
    c.swap(cn);
  }
  return p;
}
// O(N*log(N))
vector<int> sa_construction(string s) {
  s += "$";
  vector<int> sorted_shifts = sort_cyclic_shifts(s);
  sorted_shifts.erase(sorted_shifts.begin());
  return sorted_shifts;
}
// Kasai's algorithm: O(N)
```

```
vector<int> lcp_construction(string const& s, vector<int> const& suf)
    {
  int n = s.size();
  vector<int> rank(n, 0);
  for(int i = 0; i < n; i++)
    rank[suf[i]] = i;
  int k = 0;
  vector<int> lcp(n-1, 0);
  for(int i = 0; i < n; i++){
    if (rank[i] == n - 1) {
      k = 0; continue;
    }
    int j = suf[rank[i] + 1];
    while (i + k < n && j + k < n && s[i+k] == s[j+k])
      k++;
    lcp[rank[i]] = k;
    if (k) k--;
  }
  return lcp;
}
```

## 6.8 Suffix Tree

```
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
namespace SuffixTree {
const int NS = 60; //Number of strings
const int MAXN = 100010; //Number of letters
int cn, cd, ns, en = 1, lst;
string S[NS]; int lastS = -1;
/* sufn[si][i] no do sufixo S[si][i...] */
vector<int> sufn[NS];
struct Node {
  int l, r, si=0;
  int p, suf=0;
  map<char, int> adj;
  Node() : l(0), r(-1){ suf = p = 0; }
  Node(int l1, int r1, int s1, int p1) : l(l1), r(r1), si(s1), p(p1)
      {}
  inline int len() { return r - l + 1; }
  inline int operator[](int i) { return S[si][l + i]; }
  inline int& operator()(char c) { return adj[c]; }
};
Node t[2*MAXN];
inline int new_node(int l, int r, int s, int p) {
  t[en] = Node(l, r, s, p);
  return en++;
}
void init(){
  t[0] = Node();
  cn=0, cd=0, ns=0, en = 1, lst=0;
  lastS = -1;
}
//The strings are inserted independently
void add_string(string s, char id='$') {
  assert(id < 'A');
  s += id;
  S[++lastS] = s;
  sufn[lastS].resize(s.size() + 1);
  cn = cd = 0;
  int i = 0; const int n = s.size();
  for(int j = 0; j < n; j++){
    for(; i <= j; i++) {
      if(cd == t[cn].len() && t[cn](s[j]))
        cn = t[cn](s[j]), cd = 0;
      if(cd < t[cn].len() && t[cn][cd] == s[j]) {
        cd++;
        if(j < (int)s.size() - 1) break;
        else {
          if(i) t[lst].suf = cn;
          for(; i <= j; i++) {
            sufn[lastS][i] = cn;
            cn = t[cn].suf;
          }
        }
      } else if(cd == t[cn].len()) {
        sufn[lastS][i] = en;
        if(i) t[lst].suf = en;
        lst = en;
        t[cn](s[j]) = new_node(j, n - 1, lastS, cn);
        cn = t[cn].suf;
        cd = t[cn].len();
      } else {
        int mid = new_node(t[cn].l, t[cn].l + cd - 1, t[cn].si, t[cn].
            p);
        t[t[cn].p](t[cn][0]) = mid;
        if(ns) t[ns].suf = mid;
        if(i) t[lst].suf = en;
        lst = en;
        sufn[lastS][i] = en;
        t[mid](s[j]) = new_node(j, n - 1, lastS, mid);
        t[mid](t[cn][cd]) = cn;
        t[cn].p = mid; t[cn].l += cd;
        cn = t[mid].p;
        int g = cn? j - cd : i + 1;
        cn = t[cn].suf;
        while(g < j && g + t[t[cn](S[lastS][g])].len() <= j)
          cn = t[cn](S[lastS][g]), g += t[cn].len();
        if(g == j)
          ns = 0, t[mid].suf = cn, cd = t[cn].len();
        else
          ns = mid, cn = t[cn](S[lastS][g]), cd = j - g;
      }
    }
  }
}
bool match(string &s, int i=0, int no=0, int iEdge=0){
  if(i == (int)s.size())
    return true;
  if(iEdge == t[no].len()){ //I arrived at the Node
    if(t[no].adj.count(s[i]))
      return match(s, i+1, t[no].adj[s[i]], 1);
    else
      return false;
  }
  if(t[no][iEdge] == s[i])
    return match(s, i+1, no, iEdge+1);
  return false;
}
};
```

## 6.9 Trie

