CP Algorithms Handbook

Stanisław Fiedler

November 8, 2024

Contents

1 Sorting and Searching

1.1 Quick Sort

```
void qwik_sort(int p, int q, int tab[]){
    int a = p, b = q;
    int m = tab[(p+q)/2];

    while(a <= b){
        while(m > tab[a])
            a++;
        while(m < tab[b])
            b--;
        if(a <= b){
            swap(tab[a],tab[b]);
                a++;
                b--;
        }
    }
    if(b > p)
        qwik_sort(p, b, tab);
    if(a < q)
        qwik_sort(a, q, tab);
}
```

1.2 Merge Sort

```
void merge(int left, int mid, int right, int t[]){
    int a = left, b = mid;
    int i = left;
    while(a < mid && b < right)
        tab[i++] = (t[a] < t[b] ? t[a++] : t[b++]);
    while(a < mid)
        tab[i++] = t[a++];
    while(b < right)
        tab[i++] = t[b++];
    for(int k = left; k < right; k++)
        t[k] = tab[k];
}

void mergesort(int left, int right, int t[]){
    if(left+1 != right){
        int m = (left+right)/2;
        mergesort(left, m, t);
        mergesort(m, right, t);
        merge(left, m, right, t);
    }
}
```

2 Dynamic Programming

2.1 State represenation

2.1.1 All subsets

This way you can generate all subsets in order ready for DP.

```
    int sub = whole_set;
    while(sub > 0){

        sub = (sub-1)&whole_set;
    }
```

3 Graph Algorithms

3.1 Shortest Path

3.1.1 Djikstra

```cpp
priority_queue<pair<long long int ,int>> q;
dist[1] = 0;
q.push({0, 1});
while(!q.empty()){
    int a = q.top().second;
    q.pop();
    if(vis[a]) continue;
    vis[a] = true;
    for(auto v : adj[a]){
        int b = v.first;
        int w = v.second;
        if(dist[b] > dist[a] + w){
            dist[b] = dist[a] + w;
            q.push({-dist[b], b});
        }
    }
}
```

3.1.2 Belman-Ford

```cpp
dist[1] = 0;
for(int i = 1; i <= n-1; i++){
    for(auto e : edges){
        int a, b, w;
        tie(a, b ,w) = e;
        if(dist[b] > dist[a]+w){
            par[b] = a;
            dist[b] = dist[a]+w;
        }
    }
}
for(auto e : edges){
    int a, b, w;
    tie(a, b ,w) = e;
    if(dist[b] > dist[a]+w){
        found = true;
        node = b;
        par[b] = a;
        dist[b] = dist[a]+w;
}
```

3.1.3 Floyd–Warshal

```cpp
long long int dist[SIZE][SIZE] {};
for(int i = 0; i <= n; i++){
    for(int k = 0; k <= n; k++){
        dist[i][k] = __INT64_MAX__/2;
    }
}
for(int i = 1; i <= n; i++){
    dist[i][i] = 0;
    for(auto v : adj[i]){
        int b = v.first;
        long long int w = v.second;
        dist[i][b] = min(w, dist[i][b]);
        dist[b][i] = min(w, dist[i][b]);
    }
}

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]);
        }
    }
}
```

3.2 Union Find

```
int s_size[MX];
int link[MX];

int max_size = 0;
int num;

int find(int x){
    while(x != link[x])
        x = link[x];
    return x;
}

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

void unite(int a, int b){
    a = find(a);
    b = find(b);
    if(s_size[a] < s_size[b])
        swap(a, b);
    s_size[a] += s_size[b];
    max_size = max(max_size, s_size[a]);
    link[b] = a;
    num--;
}

int main(){
    for(int i = 0; i < MX; i++){
        s_size[i] = 1;
        link[i] = i;
    }

    int n, m;
    cin >> n >> m;
    num = n;
    for(int i = 0; i < m; i++){
        int a, b;
        cin >> a >> b;
        if(!same(a, b))
            unite(a, b);
        cout << num << " " << max_size << endl;
    }
    return 0;
}
```

