ICPC template

目 次

1 C++ Preparation

1.1 C++ Compiler

```
g++ -std=c++17 test.cpp -o
```

1.2 C++ Execution

```
./test.out <input> output
```

1.3 C++ Template

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using ull = unsigned long long;
using Graph = vector<vector<int>>;
constexpr int INF = 1e9;
constexpr ll LLINF = 4e18;
#define for_(i,a,b) for(int i=(a);i<(b);++i)
#define rep(i, n) for_(i, 0, n)
#define all(a) (a).begin(), (a).end()
#define rall(a) (a).rbegin(), (a).rend()

//4方向
int dx[4] = {1, 0, -1, 0};
int dy[4] = {0, -1, 0, 1};

//8方向
int ddx[8] = {1,1,1,0,0,-1,-1,-1};
int ddy[8] = {1,0,-1,1,-1,1,0,-1};

int main() {

    return 0;
}
```

2 データ構造

2.1 stack

```
stack<int> st;
// [1, 2, 3]を追加
st.push(1);
st.push(2);
cout << st.top() << endl; // 2
st.push(3);
st.pop(); // 3を削除
st.pop(); // 2を削除
cout << st.top() << endl; // 1
if(st.empty()) // 空ならtrue
```

2.2 queue

```
queue<int> que;
// [1, 2, 3]を追加
que.push(1);
que.push(2);
cout << que.front() << endl; // 1
que.push(3);
que.pop(); // 1を削除
que.pop(); // 2を削除
cout << que.front() << endl; // 3
```

2.3 priority queue

```
// 最大値が先頭にくるキュー
priority_queue<int> pq;
// 最小値が先頭にくるキュー
priority_queue<int,vector<int>,greater<int>>
    pq;
// [1, 3, 5, 6]を追加
pq.push(1);
pq.push(3);
cout << pq.top() << endl; // 3
pq.push(5);
pq.push(6);
pq.pop(); // 6を削除
cout << pq.top() << endl; // 5
```

2.4 map

```
map<string, int> mp;
mp["haruto] = 12;
mp["yuto"] = 13;
cout << mp["haruto"] << endl; // 12
```

2.5 set

```
set<int> st;
// [1, 2]を追加
st.insert(1);
st.insert(2);
st.count(1); // 1が含まれていたら1を返す
st.erase(1); // 1を削除
cout << *st.begin() << endl; // 2
```

2.6 二分探索

```
vector<int> a = { 1,4,4,7,7,8,8,11,13,19};
// lower_bound:key以上の値が初めて現れる位置
auto iter = lower_bound(all(a),4);
// key以上の最小の値を出力
cout << *iter << endl; // 4
// key以上の最小の値が初めて現れる位置を出力
cout << a.begin() - iter << endl; // 1

// upper_bound:
    keyより大きい値が初めて現れる位置
auto iter1 = upper_bound(all(a), 4);
// keyより大きい最小の値を出力
cout << *iter1 << endl; // 7
// keyより大きい最小の値が初出する位置を出力
cout << a.begin() - iter1 << endl; // 3
```

2.7 Union-Find

```
// Union-Find
// グリッドでUFを使う時,(x,y)に対して使うな
    ら(x-1)*W+(y-1)でハッシュ化できる.
struct UnionFind {
    vector<int> par, rank, siz;
    // 構造体の初期化
    UnionFind(int n) : par(n,-1), rank(n,0),
        siz(n,1) { }
    // 根を求める
```

```
    int root(int x) {
    if (par[x]==-1) return x;
    else return par[x] = root(par[x]);
    }
    // x と y が同じグループに属するか (= 根が一致するか)
    bool issame(int x, int y) {
        return root(x)==root(y);
    }
    // x を含むグループと y を含むグループを併合する
    bool unite(int x, int y) {
        int rx = root(x), ry = root(y);
        if (rx==ry) return false;
        // union by rank
        if (rank[rx]<rank[ry]) swap(rx, ry);
        par[ry] = rx; // ry を rx の子とする
        if (rank[rx]==rank[ry]) rank[rx]++;
        siz[rx] += siz[ry];
        return true;
    }
    // x を含む根付き木のサイズを求める
    int size(int x) {
        return siz[root(x)];
    }
};

// union-find木がいくつの連結成分からなるかを返す
long long partial(UnionFind tree){
    long long n = tree.siz.size();
    vector<bool> seen(n, false);
    long long ans = 0;
    for (long long i = 0; i < n; i++){
        if (seen[tree.root(i)]) continue;
        seen[tree.root(i)] = true;
        ans++;
    }
    return ans;
}

// 無向グラフGがいくつの連結成分からなるかを返す
long long partial(Graph &G){
    long long siz = G.size();
    UnionFind ki(siz);
    for (long long i = 0; i < siz; i++){
        long long siz2 = G[i].size();
        for (long long j = 0; j < siz2; j++){
            ki.unite(i, G[i][j]);
        }
    }
    long long ret = partial(ki);
    return ret;
}
```

2.8 BIT (Fenwick Tree)

```
// 数列a(a[0],a[1],…,a[n-1])についての区間和と点更新を扱う
// 区間和,点更新,二分探索はO(log{n})
class BIT {
public:
    //データの長さ
    ll n;
    //データの格納先
    vector<ll> a;
    //コンストラクタ
    BIT(ll n):n(n),a(n+1,0){}

    //a[i]にxを加算する
    void add(ll i,ll x){
        i++;
        if(i==0) return;
        for(ll k=i;k<=n;k+=(k & -k)){
            a[k]+=x;
        }
    }