```
#include <bits/stdc++.h>
using namespace std;
const int K = 26;
inline int getId(char c){
  return c - 'a';
}
struct Vertex {
  int next[K];
  int leaf;
  int count;
  Vertex() {
    fill(begin(next), end(next), -1);
    leaf = 0;
    count = 0;
  }
};
struct Trie{
  vector<Vertex> trie;
  Trie(){
    trie.emplace_back();
  }
  void add(string const& s) {
    int v = 0;
    trie[v].count++;
    for(char ch: s) {
      int c = getId(ch);
      if (trie[v].next[c] == -1) {
        trie[v].next[c] = trie.size();
        trie.emplace_back();
      }
      v = trie[v].next[c];
      trie[v].count++;
    }
    trie[v].leaf++;
  }
  int countStr(string const& s) {
    int v = 0;
    for (char ch : s) {
      int c = getId(ch);
      if (trie[v].next[c] == -1)
        return 0;
      v = trie[v].next[c];
    }
    return trie[v].leaf;
  }
  int countPre(string const& s) {
    int v = 0;
    for (char ch : s) {
      int c = getId(ch);
      if (trie[v].next[c] == -1)
        return 0;
      v = trie[v].next[c];
    }
    return trie[v].count;
  }
  bool remove(string const& s) {
    vector<int> rm;
    int v = 0;
    rm.push_back(v);
    for(char ch: s) {
      int c = getId(ch);
      if (trie[v].next[c] == -1)
        return false;
      v = trie[v].next[c];
      rm.push_back(v);
    }
    if(trie[v].leaf > 0){
      trie[v].leaf--;
      for(int x: rm)
        trie[x].count--;
      return true;
    }
    return false;
  }
};
```

## 6.10 Z Function

```
#include <bits/stdc++.h>
using namespace std;
// z[i] is the length of the longest common prefix between s[0..(n-1)]
     and the suffix of s[i..(n-1)].
// z[0] is generally not well defined.
// "aaabaab" - [0,2,1,0,2,1,0]
// "abacaba" - [0,0,1,0,3,0,1]
vector<int> z_function(string s) {
  int n = (int) s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; i++){
    if (i <= r)
      z[i] = min (r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
      z[i]++;
    if (i + z[i] - 1 > r)
      l = i, r = i + z[i] - 1;
  }
  return z;
}
```

# 7 Miscellaneous

## 7.1 Counting Inversions

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF = 0x3f3f3f3f;
// Counting Inversions: O(N*log(N))
ll ci(vector<int> &v){
  int n = v.size();
  ll inv = 0LL;
  if(n==1)
    return 0;
  vector<int> u1, u2;
  for(int i=0; i < n/2; i++)
```

```
    u1.push_back(v[i]);
  for(int i=n/2; i < n; i++)
    u2.push_back(v[i]);
  inv += ci(u1);
  inv += ci(u2);
  u1.push_back(INF);
  u2.push_back(INF);
  int ini1=0, ini2=0;
  for(int i=0; i < n; i++){
    if(u1[ini1] <= u2[ini2]){
      v[i] = u1[ini1++];
    }else{
      v[i] = u2[ini2++];
      inv += u1.size() - ini1 - 1;
    }
  }
  return inv;
}
```

## 7.2 Histogram

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
// Largest Rectangular Area in a Histogram
ll histogram(vector<int> v){
  int n = v.size();
  v.push_back(0);
  ll ans = 0;
  stack<int> st;
  for(int i = 0; i<=n; i++){
    while(st.size() && v[st.top()] >= v[i]){
      int idx = st.top(); st.pop();
      int L = st.size() ? st.top() : -1;
      ans = max(ans, (i-L-1) * (ll)v[idx]);
    }
    st.push(i);
  }
  return ans;
}
```

## 7.3 Identify Pattern

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pii;
// Return the pattern of vector in O(N): pair<cycle start, cycle size>
pii identifyPattern(vector<int> v){
  int n = v.size();
  reverse(v.begin(), v.end());
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i-1];
    while (j > 0 and v[i] != v[j])
      j = pi[j-1];
    if (v[i] == v[j])
      j++;
    pi[i] = j;
  }
  tuple<int, int, int> ans(n, 1, n-1);
  for(int i=1; i<=n; i++){
    int p = i - pi[i-1];
    if(p == 0)
      continue;
    int idx = n-i;
    ans = min(ans, {idx+p, p, idx});
  }
  auto [sum, p, idx] = ans;
  return pii(idx, p);
}
```

