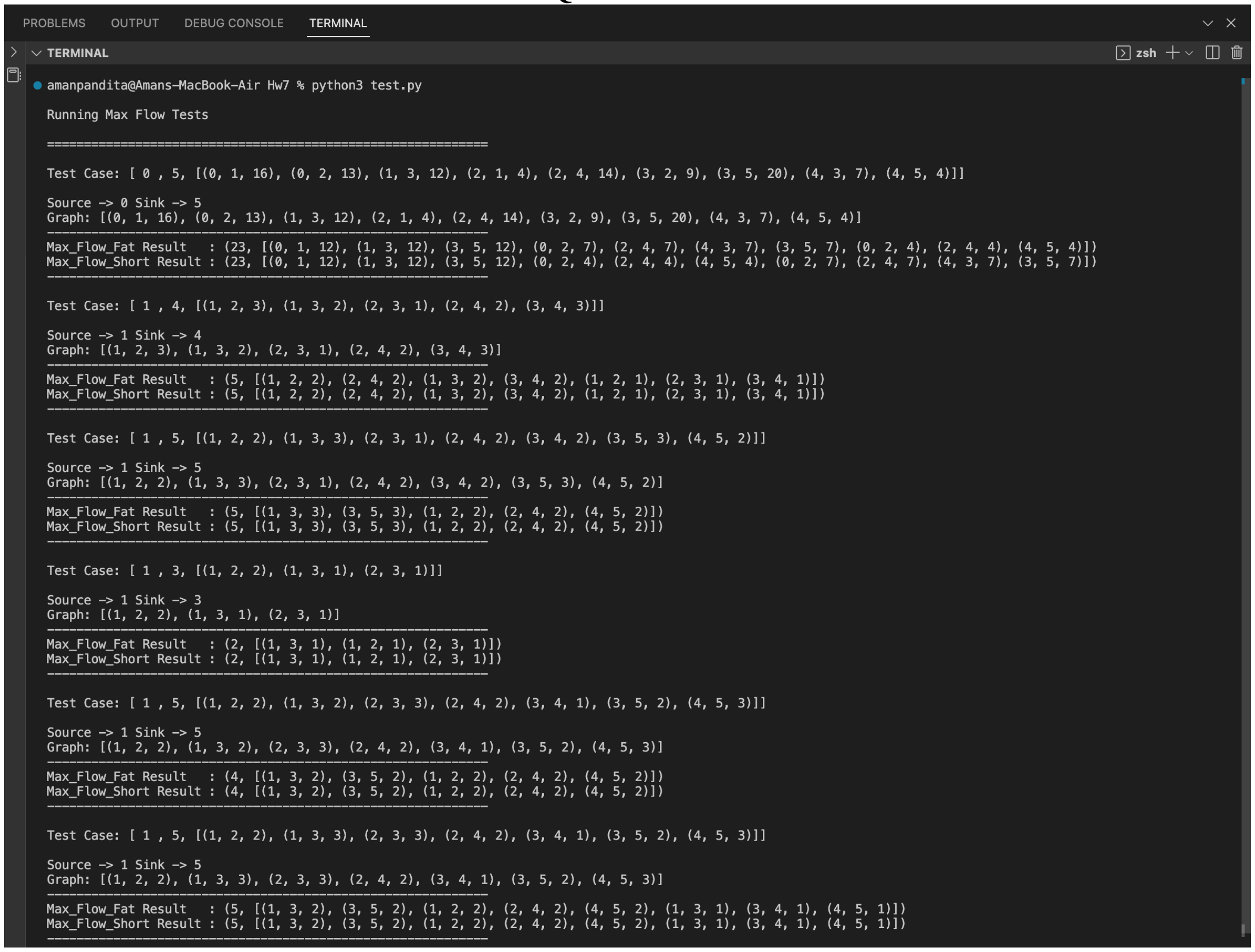
Assignment 8

CS 514 – Algorithms

Submitted By: Aman Pandita

Onid: panditaa@oregonstate.edu

Question 1:



```
amanpandita@Amans-MacBook-Air Hw7 % python3 test.py

Running Max Flow Tests

============================================================

Test Case: [ 0 , 5, [(0, 1, 16), (0, 2, 13), (1, 3, 12), (2, 1, 4), (2, 4, 14), (3, 2, 9), (3, 5, 20), (4, 3, 7), (4, 5, 4)]]

Source -> 0 Sink -> 5
Graph: [(0, 1, 16), (0, 2, 13), (1, 3, 12), (2, 1, 4), (2, 4, 14), (3, 2, 9), (3, 5, 20), (4, 3, 7), (4, 5, 4)]
------------------------------------------------------------
Max_Flow_Fat Result   : (23, [(0, 1, 12), (1, 3, 12), (3, 5, 12), (0, 2, 7), (2, 4, 7), (4, 3, 7), (3, 5, 7), (0, 2, 4), (2, 4, 4), (4, 5, 4)])
Max_Flow_Short Result : (23, [(0, 1, 12), (1, 3, 12), (3, 5, 12), (0, 2, 4), (2, 4, 4), (4, 5, 4), (0, 2, 7), (2, 4, 7), (4, 3, 7), (3, 5, 7)])
------------------------------------------------------------

Test Case: [ 1 , 4, [(1, 2, 3), (1, 3, 2), (2, 3, 1), (2, 4, 2), (3, 4, 3)]]

Source -> 1 Sink -> 4
Graph: [(1, 2, 3), (1, 3, 2), (2, 3, 1), (2, 4, 2), (3, 4, 3)]
------------------------------------------------------------
Max_Flow_Fat Result   : (5, [(1, 2, 2), (2, 4, 2), (1, 3, 2), (3, 4, 2), (1, 2, 1), (2, 3, 1), (3, 4, 1)])
Max_Flow_Short Result : (5, [(1, 2, 2), (2, 4, 2), (1, 3, 2), (3, 4, 2), (1, 2, 1), (2, 3, 1), (3, 4, 1)])
------------------------------------------------------------

Test Case: [ 1 , 5, [(1, 2, 2), (1, 3, 3), (2, 3, 1), (2, 4, 2), (3, 4, 2), (3, 5, 3), (4, 5, 2)]]

Source -> 1 Sink -> 5
Graph: [(1, 2, 2), (1, 3, 3), (2, 3, 1), (2, 4, 2), (3, 4, 2), (3, 5, 3), (4, 5, 2)]
