# Contents:



## 1. Template

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template<class T> using oset = tree<T, null_type,
less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
#define int long long
#define all(x) x.begin(),x.end()
#define pb push_back
#define tc int tt, qq=0; cin>>tt; while(qq++<tt)
#define cs cout<<"Case "<<qq<<": "
#define endl '\n'
#define deb(x) cerr<<#x" = "<<x<<endl
#define INF 1e18
#define m 1000000007
#define speed ios_base::sync_with_stdio(0);
cin.tie(0); cout.tie(0);
const int N = 1e6+5;

int32_t main(){
    speed

}
```

## 2. Number Theory

### 2.1 Sieve Prime

```
bool prime[N]; // N is set to const val <=1e7
void sieveofErat(){
  memset(prime, true, sizeof(prime));
  prime[0] = prime[1] = false;
  for(int i = 2; i*i <= N; i++)
    if(prime[i]){
      for(int j = i*i; j < N; j+=i)
        prime[j] = false;
    }
}
```

### 2.2 Modular Inverse

### Extended Eucledian

```
//Finding GCD
int extendedEuclidean(int a, int b, int& x, int&
y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    int xPrev, yPrev;
    int gcd = extendedEuclidean(b, a % b, xPrev,
yPrev);
    x = yPrev;
    y = xPrev - (a / b) * yPrev;
    return gcd;
}

//Findind Inverse from GCD
int modularInverse(int a, int m) {
    int x, y;
    int gcd = extendedEuclidean(a, m, x, y);
    if (gcd != 1) {
        cout<<"imp"<<endl; //Impossible Case
        return 0;
    }
    return (x % m + m) % m;
}
```

### 2.3 Big Mod

```
int big_mod(int a,int b,int mod){
    if(b==0) return 1%m;
    int x=big_mod(a,b/2,m);
    x=(x*x)%m;
    if(b%2==1) x=(x*a)%m;
    return x;
}
```

### 2.4 Binary Exponentiation

```
int binExp(int a, int b, int m){
    int ans = 1;
    while(b > 0){
        if(b&1) ans = (ans*1LL*a) % m;
        a = (a*1LL*a)%m;
        b >>= 1;
    }
    return ans;
}
//for a^b^c call binExp(a, binExp(b, c, m-1), m)
```

### 2.5 Bitmask

a|b = a^b + a&b
a^(a&b) = (a|b)^b
b^(a&b) = (a|b)^a
(a&b)^(a|b) = a^b
a+b = a|b + a&b
a+b = a^b + 2(a&b)
a-b = (a^(a&b))-((a|b)^a)
a-b = ((a|b)^b)-((a|b)^a)
a-b = (a^(a&b))-(b^(a&b))
a-b = ((a|b)^b)-(b^(a&b))
c = a^b then a = c^b

### 2.6 Sum of Divisor(SOD)

```
map <int,int> primes;
int SOD(int n){
    for(int i=2;i*i<=n;++i){
        int power = 0;
        while(n%i == 0){
            ++power;
            n /= i;
        }
        if(power > 0){
            primes[i] = power;
        }
    }
    int sod=1;
    primes[n]++;
    for(auto k:primes){
        int p = k.first;
        int a=k.second;
        sod=sod*(pow(p,a+1)-1)/(p-1);
    }
    return sod;
}
```

## 3. Graph

### 3.1 DFS(Modified)

```
//Height of Tree
void dfs(int nod, int par){
  vis[nod]=1;
  for(int child : adj[nod]){
    if(child == par)continue;
    depth[child] = depth[nod] + 1;
    dfs(child, nod);
    height[nod] = max(height[nod],
height[child]+1);
  }
}
```

### 3.2 Dijkstra

```
void dijkstra(int source){
    dis[source] = 0;
    priority_queue<pair<int, int>,
vector<pair<int, int>>, greater<pair<int, int>>>
st;
    st.push({0, source});
    while(!st.empty()){
        auto node = st.top();
        int v = node.second;
        int dist = node.first;
        st.pop();
        if(vis[v]) continue;
        vis[v] = 1;
        for(auto child : adj[v]){
            int adjNode = child.second;
            int wt = child.first;
            if(dis[v]+wt < dis[adjNode]){
                dis[adjNode] = dis[v] + wt;
                st.push({dis[adjNode], adjNode});
            }
        }
    }
}
```

### 3.3 Bellman Ford

