

CP Algorithms Handbook

Stanisław Fiedler

October 26, 2024

Contents

1	Sorting and Searching	3
1.1	4SUM	3
2	Dynamic Programming	3
2.1	State representation	3
2.1.1	All subsets	3
2.1.2	Permutations to Subsets	3
3	Graph Algorithms	3
3.1	Union Find	3
3.2	Path through all edges	4
3.3	Topo Sort	5
3.4	Flows and Cuts	5
3.4.1	Maximum Flow	5
3.4.2	Minimum Cut	6
3.4.3	Maximum Matching	7
3.5	2SAT	7
4	Range Queries	9
4.1	Segment trees	9
4.1.1	Point-Range Trees	9
4.1.2	Range-Range Trees	9
5	Tree Algorithms	10
5.1	Binary Liftng	10
5.2	Tree Traversal	11
5.3	LCA	11
6	String Algorithms	11
6.1	Trie	11
6.2	Pattern Finding	11
6.2.1	KMP	11
6.2.2	Hashing	12
6.3	Palindormes	12
7	Mathematics	13
7.1	Fermat's Theorem	13
7.2	Fast Fibonacci	13
7.3	Combinatorics	14
7.3.1	a	14
8	Geometry	14
9	Other Algorithms	14

Contents

1 Sorting and Searching

1.1 4SUM

2 Dynamic Programming

2.1 State representation

2.1.1 All subsets

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

```
1  int sub = whole_set;
2  while(sub > 0){
3
4      sub = (sub-1)&whole_set;
5  }
```

2.1.2 Permutations to Subsets

3 Graph Algorithms

3.1 Union Find

```
1  int s_size[MX];
2  int link[MX];
3
4  int max_size = 0;
5  int num;
6
7  int find(int x){
8      while(x != link[x])
9          x = link[x];
10     return x;
11 }
12
13 bool same(int a, int b){
14     return find(a) == find(b);
15 }
16
17 void unite(int a, int b){
18     a = find(a);
19     b = find(b);
20     if(s_size[a] < s_size[b])
21         swap(a, b);
22     s_size[a] += s_size[b];
23     max_size = max(max_size, s_size[a]);
24     link[b] = a;
25     num--;
26 }
27
28 int main(){
29     for(int i = 0; i < MX; i++){
30         s_size[i] = 1;
31         link[i] = i;
32     }
33
34     int n, m;
35     cin >> n >> m;
36     num = n;
37     for(int i = 0; i < m; i++){
38         int a, b;
39         cin >> a >> b;
40         if(!same(a, b))
41             unite(a, b);
42         cout << num << " " << max_size << endl;
43     }
44     return 0;
45 }
```

3.2 Path through all edges

```
1 set <int> adj[MX];
2
3 bool vis[MX];
4
5 void zero(){
6     for(int i = 0; i < MX; i++){
7         vis[i] = false;
8     }
9
10 int dfs1(int a){
11     int ret = 0;
12     vis[a] = true;
13     for(int b : adj[a]){
14         ret += 1;
15         if(!vis[b])
16             ret += dfs1(b);
17     }
18     return ret;
19 }
20
21 int main(){
22     int n, m;
23     cin >> n >> m;
24     for(int i = 0; i < m; i++){
25         int a, b;
26         cin >> a >> b;
27         adj[a].insert(b);
28         adj[b].insert(a);
29     }
30
31     zero();
32     if(dfs1(1)/2 != m){
33         cout << "IMPOSSIBLE";
34         return 0;
35     }
36
37     for(int i = 1; i <= n; i++){
38         if(adj[i].size() & 1){
39             cout << "IMPOSSIBLE";
40             return 0;
41         }
42     }
43
44     vector<int> ans;
45
46     stack<int> s;
47     s.push(1);
48     while(!s.empty()){
49         int a = s.top();
50         if(adj[a].size() == 0){
51             ans.push_back(a);
52             s.pop();
53         }
54         else{
55             int b = *adj[a].begin();
56             adj[a].erase(adj[a].begin());
57             adj[b].erase(a);
58             s.push(b);
59         }
60     }
61
62     for(int a : ans)
63         cout << a << " ";
64
65
66     return 0;
67 }
```