------------------------------------------------------------
Max_Flow_Fat Result   : (5, [(1, 3, 3), (3, 5, 3), (1, 2, 2), (2, 4, 2), (4, 5, 2)])
Max_Flow_Short Result : (5, [(1, 3, 3), (3, 5, 3), (1, 2, 2), (2, 4, 2), (4, 5, 2)])
------------------------------------------------------------

Test Case: [ 1 , 3, [(1, 2, 2), (1, 3, 1), (2, 3, 1)]]

Source -> 1 Sink -> 3
Graph: [(1, 2, 2), (1, 3, 1), (2, 3, 1)]
------------------------------------------------------------
Max_Flow_Fat Result   : (2, [(1, 3, 1), (1, 2, 1), (2, 3, 1)])
Max_Flow_Short Result : (2, [(1, 3, 1), (1, 2, 1), (2, 3, 1)])
------------------------------------------------------------

Test Case: [ 1 , 5, [(1, 2, 2), (1, 3, 2), (2, 3, 3), (2, 4, 2), (3, 4, 1), (3, 5, 2), (4, 5, 3)]]

Source -> 1 Sink -> 5
Graph: [(1, 2, 2), (1, 3, 2), (2, 3, 3), (2, 4, 2), (3, 4, 1), (3, 5, 2), (4, 5, 3)]
------------------------------------------------------------
Max_Flow_Fat Result   : (4, [(1, 3, 2), (3, 5, 2), (1, 2, 2), (2, 4, 2), (4, 5, 2)])
Max_Flow_Short Result : (4, [(1, 3, 2), (3, 5, 2), (1, 2, 2), (2, 4, 2), (4, 5, 2)])
------------------------------------------------------------

Test Case: [ 1 , 5, [(1, 2, 2), (1, 3, 3), (2, 3, 3), (2, 4, 2), (3, 4, 1), (3, 5, 2), (4, 5, 3)]]

Source -> 1 Sink -> 5
Graph: [(1, 2, 2), (1, 3, 3), (2, 3, 3), (2, 4, 2), (3, 4, 1), (3, 5, 2), (4, 5, 3)]
------------------------------------------------------------
Max_Flow_Fat Result   : (5, [(1, 3, 2), (3, 5, 2), (1, 2, 2), (2, 4, 2), (4, 5, 2), (1, 3, 1), (3, 4, 1), (4, 5, 1)])
Max_Flow_Short Result : (5, [(1, 3, 2), (3, 5, 2), (1, 2, 2), (2, 4, 2), (4, 5, 2), (1, 3, 1), (3, 4, 1), (4, 5, 1)])
------------------------------------------------------------
```