## 7.4 Kadane 1D and 2D

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
// Largest Sum Contiguous Subarray: O(N)
ll kadane(vector<ll> &v){
  ll ans = 0, bigger = 0;
  for(int i=0; i < (int)v.size(); i++){
    bigger = max(0LL, bigger + v[i]);
    ans = max(ans, bigger);
  }
  return ans;
}
// Largest Sum Submatrix: O(N^3)
ll kadane2d(vector<vector<int>> &mat){
  if(mat.size() == 0) return 0;
  int n = mat.size(), m = mat[0].size();
  ll ans = 0;
  vector<ll> v(m);
  for(int a=0; a<n; a++){
    fill(v.begin(), v.end(), 0);
    for(int b=a; b<n; b++){
      for(int k=0; k<m; k++)
        v[k] += mat[b][k];
      ans = max(ans, kadane(v));
    }
  }
  return ans;
}
ll circularKadane(vector<ll> v){
  ll ans1 = kadane(v);
  ll sum = 0;
  for(int i=0; i < (int)v.size(); i++){
    sum += v[i];
    v[i] = -v[i];
  }
  return max(ans1, sum + kadane(v));
}
```

## 7.5 Longest Increasing Subsequence

```
#include <bits/stdc++.h>
using namespace std;
vector<int> lis(vector<int> &v){
```

```
  vector<int> st, ans;
  vector<int> pos(v.size()+1), dad(v.size()+1);
  for(int i=0; i < (int)v.size(); i++){
    auto it = lower_bound(st.begin(), st.end(), v[i]); // Do not
        accept repeated values
    //auto it = upper_bound(st.begin(), st.end(), v[i]); //Accept
        repeated values
    int p = it-st.begin();
    if(it==st.end())
      st.push_back(v[i]);
    else
      *it = v[i];
    pos[p] = i;
    dad[i] = (p==0)? -1 : pos[p-1];
  }
  int p = pos[st.size() - 1];
  while(p >= 0){
    ans.push_back(v[p]);
    p=dad[p];
  }
  reverse(ans.begin(), ans.end());
  return ans;
}
```

## 7.6 Mo Algorithm

```
#include <bits/stdc++.h>
using namespace std;
const int BLOCK_SIZE = 700;
void remove(int idx);
void add(int idx);
void clearAnswer();
int getAnswer();
struct Query{
  int l, r, idx;
  bool operator<(Query other) const{
    if (l / BLOCK_SIZE != other.l / BLOCK_SIZE)
      return l < other.l;
    return (l / BLOCK_SIZE & 1) ? (r < other.r) : (r > other.r);
  }
};
vector<int> mo_s_algorithm(vector<Query> queries){
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());
  clearAnswer();
  int L = 0, R = 0;
  add(0);
  for(Query q : queries){
    while(q.l < L) add(--L);
    while(R < q.r) add(++R);
    while(L < q.l) remove(L++);
    while(q.r < R) remove(R--);
    answers[q.idx] = getAnswer();
  }
  return answers;
}
```

## 7.7 Polyominoes

