Saleh Alkhalifa

Q1
15 Points

Consider the following problem on a dictionary of n words, $W_1 ... W_n$, each with exactly k characters.
You can transform a word W_i into word W_j if they differ in at most $d \leq k$ characters. (both d and k are specified as part of the input, along with n and the words)

For example, if the dictionary is:

W_1 = 'hit', W_2 = 'cog', W_3 = 'hot', W_4 = 'dot', W_5 = 'dog', W_6 = 'lot', W_7 = 'log', and $d = 1$, one way to change 'hit' to 'cog' is:

'hit' $\rightarrow$ 'hot' $\rightarrow$ 'dot' $\rightarrow$ 'dog' $\rightarrow$ 'cog'.

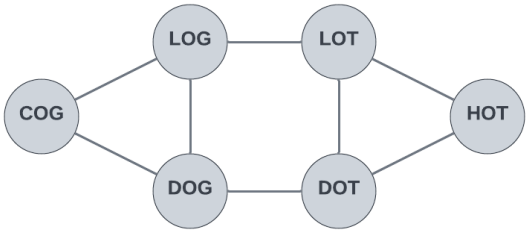
We want to find the fewest number of steps to transform W_1 to W_2.

Q1.1
5 Points

I claim that this problem be formulated as a shortest path problem. Pease provide a strategy to formulate this as a shortest path problem and give a graph visualization of the problem

Answer:

- Our objective here is to consider n words, denoted as W1, W2, … Wn where each word has exactly k characters describing it
- We are told we can transform a word into another in which they differ at most d=> k
- A strategy we can take here is a graph strategy in which we present each of the words we are given as nodes.
- Next, we can connect these nodes together between each of the words as edges, based on the changes they require to transform from one word into another.
- A word that can transform into another given one change, will be separated by one edge.
- One can interpret this as a graph in which a given word shares an edge with another word if they have two letters in common, and only one letter needs to be changed.
- We can see a depiction of this as a shortest path problem since we need to reach from one word to another in the shortest distance possible
- The diagram below illustrates this as a shortest path graph:



Q1.2
5 Points

Show that your graph (in Q 1.1) can be constructed in $O(n^2)$ time, and its size is up to $O(n^2)$.

Note: We definitely can consider the graph construction as being $O(n^2k)$ by comparing every pair of words.

- We can construct the graph above with a time complexity of $O(n^2)$
- To accomplish this, we can view the words W1, W2, … Wi as nodes within in a graph network as shown in the question above. To move from one node to another, we mutate a letter, so we can see this as a directed graph.
- To start, we must create n nodes, one for each of the words provided in the list above.
- Next, we need to iterate over the words in the form of a loop, which will be done n times.
    - For each we need to check a conversion can be made from one word to another.
- Given the nested nature of these two loops, each going through the results n times, we will see a time complexity of $O(n^2)$
- We can represent the number of edges in this graph as ${}^nC_2$
- Asymptotically, this is **$O(n^2)$**

Q1.3
5 Points

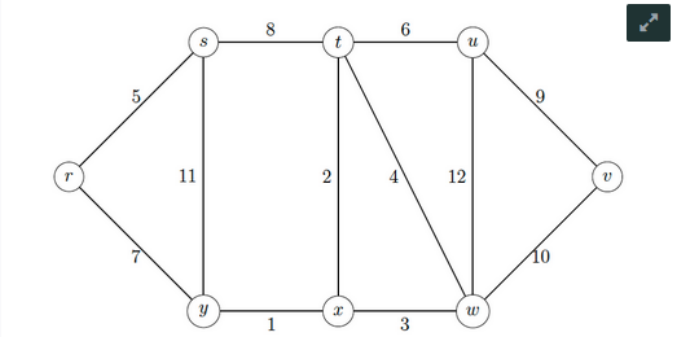
Using your formulation above, give a $O(n^2k + n^3)$ time algorithm for finding the minimum number of steps (you only need to describe how to find the number of steps, not the sequence of transformations).

