CIS 675 Homework 7

Design and Analysis of Algorithms

Saket Kiran Thombre

SU ID: 899913802

NetID: sthombre

Email: sthombre@syr.edu

Collaborators for Q1, Q2:

1. Saurav Shashank Narvekar

   NetID: sanarvek

2. Chandan Pothumarthi

   NetID: cpothuma

PROBLEM 1 All Pairs shortest Path (Point 4)

Set BSHORTi,j,k to be the shortest path from i to j that uses only k hops. Note, this is different from the ASHORT variable that we used in class, in that we do not restrict the intermediate nodes to be 1 . . . k in this formulation. Meaning, the intermediate k hops in BSHORT can be any node k ∈ V. State a recursive formula for BSHORT. Devise an algorithm that uses this recurrence. (point 1). What is the running time of this algorithm? Show that, algorithm of BSHORT will result in a larger run-time compared to ASHORT (Point 1). Via a short improvement (but keeping the recursive formula same) it is possible to achieve run-time O(V^2E). What is that improvement? (Point2).

Solution:

We are given a directed graph; we have 2 vertices 'x' and 'y'.

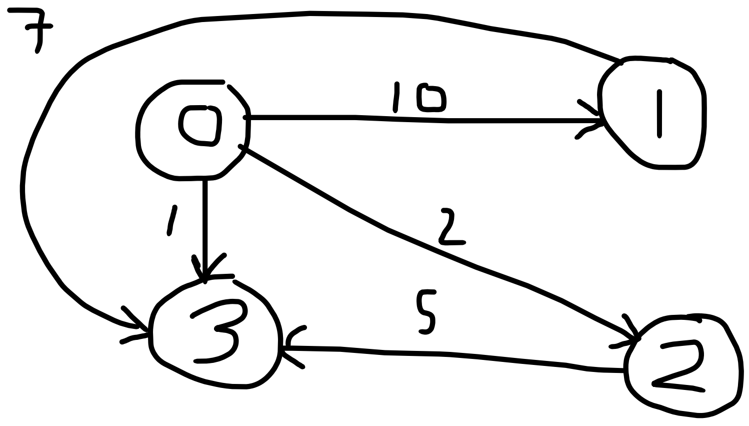
We are going to find the shortest path from 'i' to 'j'. This path will have precisely 'k' edges. That is path from 'i' to 'j' will have only k hops. This can be same as the shortest path that uses k-1 hops or a path where k-1 is the intermediate node with one hop left for the destination.

Let's take an example under consideration. Let's consider the following graph.

We will consider 'i' as 0 and 'j' as 3. Let k = 2.

We can find only 2 paths between them which are of length exactly 2. The paths are 0>2>3 and 0>1>3.

The cheapest path out of them is 0>2>3 which has a weight of 7.



Now let us generate a recurrence relationship for BSHORT variable

$$BSHORT = \begin{cases} W_{i,j} \; if \; K = 0 \\ OR \\ min \begin{cases} BSHORT_{i,j,k-1} \\ BSHORT_{i,k,0} + BSHORT_{k,j,k-1} \end{cases} \end{cases}$$

If K = 0, then we can consider that there is a direct edge between the 'i' and 'j'. That means there is no need for a hop.

Else, we need to verify the minimum weights by adding them up. Also, when we do BSHORT, when path length k-1, we may find a case where there exists an edge between 'i' and 'j'.

Otherwise, we can find the shortest path from 'i' to k with no hops is added with k vertex to 'j' with k - 1 hops.

Algorithm:

```
initialize result
result = ∞

Go to all adjacent vertices of i and perform recursion.
n = length(L)

for x in n:
    if graph[a][i] != ∞ and a != x and i != x:
        recursion = BSHORT(graph, i, j, k - 1)

        if recursion != ∞:
            result = min(result, graph[a][i] + recursion)
return result
```

The time complexity of this is O (V^k) as the algorithm will run k times in worst case where the destination could be the last connected edge in a graph.

If we organize the algorithm, we find that it will mimic the Bellman-Ford algorithm, however, it will present a result which has a larger run time than ASHORT.

