SWERC Notebook
TELECOM Nancy (France) Equipe 1

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 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));
         }
      }

   }
```

0.3 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;
    }
};
```

0.4 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();
```

0.5 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;
}
```

0.6 Floyd Warshall

```
void floydWarshall(vector<vector<int>> &graph)
{
    int V = graph.size();
    vector<vector<int>> dist = graph;

    // Update the solution matrix by considering all vertices
    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];
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(dist);
}
```

0.7 Z Algo

```
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--;
    }
  }
}
```

0.8 Bellman

```
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);
    }
}
```

0.9 Knapsack

```
int knapsack(int W, vector<int> &val, vector<int> &wt) {

    // Initializing dp vector
    vector<int> dp(W + 1, 0);

    // Taking first i elements
    for (int i = 1; i <= wt.size(); i++) {
```

```
        // Starting from back, so that we also have data of
        // previous computation of i-1 items
        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];
}
```

0.10 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;
}
```

0.11 Edmonds Karp

```
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;
}
```

0.12 Graham Convex Hull

```
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;
}
```