CS 580 Algorithm Design, Analysis, And Implementation

Fall 2021 Vassilis Zikas HW 4

Problem 1

Collaborators: List students that you have discussed problem 1 with: Vishnu Teja Narapareddy, Tulika Sureka
We can use the dynamic programming approach to solve this problem.

Subproblem:
opt[i][j] denotes the maximum points the team can earn after the i^{th} round if j^{th} person plays that round. Here i varies from 0 to n-1 and j varies from 0 to 2, mapping the given arrays s.t J $\longrightarrow$ 0, E $\longrightarrow$ 1, and C $\longrightarrow$ 2.
For returning the array of who pushed the button in every round, we can maintain three strings for the three 1-D dp's which we are populating simultaneously. Initialize three empty substrings path_sara, path_eliz and path_charles for each of the team players and then keep appending as per the cases discussed below.

Base Cases:
Choose based on the max score and store the respective path in the structure

$$opt[0][0] = max\left\{\begin{array}{rl} 0, & path\_sara = path\_sara + \text{"\_"} \\ J[0], & path\_sara = path\_sara + \text{"}J\text{"} \end{array}\right\}$$

$$opt[0][1] = max\left\{\begin{array}{rl} 0, & path\_eliz = path\_eliz + \text{"\_"} \\ E[0], & path\_eliz = path\_eliz + \text{"}E\text{"} \end{array}\right\}$$

$$opt[0][2] = max\left\{\begin{array}{rl} 0, & path\_charles = path\_charles + \text{"\_"} \\ C[0], & path\_charles = path\_charles + \text{"}C\text{"} \end{array}\right\}$$

Recursion:
Choose based on the max score and store the respective path in the structure

$$opt[i][0] = max\left\{\begin{array}{rl} opt[i-1][0], & path\_sara = path\_sara + \text{"\_"} \\ opt[i-1][1] + J[i], & path\_sara = path\_eliz + \text{"}J\text{"} \\ opt[i-1][2] + J[i], & path\_sara = path\_charles + \text{"}J\text{"} \\ opt[i-1][1], & path\_sara = path\_eliz + \text{"\_"} \\ opt[i-1][2], & path\_sara = path\_charles + \text{"\_"} \end{array}\right\}$$

Similarly, we can write for other two members.

$$opt[i][1] = max\left\{\begin{array}{rl} opt[i-1][1], & path\_eliz = path\_eliz + \text{"\_"} \\ opt[i-1][2] + E[i], & path\_eliz = path\_charles + \text{"}E\text{"} \\ opt[i-1][0] + E[i], & path\_eliz = path\_sara + \text{"}E\text{"} \\ opt[i-1][2], & path\_eliz = path\_charles + \text{"\_"} \\ opt[i-1][0], & path\_eliz = path\_sara + \text{"\_"} \end{array}\right\}$$

$$opt[i][2] = max \left\{ \begin{array}{rl} opt[i-1][2], & path\_charles = path\_charles + "\_" \\ opt[i-1][0] + C[i], & path\_charles = path\_sara + "C" \\ opt[i-1][1] + C[i], & path\_charles = path\_eliz + "C" \\ opt[i-1][0], & path\_charles = path\_sara + "\_" \\ opt[i-1][1], & path\_charles = path\_eliz + "\_" \end{array} \right\}$$

Final ans to be returned is: max(opt[n-1][0], opt[n-1][1], opt[n-1][2]) along with the path stored alongside that value in the matrix
The maximum of these three values will be the maximum points which the team can earn after n rounds.

Analysing TC:
We traverse through the given three arrays simultaneously once. So, the TC = O(n)
Analysing SC:
We store 3n values in the matrix in the algorithm. So, the space complexity = O(n)

Proof of correctness:
We can use induction to prove the correctness of the algorithm.
For base case (i=0), it is trivial to see that max(opt[0][0], opt[0][1], opt[0][2]) gives us the optimal value.
Now, let say that this is true for all values till i-1. We need to show that it holds true for the i^{th} round too.

opt[i][0] denotes the max value the team can achieve if Sara plays in the i^{th} round. For the recursion mentioned above, we see that it considers these five possibilities:

1. Sara also played in the $(i-1)^{th}$ round, so she can't play this round
2. max score till i^{th} round was achieved when Elizabeth played last round and now we add Sara's score to the current max score if it's positive
3. max score till i^{th} round was achieved when Charles played last round and now we add Sara's score to the current max score if it's positive
4. max score till i^{th} round was achieved when Elizabeth played last round and now we do not add Sara's score to the current max score if it's negative
5. max score till i^{th} round was achieved when Charles played last round and now we add Sara's score to the current max score if it's negative

In this way, all the possibilities are taken care by the given algorithm which could have maximised the score if Sara played in the last round.
Similarly, we can calculate the values for Elizabeth and Charles.
Now taking the max(opt[i-1][0], opt[i-1][1], opt[i-1][2]) gives us the max score which can be scored by the team till the i^{th} round. We can see that any other value would be less than the max computed through these.
Hence, proved.

Problem 2

Collaborators: List students that you have discussed problem 2 with: Vishnu Teja Narapareddy, Tulika Sureka
We can use the max-flow algorithm to solve this problem.

