Programming Assignment

Submitted By: **Ankit Sharma**
ASU ID: **1219472813**

Funny Problem: Determine whether there are $m\ vertex\ disjoint$ paths from the starting points to any m distinct points on the boundary.

Approach:
To find out vertex disjoint paths or Escape paths we need to convert it into max-flow path problem. Following are the steps needs to be taken to convert given undirected graph G into a directed graph.

(i) To apply max-flow algorithm, a source s and a sink t is added to the graph.
(ii) All the vertices in the graph (except s and t) are split into v_{in} and v_{out}.
(iii) Add edge between v_{in} and v_{out} where capacity is set equal to 1.
(iv) From source s, add an edge of capacity 1 to each $start\_vertex_{in}$
(v) Since we need to maintain bidirectional movement of the original graph, for each vertex we will add edge of capacity 1 from

- v_{out} to v'_{in} of all four neighbors (top, down, left and right)
- v'_{out} of all four neighbors to v_{in}

(vi) v_{out} of all boundary vertices in the graph are connected to sink t with capacity equal to 1.

Since capacity between v_{in} and v_{out} is set to 1, there can be flow only once through them. Max-flow for the graph will give us number of vertex disjoint paths from start vertices to the boundary because each start vertex can only be used once. We calculate max-flow in the graph by Edmonds-Karp algorithm, which is a specific implementation of Ford-Fulkerson algorithm where it uses Breadth First Search (BFS) for searching augmented path.

Pseudocode:

contructFlowGraph*(dimension, startVerticesList)*

1. Number of Nodes in graph = $(\text{dimension})^2 * 2 + 2$
2. Set capacity of edge between $(v_{in}, v_{out}) = 1$
3. Connect boundary vertices to sink and set capacity 1
4. Connect source to $startVerticesList_{in}[\,]$
5. Connect each vertex (except source and sink) in graph to its four neighbors (left, right, top, down)
6. $maxFlow <- edmondsKarp(graph, source, sink, dimension)$
7. return $maxFlow$

edmondsKarp(graph, source, sink, dimension)

1. Run a BFS to find shortest path from source-to-sink
   $sPath\ <-shortestPath(predecessor[], source, sink)$
2. $minCap\ <-findMinEdgeCap(graph, sPath)$
3. $flow\ <-flow\ +\ minCap$
4. $augmentPath()$ the path in the graph with minimum value
5. return $flow$

shortestPath(predecessor[], source, sink)

1. Create $shortestpath[]$ array
2. Move through path using $predecessor[]$ received from BFS and keep adding vertices to $shortestpath[]$ array
3. return $shortestpath[]$

findMinEdgeCap(graph, shortestPath[])

1. Move through shortestPath
2. Find minimum value of edge in the shortestPath
3. return $minCapacity$

augmentPath(residualGraph, shortestPath[], minEdge)

1. augment residualGraph on shortestPath[] using minEdge value
2. return residualgraph

Input/Output Format

- **FunnyProblem.py** *reads input from* **input.txt** which should be in same location as FunnyProblem.py
- **First line of input.txt** file is:
  number_of_start_vertices dimension_of_matrix
- **Second line onwards** are start vertices x,y index

  x1 y1
  x2 y2
  .
  .
  .
  xn yn

SAMPLE INPUT:
10 6
2 2
2 4
2 6
3 1
3 2
3 4
3 6
4 2
4 4
4 6

OUTPUT:
(i) YES, a solution exists.
(ii) A solution to this problem is:
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)

Prerequisites

numpy and scipy libraries are used in this program. To install these libraries, execute following commands in your terminal:

1) pip install numpy
2) pip install scipy

How to run

Execute following commands in your terminal:

1) Change directory to FunnyProblem.py file location
   Ex: cd /Users/ankitsharma/Codes/CSE551-Foundation-of-Algorithms/Assignment5/

2) python environment variable should be set. Run using following command:
   python3 FunnyProblem.py