Team notebook

Jonathan es muy guapo

November 8, 2019

Contents

1 Binary Search

1.1 Lower Bound

```
int bs_lowerbound(int x) {
        int l = 0, r = MAXN;
        while (r-l>0) {
                int mi = l+(r-l)/2;
                if (a[mi] >= x) r = mi;
                else l = mi+1;
        }
        if (a[l] == x) return l;
        return -1;
} // O(logn)
```

1.2 Unique Element

```
int root(int x) {
        int l = 0, r = MAXN;
        int mi;
        while (l<=r) {
                mi = l+(r-l)/2;
                if (a[mi] == x) return mi;
                else if (a[mi] < x) l = mi+1;
                else r = mi-1;
        }
        if (a[mi] == x) return mi;
        else return -1;
} // O(logn)
```

1.3 Upper Bound

```
int bs_upperbound(int x) {
        int l = 0, r = MAXN;
        while (r-l>0) {
                int mi = l+(r-l+1)/2;
                if (a[mi] <= x) l = mi;
                else r = mi-1;
        }
        if (a[l] == x) return l;
        return -1;
} // O(logn)
```

2 Dynamic Programming

2.1 LCS

```
int LCS(string &s, string &t) {
        forn(i, s.size()+1)  {
                forn(j, t.size()+1) {
                        if (not i or not j) continue;
                        if (s[i-1]==t[j-1]) dp[i][j]=dp[i-1][j-1]+1;
                        else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
                }
        }
        return dp[s.size()][t.size()];
}
void Reconstruct(string &s, string &t) {
        vector<char> ans;
        int i = s.size(), j = t.size();
        while(i>0 && j>0) {
                if (s[i-1] == t[j-1]) {
                        i--; j--;
                        ans.emplace_back(s[i]);
                }
                else if (dp[i-1][j]>dp[i][j-1]) i--;
                else j--;
        }
        for(int i = int(ans.size())-1; i>=0; i--) cout << ans[i];
}
```

2.2 Subsequence Count

```
int findSubsequenceCount(string S, string T) {
    int m = T.length(), n = S.length();
    if (m > n) return 0;
    int mat[m + 1][n + 1];
    for (int i = 1; i <= m; i++) mat[i][0] = 0;
    for (int j = 0; j <= n; j++) mat[0][j] = 1;
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (T[i - 1] != S[j - 1]) mat[i][j] = mat[i][j - 1];
            else mat[i][j] = mat[i][j - 1] + mat[i - 1][j - 1];
        }
    }
    return mat[m][n];
}
```

3 Graph

3.1 Articulaciones con DFS

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

3.2 BFS con parent

```
vector<vector<int>> adj;
int n;
int s;
queue<int> q;
vector<bool> used(n);
vector<int> d(n), p(n);
q.push(s);
used[s] = true;
p[s] = -1;

while (!q.empty()) {
    int v = q.front();
    q.pop();
    for (int u : adj[v]) {
        if (!used[u]) {
            used[u] = true;
            q.push(u);
            d[u] = d[v] + 1;
            p[u] = v;
        }
    }
}
```

3.3 Bipartito con BFS

```
int n;
vector<vector<int>> adj;

vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
```

```
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st);
        side[st] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int u : adj[v]) {
                if (side[u] == -1) {
                    side[u] = side[v] ^ 1
                    q.push(u);
                } else {
                    is_bipartite &= side[u] != side[v];
                }
            }
        }
    }
}

cout << (is_bipartite ? "YES" : "NO") << endl;
```

3.4 Ciclos con DFS

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

3.5 Ciclos negativos

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

bool spfa(int s, vector<int>& d) {
    int n = adj.size();
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;

    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false; // negative cycle
                }
            }
        }
    }
    return true;
}
```

3.6 Componentes conexas con DFS

```
int n;
vector<int> g[MAXN] ;
bool used[MAXN] ;
vector<int> comp ;
void dfs(int v) {
    used[v] = true ;
    comp.push_back(v);
    for (size_t i = 0; i < (int) g[v].size(); ++i) {
        int to = g[v][i];
        if (!used[to])
            dfs(to);
    }
}
void find_comps() {
    for (int i = 0; i < n ; ++i)
        used [i] = false;
    for (int i = 0; i < n ; ++i)
        if (!used[i]) {
            comp.clear();
            dfs(i);
            cout << "Component:" ;
            for (size_t j = 0; j < comp.size(); ++j)
                cout << ' ' << comp[j];
            cout << endl ;
        }
}
```

3.7 DFS con timer

