paguso fans - ICPC Library

Contents

1 Template

1.1 Template

```cpp
#include<bits/stdc++.h>
using namespace std;
#define ll long long int
#define pii pair<int, int>
#define br "\n"
#define PB push_back
#define X first
#define Y second
#define io ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
#define debug_vec(v,n) for(int i = 0; i < n; i++) {cout << v[i] << " ";}
    cout << br;
#define debug_pair(p) {cout << "(" << p.X << ", " << p.Y << ")" << br;}
const bool multitest = 1;
// void prep(){
// }
void solve(){

}
int main(){
        // freopen("input.in", "r", stdin);
        // freopen("output.out", "w", stdout);
        io;
        int t = 1;
        if(multitest) cin >> t;
        // prep();
        while(t--){
                solve();
        }
}
```

2 Data Structures

2.1 Fenwick

```cpp
// Fenwick Tree
const int MX = 2e5+5;
vector<ll> BIT(MX+1,0);
int n, q;

ll qry(int i){ // [1,i] 1 indexado
        ll ret = 0;
        for(; i > 0; i -= i & -i){
                ret += BIT[i];
        }
        return ret;
}

ll qryRange(int l, int r){ // 1 indexado inclusivo
        ll qr = qry(r);
        ll ql = qry(l-1);
        return qr-ql;
}
void increment(ll i, ll v){ // 1 indexado (+= v)
        for(; i <= n; i += i & -i){
                BIT[i] += v;
        }
}
void build(const vector<ll>& nums){
        for(int i = 0; i < nums.size(); i++){
                increment(i+1,nums[i]);
        }
}
```

2.2 Segtree Lazy

```cpp
// Lazy SegTree
const int MX = 2e5+5;
vector<ll> seg(4*mx);
vector<ll> lazy(4*mx,0);
vector<ll> nums(mx);
int n,q;

void build(int l = 0, int r = n-1, int idx = 0){
```

```
        if(l == r){
                seg[idx] = nums[l];
                lazy[idx] = 0;
                return;
        }
        int m = (l+r)/2;
        int left = 2*idx+1;
        int right = 2*idx+2;
        build(l,m,left);
        build(m+1,r,right);
        seg[idx] = seg[left] + seg[right];
}

void prop(int l = 0, int r = n-1, int idx = 0){
        seg[idx] += (ll)(r-l+1)*lazy[idx];
        if(l != r){ // nao for folha
                int left = 2*idx+1;
                int right = 2*idx+2;
                lazy[left] += lazy[idx];
                lazy[right] += lazy[idx];
        }
        lazy[idx] = 0;
}

void update(int L, int R, ll val, int l = 0, int r = n-1, int idx = 0){
        if(R < l || L > r) return;
        prop(l,r,idx);
        if(L <= l && r <= R){
                lazy[idx] = val;
                prop(l,r,idx);
        }
        else{
                int m = (l+r)/2;
                int left = 2*idx+1;
                int right = 2*idx+2;
                update(L,R,val,l,m,left);
                update(L,R,val,m+1,r,right);
                seg[idx] = seg[left] + seg[right];
        }
}

ll query(int L, int R, int l = 0, int r = n-1, int idx = 0){
        prop(l,r,idx);
        if(R < l || L > r) return 0;
        if(L <= l && r <= R){
                return seg[idx];
        }
        int m = (l+r)/2;
        int left = 2*idx+1;
        int right = 2*idx+2;
        return query(L,R,l,m,left) + query(L,R,m+1,r,right);
}
```

2.3 Segtree Topdown

```
// SegTree
const int mx = 2e5 + 5;
vector<ll> seg(4*mx);
vector<ll> nums(mx);
int n,q;
ll merge(ll a, ll b){
        return a+b;
}

void build(int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = nums[l];
                return;
        }
        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        build(l,mid,left);
        build(mid+1,r,right);
        seg[idx] = merge(seg[left], seg[right]);
}

ll query(int L, int R, int l = 0, int r = n-1, int idx = 0){
        if(R < l || L > r) return 0; // elemento neutro
        if(L <= l && r <= R) return seg[idx];

        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        ll ql = query(L,R,l,mid,left);
        ll qr = query(L,R,mid+1,r,right);
        return merge(ql,qr);
}

void update(int pos, int num, int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = num;
                return;
        }
        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        if(pos <= mid){
                update(pos,num,l,mid,left);
        }
        else update(pos,num,mid+1,r,right);
        seg[idx] = merge(seg[left],seg[right]);
}
```