- Our objective here is to give an algorithm with a time of $O(n^2k+n^3)$ to find the minimum number of steps
- We can likely use the Bellman-Form algorithm to accomplish this, which has its own time complexity of $O(n^3)$
- We select this algorithm because it operates by underestimating the length of the path from that beginning vertex to all others, and it iteratively discovers new paths that are shorter.
- The concept here of shortest distance can help with the use case of words above.
- Algorithm:
    - The first step here is to create a list of the graph's edges, generally presented in the form of an adjacency list or what not.
    - Next, The number of iterations are calculated. This is generally done in the form of V-1 given that the algorithm is formed around the concept of shortest-distance.
    - Next, a loop is entered in which we iterate such that for each edge u-v in the graph we compute the path lengths
        - If the distance v is greater than distance u plus the edge weight uv, then the distance v is set to the distance u plus that edge weight.
    - Finally, that distance is returned.
- Form the previous question we saw that the construction of the graph can take $O(n^2k)$, coming from the vertices and edges in the Graph itself, and contrasting words.
- Therefore, given the use of the algorithm, we will see a runtime of $O(n^2k+n^3)$

Saleh Alkhalifa

Q2
5 Points

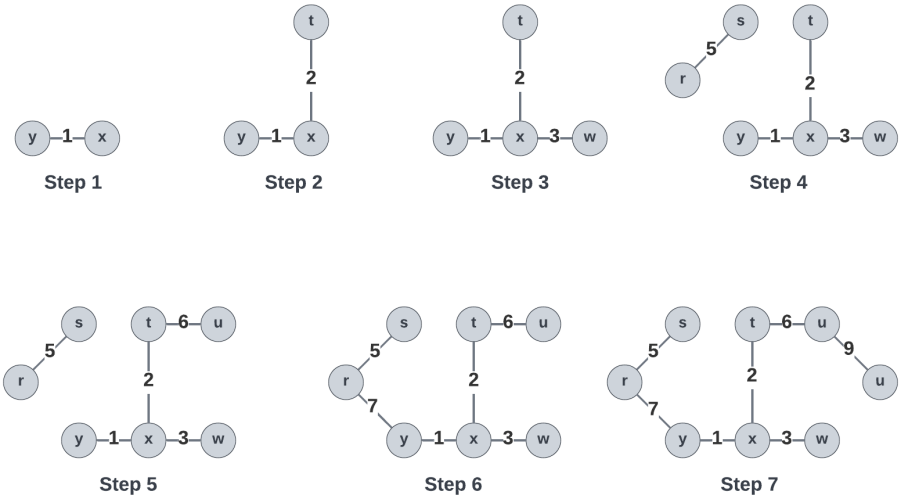
Consider an undirected graph on 8 vertices, with 12 edges given as shown below:



Q2.1
2 Points

Give the result of running Kruskal's algorithm on this edge sequence (specify the order in which the edges are selected).

- The idea here is to apply Kruskal's algorithm on this graph, and specify the order in which the edges are selected.
- The idea behind Kruskal is generally for minimum spanning trees (MST) of a given graph network
- The method operates by sorting edges based on weight from low to high, and the continuously connects them, but stops if a connection will create a cycle.
- We can see the order in which the edges are selected depicted below:

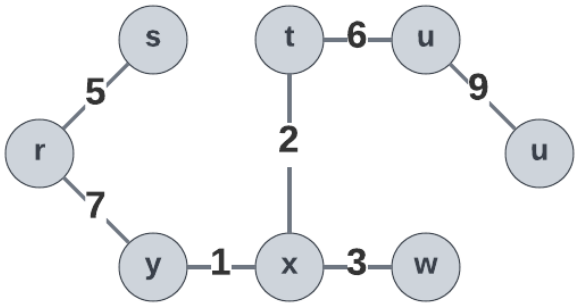


- Final Order: xy, xt, xw, rs, tu, ry, uv

Q2.2
3 Points

For the same graph, exhibit a `cut` that certifies that the edge ry is in the minimum spanning tree. Your answer should be in the form of $E(S, V \backslash S)$ for some vertex
set S. Specifically, you should find S.