Particularly the equation we wrote earlier contains (V + 1) terms present in it. So, if we consider a naïve implementation of this, we will require a O(V) run time. This will in turn result in O ($V^3.V$) since there are O (V^2) pairs (i,j) and K will also eventually reach its destination of K = V -1. This shows us that ASHORT algorithm only provides a O (V^3) run time where we hop from 'i' to 'j' that is source to destination by only traversing K hops.

Improvement:

We can improve by updating the matrix from time k to time k+1. This can we achieved when we inspect each edge of the graph.

We can store the weights of the traversed edges in an additional third dimension in the array. Each new edge e = (x, y) would only affect the V where V will be the possible entries in matrix $BSHORT_{y,k}$.

We will usually traverse the edges from the graph and fill the array matrix where we will get the minimum of weights using BSHORT. But if we consider worst case, where the destination is the last connected vertex and we would need to travel all the edges, we will get a time complexity of O (V^2E).

Hence, we can say that there is an improvement in the runtime by using memorization and DP.

problem 2 Max-flow Problem (Point 2)

Your task is to find a max flow in a directed graph. However, instead of having capacities on the edges, you have capacities on nodes. (Meaning, how much flow can be passed through a node.) All the other two rules of flow (1.for intermediate nodes, inflow=outflow, and 2. | f | =outflow from source- inflow to source node) remain same. Show how to find a max flow in such a case. Hint: Think about how to modify the graph, rather than the algorithm.

Solution:

In original max-flow problem It was okay to find any path from start to finish.

To find the max flow in a graph where the nodes have the capacities, we need to make a simple reduction for this.

To find the max flow with node capacities, it becomes necessary to change a few things in the nodes.

For every node v in our graph, we will replace it with 2 vertices namely ‘v_ingress’ and ‘v_egress’.

Similar to a router in networking terms, for the incoming edge coming to v should point at ‘v_ingress’ and outgoing edge should point at ‘v_egress’.

After making the following changes, create an additional edge from ‘v_ingress’ to ‘v_egress’ and lets name it ‘v_capacity’. This edge will ensure the max flow capacity. Also, the incoming flow cannot be greater than the total capacity.

The initial graph would look like this:

$$\begin{tikzcd} a \arrow[r, "2"'] & v \arrow[r, "2"'] & b \\ & 4 & \end{tikzcd}$$

After modification to the node

$$\begin{tikzcd} a \arrow[r, "2"'] & v\_ingress \arrow[r, "4"'] & v\_egress \arrow[r, "2"'] & b \end{tikzcd}$$

Therefore, we can say that maxflow solution is found.

problem 3 Class Allocation (Point 4)

Before the start of the Spring 2022 semester, the Registrar must assign each class to a time and a classroom. The classroom must be larger than the class it holds to properly seat all the students. Suppose there are n classes such that class i has si students enrolled. Syracuse University has m rooms, and room j can hold rj students. Finally, there are nonoverlapping time slots t1, ..., tk for the classes. For example t1 is 'CIS 675' and t2 is 'CIS 700' and so on. Given all this data, namely, given (s1,s2, ...,sn), (r1,r2...,rm), (t1, t2..., tk), design an efficient algorithm that assigns classes to times and classrooms. (point 2). Analyze the running time (point 1) and argue correctness (point 1).

Solution:

To design an efficient algorithm to assign classes to classes to times are classrooms.

This problem will be an instance of the bipartite graph matching problem.

Let us consider a graph with sink and source.

We create multiple set nodes N1 = (cs1, cs2, cs3, … , csn) where cs are classes available at Syracuse university.

Now we will create sets of nodes N2 = { (R_i, t_j) | where R_i is the room and j ∈ [1,k]}. This represents all room and time combinations possible.

Furthermore, we will connect node c_h to the node (R_i, t_j) if the room will be able to accommodate s_h students. This can be also written as $r_i > s_h$.

