SWERC Notebook
TELECOM Nancy (France) Equipe 2

Contents

0.1 Segtree

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;

    SegmentTree() : n(0) {}

    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }

    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }

    void init(int n_, Info v_ = Info()) {
        init(std::vector<Info>(n_, v_));
    }

    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        int sz = (1 << (std::__lg(n - 1) + 1));
        info.assign(sz * 2, Info());
        std::function<void(int, int, int)> build = [&](int v, int l,
              int r) {
            if (l == r) {
                info[v] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(v + v, l, m);
            build(v + v + 1, m + 1, r);
            info[v] = info[v + v] + info[v + v + 1];
        };
        build(1, 0, n - 1);
    }

    Info rangeQuery(int v, int l, int r, int tl, int tr) {
        if (r < tl || l > tr) {
            return Info();
        }
        if (l >= tl && r <= tr) {
            return info[v];
        }
        int m = (l + r) / 2;
        return rangeQuery(v + v, l, m, tl, tr) + rangeQuery(v +
              v + 1, m + 1, r, tl, tr);
    }

    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n - 1, l, r);
    }

    void modify(int v, int l, int r, int i, const Info &x) {
        if (l == r) {
            info[v] = x;
            return;
        }
        int m = (l + r) / 2;
        if (i <= m) {
            modify(v + v, l, m, i, x);
        } else {
            modify(v + v + 1, m + 1, r, i, x);
        }
        info[v] = info[v + v] + info[v + v + 1];
    }

    void modify(int i, const Info &x) {
        modify(1, 0, n - 1, i, x);
    }

    Info query(int v, int l, int r, int i) {
        if (l == r) {
            return info[v];
        }
        int m = (l + r) / 2;
        if (i <= m) {
            return query(v + v, l, m, i);
        } else {
            return query(v + v + 1, m + 1, r, i);
        }
    }

    Info query(int i) {
        return query(1, 0, n - 1, i);
    }
};

const int INF = 1E9;

struct Info {
    /* exemple
    int min1, min2, max1, max2, ans1, ans2;

    Info() : min1(INF), min2(INF), max1(-INF), max2(-INF),
          ans1(0), ans2(0) {}

    Info(std::pair<int, int> x) : min1(x.first), min2(x.second),
          max1(x.first), max2(x.second), ans1(0), ans2(0) {}
    */
};

Info operator+(const Info &a, const Info &b) {
    /* exemple
    Info res;
    res.min1 = std::min(a.min1, b.min1);
    res.min2 = std::min(a.min2, b.min2);
    res.max1 = std::max(a.max1, b.max1);
    res.max2 = std::max(a.max2, b.max2);
    res.ans1 = std::max({a.ans1, b.ans1, b.max1 - a.min1});
    res.ans2 = std::max({a.ans2, b.ans2, a.max2 - b.min2});
    return res;
    */
}
```

0.2 Findbyorder

```
// to use function find_by_order() and order_of_key() in a set
       in log
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
       tree_order_statistics_node_update>
    mod_set;
```

0.3 KMP

```
// Knuth-Morris-Pratt algorithm find string in a text in linear
      time
vector<int> findOccurences(const string& s, const string& t) {
    int n = s.length();
    int m = t.length();
    string S = s + "#" + t;
    auto pi = prefixFunction(S);
    vector<int> ans;
    FOR(i, n+1, n+m+1) {
        if(pi[i] == n) {
            ans.pb(i-2*n);
        }
    }
    return ans;
}

vector<int> prefixFunction(const string& s) {
    int n = (int)s.length();
    vector<int> pi(n);
    pi[0] = 0;
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) {
            j = pi[j - 1];
        }
        if (s[i] == s[j]) {
            j++;
        }
        pi[i] = j;
    }
    return pi;
}
```

0.4 Dijkstra

```
priority_queue<pair<ll,ll>,vector<pair<ll,ll> >,greater<pair<
      ll,ll> > > pq;
vector<ll> distTo(n+1,LLONG_MAX);
vector<bool> visited(n+1,false);

   distTo[1] = 0;
   pq.push(make_pair(0,1));
   while( !pq.empty() ){
       ll prev = pq.top().second;
      pq.pop();
        if(visited[prev])continue;
        visited[prev] = true;
      vector<pair<ll,ll> >::iterator it;
      for( it = g[prev].begin() ; it != g[prev].end() ; it++){
         ll next = it->first;
         if( distTo[next] > distTo[prev] + it->second){
            distTo[next] = distTo[prev] + it->second;
            pq.push(make_pair(distTo[next], next));
         }
      }