2.4 SparseTable

```
vector<vector<ll>> table;
vector<ll> lg2;
void build(int n, vector<ll> v) {
  lg2.resize(n + 1);
  lg2[1] = 0;
  for (int i = 2; i <= n; i++) {
    lg2[i] = lg2[i >> 1] + 1;
  }
  table.resize(lg2[n] + 1);
  for (int i = 0; i < lg2[n] + 1; i++) {
    table[i].resize(n + 1);
  }
  for (int i = 0; i < n; i++) {
    table[0][i] = v[i];
  }
  for (int i = 0; i < lg2[n]; i++) {
    for (int j = 0; j < n; j++) {
      if (j + (1 << i) >= n) break;
      table[i + 1][j] = min(table[i][j], table[i][j + (1 << i)]);
    }
  }
}
ll get(int l, int r) { // (l,r) inclusivo
  int k = lg2[r - l + 1];
  return min(table[k][l], table[k][r - (1 << k) + 1]);
}
```

3 DP

3.1 Knapsack

```
// Knapsack
const int MXW = 1e5+5;
const int MXN = 105;

int n, max_w;
vector<int> weight(MXN), value(MXN);
vector<vector<ll>> dp(MXN,vector<ll>(MXW,-1));

ll solveDp(int i, int k){ // k -> peso atual
        if(i == n) return 0;
        if(dp[i][k] != -1) return dp[i][k];

        ll ignore = solveDp(i+1,k);
        ll add = -1;
        if(weight[i] + k <= max_w){
                add = value[i] + solveDp(i+1, weight[i] + k);
        }
        return dp[i][k] = max(ignore,add);
}

// iterativo

ll knapsack(){
  vector<ll> dp(dpmx,0);
  for(int i = 0; i < n; i++){
    ll w = weight[i];
    ll v = value[i];
    for(int sz = max_w; sz >= w; sz--){
      dp[sz] = max(dp[sz],dp[sz-w]+v);
    }
  }
  return *max_element(begin(dp),end(dp));
}
```

3.2 Longest Increasing Sequence

```
// Longest Increasing Sequence
int lis(vector<ll>& nums){
        int n = nums.size();
        vector<ll> s;
        for(int i = 0; i < n; i++){
                auto it = lower_bound(s.begin(),s.end(),nums[i]);
                if(it == s.end()){
                        s.PB(nums[i]);
                }
                else{
                        *it = nums[i];
                }
        }
        return (int)s.size();
}
```

4 Geometry

4.1 Convex Hull

```
// O(nlogn) sorted = false
// O(n) sorted = true
// retorna todos os pontos q tao no c.h
vector<Point> convexHull(vector<Point>& pts, bool sorted = false){ // pts.
    size() >= 3
  int n = pts.size();
  if(!sorted){
    sort(begin(pts),end(pts));
  }
  vector<Point> lower(n + 1), upper(n + 1);
  int s = 0;
  for(int i = 0; i < n; i++){
    lower[s++] = pts[i];
    while(s >= 3){
      Point a = lower[s-3], b = lower[s-2], c = lower[s-1];
      Point v1 = b-a, v2 = c-b;
      if(cross(v1,v2) >= 0){
        break;
      }
      if(cross(v1,v2) < 0){ // tirar b
        lower[s-2] = lower[s-1];
        s--;
      }
    }
  }
  lower.resize(s);
  s = 0;
  for(int i = 0; i < n; i++){
    upper[s++] = pts[i];
    while(s >= 3){
      Point a = upper[s-3], b = upper[s-2], c = upper[s-1];
      Point v1 = b-a, v2 = c-b;
      if(cross(v1,v2) <= 0){
        break;
      }
      if(cross(v1,v2) > 0){ // tirar b
        upper[s-2] = upper[s-1];
        s--;
      }
    }

  }
  upper.resize(s-1);
  reverse(begin(upper),end(upper));
  upper.pop_back();
  lower.insert(end(lower),begin(upper),end(upper));
  return lower;
}
```

