

Contents

1 Basic	1	3.3 AP/Bridge	2
1.1 Default Code	1	3.4 Max flow min cut	2
2 Math	1	4 String	3
2.1 快速幂	1	4.1 Hash	3
2.2 擴展歐幾里得	1	4.2 Zvalue	3
3 Graph	1	5 Geometry	3
3.1 Tarjan SCC	1	5.1 Static Convex Hull	3
3.2 2 SAT	1	6 Data Structure	3
		6.1 Sparse Table	3

1 Basic

1.1 Default Code

```
#include <bits/stdc++.h>
#define int long long
// #pragma GCC target("popcnt")
// #pragma GCC optimize("O3")
using namespace std;

void solve() {
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int tt = 1;
    cin >> tt;
    while (t--) {
        solve();
    }
    return 0;
}
```

2 Math

2.1 快速幂

```
// 根據費馬小定
// 理，若  $a, p$  互質， $a^{p-2}$  為  $a$  在  $\text{mod } p$  時的乘法逆元
int fast_pow(int a, int b, int mod)
{
    //  $a^b \text{ mod }$ 
    int res = 1;
    while(b)
    {
        if(b & 1) res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
```

2.2 擴展歐幾里得

```
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b)
{
    return a * b / gcd(a, b);
}

pair<int, int> ext_gcd
(int a, int b) //擴展歐幾里德  $ax+by = \text{gcd}(a, b)$ 
{
    if (b == 0)
        return {1, 0};
    if (a == 0)
        return {0, 1};
    int x, y;
    tie(x, y) = ext_gcd(b % a, a);
    return make_pair(y - b * x / a, x);
}
```

3 Graph

3.1 Tarjan SCC

```
class tarjan{
    // 1-base
    int time = 1;
    int id = 1;
    stack<int> s;
    vector<int> low;
    vector<int> dfn;
    vector<bool> in_stack;
    void dfs(int node, vector<vector<int>> &graph){
        in_stack[node] = true;
        s.push(node);
        dfn[node] = low[node] = time++;
        for(auto &j : graph[node]){
            if(dfn[j] == 0){
                dfs(j, graph);
                // 看看往下有沒有辦法回到更上面的點
                low[node] = min(low[node], low[j]);
            }
            else if(in_stack[j]){
                low[node] = min(low[node], low[j]);
            }
        }
        vector<int> t; // 儲存這個強連通分量
        if(dfn[node] == low[node]){
            while(s.top() != node){
                t.push_back(s.top());
                in_stack[s.top()] = false;
                scc_id[s.top()] = id;
                s.pop();
            }
            t.push_back(s.top());
            scc_id[s.top()] = id;
            in_stack[s.top()] = false;
            s.pop();
            id++;
        }
        if(!t.empty()) ans.push_back(t);
    }
public:
    vector<int> scc_id;
    vector<vector<int>> ans;
    // ans[i] 代表第 i 個強連通分量裡面包含的點
    // scc_id[i] 代表第 i 個點屬於第幾個強連通分量
    vector
    <vector<int>> scc(vector<vector<int>> &graph){
        int num = graph.size();
        scc_id.resize(num, -1);
        dfn.resize(num, 0);
        low.resize(num, 0);
        in_stack.resize(num, false);
        for(int i = 1; i < num; i++){
            if(dfn[i] == 0) dfs(i, graph);
        }
        return ans;
    }
};
```

3.2 2 SAT

```
// 用
// 下面的 tarjan scc 算法來解 2 sat 問題，若事件 a 發生時，事件 b 必然發生，我們須在  $a \rightarrow b$  建立一條有向
// 用
// cses 的 Giant Pizza 來舉例子，給定 n 個人 m 個配料
// 表，每個人可以提兩個要求，兩個要求至少要被滿足一個
// 3 5
// + 1 + 2
// - 1 + 3
// + 4 - 2
// 以這
// 個例子來說，第一個人要求要加 配料1 或者 配料2 其中
// 一項，第二個人要求不要 配料1 或者 要配料3 其中一項
// 試問能不能滿足所有人的要求，我們可以把 要加
// 配料 i 當作點 i，不加配料 i 當作點  $i + m$  (配料數量)
// 關於第一個人的要求 我們可以看出若不加 配
// 料1 則必定要 配料2 以及 若不加 配料2 則必定要 配料1
// 關於第二個人要求 可看做加了 配料
// 1 就必定要加 配料3 以及 不加 配料3 就必定不加 配料1
```

```
// 以這些條件建立有像圖，並且
// 找尋 scc，若 i 以及 i + m 在同一個 scc 中代表無解
// 若要求解，則若 i 的 scc_id
// 小於 i + m 的 scc_id 則 i 為 true，反之為 false
// tarjan 的模板在上面
cin >> n >> m;

vector<vector<int>> graph(m * 2 + 1);
function<int(int)> tr = [&](int x){
    if(x > m) return x - m;
    return x + m;
};

for(int i = 0; i < n; i++){
    char c1, c2;
    int a, b;
    cin >> c1 >> a >> c2 >> b;
    // a 代表 a 為真，m + a 代表 a 為假
    if(c1 == '-') a += m;
    if(c2 == '-') b += m;
    graph[tr(a)].push_back(b);
    graph[tr(b)].push_back(a);
}

tarjan t;
auto scc = t.scc(graph);

for(int i = 1; i <= m; i++){
    if(t.scc_id[i] == t.scc_id[tr(i)]){
        cout << "IMPOSSIBLE\n";
        return 0;
    }
}

for(int i = 1; i <= m; i++){
    if(t.scc_id[i] < t.scc_id[tr(i)]){
        cout << '+';
    }
    else cout << '-';
    cout << ' ';
}
cout << '\n';
```