   }
```

0.5 Fenwick

```
struct Fenw {
    vector<int> tree;
    int size;

    Fenw(int n) : size(n), tree(n + 1, 0) {}

    void add(int idx, int val) {
        while (idx <= size) {
            tree[idx] += val;
            idx += idx & (-idx);
        }
    }

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

0.6 Trie

```
struct Trie {
    const int ALPHA = 26;
    const char BASE = 'a';
    vector<vector<int>> nextNode;
    vector<int> mark;
    int nodeCount;
    Trie() {
        nextNode = vector<vector<int>>(MAXN, vector<int>(
              ALPHA, -1));
        mark = vector<int>(MAXN, -1);
        nodeCount = 1;
    }
    void insert(const string& s, int id) {
        int curr = 0;
        FOR(i, 0, (int)s.length()) {
            int c = s[i] - BASE;
            if(nextNode[curr][c] == -1) {
                nextNode[curr][c] = nodeCount++;
            }
            curr = nextNode[curr][c];
        }
        mark[curr] = id;
    }

    bool exists(const string& s) {
        int curr = 0;
        FOR(i, 0, (int)s.length()) {
            int c = s[i] - BASE;
            if(nextNode[curr][c] == -1) return false;
            curr = nextNode[curr][c];
        }
        return mark[curr] != -1;
    }
};
```

0.7 Toposort

```
void DFS_aux(vector<vector<int>> &g,int v, vector<bool> &
      visited,stack<int>& Stack)
{
     visited[v] = true;
     for (auto i:g[v])
          if (!visited[i])
               DFS_aux(g,i, visited, Stack);
     Stack.push(v);
}
vector<bool> visited(g.size(), false);
     stack<int> S;
     for (int i = 0; i < n; i++){
        if (visited[i] == false){
               DFS_aux(g, i,visited, S);
        }
     }
     while(!S.empty()) cout << S.top()+1 << "␣",S.pop();
```

0.8 Template

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define uint unsigned int
#define ENDL '\n'
#define PRINT(x) cout << x << ENDL
#define PRINT2(x,y) cout << x << "␣" << y << ENDL
```

```
#define PRINTP(p) cout << p.first << "␣" << p.second <<
    ENDL
#define OUI cout << "YES\n"
#define NON cout << "NO\n"
#define pb push_back
#define mp make_pair
const ll MOD = 1e9 + 7;

void fastio() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
}

void solve() {

}

int main() {
    fastio();
    ll t; cin >> t;
    for (ll i = 1; i <= t; i++)
    solve();
    return 0;
}
```

0.9 Miscs

```
// 31 - ___builtin_clz ou ___lg(x) donne l'indice du premier bit
    à 1 de droite à gauche.
// calculer double acos(0.0) pour avoir pi très précis.
// int a = x avec x un char pour convert en ascii et a-48 ou - '0'
     pour avoir le nombre correspondant si entre 0 et 9.
// précision d'un nombre à couper : printf("%.*lf,n,nb) avec lf
    pour double et n le nb de chiffres.
// lower_bound(x) et upper_bound(x) pour resp premier indice
     i tq tab[i]>= x resp tq tab[i]> x.
// format exemple priority_queue priority_queue<int,vector<
    int>,greater<int>>pq; lorsque fonction de tri fait maison
    faire decltype(func) a la place de greater.

// priority_queue<int> pq;

// pq.push(3); // add 3
// pq.empty(); // -> bool
// pq.top(); // elem en haut
// pq.pop(); // tej net l'element en haut

// // autre exemple
// auto cmp = [](int left, int right) { return (left ^ 1) < (right ^
     1); };
// std::priority_queue<int, std::vector<int>, decltype(cmp)>
    lambda_priority_queue(cmp);

// for (int n : data)
// lambda_priority_queue.push(n);
```

0.10 Floyd Warshall

```
void floydWarshall(vector<vector<int>> &graph)
{
    //O(n^3)
    int V = graph.size();
    vector<vector<int>> dist = graph;
    for (int k = 0; k < V; ++k)
    {
        for (int i = 0; i < V; ++i)
        {
            for (int j = 0; j < V; ++j)
            {
                if (dist[i][k] != INF && dist[k][j] != INF && dist[i
                    ][k] + dist[k][j] < dist[i][j])
                dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printSolution(dist);
}
```

0.11 Ternary Search

