Start Assignment

Due Thursday by 11:59pm **Points** 73 **Submitting** a file upload **File Types** java
Available Aug 29 at 12am - Dec 8 at 11:59pm

7 Assignment 7 - Max Flow / Min Cut

Computer SCIENCE

IMPORTANT: Please review the statement on Academic Integrity. Submitting code, without permission, that is not your own may result in an automatic F for the class: Statement on Academic Integrity (https://usu.instructure.com/courses/717937/pages/statement-on-academic-integrity)

Introduction

Note: This assignment may not be turned in late. The late gift doesn't apply to the last assignment either (refer to the syllabus, this is described there). Plan to complete this assignment on or before the due date, no possibility to turn it in late is allowed.

This assignment is designed to give you experience with graph data structures and algorithms. To do this, you work with some provided graph code that you will modify and use to solve the max flow and min cut problems (network flow).

I believe this to be the most challenging assignment of the semester, start on it early, and work on it daily.

Assignment

Using the provided input files that represent graphs and their edge weights (flow), compute the max flow between starting and ending vertices in the graphs, along with the min cut for those same starting and ending vertices.

You must use the provided `Graph` and `GraphNode` classes that use an adjacency list to represent the graph. I know you can go out to the internet and find Java code that computes max flow and min cut, that isn't what this assignment is about; you are to write the code the implements these algorithms, starting from the provided code. You may not change the representation away from an adjacency list, but you might find it necessary to add to one, or more, of the provided methods.

Provide the implementation for the following methods in the `Graph` class:

- `public int findMaxFlow(int s, int t, boolean report)`
- `private boolean hasAugmentingPath(int s, int t)`
- `public void findMinCut(int s)`

You may modify the code in this class are free to add other private methods you find helpful.

The display of the augmenting paths and edges is done as part of the `findMaxFlow` method. Following the call of `findMaxflow`, the total flow is displayed. Similarly for the `findMinCut`, the display of which edges to cut is performed during the `findMinCut` method. I prefer to have those data items returned from the methods and then separate reporting, but that begins to distract from the core of the assignment. I want to keep the focus on the algorithms, and not have you get worried about housekeeping and tracking of the data items for reporting by other code...even though that would be a *better* overall solution.

You may not modify the `GraphNode` class; note that it isn't submitted, only the `Graph` class is submitted.

Algorithms

You must use the Edmonds Karp algorithm to determine the max flow. Use the min cut algorithm outlined below, which is also presented in my lecture slides. The following are compact descriptions of these algorithms that may be a little easier to understand than putting it together from my lecture slides. But you should refer to my lecture slides for the overall understanding.

Max Flow - Edmonds Karp

The input to this algorithm is a graph, along with a starting vertex (s) and a terminating vertex (t).

- set total flow to 0
- while there is an ***augmenting path*** between s and t
  - set the available flow to the largest possible integer that Java can represent
  - follow the augmenting path from t to s; using vertex v as the current vertex
    - available flow = minimum of available flow or the residual flow at vertex v
  - follow the augmenting path from t to s; using vertex v as the current vertex
    - update the residual graph
      - subtract available flow at vertex v in direction of s to t
      - add available flow at vertex v in direction of t to s
  - add available flow to total flow

There is a big abstraction in the above algorithm, computing of the ***augmenting path***. This is a *breadth-first search* (shortest path) over the graph looking for a way to get flow from s to t. It is as follows...

- reset the "parent" on all vertices; parent is the vertex that flow comes from, computed during this algorithm

- add s to the queue
- while queue is not empty and vertex t does not have a parent
    - remove from queue as vertex v
    - for all successor edges from v
        - for the edge, call the other vertex w
        - if there is residual capacity from v to w and not already part of the augmenting path, and it isn't vertex s, then it can be used
            - remember the path; set parent of w to v
            - add w to the queue
- if vertex t has a parent, then there is an augmenting path from s to t

Min Cut

1. Based on the max flow algorithm, compute the final residual graph
2. Find the set of vertices that are reachable from the source vertex in the residual graph; the set R includes s. Call those vertices R.
3. All edges from a vertex in R to a vertex not in R are the minimum cut edges

The above makes it seem like min cut is just a few lines of code, it isn't; my implementation is about 50 lines long. Each one of those steps is a chunk of code.

Input

Each input file contains the definition of a graph. The first number is the number of vertices in the graph, then each triple of numbers after that represents an edge. The first two numbers of the triple are the vertex numbers (0-based index), the third number is the weight of that edge. The flow is in the direction of the first vertex to the second vertex.

The following is the data in **demands1.txt (https://usu.instructure.com/courses/717937/files/85091496?wrap=1) (https://usu.instructure.com/courses/717937/files/85091496/download?download_frd=1)** :

```
6
0 1 3
0 2 3
1 2 3
1 3 3
2 3 3
2 4 2
3 4 2
3 5 2
4 5 4
```

This file contains a description of a graph with 6 vertices and 9 edges. Again, notice the vertices are a 0-based index. While the first vertices in the files happened to be ordered/sorted by the first vertex, that doesn't have to be the case, your code should be able to handle the edges in any order.

Output

The output from your program must match the format and style you see below from the two examples. The first part displays the max flow, the second is the min cut.

