Contents

1 Template

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

using namespace std;

#define all(v) v.begin(),v.end()
#define rall(v) v.rbegin(),v.rend()
#define pb push_back
#define sz(a) int(a.size())
#define F first
#define S second
typedef long long ll;
typedef vector<int> vi;
typedef vector<ll> vll;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
const int MOD = 1e9+7;
const int INF = INT_MAX;
const ll INF64 = LLONG_MAX;
const long double EPS = 1e-9;
const long double PI = acosl(-1.0L);
void setIO(string p){
  freopen((p + ".in").c_str(), "r", stdin);
  freopen((p + ".out").c_str(), "w", stdout);
}

void solve(){

}


int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  int tc = 1;
  //cin >> tc;
  for (int t = 1; t <= tc; t++)solve();
}
```

2 Data structures

2.1 Segment Tree

```cpp
struct Node{
  ll val = 0;
};
Node operator+(Node a,Node b){return {a.val+b.val};}
class Segment_Tree{
public:
  int n;
  vector<Node> tree;

  Segment_Tree(int x = 1e5+10){
    n = x;
    tree.resize(2*n+1);
  }

  void build(vi &arr){
    for(int i=0;i<n;i++)tree[n+i] = {arr[i]};
    for(int i = n-1; i>0;i--)tree[i] = tree[i*2]+tree[i*2+1];
  }

  void update(int p, ll val){
    for(tree[p+=n] = {val};p>1;p>>=1)tree[p>>1] = tree[p]+tree[p^1];
  }

  ll query(int l,int r){
    Node s;
    for(l+=n,r+=n ; l<r ; l>>=1 , r>>=1)
      if(l&1)s = s+ tree[l++];
      if(r&1)s = s+ tree[--r];
    }
    return s.val;
  }
};
```

2.2 Merge Sort Tree

```
struct Node{
  vi val;

  int search(int x){
    int l = 0, r = sz(val)-1;
    while(l<=r){
      int m = l+(r-l)/2;
      if(val[m] < x)l =m+1;
      else r = m-1;
    }
    return sz(val)-l;
  }
};

Node operator+(Node a,Node b){
  int i=0,j=0;
  Node aux;
  while (i< sz(a.val) && j<sz(b.val)){
    if (a.val[i]< b.val[j] ) aux.val.pb(a.val[i++]);
    else aux.val.pb(b.val[j++]);
  }
  while (i<sz(a.val)) aux.val.pb(a.val[i++]);
  while (j<sz(b.val)) aux.val.pb(b.val[j++]);
  return aux;
}
//Call segment Tree
```

2.3 Fenwick Tree

```
struct Node{
  ll n = 0;
  Node operator+(Node b){return {n+b.n};}
};

class Fenwick_Tree{
public:
  vector<Node> tree;
  int n;
  Fenwick_Tree(int x){
    n = x+1;tree.resize(n);
  }

  ll sum(int r){
    r++;
    Node ans = {0};
    while(r){
      ans = ans + tree[r];
      r -= r&-r;
    }
    return ans.n;
  }

  void update(int id, Node val){
    id++;
    while(id < n){
      tree[id]  = tree[id]+val;
      id += id&-id;
    }
  }
};
```

2.4 Sparse Table

```cpp
struct Node{
  int val = 0;
  Node operator+(Node b){return {__gcd(val,b.val)};}
};

class SparseTable{
  public:
  vector<vector<Node>> sparse;

  SparseTable(int n=2e5+10){
    sparse.assign(n , vector<Node> (log2(n)+1));
  }

  void build(const vi &a){
    for(int i=0;i<sz(a);i++)
      sparse[i][0] = {a[i]};

    for(int j=1;(1<<j)<=sz(a);j++)
      for(int i=0;i+(1<<j)-1 <sz(a);i++)
        sparse[i][j]= sparse[i][j-1]+sparse[i+(1<<(j-1))][j-1];
  }

  int query(int l, int r){
    int len = r-l+1;
    int k = log2(len);
    return (sparse[l][k]+sparse[r-(1<<k)+1][k]).val;
  }

};
```

2.5 Disjoint Set Union

```cpp
class DSU{
  private:
  vi parent,rank,size;
  int c;
  public:
  int mx = 0;
  DSU(int n):parent(n+1),rank(n+1,0),size(n+1,1),c(n){
    iota(all(parent),0);
  }

  int find(int u){return parent[u] == u?u:parent[u] = find(parent[u]) ;}

  bool same(int u,int v){return find(u)==find(v);}

  int get_size(int u){return size[find(u)];}

  int count(){return c;}

  void merge(int u,int v){
    u = find(u);
    v = find(v);
    if(u!=v){
      c--;
      if(rank[u] > rank[v])swap(u,v);
      parent[u] = v;
      size[v] += size[u];
      if(rank[u] == rank[v])rank[v]++;
      mx = max(mx,size[v]);
    }
  }
};
```

2.6 Lazy SegTree

2.6.1 Min/Max Range / Single Acces Element

