Algorithm Homework 5 NF

Jingwei Zhang 201528013229095

2015-12-17

1 Problem 1

1.1 Algorithm

Establish m nodes $g_i(i = 1, \ldots, m)$ representing m girls and n nodes $b_j(j = 1, \ldots, n)$ representing n boys. If one girl(g_i) loves one boy(b_j), a directed edge from g_i to b_j with capacity 1 is linked. Source node s is linked to every girl g_i with capacity 1 from s to g_i. Sink node t is linked to every boy b_j with capacity 1 from b_j to t. Then a maximum flow algorithm like Ford-Fulkerson algorithm from s to t should solve this problem. The flow this algorithm gets is the maximum number of pairs. Every flow from g_i to b_j in this maximum flow network forms a pair between g_i to b_j.

Pseudo-code: Suppose there are m girls and n boys. L is the collection of loving, whose element l_{ij} represents girl i loves boy j.

GIRLS-AND-BOYS-MATCHING(m, n, L)

$V = \{s, t\}$ // Vertex set
$E = \emptyset$ // Edge set
for $i = 1$ **to** m
    establish node g_i
    $V = V \cup \{g_i\}$
    establish an edge e from s to g_i
    $e.\mathit{capacity} = 1$
    $E = E \cup \{e\}$
for $j = 1$ **to** n
    establish node b_j
    $V = V \cup \{b_j\}$
    establish an edge e from b_j to t
    $e.\mathit{capacity} = 1$
    $E = E \cup \{e\}$
for every $L_{ij} \in L$
    establish an edge e from g_i to b_j
    $e.\mathit{capacity} = 1$
    $E = E \cup \{e\}$
$G =< V, E >$ // Graph
return FORD-FULKERSON(G, s, t)

1.2 Correctness

Proof. Firstly, capacities of every edge in this graph are integer 1. Thus, the total flow and flow on every edge must be integers. Then, the flow that FORD-FULKERSON generates can be separated into several $s \sim g_i \sim b_j \sim t$ paths with flow 1, since no back edge, including g_i to s, b_j to g_i and t to b_j, nor inner edge,including g_i to $g_{i'}$ and b_j to $b_{j'}$, exists in the original graph.

For every $s \sim g_i \sim b_j \sim t$ path, we will prove that it is equivalent to a pair between boys and girls. Since node g_i can only be achieved be node s and the edge from s to g_i has capacity 1, one girl node g_i exists in at most $s \sim g_i \sim b_j \sim t$ path. Similarly, one boy node b_j exists in at most $s \sim g_i \sim b_j \sim t$ path.

The number of pairs is the same as the number of $s \sim g_i \sim b_j \sim t$ path with flow 1. Moreover, FORD-FULKERSON maximize total flow. Thus, finding the maximum number of pairs is equivalent to the maximum flow problem above. ■

1.3 Complexity

The **for** loop in line 3 will loop for m times. The **for** loop in line 9 will loop for n times. The **for** loop in line 15 will loop for mn times. Thus, building the graph takes $O(m + n + mn) = O(mn)$, since the time complexity in every **for** loop is $O(mn)$. This graph contains $m + n + 2$ vertices and $O(mn)$ edges. Thus, FORD-FULKERSON(G, s, t) takes $O(mnC)$ time, where $C = \sum_{eoutofs} e.capacity = m$. Thus, the total time complexity is $O(m^2 n)$ if FORD-FULKERSON is applied to solve this maximum flow problem.

2 Problem 2

2.1 Algorithm

Establish m nodes $r_i (i = 1, \ldots, m)$ representing m rows and n nodes $c_j (j = 1, \ldots, n)$ representing n columns. Every r_i is linked to all c_j(from r_i to c_j) with capacity 1 is linked. Source node s is linked to every row node r_i(from s to r_i) with capacity rs_i, the sum of numbers in row i. Sink node t is linked to every boy c_j(from c_j to t) with capacity cs_j, the sum of numbers in column j.