4.2 Point

```
const double inf = 1e100, eps = 1e-9;
const double PI = acos(-1.0L);
int cmp (double a, double b = 0) {
  if (abs(a-b) < eps) return 0;
  return (a < b) ? -1 : +1;
}
struct Point{
        double x,y;
        Point(double x = 0, double y = 0) : x(x),y(y){}
        Point(const Point& p): x(p.x), y(p.y){}
  bool operator < (const Point &p) const {
    if(cmp(x, p.x) != 0) return x < p.x;
    return cmp(y, p.y) < 0;
  }
  bool operator == (const Point &p) const {return !cmp(x, p.x) && !cmp(y, p
      .y);}
  bool operator != (const Point &p) const {return !(p == *this);}

        // basic ops
```

```
        Point operator + (const Point& p) const {return Point(x+p.x,y+p.y)
            ;}
        Point operator - (const Point& p) const {return Point(x-p.x,y-p.y)
            ;}
        Point operator * (const double k) const {return Point(x*k,y*k);}
        Point operator / (const double k) const {return Point(x/k,y/k);}
};
        // points ops
double dot (const Point& p,const Point& q) { return p.x*q.x + p.y*q.y; }
double cross (const Point& p,const Point& q) { return p.x*q.y - p.y*q.x; }
double norm(const Point& p) { return hypot(p.x,p.y); }
double dist(const Point& p, const Point& q) { return hypot(p.x-q.x,p.y-q.y)
    ; }
double dist2(const Point& p, const Point& q) { return dot(p-q,p-q); }
Point normalize(const Point &p) { return p/hypot(p.x, p.y); }
double angle (const Point& p, Point& q) { return atan2(cross(p, q), dot(p,
    q)); }
double angle (const Point& p) { return atan2(p.y, p.x); }

ostream &operator<<(ostream &os, const Point &p) {
        return os << "(" << p.x << "," << p.y << ")";
}


/*
----
---- retas
*/


struct Line{
        Point p, vd;
        Line(){}
        Line(const Point& p, const Point& vd) : p(p), vd(vd) {};
};

// pra segmento usar isso
Line createLine(const Point& p1, const Point& p2){
        return Line(p1, p2-p1);
}
bool pointInSegment(const Point& p, const Line& s){
        Point v1 = p - s.p;
        double k = dot(v1,s.vd) / dot(s.vd, s.vd);
        return 0 <= k && k <= 1;
}
bool inLine(const Point& p, const Line& l){
        return cross(p-l.p, l.vd) == 0;
}
double distPointLine(const Point& p, const Line& l){
        Point vp = p-l.p;
        return abs(cross(vp,l.vd))/norm(l.vd);
}

double distPointSegment(const Point& p, const Line& s){
        Point v1 = p - s.p;
        double k = dot(v1,s.vd) / dot(s.vd, s.vd);
        if(k < 0) return dist(p,s.p);
        if(k > 1) return dist(p,s.p + s.vd);
        return distPointLine(p,s);
}

ostream &operator<<(ostream &os, const Line &l) {
        return os << "(" << l.p.x << "," << l.p.y << ")" << "+ t(" << l.vd.
            x << "," << l.vd.y << ")";
}
```

5 ETC

5.1 Bitset

```
// Bitset operations
__builtin_popcount(int x);
__builtin_popcountll(ll x);
const int SZ = 1e6;
bitset<SZ> b;
b.reset(); // 00 ... 00
b.set();   // 11 ... 11
b.flip();
b._Find_first(); // retorna SZ se nao tiver
b._Find_next(i);
b.to_ulong();
b.to_string();
b.count();
```

5.2 Coordinate Compression

```
template<typename T> // v1w g
struct CoordinateCompression {
  vector<T> v;
  void push(const T& a) { v.push_back(a); }
  int build() {
    sort(begin(v), end(v));
    v.erase(unique(begin(v), end(v)), end(v));
    return (int)v.size();
  }
  int operator[](const T& a) const {
    auto it = lower_bound(begin(v), end(v), a);
    return int(it - begin(v));
  }
};
```

5.3 Ternary Search

```
double f(double t){
        // alguma funcao unimodal -> maximo ou minimo
        //     /\
        //    /  \
        // __/    \__
}

double tern_search(double l, double r){ // achar o maximo
        double eps = 1e-8;
        while(r - l > eps){
                double m1 = l + (r-l)/3;
                double m2 = r - (r-l)/3;
                double f1 = f(m1), f2 = f(m2);
                if(f1 < f2)     l = m1; // (m1,r)
                else r = m2; // (l,m2);
        }
        return max(f(l),f(r));
}
```