```
struct Node{
  ll val = 0;
};

struct LazyNode{
  ll val = 0;
  bool can(){return val!=0;}
};

Node operator+(Node a,LazyNode b){return {a.val+b.val};}
Node operator+(Node a,Node b){return {a.val+b.val};}
LazyNode operator+(LazyNode a,LazyNode b){return {a.val+b.val};}

class Segment_Tree{
private:
  int n,h;

  void apply(int p, ll val){
    tree[p].val += val;
    if(p < n)d[p].val += val;
  }

  void build_lazy(int p){
    while(p>1){
      p >>= 1;
      tree[p] = (tree[p*2]+tree[p*2+1])+d[p];
    }
  }

  void push(int p){
    for(int s = h;s>0;--s){
      int i = p>>s;
      if(d[i].can()){
        apply(2*i,d[i].val);
        apply(2*i+1,d[i].val);
        d[i].val = 0;
      }
    }
  }

public:
  vector<Node> tree;
  vector<LazyNode> d;
  Segment_Tree(int x = 1e5+10){
    n = x;
    tree.resize(2*n+1);
    d.resize(n+1);
    h = sizeof(int)*8 - __builtin_clz(n);
  }
  void build(vi &arr){
    for(int i=0;i<n;i++)tree[n+i] = {arr[i]};
    for(int i = n-1; i>0;i--)tree[i] = tree[i*2]+tree[i*2+1];
  }
  ll query(int l, int r){
    l += n;r+=n;
    push(l);
    push(r-1);
    Node s;
    for(;l<r; l>>=1,r>>=1){
      if(l&1)s = s+ tree[l++];
      if(r&1)s = s+ tree[--r];
    }
    return s.val;
  }

  void range_increment(int l,int r,ll val){
    l += n;
    r += n;
    int l0=l,r0=r;

    for(;l<r;l>>=1,r>>=1){
      if(l&1)apply(l++,val);
      if(r&1)apply(--r,val);
    }
    build_lazy(l0);
    build_lazy(r0-1);
  }

};
```

3 Graphs

3.1 DFS

```
void dfs(int u){
  vis[u] = 1;
  for(auto v:adj[u]){
    if(!vis[v])dfs(v);
  }
}
```

3.2 BFS

```
queue<int> q;
q.push(0);
while(!q.empty()){
  int u = q.front();
  q.pop();
  for(int v:adj[u]){
    if (!vis[v]) {
      vis[v] = 1;
      q.push(v);
    }
  }
}
```

3.3 Topological Sort

```
vi ts;
queue<int> q;
for (int i = 1; i < n+1; i++)
  if(!in[i])q.push(i);

while (!q.empty()) {
  int u=q.front();
  q.pop();
  ts.pb(u);
  for (int v : adj[u]) {
    in[v]--;
    if(!in[v])q.push(i);
  }
}

if(sz(ts)!=n)cout <<"IMPOSSIBLE";
```

3.4 Flood Fill

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

int floodfill(int x,int y,char color1,char color2){
  if(grid[x][y] == "outside")return 0;
  if(grid[x][y] != color1)return 0;
  int ans = 1;
  grid[x][y] = color2;
  for(int d=0;d<8;d++)ans += floodfill(x+dx[d],y+dy[d],color1,colr2);
  return ans;
}
```

3.5 MST

3.5.1 Kruskal

```
struct Edge{
  int u,v,w;
  bool operator < (Edge b){return w<b.w;}
};

//inside main

vector<Edge> edges;
sort(all(edges));

DSU dsu(vertices);

for(int i=0;i<E;i++){
  Edge front = edges[i];
  if(!dsu.same(front.u,front.v)){
    cost+= front.w;
    dsu.merge(front.u,front.v);
  }
}
```

3.5.2 Prims

```
vector<bool> vis(n);
priority_queue<pii, vector<pii>, greater<pii>> q;

auto process = [&](int u){
  vis[u] = 1;
  for(int i=0;i<sz(adj[u]);i++){
    pii v = adj[u][i];
    swap(v.S,v.F);
    q.push(v);
  }
}
process(0);

while(!q.empty()){
  auto [w,u] = q.top();q.pop();
  if(!vis[u]){cost += w;process(u);}
}
```

3.6 Dijkstra

3.6.1 Implementation

```
vll dist(n,INF64);
vector<bool> vis(n,0);
priority_queue<pll,vector<pll>,greater<pll>> pq;
pq.emplace(0,0);
dist[0] = 0;
vll cnt(n,0);//number of shortest paths
while(!pq.empty()){
  auto [d1,u] = pq.top();
  pq.pop();
  if(vis[u])continue;
  vis[u] = 1;
  for(auto [v,c]:adj[u]){
    if(dist[u]+c < dist[v]){
      dist[v] = dist[u]+c;
      pq.emplace(dist[v],v);
      cnt[v] = cnt[u];
    }else if(dist[u]+c == dist[v])(cnt[v]+=cnt[u])%=MOD
  }
}
```