Some Test cases I ran on my algorithm.

- Time complexity of ***"bfs_shortest_path":***

  O(V+E), where V is the number of vertices and E is the number of edges in the graph. BFS traverses each vertex and edge once in the worst case.

- Time complexity of ***"dijkstra_max_path":***

  The time complexity is **O(V^2)** because it iterates over all nodes V and checks all edges in a nested loop for each node.

- Time complexity of ***"Max_Flow_Short (Ford-Fulkerson using Short Pipes strategy)":***

  This depends on the number of iterations and the complexity of ***bfs_shortest_path***. In the worst case, the algorithm could find a path that only augments by 1 unit of flow each time. Thus, if the maximum flow is ***F***, the algorithm could potentially iterate ***F*** times. Combining this with the complexity of ***bfs_shortest_path***, the overall time complexity becomes **O(F·(V+E))**
- Time complexity of ***"Max_Flow_Fat (Ford-Fulkerson using Fat Pipes strategy)":***

It is also dependent on the ***dijkstra_max_path*, **Using the same reasoning for the number of iterations, and considering the O(V^2) complexity of dijkstra_max_path, the overall time complexity is **O(F·V^2).**

2. (10 points) Suppose you are running a conference and want to assign 40 papers to 12 reviewers. Each reviewer bids for 20 papers. You want each paper to be reviewed by 3 reviewers. Formulate this problem as a max-flow problem, i.e., describe the architecture of the network and the capacities of the edges. Assume a reviewer cannot review more than 11 papers. What is the maximum flow you can expect in your network?

To formulate this conference paper review problem as a max-flow problem, let's structure the flow network and define the capacities of the edges, considering the constraints you've mentioned:

1. Network Structure:

Source Node (S): Represents the origin of the flow, symbolizing the initiation of paper assignments.

Set A (Papers): Contains 40 nodes, each representing a paper. These nodes are directly connected to the source node (S).

Set B (Reviewers): Contains 12 nodes, each representing a reviewer. These nodes are connected to the sink node (T).

Edges from Papers to Reviewers: Create an edge from each paper in Set A to each reviewer in Set B who bid for that paper.

Sink Node (T): Represents the endpoint of the flow, symbolizing the completion of paper reviews.

2. Capacities of Edges:

Edges from Source to Papers (S → A): Each edge from the source node (S) to a paper in Set A has a capacity of 3, as each paper needs to be reviewed by 3 different reviewers.

Edges from Papers to Reviewers (A → B): Each edge from a paper in Set A to a reviewer in Set B has a capacity of 1. This represents that a paper can be assigned to a reviewer only once.

Edges from Reviewers to Sink (B → T): Each edge from a reviewer in Set B to the sink node (T) has a capacity of 11, reflecting the maximum number of papers a reviewer can review.

3. Max Flow Calculation:

Applying the Ford-Fulkerson algorithm to calculate the maximum flow from the ***source (S)*** to the ***sink (T)***. This will determine the maximum number of paper-reviewer assignments possible under the given constraints.

<u>4. Maximum Flow Expectation:</u>

Theoretically, the maximum flow in this network should be 120 (which is ***<u>(40 * 3)</u>***), as there are 40 papers and each need 3 reviews. However, this maximum flow might not be achievable due to the constraints on reviewer capacities and the specific papers they bid for. The actual maximum flow will depend on how the reviewers' bids are distributed among the papers.

