CSE 551 Programming Assignment – Spring'21

Problem:

To determine whether there are m-vertex disjoint paths from the starting points to any m distinct points in the boundary.

Fact:

Two paths from vertex u to vertex v are said to be vertex disjoint if they do not contain any common internal vertex in them.

Solution:

Let's take an example with a network, G=(V, E) with vertex capacity as d: V -> R and edge capacities as c: E-> R.

For every v ∈ V, we need $\sum_{u \in V}\{f(u,v)|f(u,v) > 0\} \leq d(v)$

To solve the problem, we can use Maximum Flow algorithm. We need to convert the given 'Funny Problem' to Single Source – Single Sink Maximum Flow Path. This problem can be reduced to network flow problem with each vertex having unit capacity of 1.

We would add source vertex s and destination vertex t to the grid. We will connect s to each of the m starting points and we will connect t to every boundary point on the grid. Every edge from the source and every edge to the sink will have the capacity as 1.

The solution to the funny problem is equivalent to the answer of whether there is a maximum flow of value at least m in the given network.

By applying the below algorithm, we can reduce this problem to vertex capacities to a traditional flow problem with edge capacities in linear time. These computations, along with the flow back to the grid provides the vertex disjoint paths for this problem. We create a new flow network here, $G' = (V', E')$ with edge capacities such that for every maximum flow in G would correspond to a maximum flow in G'. Let the edge capacity of G' be $c' : E'$ -> R.

We will use Edmond-Karp algorithm here to calculate the maximum flow. Edmond-Karp is the implementation of Ford-Fulkerson method that uses BFS (Breadth First Search) to find the augmented paths.

Implementation algorithm:

- Add two additional vertices for source, s and sink, t
- For each vertex u in the set of starting vertices for this problem(Total starting points = m), add edge from s to u with capacity as 1
- Split each vertex v ∈ V in the graph into two vertices : v_{in} and v_{out} (except source s and sink t)
- For each vertex v, add an edge from v_{in} to v_{out} with unit capacity of 1. Therefore, $V' = \{v_{in}, v_{out} \mid v \in V\}$,

$$E' = \{(u_{out}, v_{in}) \mid (u, v) \in E\} \cup \{(v_{in}, v_{out}) \mid v \in V\}$$
$$c'(u_{out}, v_{in}) = c(u, v), for\ every\ (u, v) \in E$$
$$c'(v_{in}, v_{out}) = d(v), for\ every\ (v) \in V$$

- For each of the grid boundary vertices v (Total boundary points = 4n – 4), add an edge from v_{out} to t with unit capacity of 1. [Any undirected edge (u, v) in original network G is now replaced by two directed edges (v_{out}, v_{in}) and (v_{in}, v_{out}). Therefore, any flow through the original network satisfying

vertex capacities corresponds to a flow with the same value in the new network, satisfying edge capacities. It is computable in linear time.]

- Ensure that for every edge in graph G, the edge capacity = 1, and for every vertex, the vertex capacity = 1
- Maximum Flow from s-t is calculated using Edmonds-Karp Algorithm
- s-t max-flow would give the maximum number of s-t vertex disjoint paths
- If Maximum s-t flow = |V|, where V is set of vertices in the set, (m in this case), then all the vertices in V can reach the boundary vertices in grid using vertex disjoint paths and there is a solution to this 'funny' problem.
- The maximum flow can't be greater than the value of m by looking at the cut which has s by itself.
- If Maximum Flow value is less than m, then there is no solution which exists for the given network, because otherwise we could construct a flow with value m from the list of disjoint paths that are the ideal path solutions for the problem.

Complexity of the algorithm:

Cost of splitting vertex V into two vertices v_{in} and v_{out} to obtain V' is O(V) and cost of adding extra edges from vertices v_{in} to v_{out} and assigning it the capacity d(v), repeated |V| times in also O(V).

Modified vertex and edge sets in the maximum-flow network,

$|V'|$ = 2.|V|

$|E'|$ = |E| + |V|

Cost of implementing Ford-Fulkerson Method algorithm for maximum flow on G' = O ($E'.|f^*|$) = O((E + V) .$|f^*|$), where f^* is the maximum flow, provided the capacities are of integral values (unit value in this case).

After running Edmonds-Karp implementation on G', the running time is O ($V'E'^2$).

Total complexity = Graph Modification Cost + Max Flow Algorithm Cost

= O (V) + O ($V'E'^2$) = O ($V'E'^2$)

= O (2.V. $(E+V)^2$) [The complexity might vary depending on the selection of the max-flow algorithm]

Based on the input size for this problem, the complexity can have a tighter bound. As the input grid has n^2 points and $2n^2$ – 2n undirected edges, the original network G would have 2 + n^2 = O(n^2) vertices and m + (4n-4) + 2($2n^2$-2n) = m + $4n^2$ – 4 = O(n^2) directed edges. Therefore, the running time using Edmonds-Karp implementation would be O (VE^2) = O(n^6).