Then a maximum flow algorithm, like Push-Relabel algorithm, from s to t should be applied to solve this problem. If the flow this algorithm gets is equal to the sum of all numbers in this matrix($\sum_{i=1}^{m} rs_i$ or $\sum_{j=1}^{n} cs_j$), such matrix exists. Every flow from r_i to c_j in this maximum flow network indicates $M[i][j] = 1$.

Pseudo-code: Suppose this matrix M contains m rows and n columns. rs(index from $i = 1$ to m) is array containing the sum of numbers in the i-th row. cs(index from $j = 1$ to n) is the array containing the sum of numbers in the j-th column.

```
FIND-BOOL-MATRIX(m, n, rs, rc)
1  V = {s, t} // Vertex set
2  E = ∅ // Edge set
3  sum = 0 // Sum of all elements in matrix
4  for i = 1 to m
5      establish node r_i
6      V = V ∪ {r_i}
7      establish an edge e from s to r_i
8      e.capacity = rs[i]
9      E = E ∪ {e}
10     sum+ = rs[i]
11 for j = 1 to n
12     establish node c_j
13     V = V ∪ {c_j}
14     establish an edge e from c_j to t
15     e.capacity = cs[j]
16     E = E ∪ {e}
17 for i = 1 to m
18     for j = 1 to n
19         establish an edge e_ij from r_i to c_j
20         e_ij.capacity = 1
21         E = E ∪ {e_ij}
22 G =< V, E > // Graph
23 flow = FORD-FULKERSON(G, s, t)
24 if sum == flow
25     M = 0 // Matrix, m rows and n columns
26     for i = 1 to m
27         for j = 1 to n
28             if e_ij.capacity > 0
29                 M[i][j] = 1
30             else M[i][j] = 0
31     return M
32 else return NO-SUCH-MATRIX
```

2.2 Correctness

2.3 Correctness

Proof. Firstly, capacities of every edge in this graph are integers. Thus, the total flow and flow on every edge must be integers. Then, no back edge, including r_i to s, c_j to r_i and t to c_j, nor inner edge,including r_i to $g_{i'}$ and c_j to $b_{j'}$, exists in the original graph. Moreover, the capacity of edge from r_i to c_j is 1. Thus, the flow that FORD-FULKERSON generates can be separated into several $s \sim r_i \sim c_j \sim t$ paths with flow 1.

For every $s \sim r_i \sim c_j \sim t$ path with flow 1, we will prove that it is equivalent $M[i][j] = 1$. Since the flow of this path is 1, this flow adds 1 to the flow of the edge $s- > r_i$, whose capacity is the sum of row i. Similarly, this flow adds 1 to the flow of the edge $c_j- > t$, whose capacity is the sum of column i. Thus, a size 1 flow from r_i to c_j is the same as $M[i][j] = 1$. ■

2.4 Complexity

The **for** loop in line 17 will loop for m times. The **for** loop in line 11 will loop for n times. The **for** loop in line 18 will loop for mn times. Thus, building the graph takes $O(m+n+mn) = O(mn)$, since the time complexity in every **for** loop is $O(mn)$. This graph contains $m + n + 2$ vertices and $O(mn)$ edges. Thus, FORD-FULKERSON(G, s, t) takes $O(mnC)$ time, where C, the maximum flow, is at most $S = \sum_{i=1}^{m} rs_i$. Thus, the total time complexity is $O(mnS)(S = \sum_{i=1}^{m} rs_i)$ if FORD-FULKERSON is applied to solve this maximum flow problem.

3 Problem 3

3.1 Algorithm

Firstly, run a maximum flow algorithm on G. Then, in the residual network, find collection S containing all vertices that can be achieved through edge with remaining capacity from source s. Similarly, find collection T containing all vertices that can achieved sink t through edge with remaining capacity.