```
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
const int MAXP = 10;
typedef pair<int, int> pii;
//This implementation considers the rotations as distinct
//              0, 10, 10+9, 10+9+8...
int pos[11] = {0, 10, 19, 27, 34, 40, 45, 49, 52, 54, 55};
struct Polyominoes{
  pii v[MAXP];
  int64_t id;
  int n;
  Polyominoes(){
    n = 1;
    v[0] = {0, 0};
    normalize();
  }
  pii& operator[](int i){
    return v[i];
  }
  bool add(int a, int b){
    for(int i=0; i<n; i++)
      if(v[i].F == a and v[i].S == b)
        return false;
    v[n++] = pii(a, b);
    normalize();
    return true;
  }
  void normalize(){
    int mnx=100, mny=100;
    for(int i=0; i<n; i++)
      mnx = min(mnx, v[i].F), mny = min(mny, v[i].S);
    id = 0;
    for(int i=0; i<n; i++){
      v[i].F -= mnx, v[i].S -= mny;
      id |= (1LL<<(pos[v[i].F] + v[i].S));
    }
  }
};
vector<Polyominoes> polyominoes[MAXP+1];
int dx[] = {0, 0, -1, 1};
int dy[] = {-1, 1, 0, 0};
void buildPolyominoes(int mxN=10){
  for(int i=0; i<=mxN; i++)
    polyominoes[i].clear();
  Polyominoes init;
  queue<Polyominoes> q;
  unordered_set<int64_t> used;
  q.push(init);
  used.insert(init.id);
  while(!q.empty()){
    Polyominoes u = q.front(); q.pop();
    polyominoes[u.n].push_back(u);
    if(u.n == mxN)
      continue;
    for(int i=0; i<u.n; i++){
      for(int j=0; j<4; j++){
```

```
        Polyominoes to = u;
        bool ok = to.add(to[i].F + dx[j], to[i].S + dy[j]);
        if(ok and !used.count(to.id)){
          q.push(to);
          used.insert(to.id);
        }
      }
    }
  }
}
```

## 7.8 Scheduling Jobs

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct Job {
  int t, c, idx;
  Job(int t1=0, int c1=0, int i=0):t(t1), c(c1), idx(i){}
};
//Penalty functions fi(t) = c[i]*t
bool cmp1(Job a, Job b){
  return a.c*(ll)b.t > b.c*(ll)a.t;
}
//Penalty functions fi(t) = c[i]*e^(alfa*t)
const double alfa = 2;
const double EPS = 1e-9;
bool cmp2(Job a, Job b){
  return (1 - exp(alfa*a.t))/a.c > (1 - exp(alfa*b.t))/b.c + EPS;
}
```