```
vector<vector<int>> adj;
int n;
vector<int> color;
vector<int> time_in, time_out;
int dfs_timer = 0;

void dfs(int v) {
    time_in[v] = dfs_timer++;
    color[v] = 1;
    for (int u : adj[v])
        if (color[u] == 0)
            dfs(u);
    color[v] = 2;
    time_out[v] = dfs_timer++;
}
```

3.8 DIjkstra

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);
    vector<bool> u(n, false);
    d[s] = 0;
    for (int i = 0; i < n; i++) {
        int v = -1;
        for (int j = 0; j < n; j++) {
            if (!u[j] && (v == -1 || d[j] < d[v]))
                v = j;
        }

        if (d[v] == INF)
            break;

        u[v] = true;
        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
            }
        }
    }
}
```

3.9 Dikstra con struct edges

```
struct edge{
    int v, d;
    bool operator<(const edge &o) const {
        return d > o.d;
    }
};
int N, M;
vector<edge> G[MAXN];
int D[MAXN], P[MAXN];

void dijkstra(int raiz){
    priority_queue<edge> Q;
    fill(D, D+N, -1);
    fill(P, P+N, -1);
    D[raiz] = 0;
    Q.push({raiz, 0});
    while( not Q.empty() ){
        auto c = Q.top(); Q.pop();
        if( c.d > D[c.v] ) continue;
        for(auto &e : G[c.v])
            if( D[e.v] == -1 or D[e.v] > c.d + e.d ){
                D[e.v] = c.d + e.d;
                P[e.v] = c.v;
                Q.push({e.v, D[e.v]});
            }
    }
}
```

3.10 DSU

```
struct dsu {
        vector<int> parent;
        vector<int> rank;
        int size;
        dsu(int n) {
                size = n;
                parent.resize(n);
                rank.resize(n);
                for(int i =0; i < size; i++) {
                        make_set(i);
                }
        }
        void make_set(int v) {
            parent[v] = v;
            rank[v] = 0;
        }
        void showP(){
                for(int i = 0; i< size; i++) {
                        cout << find_set(i) << " ";
                }
                cout << endl;
        }
        void union_sets(int a, int b) {
            a = find_set(a);
            b = find_set(b);
            if (a != b) {
                if (rank[a] < rank[b])
                    swap(a, b);
                parent[b] = a;
                if (rank[a] == rank[b])
```

```
                    rank[a]++;
        }
        }
        int find_set(int v) {
            if (v == parent[v])
                return v;
            return parent[v] = find_set(parent[v]);
        }

};
```

3.11 Euler Path

```
int main() {
    int n;
    vector<vector<int>> g(n, vector<int>(n));
    // reading the graph in the adjacency matrix

    vector<int> deg(n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            deg[i] += g[i][j];
    }

    int first = 0;
    while (!deg[first])
        ++first;

    int v1 = -1, v2 = -1;
    bool bad = false;
    for (int i = 0; i < n; ++i) {
        if (deg[i] & 1) {
            if (v1 == -1)
                v1 = i;
            else if (v2 == -1)
                v2 = i;
            else
                bad = true;
        }
    }

    if (v1 != -1)
        ++g[v1][v2], ++g[v2][v1];

    stack<int> st;
    st.push(first);
    vector<int> res;
    while (!st.empty()) {
        int v = st.top();
        int i;
        for (i = 0; i < n; ++i)
            if (g[v][i])
                break;
        if (i == n) {
            res.push_back(v);
            st.pop();
        } else {
            --g[v][i];
            --g[i][v];
            st.push(i);
        }
    }

    if (v1 != -1) {
        for (size_t i = 0; i + 1 < res.size(); ++i) {
            if ((res[i] == v1 && res[i + 1] == v2) ||
                (res[i] == v2 && res[i + 1] == v1)) {
                vector<int> res2;
                for (size_t j = i + 1; j < res.size(); ++j)
                    res2.push_back(res[j]);
                for (size_t j = 1; j <= i; ++j)
                    res2.push_back(res[j]);
                res = res2;
                break;
            }
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (g[i][j])
                bad = true;
        }
    }

    if (bad) {
        cout << -1;
    } else {
        for (int x : res)
            cout << x << " ";
    }
}
```