- We can define a cut as a partition of the vertices for a given graph
- In this question, our objective is to exhibit a cut that certifies that the edge ry is in the MST
- To begin, we can review the graph which we developed from Kruskal's algorithm we prepared in the earlier question:



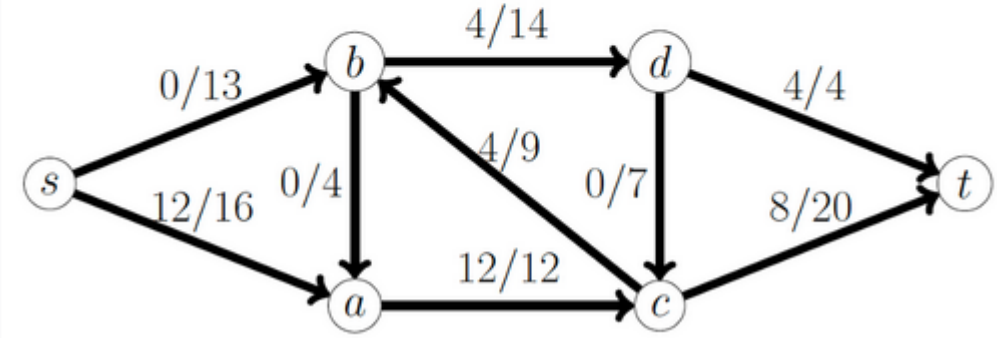
- For the cut of edge x-y,we see the minimal edge of the graph network outline above.
- If we removed this edge, we will not have a minimum spanning tree
- Using the form E(S, V\S), we arrive at E(X, Y\X)
- However, the sy edge is an edge cut that was not taken, therefore one can argue that the edge ry is the minimum spanning tree.

Saleh Alkhalifa

Q3

20 Points

Consider the following directed network with flows written as the first number and edge capacity as the second on each edge:

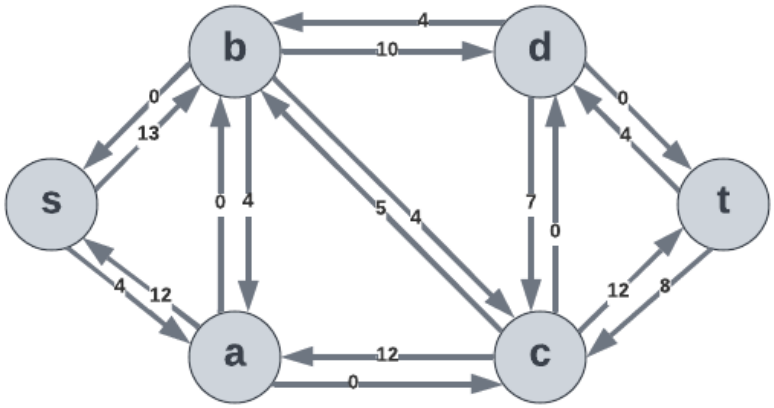


Q3.1

5 Points

Draw the residual network obtained from this flow.

- We can define the residual graph R of a network G as a graph that has the same of vertices as the original network, but with a forward and backwards edge
- We can use residual graphs to help solve maximum flow problems for a given graph G
- For each edge e of a given graph, we note that e = (u, v) ∈ G:
  - Forward edge = (u, v) with capacity c-f
  - Backwards edge = (v, u) with capacity f
- Given that definition, we arrive at the following residual network:



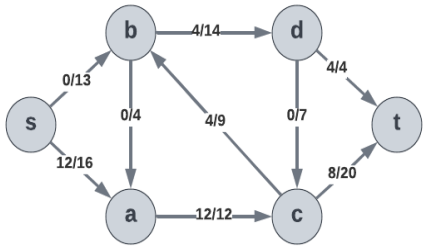
Saleh Alkhalifa

Q3.2
10 Points

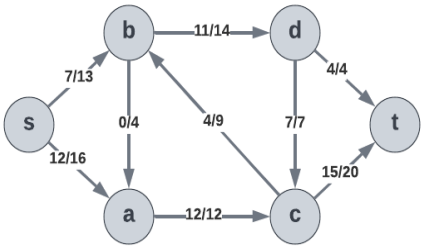
Perform two steps of the Ford Fulkerson algorithm on this network, each using the residual graph of the cumulative flow, and the augmenting paths and flow amounts specified below. After each augment, draw two graphs, preferably side by side; these are graphs of:
a) The flow values on the edges
b) Residual network

The augmenting paths and flow amounts are:
i) $s \rightarrow b \rightarrow d \rightarrow c \rightarrow t$ with flow amount 7 Units

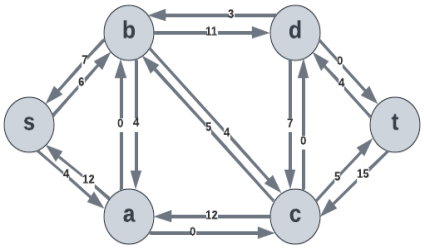
- We can recall that Ford-Fulkerson is a greedy algorithm to calculates the maximum flow in a flow network.
- The algorithm operates by:
  - Start with the flow initially at 0
  - While there exists an augmenting path s -> t
    - Add this path flow to the flow
  - Return the final flow



Original



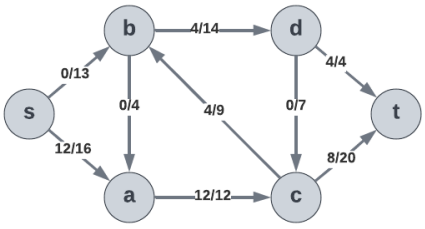
Flow



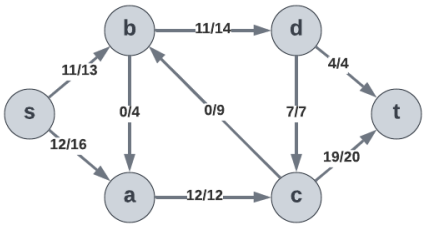
Residual Graph

ii) $s \rightarrow b \rightarrow c \rightarrow t$ with 4 units.
Note for continuity your second graph should be coming from the one in (i) NOT from the initial graph.

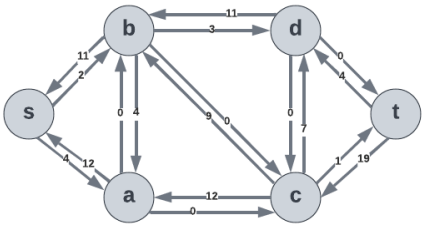
- Per the requirement above, we will continue from the graph outlined in (i)
- Continuing with the same idea and process as outlined above, we arrive with the following figure representing the original graph, the flow, and residual graph:



Original



Flow



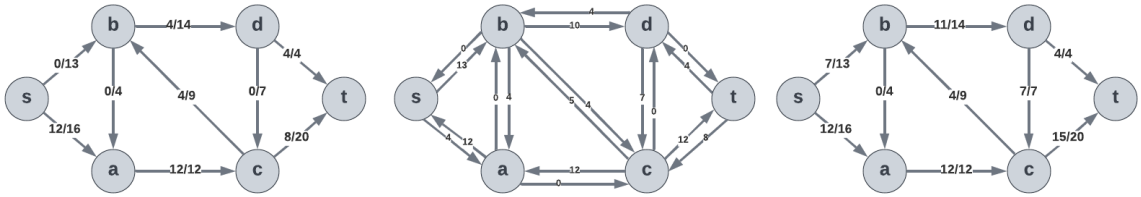
Residual Graph

Saleh Alkhalifa
12/5/2022

Q3.3
5 Points

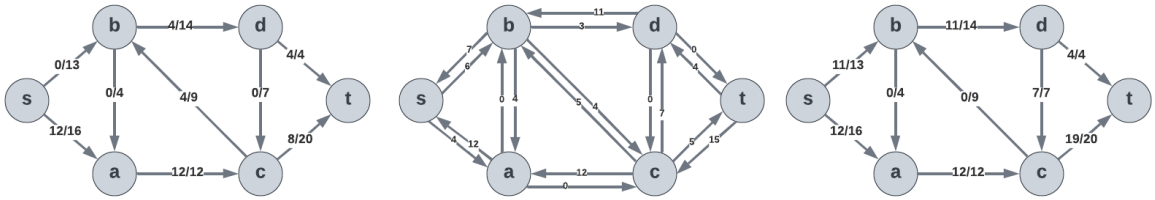
Exhibit a maximum flow with flow values on the edges, state its value, and exhibit a cut (specified as a set of vertices) with the same value.

- Objective here is to show the process of obtaining the max flow, and specify the flow values on the edges, and exhibit a cut
- In order to do so, we will need to start off by drawing the residual network
- We then find the augmenting path from the RN and show flow from the path
    - We augment from s -> t
    - Process: s -> b -> d -> c -> t
    - Results in flow = 7



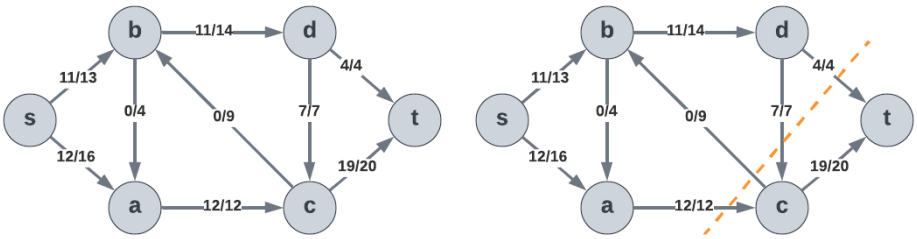
Original

- Similarly, we can augment with a similar manner
    - We augment s -> t
    - Process: s -> b -> c -> t
    - Results in flow = 4



Original

- From this, we see the inability to augment the path
- We can then show the maximum flow diagram below
- Maximum flow = 23



Max Flow = 23 **Exhibliting Cut**

Q4
10 Points

Recall the `Clique problem`: given a graph G and a value k, check whether G has a set S of k vertices that's a clique. A clique is a subset of vertices S such that for all $u, v \in S$, uv is an edge of G.

The goal of this problem is to establish the NP-hardness of Clique by reducing **VertexCover**, which is itself an NP-hard problem, to Clique. Recall that a vertex cover is a set of vertices S such that every edge uv has at least one endpoint (u or v) in S, and the VertexCover problem is given a graph H and a value l, check whether H has a vertex cover of size at most l.

Note that all these problems are already phrased as decision problems, and you only need to show the NP-Hardness of Clique. In other words, we will only solve the reduction part in this problem, and you DO NOT need to show that Clique is in NP.

This question took quite a bit of time to figure out, so I will present my best answer in a way such that I explain my reasoning as I present my results for 4.1, 4.21, and 4.23 since they discuss the same idea.

Q4.1
5 Points

Let S be a subset of vertices in G, and let C be the complement graph of G (where uv is an edge in C if and only if uv is not an edge in G).

Prove that for any subset of vertices S, S is a vertex cover in G if and only if $V \backslash S$ is a clique in C.

Note: this is an if and only if proof, i.e. you need to show both directions for full credit.

