

CP Algorithms Handbook

Stanisław Fiedler

October 6, 2024

Contents

1	Sorting and Searching	3
1.1	4SUM	3
2	Dynamic Programming	3
3	Graph Algorithms	3
3.1	Flows and Cuts	3
3.1.1	Maximum Flow	3
3.1.2	Minimum Cut	4
3.1.3	Maximum Matching	5
3.2	2SAT	5
4	Range Queries	5
4.1	Segment trees	5
4.1.1	Point-Range Trees	5
4.1.2	Range-Range Trees	5
5	Tree Algorithms	7
6	String Algorithms	7
7	Mathematics	7
8	Geometry	7
9	Other Algorithms	7

1 Sorting and Searching

1.1 4SUM

2 Dynamic Programming

3 Graph Algorithms

3.1 Flows and Cuts

All problems in this section can be solved using the same basic algorithm defined in 3.1.1.

3.1.1 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.

```
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     }

```

3.1.2 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

```

```

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

```

3.1.3 Maximum Matching

3.2 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:

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

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
19     +1][1], max_tree[2*k][0] + max_tree[2*k][1]);
20 }

```

```

1 ll 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 }

11 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

6 String Algorithms

7 Mathematics

8 Geometry

9 Other Algorithms