3.3 Topo Sort

```
1 int n, m;
2 stack<int> res;
3 vector<int> g[100007];
4 int vis[100007];
5 bool imp = false;
6
7 void dfs(int n){
8     vis[n] = 1;
9     for(int v : g[n]){
10         if(vis[v] == 0)
11             dfs(v);
12         else if(vis[v] == 1)
13             imp = true;
14     }
15     vis[n] = 2;
16     res.push(n);
17 }
18
19 int main(){
20     cin >> n >> m;
21     for(int i = 0 ; i < m; i++){
22         int a, b;
23         cin >> a >> b;
24         g[a].push_back(b);
25     }
26
27     for(int i = 1; i <= n; i++){
28         if(vis[i] == 0){
29             dfs(i);
30         }
31     }
32
33     if(!imp){
34         while(!res.empty()){
35             cout << res.top() << " ";
36             res.pop();
37         }
38     }
39     else{
40         cout << "IMPOSSIBLE";
41     }
42
43     return 0;
44 }
```

3.4 Flows and Cuts

All problems in this section can be solved using the same basic algorithm defined in 3.4.1.

3.4.1 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 the flow.

```
1 ll flow[MX][MX];
2 ll used[MX][MX];
3
4 vector<int> adj[MX];
5 vector<int> radj[MX];
6
7 bool vis[MX];
8
9 bool bfs(int s, int lvl[], const int n){
10     queue<int> q;
11     q.push(s);
12     lvl[s] = 1;
13     while(!q.empty()){
14         int p = q.front();
15         q.pop();
```

```

16         for(int v : adj[p]){
17             if(lvl[v] == 0 and flow[p][v] - used[p][v] > 0){
18                 lvl[v] = lvl[p]+1;
19                 q.push(v);
20             }
21         }
22         for(int v : radj[p]){
23             if(lvl[v] == 0 and used[v][p] > 0){
24                 lvl[v] = lvl[p]+1;
25                 q.push(v);
26             }
27         }
28     }
29     return lvl[n] > 0;
30 }
31
32 int dfs(int x, ll val, const int lvl[], const int n){
33     vis[x] = true;
34     if(x == n){
35         return val;
36     }
37     for(int v : adj[x]){
38         if(lvl[v] == lvl[x]+1 and flow[x][v] - used[x][v] > 0 and !vis[v]){
39             int r = dfs(v, min(val, flow[x][v]-used[x][v]), lvl, n);
40             if(r > 0){
41                 used[x][v] += r;
42                 return r;
43             }
44         }
45     }
46     for(int v : radj[x]){
47         if(lvl[v] == lvl[x]+1 and used[v][x] > 0 and !vis[v]){
48             int r = dfs(v, min(val, used[v][x]), lvl, n);
49             if(r > 0){
50                 used[v][x] -= r;
51                 return r;
52             }
53         }
54     }
55     return 0;
56 }

```

To find the flow:

```

1     long long int res = 0;
2
3     while(bfs(1, lvl, n)){
4         res += dfs(1, __INT_MAX__, lvl, n);
5         for(int i = 0 ; i < MX; i++){
6             vis[i] = false;
7             lvl[i] = 0;
8         }
9     }

```

3.4.2 Minimum Cut

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

```

1     long long int res = 0;
2
3     while(bfs(1, lvl, n)){
4         res += dfs(1, __INT_MAX__, lvl, n);
5         for(int i = 0 ; i < MX; i++){
6             vis[i] = false;
7             lvl[i] = 0;
8         }
9     }
10    cout << res << endl;
11    for(int i = 0 ; i < MX; i++){
12        vis[i] = false;
13        lvl[i] = 0;
14    }

```

```

15     bfs(1, lvl, n);
16     for(int a = 1; a <= n; a++){
17         if(lvl[a] > 0){
18             for(int b : adj[a]){
19                 if(lvl[b] == 0){
20                     cout << a << " " << b << endl;
21                 }
22             }
23         }
24     }
25 }

```

3.4.3 Maximum Matching

3.5 2SAT

Given logical formula in the conjunctive normal form:

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

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

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

```

1  #include <iostream>
2  #include <vector>
3  #include <stack>
4
5  using namespace std;
6
7  const int MX = 2e5+7;
8
9  vector<int> adj[MX];
10 vector<int> Tadj[MX];
11 stack<int> scc_stack;
12
13 int scc[MX];
14
15 bool vis[MX];
16 int scc_val[MX];
17
18 int not_node(int a){
19     if(!(a&1))
20         return a+1;
21     else
22         return a-1;
23 }
24
25 void dfs1(int a){
26     vis[a] = true;
27     for(int b : adj[a]){
28         if(!vis[b])
29             dfs1(b);
30     }
31     scc_stack.push(a);
32 }
33
34 void dfs2(int a, int scc_n){
35     vis[a] = true;
36     for(int b : Tadj[a]){
37         if(!vis[b])
38             dfs2(b, scc_n);
39     }
40     scc[a] = scc_n;
41 }
42
43 void zero(){
44     for(int i = 0 ; i < MX; i++)
45         vis[i] = false;
46 }
47

```

```

48 int main(){
49     int n, m;
50     cin >> n >> m;
51     for(int i = 0 ; i < n; i++){
52         char op;
53         int a, b;
54         cin >> op >> a;
55         if(op == '+'){
56             a *= 2;
57         }
58         else{
59             a *= 2;
60             a++;
61         }
62         cin >> op >> b;
63         if(op == '+'){
64             b *= 2;
65         }
66         else{
67             b *= 2;
68             b++;
69         }
70         Tadj[a].push_back(not_node(b));
71         Tadj[b].push_back(not_node(a));
72         adj[not_node(b)].push_back(a);
73         adj[not_node(a)].push_back(b);
74     }
75
76     for(int i = 1 ; i <= 2*m+1; i++){
77         if(!vis[i])
78             dfs1(i);
79     }
80
81     zero();
82     int scc_n = 1;
83     while(!scc_stack.empty()){
84         int a = scc_stack.top();
85         scc_stack.pop();
86         if(scc[a] == 0){
87             dfs2(a, scc_n);
88             scc_n++;
89         }
90     }
91
92     for(int i = 2; i <= m*2; i+=2){
93         if(scc[i] == scc[i+1]){
94             cout << "IMPOSSIBLE" << endl;
95             return 0;
96         }
97     }
98     // 1 - false; 2 - true;
99     for(int i = 2; i <= 2*m; i+=2){
100         int a = scc[i];
101         int b = scc[i+1];
102         if(scc_val[a] != 0){
103             scc_val[b] = (scc_val[a] == 1 ? 2 : 1);
104         }
105         if(scc_val[b] != 0){
106             scc_val[a] = (scc_val[b] == 1 ? 2 : 1);
107         }
108         if(a < b){
109             scc_val[a] = 1;
110             scc_val[b] = 2;
111         }
112         else{
113             scc_val[a] = 2;
114             scc_val[b] = 1;
115         }
116     }
117
118     for(int i = 2; i <= m*2; i+=2)
119         cout << (scc_val[scc[i]] == 2 ? "+" : "-") << " ";
120

```



```

121     return 0;
122 }

```

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

```

1 void insert(int tree[], int p, int val){
2     p += BASE;
3     tree[p] = val;
4     while(p > 0){
5         p >>= 1;
6         tree[p] = max(tree[2*p], tree[2*p + 1]);
7     }
8 }

```

```

1 int query(int tree[], int val){
2     int p = 1;
3     while(p < BASE){
4         if(tree[2*p] >= val){
5             p = 2*p;
6         }
7         else{
8             p = 2*p+1;
9         }
10    }
11    if(tree[p] >= val)
12        return p -= BASE;
13    else
14        return -1;
15 }

```

4.1.2 Range-Range Trees

1. range ADD insert, range MAX value query

```

1 void max_add(int a, int b, int val, int k=1, int x=0, int y=BASE-1){
2     if(a <= x and y <= b){
3         max_tree[k][1] += val;
4         return;
5     }
6     max_tree[2*k][1] += max_tree[k][1];
7     max_tree[2*k+1][1] += max_tree[k][1];
8     max_tree[k][0] += max_tree[k][1];
9     max_tree[k][1] = 0;
10
11    int d = (x+y)/2;
12    if(a <= d){
13        max_add(a, b, val, 2*k, x, d);
14    }
15    if(b > d){
16        max_add(a, b, val, 2*k+1, d+1, y);
17    }
18    max_tree[k][0] = max(max_tree[2*k+1][0] + max_tree[2*k+1][1], max_tree[2*k][0]
19    + max_tree[2*k][1]);
20 }

```

```

1 int max_query(int a, int b, int k=1, int x=0, int y=BASE-1){
2     if(a <= x and y <= b){
3         return max_tree[k][0] + max_tree[k][1];
4     }
5     max_tree[2*k][1] += max_tree[k][1];