3.6.2 Space-State graph

```
typedef array<ll,3> T;
priority_queue<T ,vector<T>,greater<T>> pq;
pq.push({0,0,state[0]});
dist[0][state[0]]= 0;
while(!pq.empty()){
  auto [d,u,s] = pq.top();
  pq.pop();
  if(vis[u][s])continue;
  vis[u][s] = 1;
  for(auto [v,w]:adj[u]){
    ll c  = min(s,slow[v]);
    //Nuevo estad cuando llegue a V
    if(dist[u][s]+w*s<dist[v][c]){
      dist[v][c] = dist[u][s]+w*s;
      pq.push({dist[v][c],v,c});
    }
  }
}
```

3.7 Bellman-Ford

3.7.1 Path

```
vector<int> d(n, INF);
d[v] = 0;
vector<int> p(n, -1);

for (;;) {
bool any = false;
for (Edge e : edges)
    if (d[e.a] < INF)
  if (d[e.b] > d[e.a] + e.cost) {
      d[e.b] = d[e.a] + e.cost;
      p[e.b] = e.a;
      any = true;
  }
if (!any)
    break;
}

if (d[t] == INF)
  cout << "No path from " << v << " to " << t << ".";
```

```
else {
  vector<int> path;
  for (int cur = t; cur != -1; cur = p[cur])path.push_back(cur);
  reverse(path.begin(), path.end());
}
```

3.7.2 Cycle

```
vector<int> d(n, INF);
d[v] = 0;
vector<int> p(n, -1);
int x;
for (int i = 0; i < n; ++i) {
x = -1;
for (Edge e : edges)
    if (d[e.a] < INF)
  if (d[e.b] > d[e.a] + e.cost) {
      d[e.b] = max(-INF, d[e.a] + e.cost);
      p[e.b] = e.a;
      x = e.b;
  }
}
if (x == -1)cout << "No negative cycle from " << v;
else{
int y = x;
for (int i = 0; i < n; ++i)y = p[y];
vector<int> path;
for (int cur = y;; cur = p[cur]) {
    path.push_back(cur);
    if (cur == y && path.size() > 1)break;
}
reverse(path.begin(), path.end());
}
```

3.8 Floyd-Warshall

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

for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
```

3.9 SCC Kosaraju

```
class SCC{
private:
  int n;
  vector<vi> adj;
  vector<vi> adj_t;
  vector<vi> adj_cond;
  vi order;
  vector<bool> vis;
  vi who;
  int comp;
public:
  SCC(int n,vector<vi> &a,vector<vi>&b):n(n),adj(a),adj_t(b){
    comp = -1;
    who.resize(n);
    vis.resize(n);
    for(int i=0;i<n;i++)
      if(!vis[i])dfs1(i);

    vis.assign(n,0);
    for(int i=n-1;i>=0;i--)
      if(!vis[order[i]]){
        comp++;
        dfs2(order[i]);
      }
    adj_cond.resize(comp+1);
    for(int u=0;u<n;u++)
      for(auto v:adj[u])
        if(who[v]!=who[u])
          adj_cond[who[u]].pb(who[v]);
  }

  void dfs1(int u) {
    vis[u] = true;
    for (int v:adj[u])
      if (!vis[v])dfs1(v);
    order.pb(u);
  }

  void dfs2(int u) {
    vis[u] = 1;
    for (int v : adj_t[u])
```

```
      if (!vis[v])dfs2(v);
    who[u] = comp;
  }
};
```

3.10 Flows

3.10.1 Dinic

```
struct flowEdge{
int u,v;
ll cap,flow = 0;
flowEdge(int u, int v, ll cap) : u(u), v(v), cap(cap) {};
};

struct Dinic{
vector<flowEdge> edges;
vector<vi> adj;
int n,s,t;
int id = 0;
vi level, next;

queue<int> q;

Dinic(int n, int s, int t) : n(n), s(s), t(t) {
  adj.resize(n);
  level.resize(n);
  next.resize(n);
  fill(all(level),-1);
  level[s] = 0;
  q.push(s);
}

void add(int u, int v, ll cap){
  edges.emplace_back(u,v,cap);
  edges.emplace_back(v,u,0);
  adj[u].pb(id++);
  adj[v].pb(id++);
}

bool bfs(){
  while (!q.empty()){
    int curr = q.front();
    q.pop();
    for (auto e : adj[curr]){
      if (edges[e].cap - edges[e].flow  < 1) continue;
      if (level[edges[e].v] != -1) continue;
      level[edges[e].v] = level[edges[e].u] + 1;
      q.push(edges[e].v);
    }
  }
  return level[t] != -1;
}

ll dfs(int u, ll flow){
  if (flow == 0) return 0;
  if (u == t) return flow;
  for (auto& cid = next[u]; cid < adj[u].size(); cid++){
      int e = adj[u][cid];
      int v = edges[e].v;

      if (level[edges[e].u] + 1 != level[v] || edges[e].cap - edges[e].
          flow < 1) continue;
      ll f = dfs(v,min(flow,edges[e].cap - edges[e].flow));
      if (f == 0) continue;
      edges[e].flow += f;
      edges[e ^ 1].flow -= f;
      return f;
  }
  return 0;
}