6 Graph

6.1 BFS 2D

```
// BFS 2D
int n,m;
// g : labirinto de . e #
string g[505];
pii start,finish;
vector<pii> dir = {{-1,0},{1,0},{0,-1},{0,1}};
```

```
bool inRange(pii p){
        return p.X >= 0 && p.X < n && p.Y >= 0 && p.Y < m;
}

pii operator + (pii lhs, pii rhs){
        return {lhs.X+rhs.X,lhs.Y+rhs.Y};
}

int bfs(){
        queue<pii> q;
        q.push(start);
        vector<vector<bool>> vis(505,vector<bool>(505,0));
        vis[start.X][start.Y] = 1;
        int cnt = 0;
        while(!q.empty()){
                int lvlSz = q.size();
                while(lvlSz--){
                        pii c = q.front();q.pop();
                        vector<pii> newPts;
                        for(pii currDir : dir) newPts.PB(c + currDir);
                        for(pii pt : newPts){
                                if(inRange(pt) && !vis[pt.X][pt.Y] && g[pt.
                                    X][pt.Y] != '#'){
                                    if(pt == finish) return cnt;
                                    vis[pt.X][pt.Y] = 1;
                                    q.push(pt);

                                }
                        }
                }
                cnt++;
        }
        return cnt;
}
```

6.2 Bicolorable

```
// Bicolorable
int v,e;
vector<int> adj[4005];
vector<int> clr (4005,-1);

void paint(int n){
        for(int a :  adj[n]){
                if(clr[a] == -1){
                        clr[a] = 1 - clr[n];
                        paint(a);
                }
        }
}

void paintAll(){
        for(int i = 0; i < v; i++){
                if(clr[i] == -1){
                        clr[i] = 0;
                        paint(i);
                }
        }
}

bool check(int n){
        for(int a : adj[n]){
                if(clr[a] != 1 - clr[n]) return true;
        }
        return false;
}
```

6.3 Dijkstra

```
// Dijkstra
#define pii pair<ll,ll>
const ll MXN = 1e5+5;
const ll INF = LLONG_MAX;
int v,e;
vector<pii> adj[MXN];
vector<ll> parent(MXN,-1);
vector<ll> dist(MXN,INF);

void dijkstra(ll node){
        dist[node] = 0;
        priority_queue<pii,vector<pii>, greater<pii>> pq;
        pq.push({0,node});
        while(!pq.empty()){
                auto [d,out] = pq.top();
                pq.pop();
                ll d = curr.X, out = curr.Y;
                if(d > dist[out]) continue;
                for(auto [cost, nb] : adj[out]){
                        ll currD = dist[out] + cost;
                        if(currD < dist[nb]){
                                dist[nb] = currD;
                                parent[nb] = out;
                                pq.push({currD,nb});
                        }
                }
        }
}
```

6.4 DSU

```
struct DSU{
        vector<int> p;
        vector<int> sz;
        int n;
        DSU(int nodes){
                n = nodes;
                p.resize(nodes);
                sz.resize(nodes,1);
                iota(begin(p), end(p), 0);
        }
        int size(int a){ return sz[root(a)]; }
        int root(int a){ return p[a] = (p[a] == a ? a : root(p[a])); }
        bool unite(int a, int b){
                int ra = root(a), rb = root(b);
                if(ra != rb){
                        if(sz[ra] < sz[rb]) swap(ra,rb);
                        p[rb] = ra;
                        sz[ra] += sz[rb];
                        return 1;
                }
                return 0;
        }

};
```

6.5 Floyd Warshall

```
const int inf = 0x3f3f3f3f;
int g[ms][ms], dis[ms][ms], n;

void clear() {
  memset(g, 0x3f, sizeof g);
  for(int i = 0; i < n; i++) g[i][i] = 0;
}

void add(int u, int v, int w) {
  g[u][v] = min(w, g[u][v]);
```

```
}

void floydWarshall() {
  memcpy(g, dis, sizeof g);
  for(int k = 0; k < n; k++) {
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) {
        dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
      }
    }
  }
}
```