    //a[i]+a[i+1]+…+a[j]を求める
    ll sum(ll i,ll j){
        return sum_sub(j)-sum_sub(i-1);
    }

    //a[0]+a[1]+…+a[i]を求める
    ll sum_sub(ll i){
        i++;
        ll s=0;
        if(i==0) return s;
        for(ll k=i;k>0;k-=(k & -k)){
            s+=a[k];
        }
        return s;
    }

//a[0]+a[1]+…+a[i]>=xとなる最小のiを求める(任意のkでa[k]>=0が必要)
    ll lower_bound(ll x){
        if(x<=0){
//xが0以下の場合は該当するものなし→0を返す
            return 0;
        }else{
            ll i=0;ll r=1;
// 最大としてありうる区間の長さを取得する
// n以下の最小の二乗のべき(BITで管理する数列の区間で最大のもの)を求める
            while(r<n) r=r<<1;
//区間の長さは調べるごとに半分になる
            for(int len=r;len>0;len=len>>1) {
                //その区間を採用する場合
                if(i+len<n && a[i+len]<x){
                    x-=a[i+len];
                    i+=len;
                }
            }
            return i;
        }
    }
};
```

3 Graph

3.1 深さ優先探索 (再帰関数型)

```
// 深さ優先探索
vector<bool> seen;
void dfs(const Graph &G, int v) {
    seen[v] = true; // v を訪問済にする

    // v から行ける各頂点 next_v について
    for (auto next_v : G[v]) {
        // next_v が探索済だったらスルー
        if (seen[next_v]) continue;
        dfs(G, next_v); // 再帰的に探索
    }
}
```

3.2 深さ優先探索 (スタック型)

```
// 深さ優先探索
stack<int> st;
st.push(start);
while (!st.empty()) {
    int v = st.top(); st.pop();
    if (seen[v]) continue;
    seen[v] = true;
    for (auto next_v : G[v]) {
        if (seen[next_v]) continue;
        st.push(next_v);
    }
}
```

3.3 幅優先探索

```
// 幅優先探索
// 全頂点を「未訪問」に初期化
vector<int> dist(N, -1);
queue<int> que;

// 初期条件 (頂点 0 を初期ノードとする)
dist[0] = 0;
que.push(0); // 0 を橙色頂点にする

// BFS 開始 (キューが空になるまで探索を行う)
while (!que.empty()) {
    // キューから先頭頂点を取り出す
    int v = que.front();
    que.pop();

    // v から辿れる頂点をすべて調べる
    for (int nv : G[v]) {
        // すでに発見済みの頂点は探索しない
        if (dist[nv] != -1) continue;

        // 新たな白色頂点 nv について距離情報
            を更新してキューに追加する
        dist[nv] = dist[v] + 1;
        que.push(nv);
    }
}
```

3.4 ダイクストラ法

```
// 負の重みがない場合の最短経路を求める

// 辺を表す構造体
struct Edge{
    long long to;
    long long cost;
    // その他、必要な情報があれば要素を追加
};
// 隣接リストを表す型
using Gpaph=vector<vector<Edge>>;
// 距離と頂点のペアを表す型
using Pair = pair<long long, long long>;
// 暫定距離を格納する配列
vector<long long> dist;
const long long INF = 1LL << 60;

void dijkstra(const Graph& G, vector<long
    long>& dist, long long start){
    priority_queue<Pair,vector<Pair>,greater<
        Pair>> Q;
```

```
    dist.assign(G.size(),INF);
    // dist[start]=0をして、qに(0,start)を
        push
    Q.emplace(dist[start]=0,start);

    while(!Q.empty()){
    Pair q=Q.top();
    Q.pop();
    long long d=q.first;
    long long v=q.second;

    if(d>dist[v]) continue;

    for(const auto& edge:G[v]){
        long long nextdist = d+edge.cost;
        if(nextdist<dist[edge.to]){
        Q.emplace(dist[edge.to]=nextdist,edge
            .to);
        }
    }
    }
}
```

3.5 ベルマンフォード法

```
// 負の重みがある場合の最短経路を求める
struct Edge {
    long long from;
    long long to;
    long long cost;
};
using Edges = vector<Edge>;
const long long INF = 1LL << 60;

/* bellman_ford(Es,s,t,dis)
    入力: 全ての辺Es, 頂点数V, 開始点 s, 最短
        経路を記録するdis
    出力: 負の閉路が存在するなら ture
    計算量: O(|E||V|)
    副作用:dis が書き換えられる
*/
bool bellman_ford(const Edges &Es, int V, int
    s, vector<long long> &dis) {
    dis.resize(V, INF);
    dis[s] = 0;
    int cnt = 0;
    while (cnt < V) {
        bool end = true;
        for (auto e : Es) {
            if (dis[e.from] != INF && dis[e.
                from] + e.cost < dis[e.to]) {
```

```
                dis[e.to] = dis[e.from] + e.
                    cost;
                end = false;
            }
        }
        if (end) break;
        cnt++;
    }
    return (cnt == V);
}
```

Example Python Code 1

```python
def hello_world():
    print("Hello, World!")

if __name__ == "__main__":
    hello_world()
```

Example Python Code 2

```python
def sort_descending(numbers):
    return sorted(numbers, reverse=True)

numbers = [1, 2, 3, 4, 5]
sorted_numbers = sort_descending(numbers)
print(sorted_numbers)
```

Example Python Code 3

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name}
            and I am {self.age} years old.")

alice = Person("Alice", 30)
alice.greet()
```