CS 630, Fall 2024, Homework 1

Yifei Bao

September 15, 2024

Problem 1 ***Database restore***

Algorithm 1: databaseRestore(C, D)

```
   /* Input:  C, D: Array of transactions of clients and days   Output:M:
      Table of transactions of client i on day j                          */
   /* Compute total transactions of C and D                               */
 1 total_C ← sum(C);
 2 total_D ← sum(D);
 3 if total_C ≠ total_D then
 4 |   print "There doesn't exist such a database";
 5 |   return;
 6 G ← Graph(V, E, s, t)/* Initialize a network flow graph with empty vertices,
      edges, source node s, and sink node t                               */
   /* Creating nodes of clients and days and capacity edges of transaction */
 7 for i = 0...n do
 8 |   V.addNode(c_i);
 9 |   E.addEdge(s, c_i, C[i]);
10 for j = 0...m do
11 |   V.addNode(d_j);
12 |   E.addEdge(d_j, t, D[j]);
13 for i = 0...n do
14 |   for j = 0...m do
15 |   |   E.addEdge(c_i, d_j, infinity);
16 f ← FordFulkerson(G)/* Get maxflow f using Ford Fulkerson with input G   */
17 if value(f) ≠ total_C then
18 |   print "There doesn't exist such a database";
19 |   return;
   /* Create table M of each transaction amount of client i on day j      */
20 M = List[n][m];
21 for j = 0...m do
22 |   for i = 0...n do
23 |   |   M[i][j] = f(c_i, d_j)/* Flow of edge from c_i to d_j            */
24 return M;
```

Algorithm Description

This algorithm models the problem as a network flow graph. The graph consists of two sets of nodes: clients c_i and days d_j. The source node s is connected to each client c_i with an edge of capacity $C[i]$, and each day d_j is connected to the sink node t with an edge of capacity $D[j]$. The edges between clients and days have infinite capacity to allow the flow of transactions.

Then the algorithm runs Ford-Fulkerson to find the max flow of the graph. If the total transaction amount of clients is not equal to the total transaction amount of days, then the database cannot be restored. Otherwise, the algorithm creates a table $M[i][j]$ to store the transaction amount of each client i on each day j.

Correctness Proof

The edge(s, c_i) has capacity $C[i]$ and edge(c_i, d_j) has capacity ∞, and edge(d_j, t) has capacity $D[j]$. Thus, the maximum flow value of the graph would be no more than $\sum_{i=0}^{n-1} C[i] = \sum_{j=0}^{m-1} D[j]$.

If the maximum flow value is $\sum_{i=0}^{n-1} C[i] = \sum_{j=0}^{m-1} D[j]$, then it means that $f(s, c_i) = \sum_{j}^{m-1} f(c_i, d_j) = C[i]$ and also $f(d_j, t) = \sum_{i}^{n-1} f(c_i, d_j) = D[j]$ due to flow conservation.

Hence the output $M[i][j] = f(c_i, d_j)$ contains the transaction amount of client i on day j.

Otherwise, if the maximum flow value is less than $\sum_{i=0}^{n-1} C[i] = \sum_{j=0}^{m-1} D[j]$, then the flow value from s is not $\sum_{i=0}^{n-1} C[i]$, thus the database cannot be restored.

In conclusion, the algorithm is correct and outputs the correct transaction amount of client i on day j or returns that there doesn't exist such a database.

Time Complexity Use the Edmonds-Karp algorithm to find the maximum flow in the residual graph. The time complexity of the Edmonds-Karp algorithm is $O(|V| \cdot |E|^2)$. In this case, the number of vertices $|V| = n + m + 2$ and the number of edges $|E| = n \cdot m + n + m$. Thus, the time complexity of the algorithm is $O((n + m + 2) \cdot (n \cdot m + n + m)^2)$, simplified as $O((n + m)^3)$.