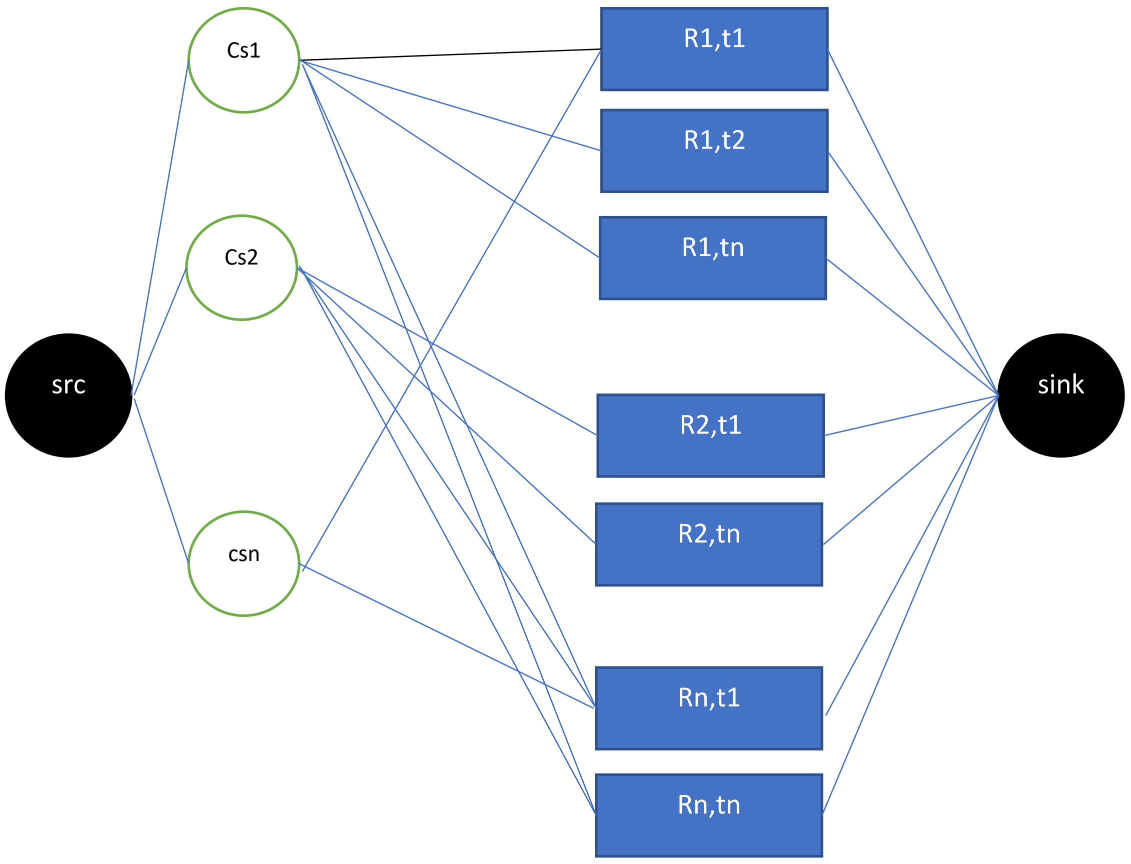
Now we initiate the bipartite graph matching and further on we use Ford-Fulkerson algorithm to find the maximum flow.

Algorithm:

```
start
initial flow = 0
while augmenting path from source to sink = true
    Add path to flow
return flow
```

We also run BFS in this algorithm. The idea behind running BFS is to pick a path with minimum number of edges.

Let us consider the following graph



As we know, there are (u + v. k) vertices which are present in this graph. There are (u.v.k) edges present in the graph.

If there is a matching of k number of edges in the graph, then we can say that there is a flow of value k. Similarly, if there is a flow of value k, then there is a matching with k number of edges.

initialize f(u,v,w) = 0∀u, v, w

while exists and augmenting path P in G’

augment f with Cf (P) = $Min_{(u,v,k)\in P}\ C_f$ (u, v, k)

Running time = (maxflow * E)

Running time = O(uvk . (u + vk)) = O($u^2vk + u(vk)^2$)

Correctness:

The output of the bipartite graph matching algorithm would always set of edges such that every vertex would be an incident to at the most one edge. Therefore, by definition, we can say that all classes are scheduled, and the room and time has been allotted to every class at the most once.