```
vector<int> bellman_ford(int V,
vector<vector<int>>& edges, int S) {
    vector<int> dist(V, 1e8);
    dist[S] = 0;
    for (int i = 0; i < V - 1; i++) {
        for (auto it : edges) {
            int u = it[0];
            int v = it[1];
            int wt = it[2];
            if (dist[u] != 1e8 && dist[u] + wt <
dist[v]) {
                dist[v] = dist[u] + wt;
            }
        }
    }
    // Nth relaxation to check negative cycle
    for (auto it : edges) {
        int u = it[0];
        int v = it[1];
        int wt = it[2];
        if (dist[u] != 1e8 && dist[u] + wt <
dist[v]) {
            return {-1};
        }
    }
    return dist;
}
```

### 3.4 MST

#### Prims

```
int spanningTree(int V, vector<vector<int>>
adj[]){
    priority_queue<pair<int, int>,
vector<pair<int, int> >, greater<pair<int, int>>>
pq;

    vector<int> vis(V, 0);
    // {wt, node}
    pq.push({0, 0});
    int sum = 0;
    while (!pq.empty()) {
        auto it = pq.top();
        pq.pop();
        int node = it.second;
        int wt = it.first;

        if (vis[node] == 1) continue;
        // add it to the mst
        vis[node] = 1;
        sum += wt;
        for (auto it : adj[node]) {
            int adjNode = it[0];
            int edW = it[1];
            if (!vis[adjNode]) {
                pq.push({edW, adjNode});
            }
        }
    }
    return sum;
}
```

#### Kruskal

```
//Include DSU before use
int spanningTree(int V, vector<vector<int>>
adj[])
{
    // 1 - 2 wt = 5
    /// 1 - > (2, 5)
    // 2 -> (1, 5)

    // 5, 1, 2
    // 5, 2, 1
    vector<pair<int, pair<int, int>>> edges;
    for (int i = 0; i < V; i++) {
```

```
        for (auto it : adj[i]) {
            int adjNode = it[0];
            int wt = it[1];
            int node = i;

            edges.push_back({wt, {node,
adjNode}});
        }
    }
    DisjointSet ds(V);
    sort(edges.begin(), edges.end());
    int mstWt = 0;
    for (auto it : edges) {
        int wt = it.first;
        int u = it.second.first;
        int v = it.second.second;

        if (ds.findUPar(u) != ds.findUPar(v)) {
            mstWt += wt;
            ds.unionBySize(u, v);
        }
    }

    return mstWt;
}
```

# 4. Data Structure

## 4.1 DSU

```
class DisjointSet {
    vector<int> rank, parent, size;
public:
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        size.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
            size[i] = 1;
        }
    }

    int findUPar(int node) {
        if (node == parent[node])
            return node;
        return parent[node] =
findUPar(parent[node]);
    }

    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] < rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        }
        else if (rank[ulp_v] < rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        }
        else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }

    void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (size[ulp_u] < size[ulp_v]) {
            parent[ulp_u] = ulp_v;
            size[ulp_v] += size[ulp_u];
        }
        else {
            parent[ulp_v] = ulp_u;
            size[ulp_u] += size[ulp_v];
        }
    }
};
```

## 4.2 Segment Tree

```
//Change functions according to need
void init(int node, int b, int e){
    if(b == e){
        segT[node] = ara[b];
        return;
    }
    int left = node*2;
    int right = node*2 + 1;
    int mid = (b+e)/2;
    init(left, b, mid);
    init(right, mid+1, e);
    segT[node] = segT[left]+segT[right];
}

int query(int node, int b, int e, int l, int r){
    if(b >= l && e <= r) return segT[node];
    if(l > e || r < b) return 0;
    int mid = (b+e)/2;
    int left = node*2;
    int right = node*2 + 1;
    int val1 = query(left, b, mid, l, r);
    int val2 = query(right, mid+1, e, l, r);
    return val1+val2;
}

void update(int node, int b, int e, int val, int
ind){
    if(b > ind || e < ind) return;
    if(b == e){
        segT[node] = val;
        return;
    }
    int mid = (b+e)/2;
    int left = node*2;
    int right = node*2 + 1;
    update(left, b, mid, val, ind);
    update(right, mid+1, e, val, ind);
    segT[node] = segT[left]+segT[right];
}
```

## 4.3 Merge Sort