3.3 Path through all edges

```
set <int> adj[MX];

bool vis[MX];

void zero(){
    for(int i = 0; i < MX; i++)
        vis[i] = false;
}

int dfs1(int a){
    int ret = 0;
    vis[a] = true;
    for(int b : adj[a]){
        ret += 1;
        if(!vis[b])
            ret += dfs1(b);
    }
    return ret;
}

int main(){
    int n, m;
    cin >> n >> m;
```

```
    for(int i = 0; i < m; i++){
        int a, b;
        cin >> a >> b;
        adj[a].insert(b);
        adj[b].insert(a);
    }

    zero();
    if(dfs1(1)/2 != m){
        cout << "IMPOSSIBLE";
        return 0;
    }

    for(int i = 1; i <= n; i++){
        if(adj[i].size() & 1){
            cout << "IMPOSSIBLE";
            return 0;
        }
    }

    vector<int> ans;

    stack<int> s;
    s.push(1);
    while(!s.empty()){
        int a = s.top();
        if(adj[a].size() == 0){
            ans.push_back(a);
            s.pop();
        }
        else{
            int b = *adj[a].begin();
            adj[a].erase(adj[a].begin());
            adj[b].erase(a);
            s.push(b);
        }
    }

    for(int a : ans)
        cout << a << " ";


    return 0;
}
```

3.4 Topo Sort

```
int n, m;
stack <int> res;
vector<int> g[100007];
int vis[100007];
bool imp = false;

void dfs(int n){
    vis[n] = 1;
    for(int v : g[n]){
        if(vis[v] == 0)
            dfs(v);
        else if(vis[v] == 1)
            imp = true;
    }
    vis[n] = 2;
    res.push(n);
}

int main(){
    cin >> n >> m;
    for(int i = 0 ; i < m; i++){
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
    }

```

```
    for(int i = 1; i <= n; i++){
        if(vis[i] == 0){
            dfs(i);
        }
    }

    if(!imp){
        while(!res.empty()){
            cout << res.top() << " ";
            res.pop();
        }
    }
    else{
        cout << "IMPOSSIBLE";
    }

    return 0;
}
```

3.5 Flows and Cuts

All problems in this section can be solved using the same basic algorithm defined in 3.5.1.

3.5.1 Maximum Flow

This implementation of Ford-Fulkerson algorithm (known as Edmonds-Karp algorithm) uses BFS to check if it's possible to expand the flow through the graph and assign depths from the source. Later it uses DFS to expand te flow.

```
ll flow[MX][MX];
ll used[MX][MX];

vector<int> adj[MX];
vector<int> radj[MX];

bool vis[MX];

bool bfs(int s, int lvl[], const int n){
    queue<int> q;
    q.push(s);
    lvl[s] = 1;
    while(!q.empty()){
        int p = q.front();
        q.pop();
        for(int v : adj[p]){
            if(lvl[v] == 0 and flow[p][v] - used[p][v] > 0){
                lvl[v] = lvl[p]+1;
                q.push(v);
            }
        }
        for(int v : radj[p]){
            if(lvl[v] == 0 and used[v][p] > 0){
                lvl[v] = lvl[p]+1;
                q.push(v);
            }
        }
    }
    return lvl[n] > 0;
}

int dfs(int x, ll val, const int lvl[], const int n){
    vis[x] = true;
    if(x == n){
        return val;
    }
    for(int v : adj[x]){
        if(lvl[v] == lvl[x]+1 and flow[x][v] - used[x][v] > 0 and !vis[v]){
            int r = dfs(v, min(val, flow[x][v]-used[x][v]), lvl, n);
            if(r > 0){
                used[x][v] += r;
                return r;
            }
        }
```

```
    }
    for(int v : radj[x]){
        if(lvl[v] == lvl[x]+1 and used[v][x] > 0 and !vis[v]){
            int r = dfs(v, min(val, used[v][x]), lvl, n);
            if(r > 0){
                used[v][x] -= r;
                return r;
            }
        }
    }
    return 0;
}
```

To find the flow:

```
    long long int res = 0;

    while(bfs(1, lvl, n)){
        res += dfs(1, __INT_MAX__, lvl, n);
        for(int i = 0 ; i < MX; i++){
            vis[i] = false;
            lvl[i] = 0;
        }
    }
```

3.5.2 Minimum Cut

To find Minimum Cut in a graph we need to find the Maximum Flow and check which edges connect two created disjont sets of nodes. This modification to finds them:

```
    long long int res = 0;

    while(bfs(1, lvl, n)){
        res += dfs(1, __INT_MAX__, lvl, n);
        for(int i = 0 ; i < MX; i++){
            vis[i] = false;
            lvl[i] = 0;
        }
    }
    cout << res << endl;
    for(int i = 0 ; i < MX; i++){
        vis[i] = false;
        lvl[i] = 0;
    }

    bfs(1, lvl, n);
    for(int a = 1; a <= n; a++){
        if(lvl[a] > 0){
            for(int b : adj[a]){
                if(lvl[b] == 0){
                    cout << a << " " << b << endl;
                }
            }
        }
    }
```