6.6 Kosaraju

```
// Kosaraju
vector<vector<int>> G, Gt;
vector<int> id;
vector<int> order;
vector<bool> vis;
int n;

void dfs1(int v){ // ordem de saida
  vis[v] = true;
  for(int u : G[v]){
    if(!vis[u]) dfs1(u);
  }
  order.PB(v);
}
void dfs2(int v, int idx, vector<int>& component){ // pegar um componente
    todo
    vis[v] = true;
    id[v] = idx;
    component.PB(v);
    for(int u : Gt[v]){
        if(!vis[u]) dfs2(u);
    }
}
vector<vector<int>> kosaraju(){
  vector<vector<int>> components;
  vis.assign(n,false);
  for(int i = 0; i < n; i++){
    if(!vis[i]) dfs1(i);
  }
  vis.assign(n,false);
  reverse(begin(order),end(order));
  int idx = 0;
  for(int v : order){
    if(!vis[v]){
      vector<int> component;
      dfs2(v, idx++, component);
      // sort(begin(component),end(component));
      components.PB(component);
    }
  }
  return components;
}
```

6.7 Kruskal

```
int v = 1e5;
DSU dsu = DSU(v+5);
priority_queue<pair<int,pii>,vector<pair<int,pii>>, greater<pair<int,pii>>>
    pq;
for(int i = 0; i < e; i++){
        int x,y,w; cin >> x >> y >> w;
        pq.push({w,{x,y}});
}
int edges = v-1; // mst count
ll mstSum = 0;
while(edges > 0){
        pair<int,pii> curr = pq.top();pq.pop();
        int w = curr.X, x = curr.Y.X, y = curr.Y.Y;
        if(dsu.unite(x,y)){
                mstSum += w;
                edges--;
        }
}
cout << mstSum;
```

6.8 Lowest Common Ancestor (LCA)

```
int par[ms][mlg+1], lvl[ms];
void dfs(int v, int p, int l = 0) { // chamar como dfs(root, root)
  lvl[v] = l;
  par[v][0] = p;
  for(int k = 1; k <= mlg; k++) {
    par[v][k] = par[par[v][k-1]][k-1];
  }
  for(int u : g[v]) {
    if(u != p) dfs(u, v, l + 1);
  }
}
int lca(int a, int b) {
  if(lvl[b] > lvl[a]) swap(a, b);
  for(int i = mlg; i >= 0; i--) {
    if(lvl[a] - (1 << i) >= lvl[b]) a = par[a][i];
  }
  if(a == b) return a;
  for(int i = mlg; i >= 0; i--) {
    if(par[a][i] != par[b][i]) a = par[a][i], b = par[b][i];
  }
  return par[a][0];
}
```

6.9 Max Flow

```
template <class T = int>
class MCMF {
public:
  struct Edge {
    Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
    int to;
    T cap, cost;
  };
  MCMF(int size) {
    n = size;
    edges.resize(n);
    pot.assign(n, 0);
    dist.resize(n);
    visit.assign(n, false);
  }
  pair<T, T> mcmf(int src, int sink) {
    pair<T, T> ans(0, 0);
    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));
  }
private:
  int n;
  vector<vector<int>> edges;
  vector<Edge> list;
  vector<int> from;
  vector<T> dist, pot;
  vector<bool> visit;

  /*bool dij(int src, int sink) {
    T INF = std::numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    visit.assign(n, false);
    dist[src] = 0;
    for(int i = 0; i < n; i++) {
      int best = -1;
      for(int j = 0; j < n; j++) {
        if(visit[j]) continue;
        if(best == -1 || dist[best] > dist[j]) best = j;
      }
      if(dist[best] >= INF) break;
      visit[best] = true;
      for(auto e : edges[best]) {
        auto ed = list[e];
        if(ed.cap == 0) continue;
        T toDist = dist[best] + ed.cost + pot[best] - pot[ed.to];
        assert(toDist >= dist[best]);
        if(toDist < dist[ed.to]) {
          dist[ed.to] = toDist;
          from[ed.to] = e;
        }
      }
    }
    return dist[sink] < INF;
  }*/

  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 = 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];
    }
  }
};
```