ll maxFlow(){
  ll flow = 0;
  while (bfs()){
      fill(all(next),0);
      for (ll f = dfs(s,INF64); f != 0ll; f = dfs(s,INF64)) flow += f;
      fill(all(level),-1);
      level[s] = 0;
      q.push(s);
  }
  return flow;
    }


vector<pii> minCut(){
  vector<pii> ans;
```

```
  fill(all(level),-1);
  level[s] = 0;
  q.push(s);
  while (!q.empty()){
    int curr = q.front();
    q.pop();
    for (int id = 0; id < adj[curr].size(); id++){
      int e = adj[curr][id];
      if (level[edges[e].v] == -1 && edges[e].cap - edges[e].flow > 0){
        q.push(edges[e].v);
        level[edges[e].v] = level[edges[e].u] + 1;
      }
    }
  }

  for (int i = 0; i < level.size(); i++){
    if (level[i] != - 1){
      for (int id = 0; id < adj[i].size(); id++){
      int e = adj[i][id];
      if (level[edges[e].v] == -1 && edges[e].cap - edges[e].flow == 0)
        ans.emplace_back(edges[e].u,edges[e].v);
      }
    }
  }
  return ans;
}

vector<pii> MaximumMatching(){
  vector<pii> ans;
  fill(all(level),-1);
  level[s] = 0;
  q.push(s);
  while (!q.empty()){
    int curr = q.front();
    q.pop();
    for (int id = 0; id < adj[curr].size(); id++){
    int e = adj[curr][id];
    if (level[edges[e].v] == -1 && edges[e].cap - edges[e].flow == 0 &&
        edges[e].flow != 0ll){
        q.push(edges[e].v);
        level[edges[e].v] = level[edges[e].u] + 1;
      }
    }
  }
  for (int i = 0; i < level.size(); i++){
    if (level[i] != - 1)
      for (int id = 0; id < adj[i].size(); id++){
        int e = adj[i][id];
        if (edges[e].u != s && edges[e].v != t && edges[e].cap - edges[e
            ].flow == 0 && edges[e].flow != 0ll)
          ans.emplace_back(edges[e].u,edges[e].v);
      }
  }
  return ans;
}

};
```

3.11 2-SAT

```
class SAT{
private:
  int n;
  vector<vi> adj;
  vector<vi> adj_t;
  vector<vi> adj_cond;
  vi order;
  vector<bool> vis;
  vi who;
  vector<bool> ans;
  int comp;
public:

  SAT(int m=0){
    n = (m<<1);
    adj.resize(n);
    adj_t.resize(n);
    vis.assign(n,0);
    who.assign(n,-1);
  }

  void dfs1(int u) {
    vis[u] = true;
    for (int v:adj[u])
      if (!vis[v])dfs1(v);
    order.pb(u);
```

```
  }

  void dfs2(int u) {
    vis[u] = 1;
    for (int v : adj_t[u])
      if (!vis[v])dfs2(v);
    who[u] = comp;
  }

  //~a->b ~b->a
  void addOR(int a, bool af, int b, bool bf) {
    a += a + (af ^ 1);
    b += b + (bf ^ 1);
    adj[a ^ 1].push_back(b);
    adj[b ^ 1].push_back(a);
    adj_t[b].push_back(a ^ 1);
    adj_t[a].push_back(b ^ 1);
  }

  //( a OR b ) ^ (~a or ~b)
  void addXOR(int a, bool af, int b, bool bf) {
    addOR(a, af, b, bf);
    addOR(a, !af, b, !bf);
  }
  //(a -> b)
  void _add(int a,bool af,int b,bool bf) {
    a += a + (af ^ 1);
    b += b + (bf ^ 1);
    adj[a].push_back(b);
    adj_t[b].push_back(a);
  }

  //(a->b)^(b->a)
  void add(int a,bool af,int b,bool bf) {
    _add(a, af, b, bf);
    _add(b, !bf, a, !af);
  }

  bool solve_SAT(){
    for(int i=0;i<n;i++)
      if(!vis[i])dfs1(i);

    comp = -1;
    vis.assign(n,0);
    for(int i=sz(order)-1;i>=0;i--)
      if(!vis[order[i]]){
        comp++;
        dfs2(order[i]);
      }
    ans.assign(n/2,0);
    for(int i=0;i<n;i+=2){
      if(who[i] == who[i+1])return 0;
      ans[i/2] = (who[i] > who[i+1]);
    }
    return 1;
  }

  bool operator[](int i){
    return ans[i];
  }

};
```

3.12 Euler Paths and cycles

3.12.1 Directed Graphs

```
struct EulerSolver{
private:
  vector<vector<pii>> adj;
  int n,edges;
  vi in,out;
  vi ans,edge_ans;
  int root;
  vi done;
  void dfs(int u){
    while(done[u] < sz(adj[u])){
      auto [v,id] = adj[u][done[u]++];
      dfs(v);
      edge_ans.pb(id);
    }
    ans.pb(u);
  }
public:
  EulerSolver(vector<vector<pii>> &a,int t):adj(a){
    n = t;
    edges = 0;
```