3.5.3 Maximum Matching

3.6 2SAT

Given logical formula in the conjunctive normal form:

$$(a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge ... \wedge (a_n \vee b_n)$$

we can eliminate disjunction by repacing each $(a_i \vee b_i)$ element with pair:

$$\neg a_i \rightarrow b_i \wedge \neg b_i \rightarrow a_i$$

```
vector<int> adj[MX];
vector<int> Tadj[MX];
stack<int> scc_stack;

int scc[MX];

bool vis[MX];
int scc_val[MX];

int not_node(int a){
    if(!(a&1))
        return a+1;
    else
        return a-1;
}

void dfs1(int a){
    vis[a] = true;
    for(int b : adj[a]){
        if(!vis[b])
            dfs1(b);
    }
    scc_stack.push(a);
}

void dfs2(int a, int scc_n){
    vis[a] = true;
    for(int b : Tadj[a]){
        if(!vis[b])
            dfs2(b, scc_n);
    }
    scc[a] = scc_n;
}

void zero(){
    for(int i = 0 ; i < MX; i++)
        vis[i] = false;
}

int main(){
    int n, m;
    cin >> n >> m;
    for(int i = 0 ; i < n; i++){
        char op;
        int a, b;
        cin >> op >> a;
        if(op == '+'){
            a *= 2;
        }
        else{
            a *= 2;
            a++;
        }
        cin >> op >> b;
        if(op == '+'){
            b *= 2;
        }
        else{
            b *= 2;
            b++;
        }
        Tadj[a].push_back(not_node(b));
        Tadj[b].push_back(not_node(a));
        adj[not_node(b)].push_back(a);
        adj[not_node(a)].push_back(b);
    }

    for(int i = 1 ; i <= 2*m+1; i++){
        if(!vis[i])
            dfs1(i);
    }

    zero();
```

```
    int scc_n = 1;
    while(!scc_stack.empty()){
        int a = scc_stack.top();
        scc_stack.pop();
        if(scc[a] == 0){
            dfs2(a, scc_n);
            scc_n++;
        }
    }

    for(int i = 2; i <= m*2; i+=2){
        if(scc[i] == scc[i+1]){
            cout << "IMPOSSIBLE" << endl;
            return 0;
        }
    }
    // 1 - false; 2 - true;
    for(int i = 2; i <= 2*m; i+=2){
        int a = scc[i];
        int b = scc[i+1];
        if(scc_val[a] != 0){
            scc_val[b] = (scc_val[a] == 1 ? 2 : 1);
        }
        if(scc_val[b] != 0){
            scc_val[a] = (scc_val[b] == 1 ? 2 : 1);
        }
        if(a < b){
            scc_val[a] = 1;
            scc_val[b] = 2;
        }
        else{
            scc_val[a] = 2;
            scc_val[b] = 1;
        }
    }

    for(int i = 2; i <= m*2; i+=2)
        cout << (scc_val[scc[i]] == 2 ? "+" : "-") << " ";

    return 0;
}
```

4 Range Queries

4.1 Segment trees

BASE size table:

a	$2 \cdot 10^5$	$5 \cdot 10^5$	10^6
$log_2 a$	18	19	20

4.1.1 Point-Range Trees

```
int sal[2*BASE];

void add(int p, int val){
    p += BASE;
    sal[p] += val;
    while(p > 0){
        p >>= 1;
        sal[p] = sal[2*p] + sal[2*p+1];
    }
}

int query(int p, int q){
    int ret =0 ;
    p += BASE -1;
    q += BASE +1;
    while(p>>1 != q>>1){
        if(not(p&1)){
```

```
            ret += sal[p+1];
        }
        if(q&1){
            ret += sal[q-1];
        }
        p >>= 1;
        q >>= 1;
    }
    return ret;
}
```

4.1.2 Range-Range Trees

1. range ADD insert, range MAX value query

```
void max_add(int a, int b, int val, int k=1, int x=0, int y=BASE-1){
    if(a <= x and y <= b){
        max_tree[k][1] += val;
        return;
    }
    max_tree[2*k][1] += max_tree[k][1];
    max_tree[2*k+1][1] += max_tree[k][1];
    max_tree[k][0] += max_tree[k][1];
    max_tree[k][1] = 0;

    int d = (x+y)/2;
    if(a <= d){
        max_add(a, b, val, 2*k, x, d);
        }
    if(b > d){
        max_add(a, b, val, 2*k+1, d+1, y);
    }
    max_tree[k][0] = max(max_tree[2*k+1][0] + max_tree[2*k+1][1], max_tree[2*k][0]
    + max_tree[2*k][1]);

}
```

```
ll max_query(int a, int b, int k=1, int x=0, int y=BASE-1){
    if(a <= x and y <= b){
        return max_tree[k][0] + max_tree[k][1];
    }
    max_tree[2*k][1] += max_tree[k][1];
    max_tree[2*k+1][1] += max_tree[k][1];
    max_tree[k][0] += max_tree[k][1];
    max_tree[k][1] = 0;
    int d = (x+y)/2;
    ll ret = -INF;
    if(a <= d){
        ret = max(ret, max_query(a, b, 2*k, x, d));
    }
    if(b > d){
        ret = max(ret, max_query(a, b, 2*k+1, d+1, y));
    }
    return ret;
}
```