3.12 Floyd Warshall

```
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}
```

3.13 Kosaraju

```
    vector < vector<int> > g, gr;
    vector<bool> used;
    vector<int> order, component;

    void dfs1 (int v) {
        used[v] = true;
        for (size_t i=0; i<g[v].size(); ++i)
            if (!used[ g[v][i] ])
                dfs1 (g[v][i]);
        order.push_back (v);
    }

    void dfs2 (int v) {
        used[v] = true;
        component.push_back (v);
        for (size_t i=0; i<gr[v].size(); ++i)
            if (!used[ gr[v][i] ])
                dfs2 (gr[v][i]);
    }

    int main() {
        int n;
        ... reading n ...
        for (;;) {
            int a, b;
            ... reading next edge (a,b) ...
            g[a].push_back (b);
            gr[b].push_back (a);
        }

        used.assign (n, false);
        for (int i=0; i<n; ++i)
            if (!used[i])
                dfs1 (i);
        used.assign (n, false);
        for (int i=0; i<n; ++i) {
            int v = order[n-1-i];
            if (!used[v]) {
                dfs2 (v);
                ... printing next component ...
                component.clear();
            }
        }
    }
```

3.14 LCA con Segment Tree

```
struct LCA {
    vector<int> height, euler, first, segtree;
    vector<bool> visited;
    int n;

    LCA(vector<vector<int>> &adj, int root = 0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
        dfs(adj, root);
        int m = euler.size();
        segtree.resize(m * 4);
        build(1, 0, m - 1);
    }

    void dfs(vector<vector<int>> &adj, int node, int h = 0) {
        visited[node] = true;
        height[node] = h;
        first[node] = euler.size();
        euler.push_back(node);
        for (auto to : adj[node]) {
            if (!visited[to]) {
                dfs(adj, to, h + 1);
                euler.push_back(node);
            }
        }
    }

    void build(int node, int b, int e) {
        if (b == e) {
            segtree[node] = euler[b];
        } else {
            int mid = (b + e) / 2;
            build(node << 1, b, mid);
            build(node << 1 | 1, mid + 1, e);
            int l = segtree[node << 1], r = segtree[node << 1 | 1];
            segtree[node] = (height[l] < height[r]) ? l : r;
        }
    }

    int query(int node, int b, int e, int L, int R) {
        if (b > R || e < L)
            return -1;
        if (b >= L && e <= R)
            return segtree[node];
        int mid = (b + e) >> 1;

        int left = query(node << 1, b, mid, L, R);
        int right = query(node << 1 | 1, mid + 1, e, L, R);
        if (left == -1) return right;
        if (right == -1) return left;
        return height[left] < height[right] ? left : right;
    }

    int lca(int u, int v) {
        int left = first[u], right = first[v];
        if (left > right)
            swap(left, right);
        return query(1, 0, euler.size() - 1, left, right);
    }
};
```

3.15 MST Kruskal con DSU

```
vector<int> parent, rank;

void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<Edge> result;
parent.resize(n);
rank.resize(n);
for (int i = 0; i < n; i++)
    make_set(i);

sort(edges.begin(), edges.end());
```

```
for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
        cost += e.weight;
        result.push_back(e);
        union_sets(e.u, e.v);
    }
}
```

3.16 MST Prim

```
int n;
vector<vector<int>> adj; // adjacency matrix of graph
const int INF = 1000000000; // weight INF means there is no edge

struct Edge {
    int w = INF, to = -1;
};

void prim() {
    int total_weight = 0;
    vector<bool> selected(n, false);
    vector<Edge> min_e(n);
    min_e[0].w = 0;

    for (int i=0; i<n; ++i) {
        int v = -1;
        for (int j = 0; j < n; ++j) {
            if (!selected[j] && (v == -1 || min_e[j].w < min_e[v].w))
                v = j;
        }

        if (min_e[v].w == INF) {
            cout << "No MST!" << endl;
            exit(0);
        }

        selected[v] = true;
        total_weight += min_e[v].w;
        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (int to = 0; to < n; ++to) {
            if (adj[v][to] < min_e[to].w)
                min_e[to] = {adj[v][to], v};
        }
    }

    cout << total_weight << endl;
}
```

3.17 Puentes con DFS

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

3.18 Topo Sort

```
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

4 Math

4.1 Closest Power of 2

```
int closePow(int x) {
        if ((x&(x-1)) == 0) return x;
        int c = 1;
        while (x>>=1) c++;
        return (1<<c);
}
```

4.2 Euler's totient