```
    in.assign(n,0);
    out.assign(n,0);
    done.assign(n,0);
    root = -1;
    for(int u=0;u<n;u++)
      for(auto v:adj[u]){
        edges++;
        in[v.F]++;
        out[u]++;
      }

  }

  bool possible(){
    int cnt1 =0,cnt2 =0;
    bool ok = 1;
    for(int i=0;i<n;i++){
      if (in[i] - out[i] == 1) cnt1++;
      if (out[i] - in[i] == 1) cnt2++, root = i;
      if (abs(in[i] - out[i]) > 1) ok = 0;
    }
    if(cnt1 > 1 || cnt2 > 1)ok = 0;
    if(!ok)return 0;

    if(root == -1)
      for(int i=0;i<n;i++)
        if(out[i])root = i;

    if(root == -1)return 1;
    dfs(root);

    if(sz(ans) != edges+1)return 0;
    reverse(all(ans));
    reverse(all(edge_ans));
    return 1;
  }

  int size(){
    return ans.size();
  }

  int operator()(int i,bool x){
    if(x)return ans[i];
    else return edge_ans[i];
  }
};
```

3.12.2 Unirected Graphs

```
struct EulerSolver{
private:
  vector<vector<pii>> adj;
  int n,edges;
  vi ans,edge_ans;
  vi deg;
  int root;
  vi done;
  vector<bool> vis;

  void dfs(int u){
    while(done[u] < sz(adj[u])){
      auto [v,id] = adj[u][done[u]++];
      if(vis[id])continue;
      vis[id]=1;
      dfs(v);
      edge_ans.pb(id);
    }
    ans.pb(u);
  }
public:
  EulerSolver(vector<vector<pii>> &a,int t):adj(a),n(t){
    edges = 0;
    done.assign(n,0);
    deg.assign(n,0);
    root = -1;
    for(int u=0;u<n;u++)
      for(auto v:adj[u]){
        edges++;
        deg[v.F]++;
        deg[u]++;
      }
    vis.assign(edges/2,0);
  }

  bool possible(){
    int odd = 0;
```

```
    for(int i=0;i<n;i++)
      if((deg[i]/2)&1){
        odd++;
        root = i;
      }


    if(odd>2)return 0;

    if(root == -1)
      for(int i=0;i<n;i++)
        if(deg[i]){root = i;break;}

    if(root == -1)return 1;
    dfs(root);

    if(sz(ans) != edges/2+1)return 0;
    reverse(all(ans));
    reverse(all(edge_ans));
    return 1;
  }

  int size(){
    return ans.size();
  }

  int operator()(int i,bool x){
    if(x)return ans[i];
    else return edge_ans[i];
  }
};
```

4 Trees

4.1 LCA - Binary Lifting

4.1.1 Normal LCA

```
const int MAX = 1e5+5,LG=18;
vi deep(MAX);//Si tiene peso vi cost(MAX)
int par[MAX][LG+1];

void dfs(int u = 1,int p=0){// U = raiz del arbol
  par[u][0] = p;
  deep[u] = deep[p]+1;//cost[u] += cost[p]
  for(int i=1;i<=LG;i++)par[u][i]=par[par[u][i-1]][i-1];
  for(int v:adj[u]){
    if(v!=p){
      dfs(v,u);
    }
  }
}

int lca(int u,int v){
  if(deep[u] < deep[v])swap(u,v);
  for(int k=LG;k>=0;k--)
    if(deep[par[u][k]] >= deep[v])
      u = par[u][k];
  if(u==v)return u;
  for(int k=LG;k>=0;k--)
    if(par[u][k]!=par[v][k])
      u=par[u][k],v=par[v][k];
  return par[u][0];
}

int dist(int u,int v){
  int lc = lca(u,v);
  return deep[u]+deep[v]-(deep[lc]<<1);
}

int kth(int u,int k){
  assert(k>=0);
  for(int i=0;i<=LG;i++)
    if(k&(1<<i))u = par[u][i];
  return u;
```

```
}
```

4.1.2 Min/Max weight on the path

```
const int MAX = 1e5+5,LG=18;
vector<vector<pii>> adj(MAX,vector<pii> ());
vi deep(MAX);
int par[MAX][LG+1];
int mn[MAX][LG+1];
int mx[MAX][LG+1];

void dfs(int u=1,int p=0,int costmn = INF,int costmx = -INF){
  par[u][0] = p;
  deep[u] = deep[p]+1;

  if(p!=0){
    mn[u][0]  = costmn;
    mx[u][0]  = costmx;
  }
  for(int i=1;i<=LG;i++){
    par[u][i]=par[par[u][i-1]][i-1];
    mn[u][i]=min(mn[u][i-1],mn[par[u][i-1]][i-1]);
    mx[u][i]=max(mx[u][i-1],mx[par[u][i-1]][i-1]);
  }