```
void merge(int a[], int low, int mid, int high){
    int n = mid - low + 1;
    int m = high - mid;
    int A[n], B[m];
    for(int i = 0, j = low; i < n; i++, j++) A[i]
= a[j];
    for(int i = 0, j = mid+1; i < m; i++, j++)
B[i] = a[j];
    int i = 0, j = 0;
    int k = low;
    while(i < n && j < m){
        if(A[i]<B[j]) a[k]=A[i], i++;
        else a[k]=B[j], j++;
        k++;
    }
    while(i<n) a[k]=A[i], i++, k++;
    while(j<m) a[k]=B[j], j++, k++;
    return;
}

void mergeSort(int a[], int low, int high){
```

```
    if(low >= high) return;
    int mid = (low+high)/2;
    mergeSort(a, low, mid);
    mergeSort(a, mid+1, high);
    merge(a, low, mid, high);
}
```

# 5. String

## 5.1 Trie

```
struct node {
    bool endmark;
    node* next[26 + 1];
    node()
    {
        endmark = false;
        for (int i = 0; i < 26; i++)
            next[i] = NULL;
    }
} * root;
void insert(char* str, int len)
{
    node* curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr->next[id] == NULL)
            curr->next[id] = new node();
        curr = curr->next[id];
    }
    curr->endmark = 1;
}
bool search(char* str, int len)
{
    node* curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr->next[id] == NULL)
            return false;
        curr = curr->next[id];
    }
    return curr->endmark;
}
void del(node* cur)
{
    for (int i = 0; i < 26; i++)
        if (cur->next[i])
            del(cur->next[i]);
    delete (cur);
}

//root = new node() in main function
```

# 6. Misscellenous

## 6.1 Ternary Search

```
void Ternary_search(){
    /// ekhane answer always double a hoy
    /// 'U' or 'Ulta U' ei duita function e kaj
kore
    /// 100 ta iteration chalaite hoy

    double l = 0,r = 1e9;
    lp(i, 1, 100){ //eksho ta loop chalaise
        double mid1 = (2*l+r)/3.0;
        double mid2 = (l+2*r)/3.0;
        /// For ’U’ function. Find the minimum
value
        if(function(mid1) > function(mid2))
            l = mid1;
        else r = mid2;

        /// For ’Ulta U’ function. Find the
maximum value
        if(function(mid1) < function(mid2))
            l = mid1;
        else r = mid2;
    }
}
```

## 6.2 Power Set

```
// All possible sequence of a string
vector<string> AllPossibleStrings(string s) {
    int n = s.length();
    vector<string>ans;
    for (int num = 0; num < (1 << n); num++) {
        string sub = "";
        for (int i = 0; i < n; i++) {
            //check if the ith bit is set or not
            if (num & (1 << i)) {
                sub += s[i];
            }
        }
        if (sub.length() > 0) {
            ans.push_back(sub);
        }
    }
    sort(ans.begin(), ans.end());
    return ans;
}
```

## Lazy Propagation

```
struct node{
    int value=0;
    int lazy=0;
};

int arr[mx];
node tree[mx*4];

void build(int L,int R,int at){
    if(L==R){
        tree[at].value=arr[L];
        return;
    }
    int mid=(L+R)/2;
    int left = 2*at;
    int right=2*at+1;
    build(L,mid,left);
    build(mid+1,R,right);
    tree[at].value+=tree[left].value+tree[right].
value;
}
//call it using build(0,arraySize-1,1);
void update(int l,int r,int x,int L,int R,int
at){
    if(l>R||r<L) return;
    if(L>=l&&R<=r){
        tree[at].value+=(R-L+1)*x;
        tree[at].lazy+=x;
        return;
    }
    int mid = (L+R)/2;
    int left = at*2;
    int right = at*2+1;
    update(l,r,x,L,mid,left);
    update(l,r,x,mid+1,R,right);
    tree[at].value =
tree[left].value+tree[right].value+(R-
L+1)*tree[at].lazy;
}
```

```
//call it using update(l,r,x,0,arraySize-1,1)
int query(int l,int r,int L,int R,int at,int
carry){
    if(l>R||r<L) return 0;
    if(L>=l&&R<=r){
        return tree[at].value+carry*(R-L+1);
    }
    int mid = (L+R)/2;
    int left=at*2;
    int right=at*2+1;
    int carryValue = carry+tree[at].lazy;
    int x = query(l,r,L,mid,left,carryValue);
    int y = query(l,r,mid+1,R,right,carryValue);
    return x+y;
}
```