Now, upper bound for Ford-Fulkerson based algorithms = O((E + V) .$|f^*|$), where f^* is the maximum flow. In this problem, $|f^*| \leq 4n - 4$ and |E| + |V| = O(n^2)

Therefore, the tighter bound for the overall complexity = O((E + V) .$|f^*|$) = O(n^3)

Steps to run the project:

In python environment shell, run the following commands:

pip install numpy

pip install scipy

python '.\Programming_assignment_Problem.py'

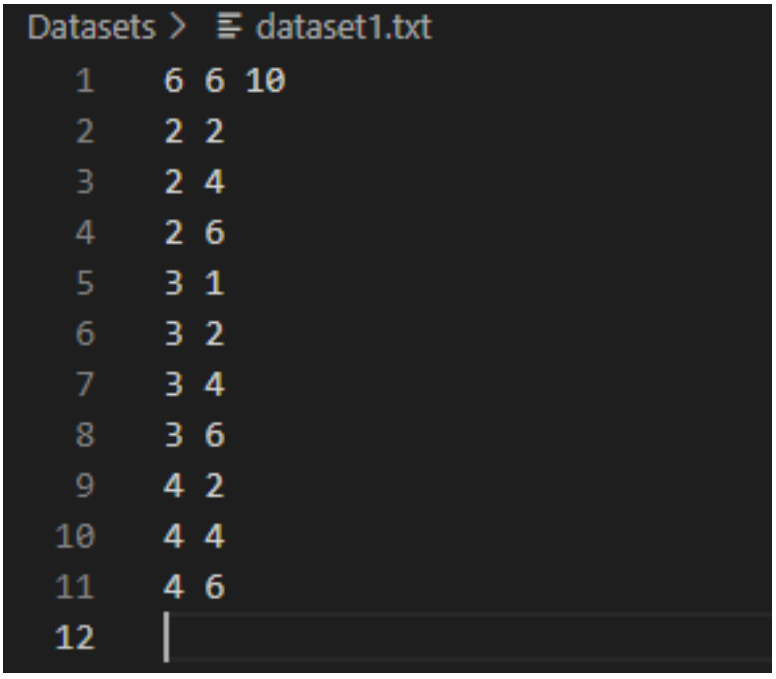
The input for the project needs to be mentioned in Datasets>dataset1.txt

Sample Test Case 1

Input:

```
n = 6
```

```
m = {(2,2),(2,4),(2,6),(3,1),(3,2),(3,4),(3,6),(4,2),(4,4),(4,6)}
```



First line will have the dimensions for rows and columns, in our case if will remain 6 and the third value will be total number of starting vertices i.e.10 in our case.

Then in every following line we will write the starting vertices in every new line with space as delimiter instead of comma (,)

Output:

```
Maximum Flow:  10
len(startingVertices):  10
Yes, the solution exists
PATH from (2, 6) : (2, 6)
PATH from (3, 1) : (3, 1)
PATH from (3, 6) : (3, 6)
PATH from (4, 6) : (4, 6)
PATH from (2, 2) : (2, 2) -> (1, 2)
PATH from (2, 4) : (2, 4) -> (1, 4)
PATH from (4, 2) : (4, 2) -> (4, 1)
PATH from (4, 4) : (4, 4) -> (5, 4) -> (6, 4)
PATH from (3, 2) : (3, 2) -> (3, 3) -> (2, 3) -> (1, 3)
PATH from (3, 4) : (3, 4) -> (3, 5) -> (2, 5) -> (1, 5)
```

Here, the output of the problem is **"Yes, the solution exists" that indicates a solution exists for the given set of vertices**. The solution exists because maximum flow equals the total number of vertices. [Maximum Flow: 10 and len(startingVertices): 10]

Also, there are paths from the starting vertices for this problem. **The above-mentioned PATHS are the vertex-disjoint paths from the starting vertices.**

Sample Test Case 2

Input:

```
n = 5
m = {(3,2),(2,3),(3,3),(4,3),(3,4)}
```

```
1   5 5 5
2   3 2
3   2 3
4   3 3
5   4 3
6   3 4
7
```

Output:

```
Maximum Flow:  4
len(startingVertices):  5
No, there exists no solution to this problem
```

"No, there exists no solution to this problem" in the line indicates that there doesn't exist solution to this problem. The solution does not exist because maximum flow is not equal to the total number of vertices. [Maximum Flow: 4 and len(startingVertices): 5]

Also, there exists no paths (vertex-disjoint paths) for this problem.