```
double ternary_search(double l, double r) {
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l); //return the maximum of f(x) in [l, r]
}
```

0.12 Z Algo

```
//find word u in s in O(|s| + |u|)

int L = 0, R = 0;
for (int i = 1; i < n; i++) {
  if (i > R) {
    L = R = i;
    while (R < n && s[R-L] == s[R]) R++;
    z[i] = R-L; R--;
  } else {
    int k = i-L;
    if (z[k] < R-i+1) z[i] = z[k];
    else {
      L = i;
      while (R < n && s[R-L] == s[R]) R++;
      z[i] = R-L; R--;
    }
  }
}

vector<int> search(string &text, string &pattern) {
    string s = pattern + '$' + text;
    vector<int> z = zFunction(s);
    vector<int> pos;
    int m = pattern.size();

    for (int i = m + 1; i < z.size(); i++) {
        if (z[i] == m){
            // pattern match starts here in text
            pos.push_back(i - m - 1);
        }
    }
    return pos;
}
```

0.13 Bellman

```
// pour detecter les cycles negatifs faire une autre itération si
     changementcycle negatifs
for (int i = 1; i <= n; i++) distance[i] = INF;
distance[x] = 0;
for (int i = 1; i <= n-1; i++) {
    for (auto e : edges) {
        int a, b, w;
        tie(a, b, w) = e;
        distance[b] = min(distance[b], distance[a]+w);
    }
}
```

0.14 Knapsack

```
int knapsack(int W, vector<int> &val, vector<int> &wt) {
    //O(n x W)
    vector<int> dp(W + 1, 0);
    for (int i = 1; i <= wt.size(); i++) {
        for (int j = W; j >= wt[i - 1]; j--) {
            dp[j] = max(dp[j], dp[j - wt[i - 1]] + val[i - 1]);
        }
    }
    return dp[W];
}
```

0.15 Strongly Connected Components

```
vector < vector<int> > g, gr; // adjList and reversed adjList
vector<bool> used;
vector<int> order, component;

void dfs1 (int v) {
    used[v] = true;
    for (size_t i=0; i<g[v].size(); ++i)
        if (!used[ g[v][i] ])
            dfs1 (g[v][i]);
    order.push_back (v);
}

void dfs2 (int v) {
    used[v] = true;
    component.push_back (v);
    for (size_t i=0; i<gr[v].size(); ++i)
        if (!used[ gr[v][i] ])
            dfs2 (gr[v][i]);
}

int main() {
    int n;
    // read n
    for (;;) {
        int a, b;
        // read edge a -> b
        g[a].push_back (b);
        gr[b].push_back (a);
    }

    used.assign (n, false);
    for (int i=0; i<n; ++i)
        if (!used[i])
            dfs1 (i);
    used.assign (n, false);
    for (int i=0; i<n; ++i) {
        int v = order[n-1-i];
        if (!used[v]) {
            dfs2 (v);
            // do something with the found component
            component.clear(); // components are generated in
                  toposort-order
        }
    }
}
```

0.16 Union Find

```
class UnionFind{
    private:
    vector<int> par;
    vector<int> sz;

    public:
     UnionFind(int n){
        par = vector<int>(n);
        iota(par.begin(), par.end(), 0);
        sz = vector<int>(n, 1);
    }

    int find(int u){
        if(par[u] != par[par[u]])
            par[u] = find(par[par[u]]);
        return par[u];
    }

    bool connected(int u, int v){
        u = find(u);
        v = find(v);
        if(u == v)
            return true;
        return false;
    }

    bool join(int u, int v){
        u = find(u);
        v = find(v);
        if(u == v)
            return false;
        if(sz[u] <= sz[v]){
            sz[v] += sz[u];
            par[u] = v;
        }else{
            sz[u] += sz[v];
            par[v] = u;
        }
        return true;
    }
};
```

0.17 Binomial