  for(auto [v,t]:adj[u]){
    if(v!=p){
      dfs(v,u,t,t);
    }
  }
}

pii minmax(int u,int v){
  pii ans = {INF,-INF};
  if(deep[u] < deep[v])swap(u,v);
  for(int k=LG;k>=0;k--)if(deep[par[u][k]] >= deep[v]){
    ans.F = min(ans.F,mn[u][k]);
    ans.S = max(ans.S,mx[u][k]);
    u = par[u][k];
  }
  if(u==v)return ans;
  for(int k=LG;k>=0;k--)if(par[u][k]!=par[v][k]){
    ans.F = min({ans.F,mn[u][k],mn[v][k]});
    ans.S = max({ans.S,mx[u][k],mx[v][k]});
    u=par[u][k],v=par[v][k];
  }
  ans.F = min({ans.F,mn[u][0],mn[v][0]});
  ans.S = max({ans.S,mx[u][0],mx[v][0]});

  return ans;
}
```

4.2 Centroid Decomposition

```
class Centroid{
private:
  vector<vi> tree;
  vi par;
  vi sub;
  vector<bool> check;
  int n;
public:
  Centroid(vector<vi> &t):tree(t){
    n = tree.size();
    par.resize(n);
    sub.resize(n);
    check.resize(n);
    build();
  }

  void build(int u = 0, int p = -1){
    dfs(u, p);
    int c = get_centroid(u,p,sub[u]);
    check[c] = 1;
    par[c] = p+1;

    for (auto v : tree[c]){
      if(!check[v])
        build(v, c);
    }

  }

  int dfs(int u,int p){
    if(check[u])return 0;
    sub[u] = 1;
```

```
    for(int v:tree[u])
      if(v!=p)sub[u]+=dfs(v,u);

    return sub[u];
  }

  int get_centroid(int u,int p,int x){
    for(auto v:tree[u])
      if(v!=p && sub[v]*2>x && !check[v])return get_centroid(v,u,x);

    return u;
  }

  int operator[](int i){
    return par[i];
  }
};
```

4.3 Euler Tour Technique

```
int timer = 0;
vi start,final;
vector<vi> adj;
void dfs(int u = 0,int prev = -1){
  start[u]=timer++;
  for(int v:adj[u]){
    if(v!=prev)dfs(v,u);
  }
  final[u] = timer;
}
// [start,final)
```

4.4 Tree Matching

```
vector<vll> dp(2,vll(MAX));

void dfs(ll u = 0, ll p=-1){
  dp[0][u] = 0;
  dp[1][u] = -INF;

  for(ll v:adj[u]){
    if(v == p)continue;
    dfs(v,u);

    dp[0][u]+=max(dp[1][v],dp[0][v]);

    ll opt = min(dp[0][v]-dp[1][v],0ll);

    dp[1][u] = max(dp[1][u],opt);
  }

  dp[1][u] += dp[0][u];
  dp[1][u]++;
}
```

4.5 Diameter

```
vector<vector<pii>> adj;
vector<bool> vis;

int bfs(int n,vll &d,int v = 0){
  vis.assign(n,0);
  d.assign(n,0);
  d[v] = 0;
  queue<int> q;
  q.push(v);
  pll last = {v,0};
  vis[v] = 1;
  while(!q.empty()){
    int u = q.front();q.pop();
    for(auto [w,c]:adj[u]){
      if(vis[w])continue;
      d[w] = d[u]+c;
      q.push(w);
      vis[w] = 1;
      if(d[w]>last.S)last = {w,d[w]};
    }
  }
  return int(last.F);
}

void solve(){
  int n;cin >> n;
  adj.assign(n,vector<pii> ());
  for(int i=1;i<n;i++){
    int a,b;ll c;
    cin >> a >> b >> c;
```

```
    adj[--a].pb({--b,c});
    adj[b].pb({a,c});
  }
  vll dx(n),dy(n);
  int a =bfs(n,dx);
  int b =bfs(n,dy,a);
  bfs(n,dx,b);

  for(int i=0;i<n;i++){
    cout << max(dx[i],dy[i]) << "␣\n"[i==(n-1)];
  }

}
```

4.6 Heavy Light Decomposition

```
class HLD{
private:
  int n;
  vector<vi> adj;
  vi parent;
  vi heavy;
  vi depth;
  vi root;
  vi treePos;
  Segment_Tree tree;

  int dfs(int u =0){
    int size = 1, mx_sub = 0;
    for(auto v:adj[u])
      if(v!=parent[u]){
        parent[v] = u;
        depth[v] = depth[u]+1;
        int sub = dfs(v);
        if(sub > mx_sub){
          heavy[u] = v;
          mx_sub = sub;
        }
        size += sub;
      }
    return size;
  }

  template <class BinaryOperation>
  void processPath(int u, int v, BinaryOperation op) {
    for (; root[u] != root[v]; v = parent[root[v]]) {
      if (depth[root[u]] > depth[root[v]]) swap(u, v);
      op(treePos[root[v]], treePos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(treePos[u], treePos[v] + 1);
  }

  void init(vi &a){
    for(int i=0,crt=0;i<n;++i){
      if(parent[i] == -1 || heavy[parent[i]]!=i){
        for(int j = i;j!=-1;j = heavy[j]){
          root[j] = i;
          treePos[j] = crt++;
        }
      }
    }
    tree = Segment_Tree(n);
    for(int i=0;i<n;i++){
      tree.update(treePos[i],a[i]);
    }
  }

public:

  HLD(int n,vector<vi> &vale,vi &a):n(n),adj(vale){
    parent.assign(n,0);
    heavy.assign(n,-1);
    depth.assign(n,0);
    root.resize(n);
    treePos.assign(n,0);
    parent[0] = -1;
    dfs();
    init(a);
  }

  void update(int u,int &val){
    tree.update(treePos[u],val);
  }

  void updatePath(int u, int v, int value) {
```

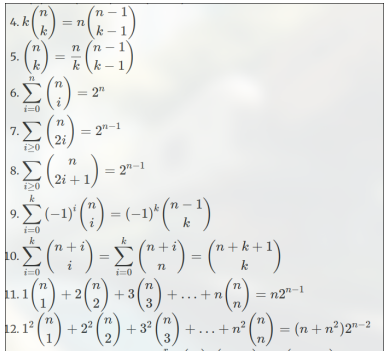
```
    processPath(u, v, [this, &value](int l, int r) {
      tree.update_range(l, r, value);
    });
  }

  int query(int u,int v){
    int ans = 0;
    processPath(u,v,[this,&ans](int l,int r){
      ans = max(ans,tree.query(l,r));
    });
    return ans;
  }
};
```

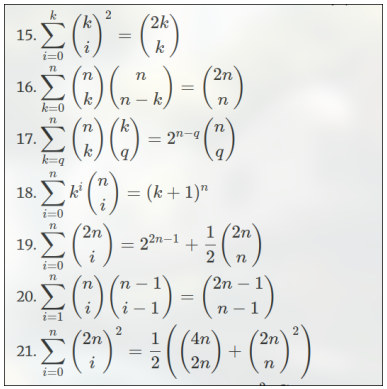
5 Math

5.1 Identities

5.1.1 Combi



$$4.\ k\binom{n}{k} = n\binom{n-1}{k-1}$$
$$5.\ \binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$
$$6.\ \sum_{i=0}^{n}\binom{n}{i} = 2^n$$
$$7.\ \sum_{i\geq 0}\binom{n}{2i} = 2^{n-1}$$
$$8.\ \sum_{i\geq 0}\binom{n}{2i+1} = 2^{n-1}$$
$$9.\ \sum_{i=0}^{k}(-1)^i\binom{n}{i} = (-1)^k\binom{n-1}{k}$$
$$10.\ \sum_{i=0}^{k}\binom{n+i}{i} = \sum_{i=0}^{k}\binom{n+i}{n} = \binom{n+k+1}{k}$$
$$11.\ 1\binom{n}{1} + 2\binom{n}{2} + 3\binom{n}{3} + \ldots + n\binom{n}{n} = n2^{n-1}$$
$$12.\ 1^2\binom{n}{1} + 2^2\binom{n}{2} + 3^2\binom{n}{3} + \ldots + n^2\binom{n}{n} = (n+n^2)2^{n-2}$$



$$15.\ \sum_{i=0}^{k}\binom{k}{i}^2 = \binom{2k}{k}$$
$$16.\ \sum_{k=0}^{n}\binom{n}{k}\binom{n}{n-k} = \binom{2n}{n}$$
$$17.\ \sum_{k=q}^{n}\binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$$
$$18.\ \sum_{i=0}^{n}k^i\binom{n}{i} = (k+1)^n$$
$$19.\ \sum_{i=0}^{n}\binom{2n}{i} = 2^{2n-1} + \frac{1}{2}\binom{2n}{n}$$
$$20.\ \sum_{i=1}^{n}\binom{n}{i}\binom{n-1}{i-1} = \binom{2n-1}{n-1}$$
$$21.\ \sum_{i=0}^{n}\binom{2n}{i}^2 = \frac{1}{2}\left(\binom{4n}{2n} + \binom{2n}{n}^2\right)$$

5.1.2 XOR

$$a + b = a \oplus b + 2(a\&b)$$

$$a + b = a|b + a\&b$$

$$a \oplus b = a|b - a\&b$$

$$1 \oplus 2 \oplus 3 \oplus \cdots \oplus (4k-1) = 0k >= 0$$

5.2 Theorems

6 Strings

6.1 Suffix Array