Construction:
First of all, we convert the given undirected graph G to a directed graph by adding another edge between same set of vertices and then giving them the respective direction, keeping their weights as it is, if any, given in the original graph. Let's call this transformed directed graph as H
Afterwards, we split each vertex in graph H into two vertices, let say v_{in} and v_{out}, and connect these split vertices by an edge weight of 1. The other edges in graph H would be transformed as follows: edge (a,b) in graph H would correspond to the edge (a_{out}, b_{in}) with an edge capacity of 1 in the new graph. Let say this graph is J.
Now, we create a source node, s', and connect it to the vertex v_{in} with the edge weight as 2. Also, we connect the vertices s_{out} and t_{out} with a sink node, t', with an edge weight of 1. In the graph J, paths are edge disjoint if and only if they are vertex disjoint because of the splitting of vertices done by us.

Algorithm:
Now, we run the ford-fulkerson algorithm to get the max flow in the graph J. If the max flow algorithm gives the value as 2, then, we can say that there exists a path from s to t that passes through v in which none of the vertices are repeated. The path can be found by traversing the edges which are in the flow. Reverse the path for flow corresponding to v to s and append it to the path from v to t. Thus, it would give us the path from s to t, passing through v, without any repeatition of vertices.

Analysing TC:
In the construction of the graph, we go through the vertices and edges, so the time taken is O(V+E). The ford fulkerson algorithm takes the time O(Ef), where f is the max flow possible in the graph, which is 2 in our case. Hence, the total time complexity is O(V+Ef).

Proof of Correctness:
We can show that a set of edge disjoint paths in the graph J are also vertex disjoint. This is because after we split each vertex into v_{in} and v_{out} and joined them with an edge weight of 1, we are trying to force the flow to pass through that vertex only once. This is because if one of the augmenting paths chooses that vertex and flow passes through that, then, for other augmenting paths to be edge disjoint, flow can't reach to that same vertex's v_{in} as it would then lead to common edge (edge between v_{in} and v_{out}) for two flows in the graph which is not possible in the edge disjoint problem. Thus, it shows that set of edge disjoint paths in graph J are also vertex-disjoint.
Every path in the graph J can be represented as s' $\rightarrow v_{in} \rightarrow v_{out} \rightarrow \ldots\ldots\ldots \rightarrow m_{in} \rightarrow m_{out} \rightarrow t$. It corresponds to the following path in the original graph s $\rightarrow$ v $\rightarrow \ldots\ldots\ldots \rightarrow$ m $\rightarrow t$.
This shows that set of vertex disjoint paths in the original graph corresponds to a set of vertex disjoint paths in the graph J.
Hence, we say that set of vertex disjoint paths in the graph J corresponds to a set of vertex disjoint paths in original graph.

Problem 3

Collaborators: List students that you have discussed problem 3 with: Vishnu Teja Narapareddy, Tulika Sureka

(a) Modelling max flow as linear program by defining variables for each s-t path:
Let us denote the set of all possible simple paths from s to t by P. Now, consider the variable x_p, which denotes the flow which is going along the path p in the total s-t flow.
Our objective functions calculates the sum of flows along each path, which we need to maximise to get the

total flow for the graph:

$$\sum_{p \in P} x_p$$

Following are the constraints for the given objective function:

$$\sum_{p \in P:(u,v) \in p} x_p \leq C(u,v) \qquad \forall (u,v) \in E$$

$$x_p \geq 0 \qquad \forall p \in P$$

The first constraint condition denotes that sum of all flows (considering all the paths) along an edge cannot be greater than the capacity of that edge. This constraint is equivalent to the capacity constraint in the flow problem.
The second constraint condition denotes that flow along any path p is non-negative.
There can be exponentially many possible paths from s to t and thus the number of variables and constraints which we have are exponential too.
This primal LP solves the problem of max flow as it takes into consideration the constraints for the normal max flow and tries to maximise the simulated flow. The LP relaxation allows fractional flows along the edges.

(b) Dual of the above formulated LP looks like this:
We have one variable $x_{u,v}$ for every edge (u,v) ∈ E. It basically tell us whether that edge is in cut or not. C(u,v) denotes the capacity of the edge (u,v)
Our objective function would be this, which needs to be minimised:

$$\sum_{(u,v) \in E} C(u,v) x_{u,v}$$

Following are the constraints for the given objective function:

$$\sum_{(u,v) \in p} x_{u,v} \geq 1 \qquad \forall p \in P$$

$$x_{u,v} \geq 0 \qquad \forall (u,v) \in E$$

The first constraint condition denotes that the separation between s and t along any path is alteast greater than or equal to 1. This basically stands for the number of cuts possible in that path. It means that for each path atleast one edge must be in the cut
The second constraint condition denotes that the edge cannot contribute negatively to a cut.
The dual tries to solve the problem of min-cut by minimising the s-t cut under the constraint that atleast one edge on the path has to be a part of the min-cut. The LP relaxation allows fractional contribution of the capacity of an edge towards the cut.

Problem 4

Collaborators: List students that you have discussed problem 4 with:None
Yes