- We will let S be a subset of vertices in G, and we will let C be the complement graph of G
- We will assume uv is an edge in C iff uv is not an edge in G
- Our objective is to prove that for any subset of vertices S, S is a vertex cover in G iff V\S is a clique in C
- We will define a clique a as a collection of vertices in an undirected graph, so that every two vertices in a given clique are seen as nearby which implies that the subgraph is complete
- We can see that by definition that S is a vertex cover in the graph G if:
- For every edge in graph G, denoted by uv, has at least either u or v in S
- Since C is the complement graph of the graph G, we see that since edge uv is in C, uv is not an edge in G
- Therefore, every edge denoted by uv in C, there is at least u or v that is not in S

- Since S is a vertex cover in graph G, every edge denoted by uv is in-fact in C, and so at least one of the nodes u or v will be in V\S
- We look this over within the confines of a proof, in which a claim is made such that Graph G has a VertexCover of a given size k, iff (if and only if) Graph H (based on the original description) has a clique of size n-k.
    - Proof.
        - Given graph G and its size of k, we can say that if G = (V, E) has a VertexCover of a size of k, and a set S as its associated VertexCover set, then we can say that all the edges in that graph G will be incident on the given set S.
        - Given that relation, we can state that there will be no edges in the vertices in that set V\S.
        - That said, the complement graph, denoted by H, and all the vertices in V\S will be connected.
        - Since they are connected, H will have a clique of size n-k (as specified above)
        - However, if S is not a Vertex Cover for graph G, then there must be an edge uv in which neigher v or u are in S or H. Thus, the set V\S can in no way be clique because of u and v in V\S not being connected to each other.
        - Therefore, G can have a VertexCover with a given size k IFF (if and only if) its complement H has a clique of n-k.
- **Therefore, we can state that V\S is a clique in C, as described in <u>both</u> directions.**

Q4.2
5 Points

Part 4.1 implies the following result (which you may use without proof): G has a vertex cover of size at most k if and only if the complement of G has a clique of size at least $n - k$.

Use this fact to give a reduction from `VertexCover` to `Clique`. Your solution should have the following two steps:

i) First, show the reduction: specify how the inputs to VertexCover, G and k, can be transformed to a valid input pair, H and l, for Clique. Make sure to explain why this takes polynomial time.

ii) Second, show that the answer to $Clique(H, l)$ can be converted to the answer of $VertexCover(G, k)$. One possibility is to explain how a YES answer to $Clique(H, l)$ must also mean YES to $VertexCover(G, k)$, AND a NO answer to $VertexCover(G, k)$ must also mean NO to $Clique(H, l)$.

- Our objective here is to show the reduction
- We are asked to specify how the inputs to VertexCover, G and k, can be transformed to a value input pair denoted by H and l for Clique.
- Our objective here is to also explain why this takes polynomial time
- Using this, we are then asked to show how Clique(H, l) can be converted to the answer of VertexCover(G, k)
- It is suggested that a possible path is to show how a YES to Clique(H,l) is also a YES to VertexCover(G, k)
- In addition, a NO to VertexCover(G, k) is also a NO to Clique(H, l).
- We can begin with the reduction. Let us assume we are given a Graph G and some input H and l
- In order to convert this to input values for Clique the first objective is to create a graph, G', and some integer value in which the G' is the complement graph of the original graph G.
- We must note that the edges that exist in G and its complement G' are reversed, as described in part 1 of this problem.
- The reduction described above can be done in Polynomial time, relative to the size of the given input value. This is of course dependent on the VertexCover(G, k).
- This is because the VertexCover(G, k) is NP-Hard. That said, Clique here will be NP-Hard as well.

References:

[1] CS5800 Course Modules and Notes

[2] Introduction to Algorithms, Cormen, Third Edition. (CLRS)

[3] LucidChart Drawing Tool (https://lucid.app/lucidchart)

[4] CS5800 Princeton Reference: https://www.cs.princeton.edu/~wayne/cs423/lectures/reductions-poly-4up.pdf