This is not a codebook

June 2, 2021

Contents

1 A Hello world **2**

1.1 Aloha . . . 2

2 B Useful **2**

2.1 ExGCD . . . 2

2.2 Fast Power . . . 2

2.3 GCD . . . 2

2.4 LCM . . . 2

2.5 Prime . . . 2

3 C Graph **2**

3.1 BFS and DFS . . . 2

3.1.1 BFS . . . 2

3.1.2 DFS-Path . . . 2

3.1.3 DFS . . . 3

3.2 Disjoint Set . . . 3

3.3 Shortest Path . . . 3

3.3.1 Bellman-Ford . . . 3

3.3.2 Dijkstra . . . 3

3.3.3 Floyd-Warshall . . . 3

1 A Hello world

1.1 Aloha

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

/* compile command */
g++ -std=c++14 -O2 -Wall -Wextra test.cpp -o test
/* script */
#!/bin/bash
g++ -std=c++14 -O2 -Wall -Wextra $1
/* compile script*/
chmod +x build
/* execute */
build test.cpp

/* cin cout */
ios::sync_with_stdio(false);
cin.tie(0); // endl -> '\n'

/* INF */
#define INF 0x3f3f3f3f // int
#define INF 0x3f3f3f3f3f3f3f3f // long long

/* bit */
p(k) denotes the largest power of two that devides k
p(k) = k & -k;
```

2 B Useful

2.1 ExGCD

```
// O(log(min(a,b)))
/* ax + by = gcd(a,b) */

tuple<int,int,int> exgcd(int a, int b){
    if(b == 0) return {1,0,a};
    else{
        int x, y, g;
        tie(x, y, g) = gcd(b, a%b);
        return {y, x-(a/b)*y, g};
    }
}

/*
to calculate a / b = ans (% MOD)
=> find b^(-1), then a * b^(-1) = ans (% MOD)
-----------------------------------------------
to find b^(-1), there are two methods

1.  Fermats  Little Theorem
* MOD is a prime and b is not divisible by MOD
=> find b^(MOD-2) with Fast Power

2.  Bezouts  Theorem
* gcd(b,MOD) == 1
=> find x with exgcd(b,MOD)
*/
```

2.2 Fast Power

```
// O(log exp)
// MOD

ll pw(ll x, ll y){
    ll ans = 1;
    while(y){
        if(y&1) ans *= x;
        x *= x;
        y >>= 1;
    }
    return ans;
}
```

2.3 GCD

```
// O(log(min(a,b)))

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

2.4 LCM

```
// O(log(min(a,b)))

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

2.5 Prime

```
#define MAX_SIZE 1000000 //1e6

bool is_prime[MAX_SIZE];
vector<ll> primes;

void prime(){
    fill(is_prime, is_prime+MAX_SIZE, true);
    is_prime[0] = is_prime[1] = false;
    for(ll i = 2; i < MAX_SIZE; i++){
        if(is_prime[i]){
            primes.push_back(i);
            for(ll j = i*i; j < MAX_SIZE; j+=i){
                is_prime[j] = false;
            }
        }
    }
}
```

3 C Graph

3.1 BFS and DFS

3.1.1 BFS

```
// O(M+N)
// keep parent to find path
int bfs(int s,int t){
    fill(dis, dis+MAX_N, -1);
    queue<int> q;
    dis[s] = 0;
    q.push(s);
    while(!q.empty()){
        int now = q.front();q.pop();
        for(int u:adj[now]){
            if(dis[u] != -1) continue;
            dis[u] = dis[now] + 1;
            q.push(u);
        }
    }
    return dis[t];
}
```

3.1.2 DFS-Path

```
void dfs_path(int now){
    path.push_back(now);
    vis[now] = 1;
    for(auto u:v[now]){
        if(vis[t]) return;
        if(!vis[u]) dfs_path(u);
    }
    if(!vis[t]) path.pop_back();
}
```

3.1.3 DFS

```
// O(M+N)
// cycle detection : a neighbor has been visited and not the
      parent of current node

void dfs(int now){
    vis[now] = true;
    for(auto u:adj[now]){
        if(!vis[u]) dfs(u);
    }
}
```

3.2 Disjoint Set

```
//O(alpha(N))

int boss[MX_N], sz[MX_N]

void init(){
    for(int i = 0; i < MX_N; i++){
        boss[i] = i;
        sz[i] = 1;
    }
}

int findBoss(int x){
    if(boss[x] == x) return x;
    return boss[x] = findBoss(boss[x]);
}

void combine(int a, int b){
    a = findBoss(a);
    b = findBoss(b);
    if(sz[a] < sz[b]) swap(a,b);
    boss[b] = a;
    sz[a] += sz[b];
}

bool same(int a, int b){
    return findBoss(a) == findBoss(b);
}
```

3.3 Shortest Path

3.3.1 Bellman-Ford

```
//O(mn)
/* Detect Negative Cycles */
vector<tuple<int, int, ll>> edge; //a b w
ll dis[MX_N];

// negative cycles might not exit between s and t
// to check connection to start node, skip INF node
// to check connection to terminal node, DFS

//return whether negative cycles exist
bool Bellman_Ford(int s = 1, int t = n){
    fill(dis, dis+n+1, INF);
    dis[s] = 0;
    for(int i = 0; i < n-1; i++){
        for(auto e: edge){
            tie(a, b, w) = e;
            //if(dis[a] == INF) continue;
            dis[b] = min(dis[b], dis[a]+w);
        }
    }
    for(auto e: edge){
        tie(a, b, w) = e;
        //if(dis[a] == INF) continue;
        if(dis[a]+w < dis[b]) return 1; // or DFS(b) and vis[
            t];
    }
    return 0;
}
```

3.3.2 Dijkstra

```
// O(n + mlogm)
/* Only Non-negative weights*/

void Dijkstra(int s){
    priority_queue<pli,vector<pli>,greater<pli>> pq;
    fill(dis,dis+n+1,INF);
    dis[s] = 0;
    pq.push({0,s});
    while(!pq.empty()){
        a = pq.top().second;
        pq.pop();
        if(processed[a]) continue;
        processed[a] = 1;
        for(auto x:adj[a]){
            b = x.first;
            w = x.second;
            if(dis[a]+w < dis[b]){
                dis[b] = dis[a] + w;
                pq.push({dis[b],b});
            }
        }
    }
}
```

3.3.3 Floyd-Warshall

```
// O(n^3)

void init(){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(i != j) dis[i][j] = INF;
        }
    }
}

void Floyd_Warshall(){
    for(int k = 0; k < n; k++){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                dis[i][j] = min(dis[i][j], dis[i][k]+dis[k][j
                    ]);
            }
        }
    }
}
```