```

```

6     max_tree[2*k+1][1] += max_tree[k][1];
7     max_tree[k][0] += max_tree[k][1];
8     max_tree[k][1] = 0;
9     int d = (x+y)/2;
10    ll ret = -INF;
11    if(a <= d){
12        ret = max(ret, max_query(a, b, 2*k, x, d));
13    }
14    if(b > d){
15        ret = max(ret, max_query(a, b, 2*k+1, d+1, y));
16    }
17    return ret;
18 }

```

2. range ADD insert, range SUM query

```

1 void sum_add(int a, int b, int val, int k=1, int x=0, int y=BASE-1){
2     if(a <= x and y <= b){
3         sum_tree[k][1] += val;
4         return;
5     }
6     sum_tree[2*k][1] += sum_tree[k][1];
7     sum_tree[2*k+1][1] += sum_tree[k][1];
8     sum_tree[k][0] += sum_tree[k][1]*(y-x+1);
9     sum_tree[k][1] = 0;
10    int d = (x+y)/2;
11    if(a <= d){
12        int w = (d <= b ? d : b)+1;
13        w -= (x <= a ? a : x);
14        sum_tree[k][0] += val*(w);
15        sum_add(a, b, val, 2*k, x, d);
16    }
17    if(b > d){
18        int w = (y <= b ? y : b)+1;
19        w -= (d+1 <= a ? a : d+1);
20        sum_tree[k][0] += val*(w);
21        sum_add(a, b, val, 2*k+1, d+1, y);
22    }
23 }

```

```

1 ll sum_query(int a, int b, int k=1, int x=0, int y=BASE-1){
2     if(a <= x and y <= b){
3         return sum_tree[k][0] + sum_tree[k][1]*(y-x+1);
4     }
5     sum_tree[2*k][1] += sum_tree[k][1];
6     sum_tree[2*k+1][1] += sum_tree[k][1];
7     sum_tree[k][0] += sum_tree[k][1]*(y-x+1);
8     sum_tree[k][1] = 0;
9     int d = (x+y)/2;
10    ll ret = 0;
11    if(a <= d){
12        ret += sum_query(a, b, 2*k, x, d);
13    }
14    if(b > d){
15        ret += sum_query(a, b, 2*k+1, d+1, y);
16    }
17    return ret;
18 }

```

5 Tree Algorithms

5.1 Binary Lifting

```

1 int main(){
2     int n, q;
3     cin >> n >> q;
4     boss[1][0] = 0;
5     for(int i = 2; i <= n; i++){
6         cin >> boss[i][0];
7     }

```

```

8   for(int i = 1; i < LOG_MX; i++){
9       for(int emp = 1; emp <= MX; emp++){
10          boss[emp][i] = boss[boss[emp][i-1]][i-1];
11      }
12  }

```

5.2 Tree Traversal

```

1  vector <int> adj[MX];
2  int subtree[MX];
3  int flat[MX];
4  int vals[MX];
5
6  int idx = 1;
7  int dfs(int a, int p){
8      flat[a] = idx;
9      idx++;
10     for(int b : adj[a]){
11         if(b == p)
12             continue;
13         subtree[a] += dfs(b, a)+1;
14     }
15     return subtree[a];
16 }

```

5.3 LCA

6 String Algorithms

6.1 Trie

```

1  int trie[TMX][30];
2  bool stop[TMX];
3  int next_node = 1;
4  int dp[MX];
5
6  void insert(string s){
7      int idx = 0;
8      for(char c : s){
9          if(trie[idx][c-'a'] == 0){
10             trie[idx][c-'a'] = next_node;
11             next_node ++;
12         }
13         idx = trie[idx][c-'a'];
14     }
15     stop[idx] = true;
16 }
17
18 int main(){
19     string text;
20     int n;
21     cin >> text >> n;
22     for(int i = 0; i < n; i++){
23         string s;
24         cin >> s;
25         insert(s);
26     }

```

6.2 Pattern Finding

6.2.1 KMP

```

1  int pi[MX];
2
3  int main(){
4      string text, pattern;
5      cin >> text >> pattern;
6      string s = pattern + "+" + text;
7      for(int i = 1; i < s.size(); i++){
8          int j = pi[i-1];
9          while(j > 0 and s[i] != s[j])

