CSC373H1 - Assignment 2

Question 4

a) The network instance has a source S and sink T. It has a node (i, t) for each city $i \in V$ and for each $t \in \{0, \ldots, D\}$, which represent the cities at different times. Consider each t to be a layer.

For each edge $i \to j \in E$ and $t \in \{0, \ldots, D-1\}$, there is an edge between nodes (i, t) and $(j, t+1)$ with capacity $c_{i,j}$. Also, for each city $i \in V$ and $t \in \{0, \ldots, D-1\}$, there is an edge between (i, t) and $(i, t+1)$ with capacity P.

Connect S to $(a, 0)$ and T to (b, D), both with a capacity of P. For each $i \in V \setminus \{b\}$, connect T to (i, D) with capacity 0. This is to ensure that Ford-Fulkerson doesn't send any flow through these nodes to T.

If the amount of flow into T is P, the plan is feasible within time D, and infeasible otherwise.

b) We use the function BinSearch, which will call Helper. Helper attempts to find a solution with a given flow using Ford-Fulkerson; it returns the graph if true, and false otherwise. In BinSearch, we repeat calls to helper with input sizes 2^k for k from 1 to some i such that 2^i is the first power of $2 \geq D'$, the smallest feasible D. Then, we perform a binary search on the interval $(2^{i-1}, 2^i] = (2^{\lceil \log D' \rceil - 1}, 2^{\lceil \log D' \rceil}]$ since D' must lie in it. For each iteration of the search, Helper creates a new graph and checks if a flow of P is feasible.

Algorithm 1 Binary search for the smallest feasible D

1: **function** BinSearch(G, P)
2:     $i = 1$
3:     **while** Helper$(G, P, 2^i)$ is $Null$ **do** ▷ Helper is on next page
4:         $i \leftarrow i + 1$
5:     **end while**
6:     **if** $i = 1$ **then**
7:         Return 1
8:     **end if**
9:     low $\leftarrow 2^{i-1}$
10:     high $\leftarrow 2^i$
11:     **while** low $\neq$ high - 1 **do**
12:         mid $\leftarrow$ (low + high) / 2
13:         **if** Helper$(G, P,$ mid$)$ is not $Null$ **then**
14:             high $\leftarrow$ mid
15:         **else**
16:             low $\leftarrow$ mid
17:         **end if**
18:     **end while**
19: **return** high ▷ High is always the smallest D upon termination. This is proved below.
20: **end function**

We show that BinSearch finds D'. For the second while loop, the precondition is that $i > 1$ since $i = 1$ is addressed in the if statement on line 6. $i > 1 \implies 2^i > 2^{i-1} + 1$, so the loop runs at least once. We prove its correctness as follows:

Loop invariant for iteration k of the second while loop (LI_k): high is feasible (i.e., Helper(high) is not null) $\wedge$ low is not feasible $\wedge$ high $-$ low is a power of 2.

- Base case: LI_0 holds since by the first while loop, Helper(2^i) is not null and Helper(2^{i-1}) is null. $2^i - 2^{i-1} = 2^{i-1}$, which is a power of 2.
- Assume LI_k and consider iteration $k+1$. If Helper(mid) is not null, high = mid is feasible, while low remains infeasible by LI_k. If Helper(mid) is null, low = mid is infeasible, while high remains feasible by LI_k. Also, high $-$ mid = high $- \frac{\text{high}+\text{low}}{2} = \frac{\text{high}-\text{low}}{2}$, and similarly for mid $-$ low. Since high $-$ low is a power of 2 by LI_k, $\frac{\text{high}-\text{low}}{2}$ is also. Thus, LI_{k+1} holds.

The second while loop terminates since by LI, the size of (low, high] is a power of 2, and it is halved in each iteration. This means (low, high] will eventually have a size of 1, which corresponds to low = high $-$ 1, the termination requirement. When the loop terminates, by the LI, high is feasible and low = high $-$ 1 is infeasible, meaning that high = D' is returned, as needed.