The max flow starts by displaying each augmenting path. The first number is the amount of flow along that augmenting path, followed by the vertex numbers of the path. After display of the augmenting paths, display all edges that carry flow (items). Remember that an augmenting path can "push" flow backwards. Because of this, an augmenting path can show flow in the opposite direction of an edge. This can only occur if a previous augmenting path sent flow (in the forward direction) over that edge. Another thing to note in the output is that the augmenting paths only grow in size, with each subsequent path. This is because the Edmonds-Karp algorithm finds shortest paths. Therefore, the shortest paths for augmenting flow are found in order.

The min cut is a report of which edges are part of the min cut.

The following shows the output of my program for **demands1.txt (https://usu.instructure.com/courses/717937/files/85091496?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091496/download?download_frd=1)**

```
-- Max Flow: demands1.txt --
Flow 2: 0 2 4 5
Flow 1: 0 2 3 5
Flow 1: 0 1 3 5
Flow 2: 0 1 3 4 5

Edge(0, 2) transports 3 items
Edge(0, 1) transports 3 items
Edge(1, 3) transports 3 items
Edge(2, 4) transports 2 items
Edge(2, 3) transports 1 items
Edge(3, 5) transports 2 items
Edge(3, 4) transports 2 items
Edge(4, 5) transports 4 items
Total Flow: 6

-- Min Cut: demands1.txt --
Min Cut Edge: (0, 2)
Min Cut Edge: (0, 1)
```

The following shows the output of my program for **demands6.txt (https://usu.instructure.com/courses/717937/files/85091495?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091495/download?download_frd=1)**

```
-- Max Flow: demands6.txt --
Flow 10: 0 1 4 7
Flow 10: 0 2 5 4 1 3 6 7

Edge(0, 2) transports 10 items
Edge(0, 1) transports 10 items
Edge(1, 3) transports 10 items
Edge(2, 5) transports 10 items
Edge(3, 6) transports 10 items
Edge(4, 7) transports 10 items
Edge(5, 4) transports 10 items
Edge(6, 7) transports 10 items
Total Flow: 20
```

```
-- Min Cut: demands6.txt --
Min Cut Edge: (0, 2)
Min Cut Edge: (0, 1)
```

Helpful Suggestions

- You'll probably find it useful to add a residual 2d array to the `Graph` class, to help with tracking residual flow (forward and backward)
- Write a method to display the residual graph to help you debug your code
- ***Think hard about how to represent the edges that have backwards flow.*** The algorithm for finding an augmenting path needs to take advantage of this information. I get a lot of questions about this assignment and I can easily look at the code to know if this has been "thought hard about" or not...so think hard about this :) If you can't answer this question, you haven't thought it through well enough yet, "How can your find augmenting path code find backwards flow?"
- Java's `LinkedList` class supports the `Queue` interface; this is much better than using an `ArrayList` as a queue. You can create it like this: `Queue<Integer> q = new LinkedList<>();`

Notes & Submission

- Use the following code and document files as the basis from which you write your implementation
  - **Assignment7Driver.java (https://usu.instructure.com/courses/717937/files/85091490?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091490/download?download_frd=1)**
  - **GraphNode.java (https://usu.instructure.com/courses/717937/files/85091483?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091483/download?download_frd=1)**
  - **Graph.java (https://usu.instructure.com/courses/717937/files/85091498?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091498/download?download_frd=1)**
  - **demands1.txt (https://usu.instructure.com/courses/717937/files/85091496?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091496/download?download_frd=1)**
  - **demands2.txt (https://usu.instructure.com/courses/717937/files/85091497?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091497/download?download_frd=1)**
  - **demands3.txt (https://usu.instructure.com/courses/717937/files/85091499?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091499/download?download_frd=1)**
  - **demands4.txt (https://usu.instructure.com/courses/717937/files/85091492?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091492/download?download_frd=1)**
  - **demands5.txt (https://usu.instructure.com/courses/717937/files/85091500?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091500/download?download_frd=1)**
  - **demands6.txt (https://usu.instructure.com/courses/717937/files/85091495?wrap=1)** **(https://usu.instructure.com/courses/717937/files/85091495/download?download_frd=1)**
- Turn in only the following file
  - Graph.java
- Your code must compile without any warnings or compiler errors. See syllabus regarding code that has compiler errors.

- Your code must adhere to the CS 2420 coding standard: **link (https://usu.instructure.com/courses/717937/pages/cs-2420-coding-standard)**

Assignment 7 - Max Flow / Min Cut

Criteria	Ratings		Pts
Compiles without any warnings	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Follows the required course style guide	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Correct hasAugmentingPath implementation	**15 pts** **Full Marks**	**0 pts** **No Marks**	15 pts
Correct findMaxFlow implementation	**15 pts** **Full Marks**	**0 pts** **No Marks**	15 pts
min cut - determine final residual graph	**8 pts** **Full Marks**	**0 pts** **No Marks**	8 pts
min cut - find reachable vertices	**8 pts** **Full Marks**	**0 pts** **No Marks**	8 pts
min cut - final determination of min cut edges	**8 pts** **Full Marks**	**0 pts** **No Marks**	8 pts
Output - Display of each augmenting path	**5 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
Output - Display of edges and how many items carried	**5 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
Output - Display of min cut edges	**5 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
			Total Points: 73