```
//phi(ab) = phi(a)*phi(b)*(d/phi(d))
//   d = gcd(a, b)
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;
}
```

4.3 FastPow Modulo

```
typedef long long ll;
ll fpow(ll a, ll b, ll p) {
    a %= p;
        ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}
```

4.4 FastPow Standard

```
typedef long long ll;
ll binpow(ll a, ll b) {
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

4.5 GCD extended

```
int gcdExtended(int a, int b, int *x, int *y)
{
    // Base Case
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExtended(b%a, a, &x1, &y1);

    // Update x and y using results of
    // recursive call
    *x = y1 - (b/a) * x1;
    *y = x1;

    return gcd;
}
```

4.6 nCr Modulo P

```
int f[MAXN]; //Pre-computed array of factorials modulo p
int inv(int n, int p) { return fpow(n, p-2, p); }
int ncr(int n, int k, int p) {
    if (k == 0 or n == k) return 1;
    int ans = f[n]*inv(f[k])%p * inv(f[n-k])%p;
    return ans%p;
}
```

4.7 Primality Test (Trial division)

```
typedef long long ll;
bool checkprime(ll a) {
        for (ll i = 2; i*i <= a; i++)
                if (not (a%i)) return false;
        return true;
}
```

4.8 Sieves

4.8.1 Biggest Prime Factor

```
int big[n + 1] = {1, 1};
void sieve_bpf(){
        for (int i = 1; i <= n; ++i)
                if (big[i] == 1)
                        for (int j = i; j <= n; j += i)
                                big[j] = i;
} // O(nloglogn)
vector<int> fact(int n) {
        vector<int> factors;
        while (n > 1) {
                factors.push_back(big[n]);
                n /= big[n];
        }
        return factors;
} // O(logn)
```

4.8.2 Divisors

```
int divisors[n + 1];
void sieve_divisors() {
for (int i = 1; i <= n; ++i)
 for (int j = i; j <= n; j += i)
  ++divisors[j];
} // O(nlogn)
```

4.8.3 Smallest Prime Factor

```
typedef long long ll;
int spf[MAXN+1];
void sieve_spf() {
    spf[0] = spf[1] = 1;
    for (int i=2; i<MAXN; i++)
        if (i&1) spf[i] = i;
        else spf[i] = 2;
    for (ll i=3; i*i<MAXN; i++)
        if (spf[i] == i)
                for (ll j=i*i; j<MAXN; j+=i)
                if (spf[j]==j) spf[j] = i;
} // O(nloglogn)
vector<int> fact(int x) {
    vector<int> ret;
    while (x != 1) {
        ret.push_back(spf[x]);
        x = x / spf[x];
    }
    return ret;
} // O(logn)
```

4.8.4 Standard

```
bool pr[MAXN]; //global, all false by default
void sieve() {
        for (ll i = 2; i*i<=MAXN; i++)
                if (not pr[i])
                        for(ll j = i*i; j<=MAXN; j+=i)
                                pr[j] = true; //falses are primes
} // O(nloglogn)
```

5 *utilities*

5.1 Built-in Bitwise Funcitons

```
//This function is used to count the number of ones(set bits) in an
    integer
__builtin_popcount(x); //int
__builtin_popcountll(x); //longlong

//This function is used to check the parity of a number.
//This function returns true(1) if the number has odd parity else it
    returns false(0) for even parity.
__builtin_parity(x); //int
__builtin_parityll(x); //longlong

//This function is used to count the leading zeros of the integer. Note :
    clz = count leading zeros
//This function only accept unsigned values
__builtin_clz(x); //int
__builtin_clzll(x); //longlong

//This function is used to count the trailing zeros of the given integer.
//Note : ctz = count trailing zeros.
__builtin_ctz(x); //int
__builtin_ctzll(x); //longlong
```

5.2 Template

```
#include <bits/stdc++.h>
#define forn(i, n) for (int i = 0; i<(int)n; ++i)
#define forr(i, t, n) for (int i = t; i<n; ++i)
#define rmod(x, y) (((x%y)+y)%y)
using namespace std;
typedef unsigned int uint;
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> pii;
const double PI = acos(-1);
const int MAXN = 1024;

int main() {
        ios::sync_with_stdio(0);
        cin.tie(0);

        return 0;
}
```

5.3 Terminal y Python

```
cd: change directory
ls: listar archivos en dir
clear: limpiar terminal
g++ -std=c++14 filename.cpp
./file.out < in > out
import sys
for line in sys.stdin:
```