## 7.9 Sprague Grundy

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1010;
int version;
int used[MAXN];
int mex() {
  for(int i=0; ; ++i)
    if(used[i] != version)
      return i;
}
int g[MAXN];
// Can remove 1, 2 and 3
void grundy(){
  //Base case depends on the problem
  g[0] = 0;
  g[1] = 1;
  g[2] = 2;
  //Inductive case
  for(int i=3; i<MAXN; i++){
    version++;
    used[g[i-1]] = version;
    used[g[i-2]] = version;
    used[g[i-3]] = version;
    g[i] = mex();
  }
}
string solve(vector<int> v){
  grundy();
  int ans = 0;
  for(int x: v)
    ans ^= g[x];
  return ((ans != 0) ? "First" : "Second");
}
```

# 8 Theorems and Formulas

## 8.1 Binomial Coefficients

$(a+b)^n = \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \cdots + \binom{n}{k}a^{n-k}b^k + \cdots + \binom{n}{n}b^n$

Pascal's Triangle: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

Symmetry rule: $\binom{n}{k} = \binom{n}{n-k}$

Factoring in: $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$

Sum over $k$: $\sum_{k=0}^{n}\binom{n}{k} = 2^n$

Sum over $n$: $\sum_{m=0}^{n}\binom{m}{k} = \binom{n+1}{k+1}$

Sum over $n$ and $k$: $\sum_{k=0}^{m}\binom{n+k}{k} = \binom{n+m+1}{m}$

Sum of the squares: $\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$

Weighted sum: $1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n} = n2^{n-1}$

Connection with the Fibonacci numbers: $\binom{n}{0} + \binom{n-1}{1} + \cdots + \binom{n-k}{k} + \cdots + \binom{0}{n} = F_{n+1}$

More formulas: $\sum_{k=0}^{m}(-1)^k \cdot \binom{n}{k} = (-1)^m \cdot \binom{n-1}{m}$

## 8.2 Catalan Number

Recursive formula: $C_0 = C_1 = 1$

$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}, n \geq 2$

Analytical formula: $C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}, n \geq 0$

The first few numbers Catalan numbers, $C_n$ (starting from zero): $1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots$

The Catalan number $C_n$ is the solution for:

- Number of correct bracket sequence consisting of $n$ opening and $n$ closing brackets.
- The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- The number of ways to completely parenthesize $n+1$ factors.
- The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).

- The number of ways to connect the $2n$ points on a circle to form $n$ disjoint chords.
- The number of non-isomorphic full binary trees with $n$ internal nodes (i.e. nodes having at least one son).
- The number of monotonic lattice paths from point $(0, 0)$ to point $(n, n)$ in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting $(0, 0)$ to $(n, n)$).
- Number of permutations of length $n$ that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index $i < j < k$, such that $a_k < a_i < a_j$ ).
- The number of non-crossing partitions of a set of $n$ elements.
- The number of ways to cover the ladder $1 \dots n$ using $n$ rectangles (The ladder consists of $n$ columns, where $i^{th}$ column has a height $i$).

---

## 8.3 Euler's Totient

If p is a prime number: $\phi(p) = p - 1$ and $\phi(p^k) = p^k - p^{k-1}$

If a and b are relatively prime, then: $\phi(ab) = \phi(a) \cdot \phi(b)$

In general: $\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{gcd(a,b)}{\phi(gcd(a,b))}$

This interesting property was established by Gauss: $\sum_{d|n} \phi(d) = n$, Here the sum is over all positive divisors d of n.

Euler's theorem: $a^{\phi(m)} \equiv 1 \pmod{m}$, if a and m are relatively prime.

Generalization: $a^n \equiv a^{\phi(m)+[n \bmod \phi(m)]} \bmod m$, for arbitrary a, m and n $\geq log_2(m)$.

---

## 8.4 Formulas

Count the number of ways to partition a set of $n$ labelled objects into $k$ nonempty labelled subsets.

$$f(n,k) = \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$$

Stirling Number 2nd: Partitions of an $n$ element set into $k$ not-empty set. Or count the number of ways to partition a set of $n$ labelled objects into $k$ nonempty unlabelled subsets.

$$S_{2nd}(n,k) = \left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^n$$

Euler's formula: $f = e - v + 2$

Euler's formula to $n$ Lines or Segment if there is no three lines/segments that contains the same point: $R = intersects + component - n$

Number of regions in a planar graph: $R = E - V + C + 1$ where C is the number of connected components

Given $a$ and $b$ co-prime, $n = a \cdot x + b \cdot y$ where $x \geq 0$ and $y \geq 0$. You are required to find the least value of n, such that all currency values greater than or equal to $n$ can be made using any number of coins of denomination $a$ and $b$: $n = (a-1) * (b-1)$

generalization of the above problem, $n$ is multiple of $gcd(a, b)$: $n = lcm(a, b) - a - b + gcd(a, b)$

---

## 8.5 Graph

---

## 8.6 Manhattan Distance

Transformation of the manhattan distance to 2 dimensions between $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$:

$|x_1 - x_2| + |y_1 - y_2| = max(|A_1 - B_1|, |A_2 - B_2|)$ where $A = (x_1 + y_1, x_1 - y_1)$ e $B = (x_2 + y_2, x_2 - y_2)$

Transformation of the manhattan distance to 3 dimensions between $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$:

$|x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2| = max(|A_1 - B_1|, |A_2 - B_2|, |A_3 - B_3|, |A_4 - B_4|)$ where $A = (x_1 + y_1 + z_1, x_1 + y_1 - z_1, x_1 - y_1 + z_1, -x_1 + y_1 + z_1)$ e $B = (x_2 + y_2 + z_2, x_2 + y_2 - z_2, x_2 - y_2 + z_2, -x_2 + y_2 + z_2)$

Transformation of the manhattan distance to D dimensions between $P_1$ and $P_2$:

$isSet(i, x) = 1$ if the i-th bit is setted in $x$ and 0 otherwise.

$A[i] = \sum_{j=0}^{d-1} (-1)^{isSet(j,i)} P_1[j]$

$B[i] = \sum_{j=0}^{d-1} (-1)^{isSet(j,i)} P_2[j]$

$$\sum_{i=0}^{d-1} |P_1[i] - P_2[i]| = \max_{i=0}^{2^d-1} |A_i - B_i|$$

---

## 8.7 Primes

If $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, then:

Number of divisors is $d(n) = (e_1 + 1) \cdot (e_2 + 1) \cdots (e_k + 1)$.

Sum of divisors is $\sigma(n) = \frac{p_1^{e_1+1}-1}{p_1-1} \cdot \frac{p_2^{e_2+1}-1}{p_2-1} \cdots \frac{p_k^{e_k+1}-1}{p_k-1}$