CMPS 102 — Fall 2018 – Homework 4

Alyssa Melton

I have read and agree to the collaboration policy.

Collaborators: none

Solution to Problem 1

Charlie only likes two coffee shops, Valve and Ruru. For each minute at either coffee shop, we know precisely how productive Charlie will be, aka, we know how much work he will get done. This is given in the following sets:

$V = \{v_1, v_2, ..., v_i..., v_n\}$

$R = \{r_1, r_2, ..., r_i, ..., r_n\}$

While thinking of ways I can think of this problem, I considered creating a graph with weighted edges that represent the amount of work that Charlie can get done in between two minute marks.

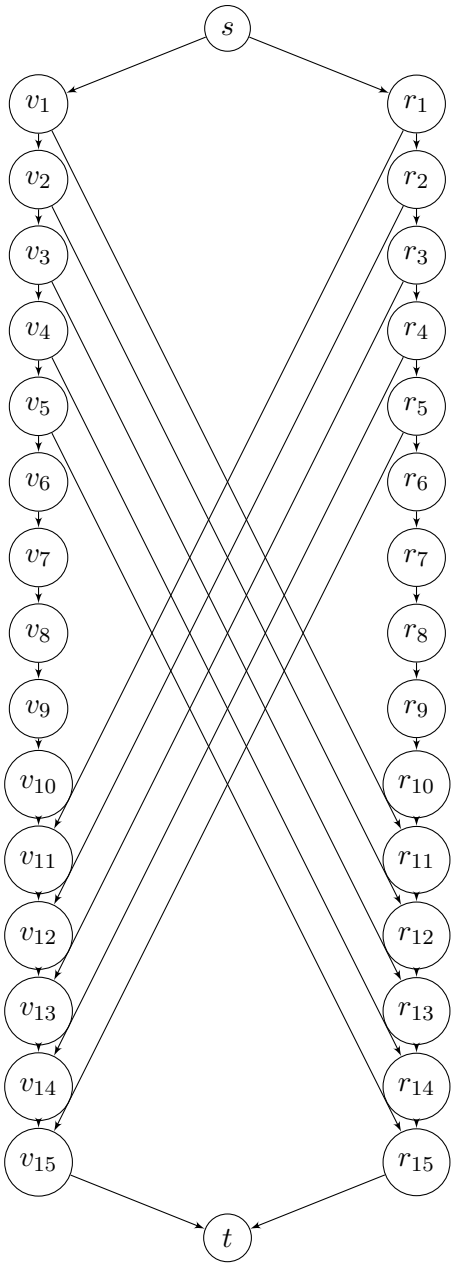
In the graph below, I am considering time being at the earliest at the top, and the latest at the bottom.

The minutes that Charlie can either spend at Valve or Ruru are vertices, namely V_1 to V_n, and R_1 to R_n respectively.

We create edges connecting each consecutive minute at the same coffee shop, making a direct path though V_1 to V_n, and the same with R_1 to R_n. For an edge (C_i, C_{i+1}), where C is the coffee shop (either V or R) and i is the minute, (C_i, C_{i+1}) is weighted with the value of the work that can be completed at time i at coffee shop C. For instance, weight $(V_3, V_4) = v_3$.

We also want to connect each vertex C_i to the vertex of the other coffee shop ten minutes later, namely, $\hat{C}_{i+10}$ if it exists. The weights of these vertices will be zero, since Charlie is not getting any work done while travelling.

Finally, we will connect the final two and beginning two nodes to a start node, s, and terminating node t. The nodes from s connecting to C_1 will have a weight of 0 since Charlie is not getting work done before he arrives, and the nodes from C_n connecting to t will have weight c_n. This will look something like this:



This of course, in our problem, will have a LOT more vertices and edges, because who the heck is going to consider switching coffee shops in between the span of 15 minutes. But anyways, as a subproblem, we want to find the maximum weight path, and note along the way when we switch from one coffee shop to the other.

To find the longest weight path, we can consider each vertex in order from earliest time to latest. For each time t, we will first consider v_t and then r_t. This is not for any particular reason, the two are interchangeable. Thus, our vertices will be looked at in our main loop in the following order:
Vertices : $\{s, v_1, r_1, v_2, r_2, ..., v_n, r_n, t\}$

Now, consider the following:

$work[v] = -1$ for all vertices v.
$prev[v] = null$ for all vertices v
for every vertex v in cronological order:
    for every (neighboring vertex of v), u:
        if $(work[u] < work[v] + weight[v])$:
            $work[u] = work[v] + weight[v]$.
            $prev[u] = v$.

To find where Charlie switches coffee shops, simply reccur backwards from t to s, and note when the previous of a vertex v is a different coffee shop than v. When we look at prev[v], we will get another vertex. For the sake of simplicity, assume we can tell by the name of the vertex (whether it is R or V), that a vertex is location Valve or Ruru. Consider:

$switches = \{\}$
$location = prev[t]$.
while $(prev[location] \neq null)\{$
    if $prev[location] \neq location$
        add $prev[location]$ to switches.
    $location = prev[location]$.
$\}$

$switches$ should now contain all the vertices that Charlie is located right before he switches to the other coffee shop.

If we didn't want to construct a graph, this can be written as the following, less intuitive, algorithm:

$work[t] = -1$ and $prev[u] = null$ for all times at either coffee shop v_1 to v_n, and r_1 to r_n.
$work[v_1] = 0$ $work[r_1] = 0$ for t (from 1 to $n-1$):
    if $work[v_{t+1}] < work[v_t] + v_t$
        $work[v_{t+1}] = work[v_t] + v_t$
        $prev[v_{t+1}] = v_t$
    if v_{t+10} is defined
        if $work[v_{t+10}] < work[v_t]$
            $work[v_{t+10}] = work[v_t]$
            $prev[v_{t+10}] = v_t$
    if $work[r_{t+1}] < work[r_t] + r_t$
        $work[r_{t+1}] = work[r_t] + r_t$
        $prev[r_{t+1}] = r_t$
    if r_{t+10} is defined
        if $work[r_{t+10}] < work[r_t]$
            $work[r_{t+10}] = work[r_t]$
            $prev[r_{t+10}] = r_t$

Claim 1. *The above algorithm correctly determines the most efficient locations to study.*

Proof. As a base case, consider when n = 1. We consider v_1, and assign work[t] = v_1, because all elements in V are non-negative and t was initialized to -1. We then consider r_1 and only if the value of r_1 is larger than v_1 do we replace the value of $work[t]$ with r_1. Thus, we have chosen the correct path.

Suppose the algorithm is correct for some $k \geq 1$. When n= $k+1$, we look at the value of the vertices with time k, and only update the value of the vertices adjacent to k if the value of k and the weight of k is greater than the current value. The vertices adjacent to k can either be of work value -1, or some positive integer. If -1, we update the value. If not, we update only if $work[k]$ + $weight[k]$ is greater than the current. Thus, t will be overwritten with whatever coffeeshop has the highest value of $k+1$. □

Claim 2. *The time complexity is $O(n)$ and the space complexity is $O(2n)$.*

Proof. Iterating through each vertex yeilds looping $2n$ times. Each vertex has at most two adjacent vertices, giving us $2n(2)$. With each adjacent vertex, we are doing a single comparison and two assignments if the comparison holds true. The algorithm is $O(2n(2)) \implies O(n)$.

We have two arrays of size $2n+1$ (work and prev) and a list of switching vertices that can be at most $n/10$ in length (when switching every two minutes). Thus, the space complexity is O($2n+1+n/10$) $\implies$ $O(n)$. □