ACM-ICPC-REFERENCE

Searlese

April 2018

Contents

1 Coding Resources

1.1 Python

1.1.1 SortListOfClass

```
class MyObject:
    def __init__(self, first, second):
        self.first = first
        self.second = second


li = [MyObject('c', 3), MyObject('a', 1),
↪  MyObject('b', 2)]

li.sort(key=lambda x: x.first, reverse=False)
```

1.1.2 Fast IO

```
from sys import stdin, stdout

N = 10
# Reads N chars from stdin (it counts '\n' as
↪  char)
stdin.read(N)
# Reads until '\n' or EOF
line = stdin.readline()
# Reads all lines in stdin until EOF
lines = stdin.readlines()
# Writes a string to stdout, it doesn't adds '\n'
stdout.write(line)
# Writes a list of strings to stdout
stdout.writelines(lines)
# Reads numbers separated by space in a line
numbers = list(map(int, stdin.readline().split()))
```

1.2 C++

1.2.1 ReadLineCpp

```
// when reading lines, don't mix 'cin' with
↪  'getline'
// just use getline and split
string input() {
    string ans;
    // cin >> ws; // eats all whitespaces.
    getline(cin, ans);
    return ans;
}
```

1.2.2 PrintVector

```
void printv(vector<int> v) {
    if (v.size() == 0) {
        cout << "[]" << endl;
        return;
    }
    cout << "[" << v[0];
    for (int i = 1; i < v.size(); i++) {
        cout << ", " << v[i];
    }
    cout << "]" << endl;
}
```

1.2.3 PriorityQueueOfClass

```
struct Object {
    char first;
    int second;
};

int main() {
    auto cmp = [](const Object& a, const Object&
    ↪  b) {return a.second > b.second;};
    priority_queue<Object, vector<Object>,
    ↪  decltype(cmp)> pq(cmp);
    vector<Object> v = {{'c',3}, {'a', 1}, {'b',
    ↪  2}};
    sort(v.begin(), v.end(), cmp);
    return 0;
}
```

1.2.4 SortPair

```
pair<int, int> p;
sort(p.begin(), p.end());
// sorts array on the basis of the first element
```

1.2.5 IntToBinary

```
typedef long long int lli;

lli bitsInInt(lli n) {
    return floor(log2(n) + 1LL);
}

vector<int> intToBitsArray(lli n) {
    n = abs(n);
    if (!n) {
        vector<int> v;
        return v;
    }
    int length = bitsInInt(n);
    int lastPos = length - 1;
    vector<int> v(length);
    for (lli i = lastPos, j = 0; i > -1LL; i--,
    ↪  j++) {
        lli aux = (n >> i) & 1LL;
        v[j] = aux;
    }
    return v;
}
```

1.2.6 SplitString

```
vector<string> split(string str, char token) {
    stringstream test(str);
    string seg;
    vector<string> seglist;
    while (getline(test, seg, token))
        seglist.push_back(seg);
    return seglist;
}
```

1.2.7 IOoptimizationCPP

```
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
}
```

1.2.8 SortVectorOfClass

```
struct Object {
    char first;
    int second;
};

int main() {
    auto cmp = [](const Object& a, const Object&
    ↪  b) {return a.second > b.second;};
    vector<Object> v = {{'c',3}, {'a', 1}, {'b',
    ↪  2}};
    sort(v.begin(), v.end(), cmp);
    printv(v);
    return 0;
}
```

2 Multiple Queries

2.1 Mo

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

2.2 SqrtDecomposition

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

3 Maths

3.1 Combinatorics

3.2 Number Theory

3.3 Probability

3.4 Game Theory

4 Geometry

5 Strings

6 Graphs

6.1 Flow

6.1.1 MaxFlowDinic

```
// cap[a][b] = Capacity from a to b
// flow[a][b] = flow occupied from a to b
// level[a] = level in graph of node a
// Num = number
typedef int Num;
vector<int> level;
vector<vector<int>> ady, cap, flow;
int N, MAXN = 101;

bool levelGraph(int s, int t) {
    level = vector<int>(MAXN);
    level[s] = 1;
    queue<int> q; q.push(s);
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for (int &v : ady[u]) {
            if (!level[v] && flow[u][v] <
            ↪  cap[u][v]) {
                q.push(v);
                level[v] = level[u] + 1;
            }
        }
    }
    return level[t];
}

Num blockingFlow(int u, int t, Num
↪  currPathMaxFlow) {
    if (u == t) return currPathMaxFlow;
    for (int v : ady[u]) {
        Num capleft = cap[u][v] - flow[u][v];
        if ((level[v] == (level[u] + 1)) &&
        ↪  (capleft > 0)) {
            Num pathMaxFlow = blockingFlow(v, t,
            ↪  min(currPathMaxFlow, capleft));
            if (pathMaxFlow > 0) {
                flow[u][v] += pathMaxFlow;
                flow[v][u] -= pathMaxFlow;
                return pathMaxFlow;
            }
        }
    }
    return 0;
}

Num dinicMaxFlow(int s, int t) {
    if (s == t) return -1;
    Num maxFlow = 0;
    while(levelGraph(s, t))
        while (Num flow = blockingFlow(s, t, 1 <<
        ↪  30))
            maxFlow += flow;
    return maxFlow;
}

void addEdge(int u, int v, Num capacity) {
    cap[u][v] = capacity;
    ady[u].push_back(v);
}
```