2. range ADD insert, range SUM qyerry

```
void sum_add(int a, int b, int val, int k=1, int x=0, int y=BASE-1){
    if(a <= x and y <= b){
        sum_tree[k][1] += val;
        return;
    }
    sum_tree[2*k][1] += sum_tree[k][1];
    sum_tree[2*k+1][1] += sum_tree[k][1];
    sum_tree[k][0] += sum_tree[k][1]*(y-x+1);
    sum_tree[k][1] = 0;
    int d = (x+y)/2;
    if(a <= d){
        int w = (d <= b ? d : b)+1;
        w -= (x <= a ? a : x);
        sum_tree[k][0] += val*(w);
        sum_add(a, b, val, 2*k, x, d);
    }
```

```
    if(b > d){
        int w = (y <= b ? y : b)+1;
        w -= (d+1 <= a ? a : d+1);
        sum_tree[k][0] += val*(w);
        sum_add(a, b, val, 2*k+1, d+1, y);
    }
}
```

```
ll sum_query(int a, int b, int k=1, int x=0, int y=BASE-1){
    if(a <= x and y <= b){
        return sum_tree[k][0] + sum_tree[k][1]*(y-x+1);
    }
    sum_tree[2*k][1] += sum_tree[k][1];
    sum_tree[2*k+1][1] += sum_tree[k][1];
    sum_tree[k][0] += sum_tree[k][1]*(y-x+1);
    sum_tree[k][1] = 0;
    int d = (x+y)/2;
    ll ret = 0;
    if(a <= d){
        ret += sum_query(a, b, 2*k, x, d);
    }
    if(b > d){
        ret += sum_query(a, b, 2*k+1, d+1, y);
    }
    return ret;
}
int sal[2*BASE];
```

4.2 Max Subarray Sum

```
ll ssum[2*BASE];
ll prfx[2*BASE];
ll sufx[2*BASE];
ll tsum[2*BASE];

void insert(int val, int p){
    p+=BASE;
    ssum[p] = val;
    prfx[p] = val;
    sufx[p] = val;
    tsum[p] = val;
    int l, r;
    while(p){
        p >>= 1;
        l = 2*p;
        r = 2*p+1;
        tsum[p] = tsum[l] + tsum[r];
        prfx[p] = max(prfx[l], tsum[l] + prfx[r]);
        sufx[p] = max(sufx[l] + tsum[r], sufx[r]);
        ssum[p] = max(sufx[l] + prfx[r], max(ssum[l], ssum[r]));
    }
}
```

5 Tree Algorithms

5.1 Binary Lifitng

```
int main(){
    int n, q;
    cin >> n >> q;
    boss[1][0] = 0;
    for(int i = 2; i <= n; i++){
        cin >> boss[i][0];
    }
    for(int i = 1; i < LOG_MX; i++){
        for(int emp = 1; emp <= MX; emp++){
            boss[emp][i] = boss[boss[emp][i-1]][i-1];
        }
    }
```

5.2 Tree Traversal

```
vector <int> adj[MX];
int subtree[MX];
int flat[MX];
int vals[MX];

int idx = 1;
int dfs(int a, int p){
    flat[a] = idx;
    idx++;
    for(int b : adj[a]){
        if(b == p)
            continue;
        subtree[a] += dfs(b, a)+1;
    }
    return subtree[a];
}
```

5.3 LCA

LCA can be found using binary lifting or by building a segment tree build on pre- and postorder.