```
struct suffix{
  int index;
  pii rank;
  bool operator<(suffix b){return rank<b.rank;}
  bool operator>(suffix b){return rank>b.rank;}
};

class Suffix_Array{
  public:
  string s;int n;
  vector<suffix> suffixes;
  vi lcp;
  Suffix_Array(string x){
    s = x+"$";
    n = sz(s);
    suffixes.resize(n);
  }


  void radix_sort(vector<suffix> &suff){
    for(int i: vi{2,1}){
      auto key = [&](const suffix &x){
        return i == 1?x.rank.F:x.rank.S;
      };

      int mx = 0;
      for (const auto &i:suff) { mx = max(mx, key(i)); }
      vector<int> occs(mx + 1);
      for (const auto &i:suff)occs[key(i)]++;
      vector<int> start(mx + 1);
      for (int i=1;i<=mx; i++) start[i] = start[i-1]+occs[i-1];

      vector<suffix> new_arr(suff.size());
      for (const auto &i : suff) {
        new_arr[start[key(i)]] = i;
        start[key(i)]++;
      }
```

```
      suff = new_arr;
    }
  }

  void build(){
    for(int i=0;i<n;i++)suffixes[i].index = i , suffixes[i].rank = {s[i
        ],(i+1<n?s[i+1]:-1)};
    sort(all(suffixes));
    vi equiv(n);

    for(int i=1;i<n;i++){
      auto c_val = suffixes[i].rank;
      int cs = suffixes[i].index;
      auto  p_val = suffixes[i - 1].rank;
      int ps = suffixes[i-1].index;
      equiv[cs] = equiv[ps] + (c_val > p_val);
    }

    for(int cmp_ant = 1;cmp_ant <n; cmp_ant<<=1){
      for(auto &x:suffixes)
        x.rank = {equiv[x.index],equiv[(x.index+cmp_ant)%n]};

      //Std sort O(nlognlogn) Radix Sort O(nlogn)
      radix_sort(suffixes);

      for(int i=1;i<n;i++){
        auto c_val = suffixes[i].rank;
        int cs = suffixes[i].index;
        auto  p_val = suffixes[i - 1].rank;
        int ps = suffixes[i-1].index;
        equiv[cs] = equiv[ps] + (c_val > p_val);
      }
    }
  }

  void build_lcp(){
    vector<int> suff_ind(n);
    for (int i = 0; i < n; i++)suff_ind[suffixes[i].index] = i;
    lcp.resize(n-1);
    int start_at = 0;
    for (int i = 0; i < n - 1; i++) {
      int prev = suffixes[suff_ind[i] - 1].index;
      int curr_cmp = start_at;
      while (s[i + curr_cmp] == s[prev + curr_cmp])curr_cmp++;
      lcp[suff_ind[i] - 1] = curr_cmp;
      start_at = max(curr_cmp - 1, 0);
    }
  }

  void show(){
    for(int i=0;i<n;i++)cout << suffixes[i].index << "␣\n"[(i+1)==n];
  }
};
```

7 Others

7.1 Binpow

```
ll binpow(ll a,ll b){
  a%=MOD;
  ll ans = 1;
  while(b){
    if(b&1)(ans*=a)%=MOD;
    (a*=a)%=MOD;
    b>>=1;
  }
  return ans;
}
```

7.2 Matrix exponentiation

```
struct Mat {
  int n, m;
  vector<vector<int>> a;
  Mat() { }
  Mat(int _n, int _m) {n = _n; m = _m; a.assign(n, vector<int>(m, 0)); }
  Mat(vector< vector<int> > v) { n = v.size(); m = n ? v[0].size() : 0;
      a = v; }
  inline void make_unit() {
    assert(n == m);
    for (int i = 0; i < n; i++)  {
      for (int j = 0; j < n; j++) a[i][j] = i == j;
    }
  }
  inline Mat operator + (const Mat &b) {
    assert(n == b.n && m == b.m);
```

```
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
      }
    }
    return ans;
  }
  inline Mat operator - (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
      }
    }
    return ans;
  }
  inline Mat operator * (const Mat &b) {
    assert(m == b.n);
    Mat ans = Mat(n, b.m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < b.m; j++) {
        for(int k = 0; k < m; k++) {
          ans.a[i][j] = (ans.a[i][j] + 1LL * a[i][k] * b.a[k][j] % mod)
              % mod;
        }
      }
    }
    return ans;
  }
  inline Mat pow(long long k) {
    assert(n == m);
    Mat ans(n, n), t = a; ans.make_unit();
    while (k) {
      if (k & 1) ans = ans * t;
      t = t * t;
      k >>= 1;
    }
    return ans;
  }
  inline Mat& operator += (const Mat& b) { return *this = (*this) + b; }
  inline Mat& operator -= (const Mat& b) { return *this = (*this) - b; }
  inline Mat& operator *= (const Mat& b) { return *this = (*this) * b; }
  inline bool operator == (const Mat& b) { return a == b.a; }
  inline bool operator != (const Mat& b) { return a != b.a; }
};

// Usage
// Mat a(n, n);
// Mat b(n, n);
// Mat c = a * b;
// Mat d = a + b;
// Mat e = a - b;
// Mat f = a.pow(k);
// a.a[i][j] = x;
```