6.2 TopologicalSort

```
int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)
vector<int> vis; // vis.resize(n)
vector<int> toposorted;

bool toposort(int u) {
    vis[u] = 1;
    for (auto &v : ady[u]) {
        if (v == u || vis[v] == 2)
            continue;
        if (vis[v] == 1 || !toposort(v))
            return false;
    }
    vis[u] = 2;
    toposorted.push_back(u);
    return true;
}

bool toposort() {
    vis.clear();
    for (int u = 0; u < n; u++)
        if (!vis[u])
            if (!toposort(u))
                return false;
    return true;
}
```

6.3 MinimumCut

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

using namespace std;

int main() {

    return 0;
}
```

6.4 UnionFind

```
struct UnionFind {
    vector<int> dad, size;
    int n;
    UnionFind(int N) : n(N), dad(N), size(N, 1) {
        while (--N) dad[N] = N;
    }

    int root(int u) {
        if (dad[u] == u) return u;
        return dad[u] = root(dad[u]);
    }

    bool areConnected(int u, int v) {
        return root(u) == root(v);
    }

    void join(int u, int v) {
        int Ru = root(u), Rv = root(v);
        if (Ru == Rv) return;
        --n, dad[Ru] = Rv;
        size[Rv] += size[Ru];
    }

    int getSize(int u) {
        return size[root(u)];
    }

    int numberOfSets() {
        return n;
    }

};
```

6.5 CycleInUndirectedGraph

```
int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)
vector<bool> vis; // vis.resize(n)
vector<vector<int>> cycles;
vector<int> cycle;
bool flag = false;
int rootNode = -1;

bool hasUndirectedCycle(int u, int prev) {
    vis[u] = true;
    for (auto &v : ady[u]) {
        if (v == u || v == prev)
            continue;
        if (vis[v] || hasUndirectedCycle(v, u)) {
            if (rootNode == -1)
                rootNode = v, flag = true;
            if (flag) {
                cycle.push_back(u);
                if (rootNode == u)
                    flag = false;
            }
            return true;
        }
    }
    return false;
}

bool hasUndirectedCycle() {
    vis.clear();
    for (int u = 0; u < n; u++)
        if (!vis[u]) {
            cycle.clear();
            if (hasUndirectedCycle(u, -1))
                cycles.push_back(cycle);
        }
    return cycles.size() > 0;
}
```

6.6 IsBipartite

```
int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)
```

```
bool isBipartite() {
    vector<int> color(n, -1);
    for (int s = 0; s < n; s++) {
        if (color[s] > -1)
            continue;
        color[s] = 0;
        queue<int> q; q.push(s);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int &v : ady[u]) {
                if (color[v] < 0)
                    q.push(v), color[v] =
                    ↪  !color[u];
                if (color[v] == color[u])
                    return false;
            }
        }
    }
    return true;
}
```

6.7 ShortestPaths

6.7.1 Dijkstra

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

using namespace std;

int n; // max node id >= 0
typedef int Weight;
typedef pair<int, int> NeighCost;
typedef pair<int, NeighCost> ady;

vector<int> parent;
vector<int> dist;

void Dijkstra(int src) {

}

int main() {
    cin >> n;
    ady.resize(n);
    parent.resize(n);
    dist.resize(n);
    return 0;
}
```

6.8 CycleInDirectedGraph

```
int n; // max node id >= 0
vector<vector<int>> ady; // ady.resize(n)
vector<int> vis; // vis.resize(n)
vector<vector<int>> cycles;
vector<int> cycle;
bool flag = false;
int rootNode = -1;

bool hasDirectedCycle(int u) {
    vis[u] = 1;
    for (auto &v : ady[u]) {
        if (v == u || vis[v] == 2)
            continue;
        if (vis[v] == 1 || hasDirectedCycle(v)) {
            if (rootNode == -1)
                rootNode = v, flag = true;
            if (flag) {
                cycle.push_back(u);
                if (rootNode == u)
                    flag = false;
            }
            return true;
        }
    }
    vis[u] = 2;
    return false;
}

bool hasDirectedCycle() {
    vis.clear();
    for (int u = 0; u < n; u++)
        if (!vis[u]) {
            cycle.clear();
            if (hasDirectedCycle(u))
                cycles.push_back(cycle);
        }
    return cycles.size() > 0;
}
```

6.9 FloodFill

```
int n, m, oldColor = 0, color = 1;
vector<vector<int>> mat;

vector<vector<int>> movs = {
    {1, 0},
    {0, 1},
    { -1, 0},
    {0, -1}
};

void floodFill(int i, int j) {
    if (i >= mat.size() || i < 0 || j >=
    ↪  mat[i].size() || j < 0 || mat[i][j] !=
    ↪  oldColor)
        return;
    mat[i][j] = color;
    for (auto move : movs)
        floodFill(i + move[1], j + move[0]);
}

void floodFill() {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (mat[i][j] == oldColor)
                floodFill(i, j);
}
```

6.10 KruskalMST

```
typedef int Weight;
typedef pair<int, int> Edge;
typedef pair<Weight, Edge> Wedge;

vector<Wedge> Wedges; // gets filled from input;
vector<Wedge> mst;

int kruskal() {
    int cost = 0;
    sort(Wedges.begin(), Wedges.end());
    // reverse(Wedges.begin(), Wedges.end());
    UnionFind uf(n);
    for (Wedge &wedge : Wedges) {
        int u = wedge.second.first, v =
        ↪  wedge.second.second;
        if (!uf.areConnected(u, v))
            uf.join(u, v), mst.push_back(wedge),
            ↪  cost += wedge.first;
    }
    return cost;
}
```

7 Rare Topics

8 Data Structures