6.10 Policy Based

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
ordered_set X;
X.insert(1); X.find_by_order(0); // Acha a key na ordem Y
X.order_of_key(-5); // Acha a ordem da key Y
end(X), begin(X);
```

6.11 Toposort

```
// Toposort
const int MX = 1e5+5;
vector<int> adj[MX];
vector<int> dep(MX,0); // dep[x] = quantos nodes se ligam a x
int v;

vector<int> toposort(){
        vector<int> order;
        queue<int> q;
        for(int i = 0; i < v; i++)
                if(dep[i] == 0) q.push(i);
        if(q.empty()) return order;
        while(!q.empty()){
                int f = q.front(); q.pop();
                order.PB(f);
                for(int nb : adj[f])
                        if(--dep[nb] == 0)
                                q.push(nb);
        }
        return order;
}
```

6.12 2-sat

```
#define PB push_back
// Kosaraju
struct TwoSat{
        int n;
        vector<vector<int>> G, Gt;
```

```
    vector<int> id, order, ans;
    vector<bool> vis;

    TwoSat(){}
    TwoSat(int n) : n(n){init();}
    void init(){
            G.resize(2*n);
            Gt.resize(2*n);
            id.resize(2*n);
            ans.resize(n);
    }
    void add_edge(int u, int v){
            G[u].PB(v);
            Gt[v].PB(u);
    }
    // Algum ser 1
    void add_or(int a, bool neg1, int b, bool neg2){
            // (neg1 A | neg2 B) = (!neg1 A -> neg2 B) & (!neg2 B ->
                neg1 A)
            add_edge(a + (neg1 ? 0 : n), b + (neg2 ? n : 0));
            add_edge(b + (neg2 ? 0 : n), a + (neg1 ? n : 0));
    }
    // Apenas algum ser 1
    void add_xor(int a, bool neg1, int b, bool neg2){
             add_or(a,neg1,b,neg2);
             add_or(a,!neg1,b,!neg2);
    }
    // Setar variavel a pra b
    void set(int a, bool b){ // (a|a)
            add_or(a,!b,a,!b);
    }
    // Mesmo valor
void add_xnor(int a, bool neg1, int b, bool neg2) {
    add_xor(a, !neg1, b, neg2);
}
//

    void dfs1(int v){ // ordem de saida
      vis[v] = true;
      for(int u : G[v]){
        if(!vis[u]){
          dfs1(u);
        }
      }
      order.PB(v);
    }
    void dfs2(int v, int idx){ // pegar um componente todo
        vis[v] = true;
        id[v] = idx;
        for(int u : Gt[v]){
            if(!vis[u]) dfs2(u,idx);
        }
    }
    void kosaraju(){
      vis.assign(2*n,false);
      for(int i = 0; i < 2*n; i++){
            if(!vis[i]) dfs1(i);
      }
      vis.assign(2*n,false);
      reverse(begin(order),end(order));
      int idx = 0;
      for(int v : order){
        if(!vis[v]) dfs2(v, idx++);
      }
    }
    bool satisfiable(){
            kosaraju();
            for(int i = 0; i < n; i++){
                    if(id[i] == id[i+n]) return false;
                    ans[i] = (id[i] > id[i+n]);
            }
            return true;
    }
};
```

7 Math

7.1 Extended Euclidean

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// inverso modular de a
int inv, y;
int g = gcd(a,mod,inv,y);
inv = (inv % m + m) % m;
```

7.2 Factorization

```
// Factorization
vector<pii> getFact(int n){
        vector<pii> primes;
        for(int p = 2; p*p <= n; p++){
                if(n % p == 0){
                        int exp = 0;
                        while(n % p == 0){
                                exp++;
                                n /= p;
                        }
                        primes.PB({p,exp});
                }
        }
        if(n > 1) primes.PB({n,1});
        return primes;
}
```