3.3 AP/Bridge

```
// adj[u] = adjacent nodes of u
// ap = AP = articulation points
// p = parent
// disc[u] = discovery time of u
// low[u] = 'low' node of u

int dfsAP(int u, int p) {
    int children = 0;
    low[u] = disc[u] = ++Time;
    for (int& v : adj[u]) {
        if (v == p) continue; // we don't want to go back through the same path.
        // if we go back is because we found another way back

        if (!disc[v]) { // if v has not been discovered before
            children++;
            dfsAP(v, u); // recursive DFS call
            if (disc[u] <= low[v]) // condition #1
                ap[u] = 1;
            low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
        } else // if v was already discovered means that we found an ancestor
            low[u] = min(low[u], disc[v]); // finds the ancestor with the least discovery time
    }
    return children;
}

void AP() {
    ap = low = disc = vector<int>(adj.size());
    Time = 0;
    for (int u = 0; u < adj.size(); u++)
        if (!disc[u])
            ap[u] = dfsAP(u, u) > 1; // condition #2
}

// br = bridges, p = parent
```

```
vector<pair<int, int>> br;

int dfsBR(int u, int p) {
    low[u] = disc[u] = ++Time;
    for (int& v : adj[u]) {
        if (v == p) continue; // we don't want to go back through the same path.
        // if we go back is because we found another way back

        if (!disc[v]) { // if v has not been discovered before
            dfsBR(v, u); // recursive DFS call
            if (disc[u] < low[v]) // condition to find a bridge
                br.push_back({u, v});
            low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
        } else // if v was already discovered means that we found an ancestor
            low[u] = min(low[u], disc[v]); // finds the ancestor with the least discovery time
    }
}

void BR() {
    low = disc = vector<int>(adj.size());
    Time = 0;
    for (int u = 0; u < adj.size(); u++)
        if (!disc[u])
            dfsBR(u, u)
}
```

3.4 Max flow min cut

```
#define int long long

// Edmonds-Karp Algorithm Time:  $O(VE^2)$  實際上會快一點
// 記得在 main 裡面 resize graph
// 最小割，找
// 到最少條的邊切除，使得從 src 到 end 的 maxflow 為 0
// 枚舉所有邊 i -> j，src 可
// 以到達 i 但無法到達 j，那這條邊為最小割裡的邊之一

class edge{
public:
    int next;
    int capacity;
    int rev;
    bool is_rev;
    edge(int _n, int _c, int _r, int _ir) :
        next(_n), capacity(_c), rev(_r), is_rev(_ir){};
};

vector<vector<edge>> graph;

void add_edge(int a, int b, int capacity){
    graph[a].push_back
        (edge(b, capacity, graph[b].size(), false));
    graph[b].
        push_back(edge(a, 0, graph[a].size() - 1, true));
}

int dfs(int now, int end
        , int flow, vector<pair<int, int>> &path, int idx){
    if(now == end) return flow;
    auto &e = graph[now][path[idx + 1].second];
    if(e.capacity > 0){
        auto ret = dfs(e.next
            , end, min(flow, e.capacity), path, idx + 1);
        if(ret > 0){
            e.capacity -= ret;
            graph[e.next][e.rev].capacity += ret;
            return ret;
        }
    }
    return 0;
}

vector<pair<int, int>> search_path(int start, int end){
    vector<pair<int, int>> ans;
    queue<int> q;
    vector
        <pair<int, int>> parent(graph.size(), {-1, -1});
    q.push(start);
```

```

while(!q.empty()){
    int now = q.front();
    q.pop();
    for(int i = 0; i < (int)graph[now].size(); i++){
        auto &e = graph[now][i];
        if(e.capacity > 0 and parent[e.next].first == -1){
            parent[e.next] = {now, i};
            if(e.next == end) break;
            q.push(e.next);
        }
    }
}
if(parent[end].first == -1) return ans;
int now = end;
while(now != start){
    auto [node, idx] = parent[now];
    ans.emplace_back(node, idx);
    now = node;
}
ans.emplace_back(start, -1);
reverse(ans.begin(), ans.end());
return ans;
}

int maxflow(int start, int end, int node_num){
    int ans = 0;
    while(1){
        vector<bool> visited(node_num + 1, false);
        auto tmp = search_path(start, end);
        if(tmp.size() == 0) break;
        auto flow = dfs(start, end, 1e9, tmp, 0);
        ans += flow;
    }
    return ans;
}

