Coding-Library

August 15, 2023

Contents

1. Bits

1..1 Bit Manipulation

```
y  =  (x & (1 << i)) // Get the i-th bit
x |=  (1 << i)      // Set the i-th bit
x &= ~(1 << i)      // Clear the i-th bit
(x && !(x &(x-1))) // if power of 2
1 << (32 - __builtin_clz (n - 1)) // Next power of 2
n & (-n)             // LSB
```

2. Data Stuctures

3. Graph

3..1 0-1 BFS

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

3..2 Bellman-Ford

```
#define ar array
#define ll long long

const int MAX_N = 2.5e3 + 1;
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LINF = 1e15;

int n, m, par[MAX_N];
vector<ar<ll,2>> adj[MAX_N];
vector<ll> dist;

void bellman_ford(int s) {
    dist.assign(n + 1, LINF);
    dist[s] = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int u = 1; u <= n; u++) {
            for (auto [v, w] : adj[u]) {
                if (dist[u] + w < dist[v]) {
                    par[v] = u;
                    dist[v] = dist[u] + w;
                }
            }
        }
    }
}

void cycle_detect() {
    int cycle = 0;
    for (int u = 1; u <= n; u++) {
        for (auto [v, w] : adj[u]) {
            if (dist[u] + w < dist[v]) {
                cycle = v;
                break;
            }
        }
    }
    if (!cycle) cout << "NO\n";
    else {
        cout << "YES\n";
        // backtrack to print the cycle
        for (int i = 0; i < n; i++) cycle = par[cycle];
        vector<int> ans; ans.push_back(cycle);
        for (int i = par[cycle]; i != cycle; i = par[i]) ans.
            push_back(i);
        ans.push_back(cycle);
        reverse(ans.begin(), ans.end());
        for (int x : ans) cout << x << " ";
        cout << "\n";
    }
}

void solve() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        adj[u].push_back({v, w});
    }
    bellman_ford(1);
    cycle_detect();
}
```

3..3 DSU

```
struct dsu {
    vt<int> par, sz;
    explicit dsu(int n)
    {
        par.assign(n+1, 0);
        iota(all(par), 0);
        sz.assign(n+1, 1);

    }
    int get_par(int x) {
        if (par[x] == x) return x;
        return par[x] = get_par(par[x]);
    }
    bool join(int a, int b) {
        a = get_par(a), b = get_par(b);
        if (a == b) return false;
        if (sz[a] > sz[b]) swap(a, b);
        par[a] = par[b];
        sz[b] += sz[a];
        return true;
    }
};
```

3..4 Dijkstra

```
vector<int> dist; // answer -> dist[destination]
vector<vector<pair<int,int>>> g; // {cost, to}

void djikstra(int src = 1) {
    dist.resize(g.size(), INFINITY);
    priority_queue<pair<int,int>, vector<pair<int,int>>,
        greater<pair<int,int>>> pq;
    pq.push({dist[src] = 0, src});
    while(!pq.empty()) {
        auto curr = pq.top(); pq.pop();
        for(auto to : g[curr.second])
            if(dist[to.second] > curr.first + to.first)
                pq.push({dist[to.second] = curr.first + to.
                    first, to.second});
    }
}
```

3..5 Floyd

```
// Find all pair shortest paths
// Time complexity: O(n^3)
// Problem link: https://cses.fi/problemset/task/1672
```

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

using namespace std;

#define ar array
#define ll long long

const int MAX_N = 500 + 1;
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LINF = 1e15;

int n, m, q;
ll dist[MAX_N][MAX_N];

void floyd_warshall() { // 4 lines
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist
                    [k][j]);
}

void solve() {
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            dist[i][j] = (i == j) ? 0 : LINF;
    for (int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        dist[u][v] = dist[v][u] = min(dist[u][v], (ll)w);
    }
    floyd_warshall();
    while (q--) {
        int u, v; cin >> u >> v;
        cout << (dist[u][v] < LINF ? dist[u][v] : -1) << "\n"
            ;
    }
}
```

4. MISC

5. Number theory

5..1 Chinese Remainder

```cpp
// k is size of num[] and rem[]. Returns the smallest
// number x such that:
//  x % num[0] = rem[0],
//  x % num[1] = rem[1],
//  ..................
//  x % num[k-2] = rem[k-1]
// Assumption: Numbers in num[] are pairwise coprime
// (gcd for every pair is 1)
int findMinX(int num[], int rem[], int k)
{
    int x = 1; // Initialize result

    // As per the Chinese remainder theorem,
    // this loop will always break.
    while (true)
    {
        // Check if remainder of x % num[j] is
        // rem[j] or not (for all j from 0 to k-1)
        int j;
        for (j=0; j<k; j++ )
            if (x%num[j] != rem[j])
               break;

        // If all remainders matched, we found x
        if (j == k)
            return x;

        // Else try next number
        x++;
    }

    return x;
}
```

5..2 Euler's totient

```cpp
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

5..3 Fast Power

```cpp
ll power(ll b, ll p, ll mod) {
    ll res = 1;
    b = b % mod;
    if (b == 0) return 0;
    while (p)
    {
        if (p & 1)
            res = (res * b) % mod;
        p >>= 1;
        b = (b * b) % mod;
    }
    return res;
}
```

5..4 Josephus

```cpp
int josephus(int n, int k) {
    if (n == 1)
        return 1;
    if (k <= (n + 1) / 2) {
        if (2 * k > n) return 2 * k % n;
        else return 2 * k;
    }
    int c = josephus(n >> 1, k - ( n + 1) / 2);
    if (n & 1) return 2 * c + 1;
    else return 2 * c - 1;
}
```

5..5 Sieve

```cpp
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
```

```
}
```

5..6 nCr with Mod Inverse

```
ll modInverse(ll n, ll mod)
{
    return power(n, mod - 2, mod);
}

ll nCr(ll n, ll r, ll mod)
{
    if (n < r)
        return 0;
    if (r == 0)
        return 1;
    ll fac[n + 1];
    fac[0] = 1;
    for (int i = 1; i <= n; i++)
        fac[i] = (fac[i - 1] * i) % mod;
    return (fac[n] * modInverse(fac[r], mod) % mod
            * modInverse(fac[n - r], mod) % mod)
        % mod;
}
```

5..7 nPr

```
int nPr(int n, int r)
{
 ll ans = 1;
 for (int i = 0; i < (n - r); i++)
  ans *= (n - i);
}
```

6. Strings

6..1 Z

```
//~ The Z-function for this string is an array of length
//~  $n$  where the
//~  $i$ -th element is equal to the greatest number of
     characters starting from the position
//~  $i$  that coincide with the first characters of
//~  $s$ .

//~ In other words,
//~ $z[i] $  is the length of the longest string that is, at
      the same time, a prefix of
//~  $s$  and a prefix of the suffix of
//~  $s$  starting at
//~  $i$ .
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - 1]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}

//~ String compression
//~ Given a string
//~  $s$  of length
//~  $n$ . Find its shortest "compressed" representation,
     that is: find a string
//~  $t$  of shortest length such that
//~  $s$  can be represented as a concatenation of one or
     more copies of
//~  $t$ .

//~ A solution is: compute the Z-function of
//~  $s$ , loop through all
//~  $i$  such that
//~  $i$  divides
//~  $n$ . Stop at the first
//~  $i$  such that
//~ $i + z[i] = n$. Then, the string
//~  $s$  can be compressed to the length
//~  $i$ .
```