7.3 Fastexp

```
// Fast Exp
const ll MOD = 1e9+7;
// matriz quadrada
class Matrix{
        public:
        vector<vector<ll>> mat;
        int m;
        Matrix(int m): m(m){
                mat.resize(m);
                for(int i = 0; i < m; i++) mat[i].resize(m,0);
        }
        Matrix operator * (const Matrix& rhs){
                Matrix ans = Matrix(m);
                for(int i = 0; i < m; i++)
                        for(int j = 0; j < m; j++)
                                for(int k = 0; k < m; k++)
                                        ans.mat[i][j] = (ans.mat[i][j] + (
                                            mat[i][k] * rhs.mat[k][j]) %
                                            MOD) % MOD;
```

```
                return ans;
        }
};

Matrix fexp(Matrix a, ll n){
        int m = a.m;
        Matrix ans = Matrix(m);
        for(int i = 0; i < m; i++) ans.mat[i][i] = 1;
        while(n){
                if(n & 1) ans = ans * a;
                a = a * a;
                n >>= 1;
        }
        return ans;
}
```

7.4 Polynomial

```
template<typename T>
struct Poly {
  int n;
  vector<T> v;
  Poly(int sz) : n(sz+1) { v.resize(sz+1,0);}
  friend Poly operator*(const Poly& lhs, const Poly& rhs) {
    int grauL = (int)lhs.n - 1;
    int grauR = (int)rhs.n - 1;
    Poly ans(grauR+grauL);
    for(int i = 0; i <= grauL; ++i) {
      for(int j = 0; j <= grauR; ++j) {
        ans.v[i + j] += lhs.v[i] * rhs.v[j];
      }
    }
    return ans;
  }
  void set_identity() { // 1
    v[0] = T(1);
    for(int i = 1; i < n; ++i) {
      v[i] = T(0);
    }
  }
};
template<typename T>
Poly<T> poly_exp(Poly<T> a, long long e) {
  Poly<T> r(0);
  r.set_identity();
  for(; e > 0; e >>= 1) {
    if(e & 1) {
      r = r * a;
    }
    a = a * a;
  }
  return r;
}
```

7.5 Sieve

```
// Sieve
const int MXN = 1e6+5;
bitset<MXN> isPrime;

vector<int> sieve(){
        isPrime.set();
        vector<int> primes;
        for(int i = 2; i < MXN; i++){
                if(isPrime[i]){
                        primes.PB(i);
                        for(int j = 2*i; j < MXN; j += i){
                                isPrime[j] = 0;
                        }
                }
        }
        return primes;
}
```

8 String

8.1 KMP

```
vector<int> getBorder(string str) {
  int n = str.size();
  vector<int> border(n, -1);
  for(int i = 1, j = -1; i < n; i++) {
    while(j >= 0 && str[i] != str[j + 1]) {
      j = border[j];
    }
    if(str[i] == str[j + 1]) {
      j++;
    }
    border[i] = j;
  }
  return border;
}
int matchPattern(const string &txt, const string &pat, const vector<int> &
    border) {
  int freq = 0;
  for(int i = 0, j = -1; i < txt.size(); i++) {
    while(j >= 0 && txt[i] != pat[j + 1]) {
      j = border[j];
    }
    if(pat[j + 1] == txt[i]) {
      j++;
    }
    if(j + 1 == (int) pat.size()) {
      //found occurence
      freq++;
      j = border[j];
    }
  }
  return freq;
}
```

8.2 RabinKarp

```
// Rabin Karp
class RabinKarp{
public:
        ll base, mod, sz;
        string s;
        vector<ll> pot, has;
        RabinKarp(const string& str,ll b = 997, ll m = 1e9 + 7)
        : base(b), mod(m), s(str){
                sz = str.length();
                pot.resize(sz+1);
                has.resize(sz+1);
                build();
        }
        void build(){
                pot[0] = 1;
                has[0] = s[0];
                for(int i = 1; i < sz; i++){
                        pot[i] = (pot[i-1]*base) % mod;
                        has[i] = ((has[i-1]*base)+s[i])%mod;
                }
        }
```

```
        ll getKey(ll l, ll r){ // inclusivo
                ll ans = has[r];
                if(l > 0) ans = (((ans - pot[r-l+1]*has[l-1])%mod + mod)%
                    mod);
                return ans;
        }
};
```

8.3 Trie

```
int trie[ms][sigma], terminal[ms], z = 1;

void insert(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = p[i]-'a';
    if(!trie[cur][id]) {
      trie[cur][id] = z++;
    }
    cur = trie[cur][id];
  }
  terminal[cur]++;
}

int count(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = p[i]-'a';
    if(!trie[cur][id]) {
      return false;
    }
    cur = trie[cur][id];
  }
  return terminal[cur];
}
```