Space Complexity The space complexity of the algorithm is $O(|V| + |E| + nm) = O(2(n + m) + 2 + n \cdot m + n \cdot m) = O(nm)$ for storing the table and flow network graph M.

Problem 2 ***Decreased flow***

2.1

Construct a residual graph based on G given the flow f. For each edge(u, v), the value is $c(u, v) - f(u, v)$ and add backward edge(v, u) with value $f(u, v)$. Then run BFS from s to t in the residual graph. f is a maximum flow only if there exists no path. Time complexity should be $O(|V| + |E|)$.

2.2

The value of maximum flow can be $value(f)$ or $value(f) - 1$ in the decreased graph.
$value(f') = value(f)$ if $f(x, y) < c(x, y)$ or there exists other augmenting paths. $value(f') = value(f) - 1$ if $c(x, y) = f(x, y)$ and there exists no other paths.

To be more specific,

If $f(x, y) < c(x, y)$, then the edge(x, y) is not saturated and the max flow remains the same.

If $f(x, y) = c(x, y)$, then the edge(x, y) is saturated and there exists two cases:

- If there exists an augmenting path, then the max flow remains the same.

- If there exists no augmenting path, then the max flow decreases by 1.

2.3

Algorithm 2: decreasedFlow(G, f, (x, y))

```
/* Input:
   - G(V, E, s, t): original network flow graph
   - f: original max flow
   - (x, y): edge to be decreased
   Output:
   - f': max flow after decreasing edge(x, y) capcity
                                                                    */
/* Create a residual graph based on G with respect to f            */
residual_G ← ResidualGraph(G, f);
/* If edge(x, y) is not saturated, then return the original max flow */
if f(x, y) < c(x, y) then
    return f;
residual_G', f' ← reducePathFlowValue(residual_G,(x,y),1)/* Decrease flow value of
    path passing through edge(x, y) by 1                            */
residual_G' ← reduceCapacity(residual_G', (x,y), 1) /* Decrease capacity(x, y) by 1 */
P = BFS(residual_G', s, t)/* Find augmenting path in the residual graph */
if P is not empty then
    f' ← Augment(f', P);
    update residual_G';
return f'
```

Algorithm Description

The goal of the algorithm is after decreasing the capacity value of the edge (x, y) by 1 given $G(V, E, s, t, c)$ and f, to check if the max flow value changes and return the new max flow f'.

The core concept is to check whether or not the edge (x, y) is saturated. If not saturated, then the max flow remains the same after decreasing the capacity. Otherwise, if edge (x, y) is saturated, then the max flow value can either remain the same or decrease by 1.

The algorithm constructs a residual graph based on the original graph and flow. If the edge (x, y) is not saturated, then the original max flow is returned. Otherwise, the algorithm decreases the flow value of the path passing through edge (x, y) by 1 and decreases the capacity of edge (x, y) by 1.

Then try to find an augmenting path in the residual graph using BFS. If it exists, augment and return the flow and it should be the new maximum flow. Else, return the original f as max flow.

Correctness Proof

As answered in 2.2, if the edge (x, y) is not saturated then the max flow remains the same after decreasing the capacity because the edge can still carry at least one flow.

If the edge (x, y) is saturated, then the max flow value can either remain the same or decrease by 1. If there exists an augmenting path in the residual graph, then the max flow remains the same. Otherwise, the max flow decreases by 1.

Time Complexity

Total running time: $O(|V| + |E|)$

1 $O(|E|)$ construction of $residual_G$

2, 3 $O(1)$ check if edge (x, y) is saturated

4 $O(|V| + |E|)$ reduce flow value of path through (x, y)

5 $O(1)$ reduce capacity of edge (x, y)

6 $O(|V| + |E|)$ BFS to find augmenting path

7 $O(|V|)$ augment flow and update residual graph

Space Complexity

The space complexity of the algorithm is $O(|V| + |E|)$ for storing flow network graph.