Algorithms: Assignment #11

Tanvi Magdum
magdum.t@northeastern.edu

April 3, 2023

Answers

1 Problem 1 Solution :

The problem describes a flow network, which is a directed graph that models the flow of a commodity, such as water, oil, or data, through a network of interconnected nodes and edges. The flow is represented by non-negative values assigned to the edges, which indicate the maximum amount of the commodity that can be transported along that edge. An edge e is a bottleneck edge if and only if it appears in every minimum capacity cut that separates the source s from the sink t. The capacity of a cut is defined as the sum of the capacities of all edges crossing the cut from the source side to the sink side. A minimum-capacity cut is a cut with the minimum possible capacity among all possible cuts separating s from t.

The problem requires an efficient algorithm to determine whether a given edge e is a bottleneck edge or not. The algorithm needs to compute the maximum flow in the network and then check if the given edge e is part of any minimum cut in the residual network with respect to the maximum flow. To determine if a given edge e is a bottleneck in a flow network G = (V,E) with source s and sink t, we can use the following algorithm-

1. Compute the maximum flow in the network G using any efficient maximum flow algorithm, such as the Ford-Fulkerson algorithm. Let f be the maximum flow value obtained.
2. Construct the residual network GR of G with respect to the maximum flow f. The residual network can be constructed by subtracting the flow values of each edge from its capacity in the original network G. This will allow finding augmenting paths in the residual network that can increase the flow.
3. Find any minimum cut in GR that separates s from t.
4. Check if the given edge e is part of the minimum cut found in step 3.
   (a) If e is part of the minimum cut, then it appears in every minimum cut and is a bottleneck edge.
   (b) Otherwise, it is not a bottleneck edge.
5. Output whether e is a bottleneck edge or not.

```
Psuedocode :

IsBottleneck(G, E):
f = MaxFlow(G)
GR = ResidualNetwork(G, f)
cut = MinCut(GR)
if E.from in cut and E.to not in cut:
    return true    // E is a bottleneck edge
else:
    return false  // E is not a bottleneck edge
```

Maxflow(G) : a function that computes the maximum flow in flow network G using an efficient maximum flow algorithm

ResidualNetwork(G, f) : a function that constructs the residual network of G with respect to a flow f

MinCut(GR) : a function that finds any minimum cut in Gf that separates s from t

The complexity of this algorithm depends on the maximum flow algorithm used. If we use Ford-Fulkerson algorithm to calculate the maximum flow, then time complexity will depend on the number of iterations required by the algorithm to converge to the maximum flow. In the worst case, the algorithm may need to iterate up to f times, where f is the maximum flow value. The time complexity of the Ford-Fulkerson algorithm is generally given as O(Ef), where E is the number of edges in the network and f is the maximum flow value. However, in the worst case, the algorithm can have an exponential running time.

2 Problem 2 Solution :

2.1 Problem 2.1 Solution :

To frame this problem as a network flow problem, we can consider a bipartite graph with two sets of nodes - the first set represents the characters of the given word, and the second set represents the faces of the dice. Each node in the first set will be connected to all the nodes in the second set, with an edge capacity of 1. This represents the fact that each character can be formed by using one face of one dice.

Next, we add a source node that is connected to all the nodes in the first set, with an edge capacity equal to the frequency of that character in the word. This ensures that the number of times a character is used in the word is not more than the number of available faces of that character in the dice. Similarly, we add a sink node that is connected to all the nodes in the second set, with an edge capacity equal to the number of faces of that dice. This ensures that each face of a dice can be used only once. Therefore, the maximum flow in this graph represents the maximum number of characters that can be formed from the given dice. If the maximum flow is equal to the length of the word, then it can be formed using the given dice, otherwise not.

Let's consider the first example given:

Input:

word = "hello"

dice = [[a, l, c, d, e], [a, b, c, d, e], [a, b, c, h, e], [a, b, c, d, o], [a, b, c, l, e]]

We will construct the bipartite graph with the following nodes and edges:

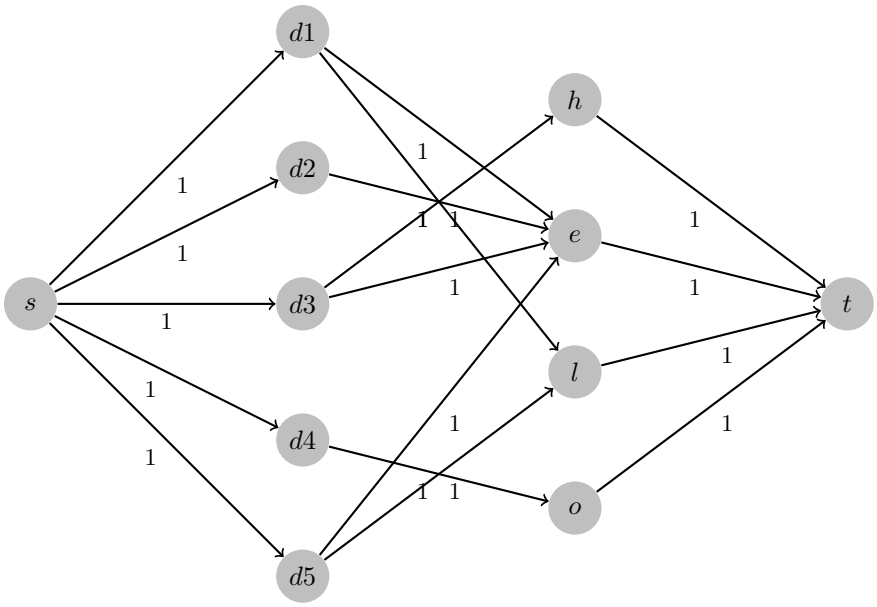
Set of characters: {h, e, l, o}

Set of dice faces: {d1, d2, d3, d4, d5} where,

d1 = [a, l, c, d, e], d2 = [a, b, c, d, e], d3 = [a, b, c, h, e], d4 = [a, b, c, d, o], d5 = [a, b, c, l, e].

Add an edge between a dice node and a character node if and only if the character appears on the face of the dice. Also, add a source node s and connect it to all dice nodes in the first set with an edge of capacity 1 and a sink node t and connect all character nodes in the second set to the sink node with an edge of capacity 1.

.



The capacity of the edges connecting set of dices and set of characters is 1 because each dice can only be used once to contribute to the formation of the word. The edges from each dice node to the character nodes it can contribute to should have a capacity of 1, meaning that the dice can only be used once to provide that particular character. Similarly, the edges from each character node to the sink should also have a capacity of 1, meaning that each character can only be formed once in the final word. This ensures that the total flow in the network corresponds to a valid formation of the given word using the given dice.

In the case of Example 2, there is no flow because there is no way to form the word "hello" using the given set of dice. The graph would still be constructed in the same way as for Example 1, but when running the Ford-Fulkerson algorithm, it would not be possible to find a path from the source to the sink that satisfies the capacity constraints, indicating that the word cannot be formed.

In the case of example 3, we can construct a bipartite graph as we did for Example 1, but we can notice there are total 4 a characters in the given word, and we need all 4 dices to have atleast one character a. From the 4 dices we have, the second dice does not contain this character, so there is no way to form the word "aaaa" using the given set of dice. Therefore, there will be no flow in the network and the output will be "false".

2.2 Problem 2.2 Solution :

Algorithm-

1. Define a bipartite graph with two sets of nodes: the first set represents the dice, and the second set represents the characters in the given word.
2. Connect each dice node to the character nodes that correspond to the characters on its faces. This means that each dice node will have edges connecting it to multiple character nodes, with a capacity of 1 on each edge.
3. Add a source node s and connect it to each dice node with an edge of capacity 1.
4. Add a sink node t and connect each character node to it with an edge of capacity 1.
5. Run a maximum flow algorithm, for ex. Ford-Fulkerson algorithm on the constructed graph.
6. If the maximum flow obtained is equal to the length of the word, then it means that the word can be constructed using the given dice. Otherwise, it cannot be constructed.

The time complexity of this algorithm depends on the choice of maximum flow algorithm used. However, constructing the bipartite graph takes O(nm) time, where n is the number of dice and m is the

number of characters in the word. This is because each dice node needs to be connected to multiple character nodes, and each connection takes O(1) time. Assuming the maximum flow algorithm used has a time complexity of O(Ef) due to use of Ford-Fulkerson algorithm, where f is the maximum flow and E is the number of edges in the graph, the overall time complexity of the algorithm will be O(nm + Ef).

This algorithm works because we are trying to find a set of paths through the bipartite graph that connect the source nodes (dice nodes) to the sink nodes (character nodes) such that each dice is used exactly once and each character is matched with a dice face that contains that character. By using a maximum flow algorithm, we can efficiently find the maximum number of such paths, which corresponds to the maximum number of dice that can be used to construct the word. If the maximum flow is equal to the length of the word, then it means that every character in the word can be matched with a dice face, and therefore the word can be constructed. Otherwise, it means that there are characters in the word that cannot be matched with any dice face, and therefore the word cannot be constructed.