CP Algorithms Handbook

Stanisław Fiedler

October 6, 2024

Contents

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.

```
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;
        }
    }
```

3.1.2 Minimum Cut

To find Minimum Cut in a graph we need to find the Maximum Flow and check which edges connect two created 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;
                }
            }
        }
    }
```

3.1.3 Maximum Matching

3.2 2SAT

Given logical formula in the conjunctive normal form:

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

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

```
void insert(int tree[], int p, int val){
    p += BASE;
    tree[p] = val;
    while(p > 0){
        p >>= 1;
        tree[p] = max(tree[2*p], tree[2*p + 1]);
    }
}
```

```
int querry(int tree[], int val){
    int p = 1;
    while(p < BASE){
        if(tree[2*p] >= val){
            p = 2*p;
        }
        else{
            p = 2*p+1;
        }
```

```
    }
    if(tree[p] >= val)
        return p -= BASE;
    else
        return -1;
}
```

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;
}
```

5 Tree Algorithms

6 String Algorithms

7 Mathematics

8 Geometry

9 Other Algorithms