```
const int MAX = 1e6;

ll fact[MAX+1], inv[MAX+1];

ll power(ll a, ll b) {
    ll res = 1;
    while(b > 0) {
        if(b%2) res = res*a % MOD;
        a = a*a % MOD;
        b /= 2;
    }
    return res;
}

void precompute() {
    fact[0] = 1;
    for(int i=1; i<=MAX; i++)
        fact[i] = fact[i-1] * i % MOD;

    inv[MAX] = power(fact[MAX], MOD-2);
    for(int i=MAX-1; i>=0; i--)
        inv[i] = inv[i+1] * (i+1) % MOD;
}

ll binomial(int n, int k) {
    if(k < 0 || k > n) return 0;
    return fact[n] * inv[k] % MOD * inv[n-k] % MOD;
}
```

0.18 Edmonds Karp

```
//O(V*E)
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
```

```
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

0.19 Aho Corasick Automaton

```
// find in linear time multiple string in a text
// alphabet size
const int K = 70;

// the indices of each letter of the alphabet
int intVal[256];
void init() {
    int curr = 2;
    intVal[1] = 1;
    for(char c = '0'; c <= '9'; c++, curr++) intVal[(int)c] =
        curr;
    for(char c = 'A'; c <= 'Z'; c++, curr++) intVal[(int)c] =
        curr;
    for(char c = 'a'; c <= 'z'; c++, curr++) intVal[(int)c] =
        curr;
}

struct Vertex {
    int next[K];
    vector<int> marks;
    // ^ this can be changed to int mark = -1, if there will be
        no duplicates
    int p = -1;
    char pch;
    int link = -1;
    int exitLink = -1;
    // ^ exitLink points to the next node on the path of suffix
        links which is marked
    int go[K];

    // ch has to be some small char
    Vertex(int _p=-1, char ch=(char)1) : p(_p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void addString(string const& s, int id) {
    int v = 0;
    for (char ch : s) {
        int c = intVal[(int)ch];
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].marks.pb(id);
}

int go(int v, char ch);

int getLink(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(getLink(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int getExitLink(int v) {
    if(t[v].exitLink != -1) return t[v].exitLink;
    int l = getLink(v);
    if(l == 0) return t[v].exitLink = 0;
    if(!t[l].marks.empty()) return t[v].exitLink = l;
    return t[v].exitLink = getExitLink(l);
}

int go(int v, char ch) {
    int c = intVal[(int)ch];
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(getLink(v), ch);
    }
    return t[v].go[c];
}

void walkUp(int v, vector<int>& matches) {
    if(v == 0) return;
    if(!t[v].marks.empty()) {
        for(auto m : t[v].marks) matches.pb(m);
    }
    walkUp(getExitLink(v), matches);
}

// returns the IDs of matched strings.
// Will contain duplicates if multiple matches of the same string
    are found.
vector<int> walk(const string& s) {
    vector<int> matches;
    int curr = 0;
    for(char c : s) {
        curr = go(curr, c);
        if(!t[curr].marks.empty()) {
            for(auto m : t[curr].marks) matches.pb(m);
        }
        walkUp(getExitLink(curr), matches);
    }
    return matches;
}
/* Usage:
 * addString(strs[i], i);
 * auto matches = walk(text);
 * .. do what you need with the matches - count, check if some
     id exists, etc ..
 * Some applications:
 * - Find all matches: just use the walk function
 * - Find lexicographically smallest string of a given length that
     doesn't match any of the given strings:
 * For each node, check if it produces any matches (it either
     contains some marks or walkUp(v) returns some marks).
 * Remove all nodes which produce at least one match. Do DFS
     in the remaining graph, since none of the remaining
     nodes
 * will ever produce a match and so they're safe.
 * - Find shortest string containing all given strings:
 * For each vertex store a mask that denotes the strings which
     match at this state. Start at (v = root, mask = 0),
 * we need to reach a state (v, mask=2^n-1), where n is the
     number of strings in the set. Use BFS to transition
     between states
 * and update the mask.
 */
```

0.20 Graham Convex Hull

```
//O(N)

struct pt {
    double x, y;
    bool operator == (pt const& t) const {
        return x == t.x && y == t.y;
    }
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) ==
    0; }

void convex_hull(vector<pt>& a, bool include_collinear = false
    ) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
```

```
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i],
                include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    if (include_collinear == false && st.size() == 2 && st[0]
            == st[1])
        st.pop_back();

    a = st;
}
```