```

4 String

4.1 Hash

```

vector<int> Pow(int num){
    int p = 1e9 + 7;
    vector<int> ans = {1};
    for(int i = 0; i < num; i++){
        ans.push_back(ans.back() * b % p);
    }
    return ans;
}

vector<int> Hash(string s){
    int p = 1e9 + 7;
    vector<int> ans = {0};
    for(char c:s){
        ans.push_back((ans.back() * b + c) % p);
    }
    return ans;
}

// 閉區間[l, r]
int query
(vector<int> &vec, vector<int> &pow, int l, int r){
    int p = 1e9 + 7;
    int length = r - l + 1;
    return
        (vec[r + 1] - vec[l] * pow[length] % p + p) % p;
}

```

4.2 Zvalue

```

vector<int> z_func(string s1){
    int l = 0, r = 0, n = s1.size();
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i <= r and z[i - l] < r - i + 1) z[i] = z[i - l];
        else{
            z[i] = max(z[i], r - i + 1);
            while(i + z[i] < n and s1[i + z[i]] == s1[z[i]]) z[i]++;
        }
        if(i + z[i] - 1 > r){
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

```

5 Geometry

5.1 Static Convex Hull

```

#define mp(a, b) make_pair(a, b)
#define pb(a) push_back(a)
#define F first
#define S second

template<typename T>
pair<T, T> operator-(pair<T, T> a, pair<T, T> b){
    return mp(a.F - b.F, a.S - b.S);
}

template<typename T>
T cross(pair<T, T> a, pair<T, T> b){
    return a.F * b.S - a.S * b.F;
}

template<typename T>
vector<pair<T, T>> getConvexHull(vector<pair<T, T>> &pnts){
    sort(pnts.begin(), pnts.end(), [](pair<T, T> a, pair<T, T> b){
        return a.F < b.F || (a.F == b.F && a.S < b.S);
    });
    auto cmp = [&](pair<T, T> a, pair<T, T> b){
        return a.F == b.F && a.S == b.S;
    };
    pnts.erase(unique(pnts.begin(), pnts.end(), cmp), pnts.end());
    if(pnts.size() <= 1) return pnts;
    int n = pnts.size();
    vector<pair<T, T>> hull;
    for(int i = 0; i < 2; i++){
        int t = hull.size();
        for(pair<T, T> pnt : pnts){
            while(hull.size() - t >= 2 && cross(hull.back() - hull[hull.size() - 2], pnt - hull[hull.size() - 2]) <= 0){
                hull.pop_back();
            }
            hull.pb(pnt);
        }
        hull.pop_back();
        reverse(pnts.begin(), pnts.end());
    }
    return hull;
}

```

6 Data Structure

6.1 Sparse Table

```

class Sparse_Table{
    // 0-base
    // 要改成找最大把min換成max就好
private:
public:
    int spt[500005][22][2];
    Sparse_Table(vector<int> &ar){
        int n = ar.size();
        for(int i = 0; i < n; i++){
            spt[i][0][0] = ar[i];
            // spt[i][0][1] = ar[i];
        }
        for(int j = 1; (1 << j) <= n; j++){
            for(int i = 0; (i + (1 << j) - 1) < n; i++){
                spt[i][j][0] = min(spt[i + (1 << (j - 1))][j - 1][0], spt[i][j - 1][0]);
                // spt[i][j][1] = max(spt[i + (1 << (j - 1))][j - 1][1], spt[i][j - 1][1]);
            }
        }
    }
    int query_min(int l, int r){
        if(l > r) return INT_MAX;
        int j = (int)lg(r - l + 1);
        // j = 31 - __builtin_clz(r - l + 1);
        return min(spt[l][j][0], spt[r - (1 << j) + 1][j][0]);
    }
    int query_max(int l, int r){
        if(l > r) return INT_MAX;
    }
}

```

```
int j = (int) __lg(r - l + 1);  
///j = 31 - __builtin_clz(r - l + 1);  
return max  
    (spt[l][j][1], spt[r - (1 << j) + 1][j][1]);  
}  
};
```