```

```

10         j = pi[j-1];
11         if(s[i] == s[j])
12             j++;
13         pi[i] = j;
14     }
15
16     int res = 0;
17     for(int i = 0; i < s.size(); i++){
18         if(pi[i] == pattern.size())
19             res++;
20     }
21
22     cout << res;
23     return 0;

```

6.2.2 Hashing

```

1  const int MOD = 1e9+9;
2  const int P = 9973;
3  const int MX = 1e6+7;
4
5  ll ppow[MX];
6  ll pfx_hash[MX];
7
8  int res = 0;
9
10 int main(){
11     string s, pattern;
12     cin >> s >> pattern;
13     ppow[0] = 1;
14     for(int i = 1 ; i < MX; i++){
15         ppow[i] = (ppow[i-1]*P)%MOD;
16     }
17
18     ll pattern_hash = 0;
19     for(int i = 0; i < pattern.size(); i++){
20         pattern_hash = (pattern_hash + (pattern[i]-'a'+1)*ppow[i])%MOD;
21     }
22
23     for(int i = 0; i < s.size(); i++){
24         pfx_hash[i+1] = (pfx_hash[i] + (s[i]-'a'+1)*ppow[i])%MOD;
25     }
26
27     for(int i = 0; i+pattern.size()-1 < s.size(); i++){
28         int a = i, b = i+pattern.size();
29         ll sub_hash = pfx_hash[b] - pfx_hash[a]+MOD;
30         sub_hash %= MOD;
31         if(sub_hash == (pattern_hash*ppow[a])%MOD){
32             res++;
33         }
34     }
35     cout << res << endl;
36     return 0;

```

6.3 Palindromes

Finding longest palindromic substring.

```

1  int p[MX];
2
3  int main(){
4     string txt1, txt = "#";
5     cin >> txt1;
6     for(char c : txt1){
7         string s{c};
8         txt += s + "#";
9     }
10    txt = "^" + txt + "$";
11    int l = 1, r = 1;
12    for(int i = 1; i < txt.size(); i++){
13        p[i] = max(0, min(r-i, p[l+r-i]));
14        while(txt[i-p[i]] == txt[i+p[i]])
15            p[i]++;
16        if(i+p[i] > r){

```

```

17         l = i-p[i];
18         r = i+p[i];
19     }
20 }
21 int maxi = 0;
22 int imaxi;
23 for(int i = 0; i < txt.size(); i++){
24     if(p[i] > maxi){
25         maxi = p[i];
26         imaxi = i;
27     }
28 }
29 string res = txt.substr(imaxi-maxi+1,maxi*2-1);
30 for(auto c : res){
31     if(c != '#')
32         cout << c;
33 }

```

7 Mathematics

7.1 Fermat's Theorem

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

```

1 int exp(int a, int b, int MOD){
2     if(b == 0)
3         return 1;
4     if(b & 1){
5         return ((1ll)a * exp(a, b-1, MOD))%MOD;
6     }
7     ll tmp = exp(a, b/2, MOD);
8     return (tmp*tmp)%MOD;
9 }

```

7.2 Fast Fibonacci

```

1 void mul(ll a[][2], ll b[][2]){
2     ll res[2][2];
3     res[0][0] = a[0][0]*b[0][0] + a[0][1]*b[1][0];
4     res[0][1] = a[0][0]*b[0][1] + a[0][1]*b[1][1];
5     res[1][0] = a[1][0]*b[0][0] + a[1][1]*b[1][0];
6     res[1][1] = a[1][0]*b[0][1] + a[1][1]*b[1][1];
7
8     a[0][0] = res[0][0] % MOD;
9     a[0][1] = res[0][1] % MOD;
10    a[1][0] = res[1][0] % MOD;
11    a[1][1] = res[1][1] % MOD;
12 }
13
14 int main(){
15     ll n;
16     cin >> n;
17     ll m_pow[2][2] =
18         {{0, 1},
19          {1, 1}};
20     ll m[2][2] =
21         {{1, 0},
22          {0, 1}};
23
24     auto pow = bitset<64> (n);
25
26     for(int i = 0 ; i < 64; i++){
27         if(pow[i]){
28             mul(m,m_pow);
29         }
30         mul(m_pow, m_pow);
31     }

```

```
32
33     cout << m[0][1];
34
35     return 0;
36 }
```

7.3 Combinatorics

7.3.1 a

8 Geometry

9 Other Algorithms