Pseudo-code: Ford-Fulkerson algorithm is applied to solve maximum flow problem. $e.rev$ means the corresponding reverted edge of e.

```
DFS(G, s)
if s is not visited
    set s being visited
    S = {s}
    for every edge e from s with remaining capacity not 0
        S = S ∪ DFS(G, e.to, r)
    return S
else return ∅
```

```
Reverted-DFS(G, t)
if s is not visited
    set s being visited
    S = {s}
    for every edge e to t with remaining capacity not 0
        S = S ∪ DFS(G, e.to, r)
    return S
else return ∅
```

```
Unique-Cut(G, s, t)
Push-Relabel(G, s, t)
set all vertics in G to be not visited
S = DFS(G, s)
T = Reverted-DFS(G, t)
if |S| + |T| == |G.V|
    return is-unique
else return is-not-unique
```

3.2 Correctness

Proof. Firstly, in residual networks, S, containing all vertices that can be achieved through edges with remaining capacity larger than 0 from source S, and $V - S$ form a minimum cut of original graph. Similarly, $V - T$ and T form a minimum cut of original graph.

If the minimum cut is unique, these two cut are the same, $S = V_T$ and $V - S = T$. Thus, we have $S + T = V$ and $|S| + |T| == |V|$

If this graph contains multiple minimum cut. We denote every minimum cut as $L_i - R_i$ with L_i containing s and R_i containing t. Then, since every $L_i - R_i$ is a minimum cut, every node in every R_i should not be achieved from s through edge with remaining capacity larger than 0. Moreover, for every $L_i - R_i$,$L_i \cup R_i$. Thus, $S = \bigcap_i L_i$. Similarly, $T = \bigcap_i R_i$. Since all L_i are different and all R_i are different, $S \cup T = (\bigcap_i L_i) \cup (\bigcap_i R_i) \neq V$. Moreover,$S \cup T \subseteq V$ Thus, $|S| + |T| < |V|$.

Thus, Unique-Cut(G, s, t) judge the uniqueness of minimum cut correctly. ∎

3.3 Complexity

Firstly, Push-Relabel(G, s, t) takes $O(V^3)$ of time if a queue is applied to select active vertex. Then, the two deep first search algorithm on residual graph G take $O(V + E)$ of thime. Thus, the total time complexity is $O(V^3)$.

4 Problem 7

4.1 Dual LP Formulation

$$\begin{array}{ll} \text{Max} & z' = \sum_e x_e u_e \\ \text{s.t.} & \sum_{e \in P} x_i \geq 1, \text{ for all path } P \\ & x_i \geq 0, \qquad\quad \text{for all edge } e \end{array}$$

Explanation: $x_e = 1$ means that edge e is a minimum cut edge. $x_e = 0$ means that edge e is not a minimum cut edge. The first constraint means that for every path P, P contains at least one minimum cut edge. The second constraint show that x_e can not be negative. The objective function z' is one possible s-t cut. Minimizing this cut will get the minimum s-t cut.

