**CS 2420**

**Min Cost Max Flow**

**Motivating Example:** In the transportation problem, goods are transported from a set of sources to a set of destinations subject to the supply and demand of the sources and destination respectively such that the total cost of transportation is minimized.

Rubric

|  |  |
| --- | --- |
|  | Points |
|  |  |
| Read in the graph | 5 |
| Find the cheapest path using Belman Ford (show path and cost) | 5 |
| Find the best matching | 5 |
|  |  |
| Good variable names | 3 |
| Comments for methods | 3 |
| Clear coding (see style) | 9 |
|  |  |
| Bonus (Optional) | 3 |
|  |  |

**Part 1:**

Use starter code to input each graph. The starter code uses an adjacency matrix representation, but has multiple matrices to store extra information (like the cost of an edge). So capacity is used along with edge cost to find paths in the graph.

Draw a picture (for your own use) of what each graph looks like.

Input Files:

The first line indicates the number of nodes (n). Node 0 is the start node. Node n-1 is the sink. The remaining lines indicate each arc (source, destination, capacity, cost). For example, 11 12 3 1 indicates an arc from 11 to 12 with a capacity of 3 and a cost of 1.

**Part 2: Find Cheapest path using Bellman Ford.**

Using the graph you input in Part 1, find a cheapest path (in terms of cost) from the source to the sink**.** While Dijkstra’s algorithm seems like a good one to use, we are going to use Bellman Ford (as we will have negative edges in the future.

Bellman Ford to find the shortest path. The pseudo code for finding the cheapest augmenting path is shown below.

private boolean hasAugmentingCheapestPath(FlowNetwork G, int s, int t) {

clear pred

set costs to high value;

cost[s]=0

for (int i = 0; i < vertexCt; i++) {

for (int u = 0; u < vertexCt; u++) {

for (int v = 0; v < vertexCt; v++) {

if ( the edge from [u,v] exists and creates a cheaper path ) {

cost[v] = cost[u] + edgeCost[u][v];

pred[v] = u;

}

}

}

}

return pred[t] exists;

}

Notice, the edge information is stored as two parts (1) does the edge exist? (2) what is its cost?. This seems a little odd but will be really helpful as we keep adding/deleting edges. The costs don’t change, but whether the edge exists does change.

Test your result. Do you find the cheapest path?

**Part 3: Transportation**

In the transportation problem, goods are transported from a set of sources to a set of destinations subject to the supply and demand of the sources and destination respectively such that the total cost of transportation is minimized. It is also sometimes called as the Hitchcock problem. In the example below, the suppliers F, G, and H can provide various amounts of sugar to bakeries M and N. The cost of the sugar depends both on the supplier costs and the transportation cost. The suppliers have a limit as to what they can provide. The bakeries have a certain amount of the product they want.

**A diagram of a network

Description automatically generated**

Several quantities should be defined to help formulate the solution:

* the amount of material provided at the supplier
* the amount of material being consumed at the bakery
* the amount of material being transferred from supplier to bakery
* the cost of transferring 1 unit of material from supplier to bakery

��� �In the example below, we have added sources, sinks, and capacity:cost to the graph. This data means that supplier1 can supply 5 units and bakery 4 needs 10 units. Also, supplier 2 can supply 8 units to consumer 4 at a cost of $1 per unit.

**A diagram of a number of objects

Description automatically generated with medium confidence**

You will repeatedly find the best path through the graph.

**Paths found in order (transport0.txt)**

0 2 4 6(8) $ 1

0 1 4 6(2) $ 2

0 2 5 6(2) $ 3

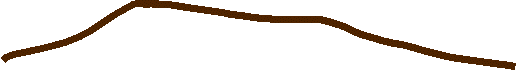
0 3 5 6(3) $ 4

0 1 4 2 5 6(1) $ 4 \*\*Notice what is happening here. Earlier decisions are undone to utilize capacity of plant 1.

0 1 5 6(2) $ 5

**A diagram of a number of objects

Description automatically generated with medium confidence**



Then you summarize the flow on each edge as shown below.

Final flow on each edge

Flow 0 -> 1 (5) $ 0

Flow 0 -> 2 (10) $ 0

Flow 0 -> 3 (3) $ 0

Flow 1 -> 4 (3) $ 2

Flow 1 -> 5 (2) $ 5

Flow 2 -> 4 (7) $ 1

Flow 2 -> 5 (3) $ 3

Flow 3 -> 5 (3) $ 4

Flow 4 -> 6 (10) $ 0

Flow 5 -> 6 (8) $ 0

Can you convince yourself this is the best routing in terms of min cost and max flow?

**Implementation**

I used an adjacency matrix implementation. I augmented the graph representation with a residual matrix which contains the remaining capacity on each edge: Initially, residual is the same as capacity.

**int** [][] **residual**;

The residual graph was so useful, I didn’t use the original graph for much. The residual graph captured what was needed to find paths.

The pseudo code for Ford Fulkerson max flow is in the notes. We need a shortest path from start to end, where shortest is in terms of edge cost. Since backward edges have negative cost, we will need to use the Belman Ford algorithm to find the shortest path.

**Output:**

1. **Output the original graph**
2. **Show each path you find in order (and the flow and cost of the path)**
3. **Print the final flow (and cost) on each edge.**

**Bonus (3 points)**

**Find the min cut for Graph Flow10.txt. See notes for details of algorithm.**