Akash Ashwinbhai Patel (#1219522499)

Foundation of Algorithm

Assignment – 5

Question – 1: Pseudo Code : Funny Problem
Answer →

To solve the problem, we will convert the funny problem to single source – single sink max-flow path problem. So, given undirected graph, we will convert it in to directed graph.

Implementation:

1. First, we need to divide every node into two node such as one node act as V_{in} and another node will act as V_{out}. We do this to represent points(x, y) coordinate and also to connect each point with its neighbor nodes.
2. To convert problem in single source – single sink max flow problem, we will add two extra node, one is source and another is sink. (Do not split it into two node).
3. To make sure we get unique path from starting vertices to boundary points we will assume each edge will have capacity of 1.
4. For each node, connect Vin and Vout with edge of capacity of 1.
5. Connect source node with edge to Vin of given starting vertexes.
6. Connect Vout every boundary node with edge to sink node.
7. So now problem covert to such as:
    a. Source → Vin(Starting Vertexes) →(find path to boundary node using Edmond-Karp algo.) → Boundary nodes(Vout) → Sink.
8. We will use Edmond-Karp Algo to find maximum Flow from Source to Sink of each node.
9. Algo from point (8) will give us Source-sink vertex disjoint path.

Algorithm.

```
Main(){
        n = input dimension.
        m = Starting vertices (Constraint m < n*n)
        For i in m:
                Point x = input()
                Point y = input()
```

Generate Adjacency matrix:

Dimension of matrix = (n * n * 2 (Split each node into two as Vin and Vout))+ (2(Source and sink))

Example: Grid for n = 2, hence : 2*2*2+2 = 10

	Source	Point 1: Vin	Point 1: Vout	Point 2:Vin	Point 2: Vout	……..	Sink
Source							
Point 1: Vin							
Point 1: Vout							
Point 2:Vin							
Point 2: Vout							
……							
Sink							

Fill the grid cell with following constraint:

1. For each node, connect Vin and Vout with edge of capacity of 1.
2. Connect source node with edge to Vin of given starting vertexes.
3. Connect Vout every boundary node with edge to sink node.

Apply Edmond Karp Algo on the adjacency matrix generated in previous step.

```
flow = Edmond Karp Algo(Adjacency Matrix, dimensions)

If(flow == m)
    Print(Possible)
    Print(finalPathFromStartingVertextoBoundaryNode)
Else
    Print(Not possible)

End of algo.
}

Edmond Karp Algo(Adjacency Matrix, dimensions){
    (global scope) List<Path> finalPathFromStartingVertextoBoundaryNode;

    parentPath = BreadthFirstAlgo(Adjacency Matrix)
```

```
        pathFromSourceToSink = getPath(parentPath)

        finalPathFromStartingVertextoBoundaryNode.add(pathFromSourceToSink)

        Repeat till (source not found in pathFromSourceToSink):
                minFlow = (get min flow from pathFromSourceToSink)
                flow = flow + minFlow
                Adjacency Matrix = generateResidualGraph(Adjacency Matrix,
pathFromSourceToSink, minFlow).

                parentPath = BreadthFirstAlgo(Adjacency Matrix)

                pathFromSourceToSink = getPath(parentPath)

                finalPathFromStartingVertextoBoundaryNode.add(pathFromSourceT
                oSink)


                Return flow(Number of disjoint path from each starting vertex to
boundary node).

}

BreadthFirstAlgo(Adjacency Matrix[][], dimensions){
        Queue q;
        visited array = [], //all false value
        parentPath array= [], // all -ve infinite value.
        q.enque(source)
        visited[source] = true;
        while(!q.isEmpty()){
                vis = q.deque();
                for i in dimensions:
                        if(Adjacency Matrix[vis][i] ==1 && !visited[i]){
                                q.enque(i);
                                parentPath[i] = vis;
                        }
        }

        Return parentPath.
}
```

Code is written in java.

Input:
N = 6
M = 10
Point 1 x = 3, y = 1
Point 2 x = 2, y = 2
Point 3 x = 3, y = 2
Point 4 x = 4, y = 2
Point 5 x = 2, y = 4
Point 6 x = 3, y = 4
Point 7 x = 4, y = 4
Point 8 x = 2, y = 6
Point 9 x = 3, y = 6
Point 10 x = 4, y = 6

Output :
YES, a solution exists.
A solution to this problem (a set of vertex-disjoint paths) is:
PATH from (2, 6): (2, 6) -> end
PATH from (3, 1): (3, 1) -> end
PATH from (3, 6): (3, 6) -> end
PATH from (4, 6): (4, 6) -> end
PATH from (2, 2): (2, 2) -> (1, 2) -> end
PATH from (2, 4): (2, 4) -> (1, 4) -> end
PATH from (4, 2): (4, 2) -> (4, 1) -> end
PATH from (4, 4): (4, 4) -> (5, 4) -> (6, 4) -> end
PATH from (3, 2): (3, 2) -> (3, 3) -> (2, 3) -> (1, 3) -> end
PATH from (3, 4): (3, 4) -> (3, 5) -> (2, 5) -> (1, 5) -> end