5 Problem 8

5.1 Result

Outputed flow:

```
2
9
11
16
18
116
```

5.2 C++ Code

```
#include<iostream>
#include<cstdio>
#include<fstream>
#include<sstream>
#include<cstdlib>
#include<climits>
#include<vector>
#include<algorithm>

using namespace std;
const string file_name("problem1.data");
struct Edge{
    int to;
    int cap;
    int rev;
    Edge(int tt,int cc, int rr):to(tt),cap(cc),rev(rr){}
};
vector<vector<Edge>> graph;
vector<bool> vis;

inline void init_graph(int num_nodes){
    graph.clear();
    graph.assign(num_nodes,vector<Edge>());
    vis.resize(num_nodes);
}


inline void add_edge(int from, int to, int cap){
    graph[from].push_back(Edge(to, cap, graph[to].size()));
```

```
    graph[to].push_back(Edge(from, 0, graph[from].size() - 1));
}

bool get_graph(istream& in){
    string str;
    int m,n;
    bool is_end = true;
    while(getline(in, str)){
        if(str[0] != '#'){
            stringstream ss(str);
            ss>>m>>n;
            is_end = false;
            break;
        }
    }
    if(!is_end){
        init_graph(m+n+2);// m girls + n boys + s + t
        int c;
        int from,to;
        // index range of girls: [1, m+1)
        // index range of boys:[m+1, m+n]
        // s: 0,  t: m+n+1
        for(int i = 1;i <= m;i++){
            getline(in,str);
            stringstream ss(str);
            ss>>c;
            from = i;
            for(int j = 0;j<c;j++){
                ss>>to;
                to += m;
                add_edge(from, to, 1);
            }
        }
        int s = 0, t =m + n + 1;
        for(int i = 1;i <= m + n;i++){
            if(i <= m){
                add_edge(s, i, 1);
            } else {
                add_edge(i, t, 1);
            }
        }
    }
    return !is_end;
}

void show_graph(){
    for(int i = 0;i< graph.size();i++){
        cout<<i<<":";
        for(Edge &e: graph[i]){
            cout<<" ["<<e.to<<","<<e.cap<<"]";
        }cout<<endl;
    }
}
int dfs(int s, int t, int min_flow){
    if(s == t){
        return min_flow;
```

```
    }
    vis[s] = true;
    for(Edge& e:graph[s]){
        if(!vis[e.to] && e.cap > 0){
            min_flow = min(min_flow, e.cap);
            int f = dfs(e.to, t, min_flow);
            if(f > 0){
                // Reduce cap on e and increase cap on rev
                e.cap -= f;
                graph[e.to][e.rev].cap += f;
                return f;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t){
    int flow = 0;
    int f = 0;
    while(true){
        for(int i = 0;i<vis.size();i++){
            vis[i] = 0;
        }
        f = dfs(s,t,INT_MAX);
        flow += f;
        if(f == 0) {
            break;
        }
    }
    show_graph();
    return flow;
}

int main()
{
    // Input part
    ifstream s_file;
    s_file.open(file_name);
    string str;
    while(get_graph(s_file)){
        cin>>str;
        int flow = max_flow(0,graph.size()-1);
        cout<<flow<<endl;
    }
    s_file.close();
    return 0;
}
```

6 Problem 9

6.1 Result

Outputed matrix(only the first one is shown):

```
0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 1 1 1 1
0 1 0 0 0 0 0 1 1 1 1
```

```
0 1 0 0 0 1 1 1 1 1 1
0 1 0 1 0 0 1 1 1 1 1
0 0 1 1 1 1 1 0 0 0 1
0 0 1 1 0 0 1 0 0 0 0
0 0 1 1 1 1 0 0 0 0 1
0 0 1 1 1 1 1 1 0 0 1
0 1 1 1 1 1 1 1 0 0 0
0 1 1 1 0 0 0 0 0 0 0
```

