Assignment 8

Rahul Motan

**1)**

In the given problem we have to find the shortest path or the path that takes the least amount of time to reach the destination. We are given ’**k**‘ lives in the Jumanji World where each landmark acts as a vertex and there can be ’**n**’ such landmarks (deadly) stepping upon such landmarks would kill us and reduce our lives by 1. If we have more lives left then we respawn on the landmark that we died on, which makes the landmark not deadly.

We can modify the existing Bellman-Ford algorithm that finds the shortest path using k-edges. Using these k-edges we’ll find the most optimal path  vertices  from  such that

There could be a path that is the shortest but has  deadly landmarks such that , so we need to find a path from source to destination so that

**Recurrence**

Let’s assume  represents the path that takes the least amount of time between the source and  using at most  edges and lives

If a vertex  is unreachable from the source vertex using at most  edges or exhausting all lives then

In the given problem there is no constraint on the edges so we’ll explore the entire set of edges but we’ll keep track of the lives we have. In our algorithm we will fill the Memo table for each death incrementally.

Suppose we have the shortest path distances using at most  edges, then we can calculate all shortest path distance which use at most  edges by only extending every shortest path by one edge.

We can write the recurrence using the above statement for Bellman-Ford

…. (1)

We need to also consider paths that are using  edges to get to destination vertex as they could be smaller than

**ALGORITHM (Bottom-up Approach)**

1) Consider total edges in the Graph. Start finding paths to all vertices incrementing edges one by one.

2) We start with the source node  and we shall find path to all vertices  using edges (1, 2, 3, 4,… n-1)

3) If we explore a node that is a death trap then we continue exploring more vertices if we have more lives and Update the Memo Table

4) Else we do not relax edge and mark those edges as

5) We start again by incrementing our lives by one

**PSUEDOCODE**

We will use a 3 dimensional matrix as a MEMO - TABLE. M[k,V,m] where m are the edges

// k = lives

// G = graph

// s =  source

//  t = destination

Shortest-Path(G, s, t, k)

    1.    n = number of nodes in G

    2.    Array M[0….k,V,0…. n-1]

    3.    Define M[0,t,0] = 0 and M[0, v, 0]=

    4.

    5.

    6.         // nodes

    7.

    8.                    Compute  using the recurrence given above in Eq. (1)

    9.                            Update Memo table such that K(lives) > death traps in path and change parent of v to u

                                        if all lives exhausts then relax rest of the nodes for that row by making them .

   10.       Now increament k by 1 and again find path for all nodes and edges.

Now from the Memo Table M[i,t, k]  find the shortest path. There will be a finite optimal path if there is a way to reach the jaguar’s eye.

While we are relaxing the edges of each vertex from u we are also marking the parent of that node. We can use that to trace back the path to source.

**A Naive Approach**

Using DFS we can find all paths to the destination and keep track of death traps encoutered in the path and then find the shortest path amongs them that keeps us alive till the Jaguar’s eye.

**2.**

The problem given here is the problem of finding the optimal arrangement or the Maximum Bi-Partite Matching problem. There are a few facts that are given to us as follows:

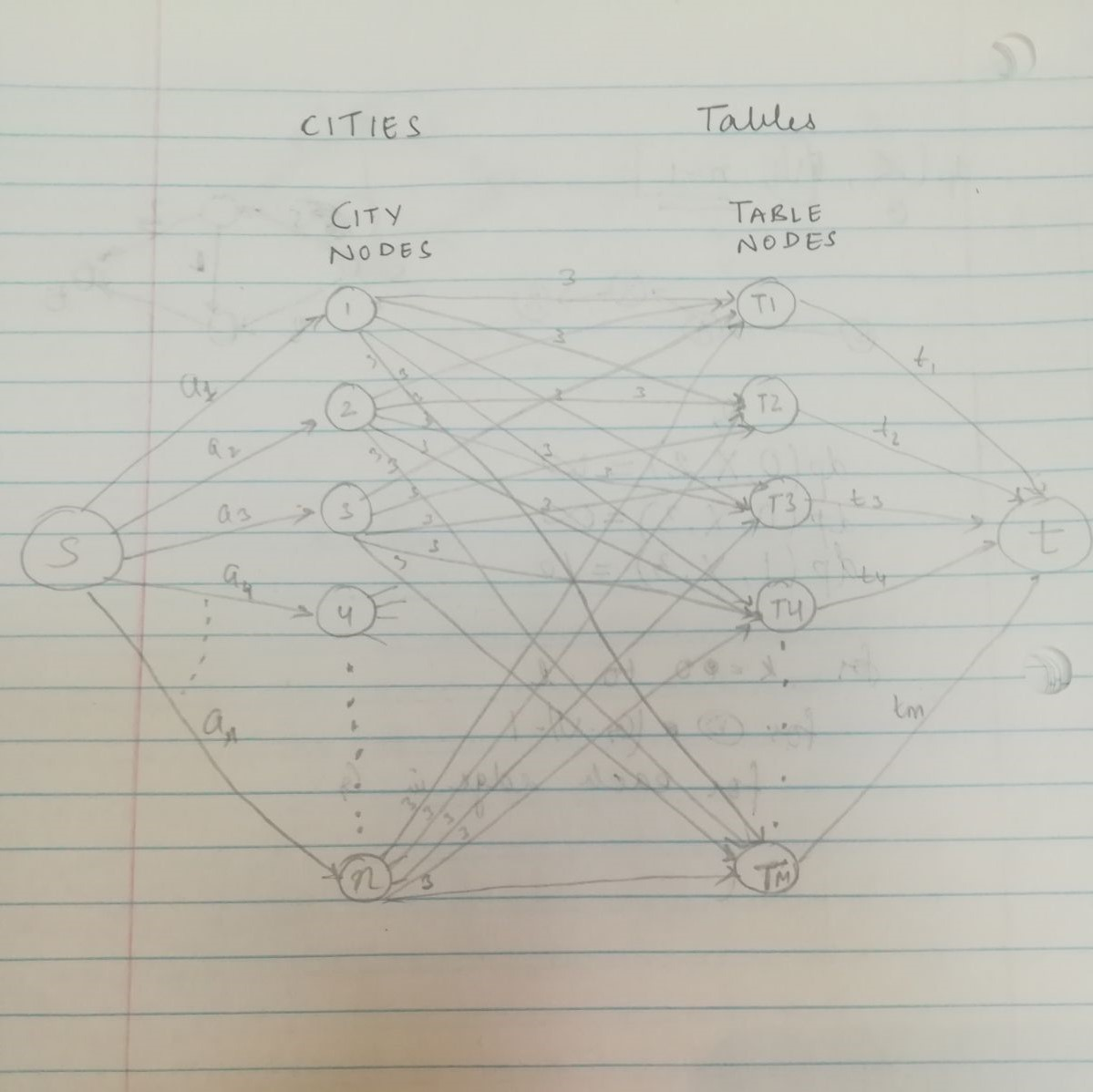
1. different cities from where the students are coming from city .
2. are the amount of students coming from city .
3. There are  tables.
4. Each table  has a capacity of

We can solve this problem using **Network Flows**

First, We need to construct a Network Flow using the following rules:

* We will first create  nodes, each representing a city. Label them as 1 through .
* Similarly we’ll create  nodes for tables  through
* Create a source node  and a sink node  in the network flow graph
* Create an edge from  to city : with edge weights or capacity as
* Similarly create edges from table  to sink node
* Now, to **restrict the flow** from each node representing a city, create edges from city  to each node that represents a table  such that  and  with a **capacity of 3**.

The network would appear like this:



The Network Flow of the optimal arrangement

We can find the solution by running Ford-Fulkerson algorithm on the above flow network and find the value of Max Flow.

If value of Max Flow is equal to sum of all the number of students from each city, then we can say that there exists a acceptable condition, such that every student from each city is seated at each table, and also no more than 3 students from the same city are sitting on the same table.

If the Max flow value is not equal to the sum of the total no. of students from every city, then there doesn’t exist enough tables to accommodate the students according to the given constraint or capacity of each table isn’t enough.

If the given constrained is satisfied then we can find the optimal seating arrangement,  which student sits on which table, we need to look at the residual graph G’ that is generated after finding the Max Flow.

There would be backward edges present after the last iteration of Ford Fulkerson algorithm in the graph G’, and to find the solution we need to look at backward edges that originate from table nodes towards city nodes with residual capacity of 3. Let’s represent these backward edges as

These backward edges of the form  depicts that 3 students from city  can sit on table .

**3.**

Let’s assume in a Graph G there is a cycle C which is the shortest cycle between Node A and C. A shortest cycle is also a simple cycle.

d(A,B) = 1 , d(B,C) = 1 , d(C,A) = 1 So Diam(G) = maximum distance which is 1. Cycle length is 3 which satisfies the condition.

Let’s assume the length of the Cycle be 2\*Diam(G)+2. This means there exist a pair of vertices in C that has distance of at least Diam(G)+ 1. But their distance is lesser than that in the graph considered here. And the cycle considered here is the shortest cycle so the shortest path between these nodes must be included in the shortest cycle which means the cycle is not the shortest cycle which is a contradiction and thus at most the length can be 2\*Diam(G) + 1.