3. (10 points: Hall's theorem) Consider a bipartite matching problem of matching *N* boys to *N* girls. Show that there is a perfect match **if and only if** every subset of S of boys is connected to at least |S| girls. Hint: Consider applying the Max-flow Min-cut theorem.

The matching ***N*** boys to ***N*** girls, a perfect match exists if and only if every subset ***S*** of boys is connected to at least ***/S/*** girls. This can be understood through two cases, aligned with the Max-flow Min-cut theorem:

1. **If a Perfect Match Exists**:
   - Assume there's a perfect matching.
   - For any subset ***S*** of boys, if they were connected to fewer than ***/S/*** girls, then not all boys in ***S*** could be matched uniquely to different girls, as there wouldn't be enough girls for each boy.
   - This contradicts the existence of a perfect match, showing that for a perfect match, every subset ***S*** of boys must be connected to at least ***/S/*** girls.
2. **If Every Subset of Boys is Connected to at Least |S| Girls**:
   - Now, assume that for every subset ***S*** of boys, there are at least ***/S/*** girls connected to them.
   - Start by matching one boy in ***S*** to a girl. Then, a reduced subset ***S−1*** remains, which must be connected to at least ***/S−1/*** girls.
   - Proceeding in this manner, each boy can be matched to a different girl, ensuring a perfect match.

This approach confirms that a perfect match is guaranteed if and only if every subset ***S*** of boys has at least ***/S/*** girls available for matching. The Max-flow Min-cut theorem aids in visualizing this by considering the flow of potential matches from boys to girls and ensuring that the capacity (the number of girls available) is never less than the demand (the size of the subset of boys).

4. (10 points) Suppose you want to find the shortest path from node *s* to *t* in a directed graph where edge *(u,v)* has length *l[u,v] > 0*. Write the shortest path problem as a linear program. Show that the dual of the program can be written as *Max X[s]-X[t]*, where *X[u]-X[v] <= l[u,v]* for all *(u,v) in E*. [An interpretation of this dual is given in page 209 of DPV.]

Primal Problem (Shortest Path as Linear Program)

1. **Objective Function**:
   - Minimize $\sum_{(u,v)\in E} l[u,v]\cdot x[u,v]$
   - Here, ***x[u,v]*** is a decision variable indicating whether edge ***(u,v)*** is used in the shortest path (often set to 1 if the edge is used, 0 otherwise).
2. **Constraints**:
   - **Flow Conservation**: For every vertex ***v*** (except ***s*** and ***t***):
     - $\sum_{(u,v)\in E} x[u,v] = \sum_{(v,w)\in E} x[v,w]$
   - **Source and Sink Constraints**:
     - $\sum_{(s,u)\in E} x[s,u] = 1$ *(exactly one edge leaves the source)*
     - $\sum_{(u,t)\in E} x[u,t] = 1$ *(exactly one edge enters the sink)*
   - **Non-Negativity and Binary Nature**:
     - $x[u,v] >= 0$ and typically, $x[u,v] \in \{0,1\}$ for all edges $(u,v) \in E$.

Dual Problem

The dual of this linear programming problem would involve a new set of variables, say ***X[v]*** for each vertex ***v***, representing a potential function.

1. **Objective Function**:
   - Maximize $X[s]-X[t]$
2. **Constraints**:
   - For every edge $(u,v) \in E$, $X[u]-X[v] <= l[u,v]$.

The primal problem focuses on the flow along the edges to find the shortest path, while the dual problem interprets these edge lengths as constraints on the potential differences between vertices. The dual constraint $X[u] - X[v] <= l[u,v]$ ensures that the increase in potential from ***u*** to ***v*** does not exceed the length of the edge connecting them, aligning with the idea of not exceeding the shortest path length.

References:

1. **Geeks For Geeks, https://www.geeksforgeeks.org/**
2. **StackOverflow, https://stackoverflow.com/**
3. **W3 Schools, https://www.w3schools.com/**