Algorithm 2 Helper function to create the network

```
function HELPER(G, P, D)
    Create S, T
    for t = 0, ..., D do
        for i ∈ V do
            Create node (i, t)
            if t > 0 then
                Create edge (i, t − 1) → (i, t) with capacity P
            end if
        end for
    end for
    for (i, j) ∈ E do
        for t = 0, ..., D − 1 do
            Create edge (i, t) → (j, t + 1) with capacity c_{i,j}
        end for
    end for
    for i ∈ V do
        Create edge (i, D) → T with capacity 0
    end for
    Change edge (b, D) → T to capacity P
    Create edge S → (a, 0) with capacity P
    Compute a maximum flow f in the above instance with Ford-Fulkerson
    if f = P then
        return the constructed network
    else
        return Null
    end if
end function
```

We show that a maximum flow of P is equivalent to a feasible D.

($\implies$) Take any flow P, which is the maximum flow that S can send out and T can receive. We want to prove that D is feasible. Since the only edge to T with non-zero capacity is $(b, D) \to T$, it must be saturated with flow P. By conservation of flow, (b, D) must be receiving P flow for it to send it to T. This means every part of this flow passes through $(a, 0)$ and (b, D) along some $s - t$ path, which is length D if disregarding S and T since edges are only between nodes at different t. This means P people can travel to b in D steps of 1 unit time each, meaning D is a feasible deadline.

($\impliedby$) Assume D is feasible. We want to show that there is a valid flow P in the network described, which is the maximum possible. Notice:

- Since the only edge to T with non-zero capacity is $(b, D) \to T$, all flow must eventually arrive at (b, D), which itself sends P flow.
- All edge capacities are the same as in G. Edges between the same cities at different times represent people who are waiting and have capacity P (each city can hold P people).
- The length of any path from $(a, 0)$ to (b, D) is D since edges are only between nodes at different t (i.e., edges are always forward).

By the above and since D is feasible, a flow P sent from S can take the same actions as in the original schedule, moving in a forward edge of the network for each increment of time, and arrive at (b, D). Moreover, conservation of flow is respected since in the original schedule, the number of people going into or staying at a city at some t is the same as the number leaving or staying at $t + 1$. Thus, there is a valid and maximum flow of P in the network.

c) Define $n = |V|, m = |E|$. Note that every graph G has $\geq$ one $a - b$ path and its edge capacities are all ≥ 1. We can send each person into the graph individually with no spaces in between, like a queue. In this way, the last person must be free to move out of node a at time $P + 1$ after all preceding people have left a. Since this last person need only travel for $\leq n - 1$ nodes before reaching b, which would take time $n - 1$, every graph G needs at most $P + 1 + n - 1 = P + n$ time for all people to arrive at b. Thus, $D' \leq P + n$.

The overall runtime of BinSearch is $O((\log D')(n + m)D'P) = O\Big((\log(P + n))(n + m)(P + n)P\Big)$, since:

- The first while loop iterates over powers of 2 and stops at the smallest such power $\geq D'$, meaning that it runs for $\lceil \log D' \rceil \in O(\log D')$ steps. Each call to Helper calls Ford-Fulkerson, which is $O((n + m)D'P)$ since the maximum capacity is P and there are $2^{\lceil \log D' \rceil} \approx D'$ layers of nodes and edges. Also, the loops at lines 3, 11, and 16 in Helper are $O(D'n)$, $O(D'm)$, and $O(n)$ respectively. In total, the first while loop of BinSearch is in $O((\log D')(n + m)D'P)$.
- The second while loop performs binary search over the space $(2^{\lceil \log D' \rceil - 1}, 2^{\lceil \log D' \rceil}]$, which has size $2^{\lceil \log D' \rceil - 1}$, meaning that it runs for around $O(\log(2^{\lceil \log D' \rceil - 1})) = O(\log D')$ iterations. Each iteration of the second while loop runs Helper, which has the same runtime as noted above. In total, the second while loop of BinSearch is $O((\log D')(D'n + D'm)P)$.
- All other operations are $O(1)$.