6 String Algorithms

6.1 Trie

```
int trie[TMX][30];
bool stop[TMX];
int next_node = 1;
int dp[MX];

void insert(string s){
    int idx = 0;
    for(char c : s){
        if(trie[idx][c-'a'] == 0){
            trie[idx][c-'a'] = next_node;
            next_node ++;
        }
        idx = trie[idx][c-'a'];
    }
    stop[idx] = true;
}

int main(){
    string text;
    int n;
    cin >> text >> n;
    for(int i = 0; i < n; i++){
        string s;
        cin >> s;
        insert(s);
    }
```

6.2 Pattern Finding

6.2.1 KMP

```
int pi[MX];

int main(){
    string text, pattern;
    cin >> text >> pattern;
    string s = pattern + "+" + text;
    for(int i = 1; i < s.size(); i++){
        int j = pi[i-1];
        while(j > 0 and s[i] != s[j])
            j = pi[j-1];
        if(s[i] == s[j])
            j++;
```

```
            pi[i] = j;
    }

    int res = 0;
    for(int i = 0; i < s.size(); i++){
        if(pi[i] == pattern.size())
            res++;
    }

    cout << res;
    return 0;
```

6.2.2 Hashing

```
const int MOD = 1e9+9;
const int P = 9973;
const int MX = 1e6+7;

ll ppow[MX];
ll pfx_hash[MX];

int res = 0;

int main(){
    string s, pattern;
    cin >> s >> pattern;
    ppow[0] = 1;
    for(int i = 1 ; i < MX; i++){
       ppow[i] = (ppow[i-1]*P)%MOD;
    }

    ll pattern_hash = 0;
    for(int i = 0; i < pattern.size(); i++){
        pattern_hash = (pattern_hash + (pattern[i]-'a'+1)*ppow[i])%MOD;
    }

    for(int i = 0; i < s.size(); i++){
        pfx_hash[i+1] = (pfx_hash[i] + (s[i]-'a'+1)*ppow[i])%MOD;
    }

    for(int i = 0; i+pattern.size()-1 < s.size(); i++){
        int a = i, b = i+pattern.size();
        ll sub_hash = pfx_hash[b] - pfx_hash[a]+MOD;
        sub_hash %= MOD;
        if(sub_hash == (pattern_hash*ppow[a])%MOD){
            res++;
        }
    }
    cout << res << endl;
    return 0;
```

6.3 Palindormes

Fiding longest palindromic substring.

```
int p[MX];

int main(){
    string txt1, txt = "#";
    cin >> txt1;
    for(char c : txt1){
        string s{c};
        txt += s + "#";
    }
    txt = "^" + txt + "$";
    int l = 1, r = 1;
    for(int i = 1; i < txt.size(); i++){
        p[i] = max(0, min(r-i, p[l+r-i]));
        while(txt[i-p[i]] == txt[i+p[i]])
            p[i]++;
        if(i+p[i] > r){
            l = i-p[i];
            r = i+p[i];
        }
    }
    int maxi = 0;
    int imaxi;
    for(int i = 0; i < txt.size(); i++){
        if(p[i] > maxi){
            maxi = p[i];
            imaxi = i;
        }
    }
    string res = txt.substr(imaxi-maxi+1,maxi*2-1);
    for(auto c : res){
        if(c != '#')
            cout << c;
    }
```

7 Mathematics

7.1 Fermat's Theorem

Modular inverse:

$$\text{If m is prime, then: } x^{-1} \, mod \, m = x^{m-2} \, mod \, m$$

$$x^{\varphi(m)} \, mod \, m = x^{k \, mod \, (m-1)} \, mod \, m$$

```
int exp(int a, int b, int MOD){
    if(b == 0)
        return 1;
    if(b & 1){
        return ((ll)a * exp(a, b-1, MOD))%MOD;
    }
    ll tmp = exp(a, b/2, MOD);
    return (tmp*tmp)%MOD;
}
```

7.2 Fast Fibonacci

```
void mul(ll a[][2], ll b[][2]){
    ll res[2][2];
    res[0][0] = a[0][0]*b[0][0] + a[0][1]*b[1][0];
    res[0][1] = a[0][0]*b[0][1] + a[0][1]*b[1][1];
    res[1][0] = a[1][0]*b[0][0] + a[1][1]*b[1][0];
    res[1][1] = a[1][0]*b[0][1] + a[1][1]*b[1][1];
```

```cpp

    a[0][0] = res[0][0] % MOD;
    a[0][1] = res[0][1] % MOD;
    a[1][0] = res[1][0] % MOD;
    a[1][1] = res[1][1] % MOD;
}

int main(){
    ll n;
    cin >> n;
    ll m_pow[2][2] =
            {{0, 1},
             {1, 1}};
    ll m[2][2] =
            {{1, 0},
             {0, 1}};

    auto pow = bitset<64> (n);

    for(int i = 0 ; i < 64; i++){
        if(pow[i]){
            mul(m,m_pow);
        }
        mul(m_pow, m_pow);
    }

    cout << m[0][1];

    return 0;
}
```

8 Geometry

9 Other Algorithms