6.2 C++ Code

```cpp
#include<iostream>
#include<cstdio>
#include<fstream>
#include<sstream>
#include<cstdlib>
#include<climits>
#include<vector>
#include<algorithm>

using namespace std;
const string file_name("problem2.data");
struct Edge{
    int to;
    int cap;
    int flow;
    int rev;
    Edge(int tt,int cc, int ff,int rr):
        to(tt),cap(cc),flow(ff),rev(rr){}
};
vector<vector<Edge>> graph;
int m,n;// m rows and n columns
vector<int> r;// row
vector<int> c;// column
inline void init_graph(int num_nodes){
    graph.clear();
    graph.assign(num_nodes,vector<Edge>());
}

inline void add_edge(int from, int to, int cap){
    graph[from].push_back(Edge(to, cap, 0,graph[to].size()));
    graph[to].push_back(Edge(from, 0, 0,graph[from].size() - 1));
}

inline int matrix_index(int r, int c){
    return (r-1) * n + c;
}

bool get_graph(istream& in, int& cap){
    string str;
    bool is_end = true;
    while(getline(in, str)){
        if(str[0] != '#'){
            stringstream ss(str);
            ss>>m>>n;
            is_end = false;
```

```
                break;
            }
        }
        if(!is_end){
            // m row nodes + n cloumn nodes + s + t
            init_graph(m + n + 2);
            // index range of row [1, m]
            // index range of col [m + 1, m + n]
            // s: 0,  t: m + n + 1
            getline(in,str);
            stringstream ssm(str);
            r.assign(m + 1, 0);
            c.assign(n + 1, 0);
            for(int i = 1;i <= m;i++){
                ssm>>r[i];
            }
            getline(in,str);
            stringstream ssn(str);
            for(int i = 1;i <= n;i++){
                ssn>>c[i];
            }
            int s = 0, t = m + n + 1;
            // s -> row nodes
            for(int i = 1; i <= m;i++){
                add_edge(s, i, r[i]);
            }
            // column nodes -> t
            for(int i = 1; i <= n;i++){
                add_edge(i + m, t, c[i]);
            }
            // row nodes -> column nodes
            for(int i = 1;i <= m;i++){
                for(int j = 1;j <= n;j++){
                    add_edge(i, j + m, 1);
                }
            }
        }
        return !is_end;
}

void show_graph(){
    // for debugging
    for(int i = 0;i< graph.size();i++){
        cout<<i<<":";
        for(Edge &e: graph[i]){
            cout<<" ["<<e.to<<","<<e.cap<<"]";
        }cout<<endl;
    }
}

int max_flow(int s, int t){
    int flow = 0;
    int N = graph.size();
    vector<int> h(N, 0); // Height of every nodes
    vector<int> excess(N, 0);
    // Initial pre-flow
```

```
    h[s] = N;
    for(Edge &e:graph[s]){
        e.flow += e.cap;
        graph[e.to][e.rev].flow -= e.cap;
        excess[e.to] += e.cap;
    }

    while(true){
        bool stop = true;
        int v; // find E(v) > 0
        for(int i = 0;i < N;i++){
            if(excess[i] > 0 && i != s && i != t){
                stop = false;
                v = i;
                break;
            }
        }
        if(stop) break;
        for(Edge &e: graph[v]){
            if(excess[v] <= 0) {
                break;
            }
            int w = e.to;
            if(h[v] > h[w]){
                // Push excess: v->w
                int amt = min(excess[v], e.cap - e.flow);
                e.flow += amt;
                graph[w][e.rev].flow -= amt;
                excess[v] -= amt;
                excess[w] += amt;
            }
        }
        if(excess[v] > 0){
            // Relabel v
            h[v] = 2*N;
            for(Edge& e:graph[v]){
                if(e.cap - e.flow > 0){
                    h[v] = min(h[v], h[e.to]+1);
                }
            }
        }
    }
    return excess[t];
}
vector<vector<int>> get_matrix(){
    vector<vector<int>> matrix(m + 1,vector<int>(n + 1,0));
    for(int i = 1;i<=m;i++){
        for(Edge& e: graph[i]){
            if(e.flow > 0){
                int j = e.to - m;
                matrix[i][j] = 1;
            }
        }
    }
    return matrix;
}
```

```
void show_matrix(vector<vector<int>> matrix){
    for(vector<int> &row:matrix){
        for(int b:row){
            cout<<b<<" ";
        }
        cout<<endl;
    }
}
bool check_matrix(){
    vector<vector<int>> matrix = get_matrix();
    show_matrix(matrix);
    bool is_right = true;
    vector<int> check_r(m + 1, 0);
    vector<int> check_c(n + 1, 0);
    for(int i = 1;i <= m;i++){
        for(int j = 1;j <= n;j++){
            if(matrix[i][j] >= 1){
                check_r[i]++;
                check_c[j]++;
            }
        }
    }
    for(int i = 1;i <= m;i++){
        if(check_r[i] != r[i]){
            is_right = false;
            break;
        }
    }
    for(int j = 1;j <= n;j++){
        if(check_c[j] != c[j]){
            is_right = false;
            break;
        }
    }
    return is_right;
}

int main()
{
    // Input part
    ifstream s_file;
    s_file.open(file_name);
    //string str;
    int cap;
    while(get_graph(s_file, cap)){
        //show_graph();
        //cin>>str;
        int flow = max_flow(0,graph.size()-1);
        cout<<"flow: "<<flow<<endl;
        bool is_right = check_matrix();
        cout<<"check_result: "<<(is_right?"right":"error")<<"."<<endl;

    }
    s_file.close();
    return 0;
}
```