Jason Lai
204995126
Dis 1B
CS 180
June 6, 2019

Homework 8

1 Problem 7.22

(a) Give an example of a matrix M that is not rearrangeable, but for which at least one entry in each row and each column is equal to 1.
(b) Give a polynomial-time algorithm that determines whether a matrix M with 0-1 entries is rearrangeable.

a) An example of a matrix M that is not rearrangeable, but for which at least one entry in each row and each column is equal to 1 is as follows: for any $n \times n$ matrix, let the first row be all 1's except for the last element $a_{1,n}$, which is 0. Let all subsequent rows 2 through n be all 0's except for the last element in each row $a_{2,n}$, $a_{3,n}$, ..., $a_{n,n}$, which is 1. This configuration of the matrix M is not rearrangeable and there is at least one entry in each row and each column that is equal to 1.

b) A polynomial-time algorithm that determines whether a matrix M with 0-1 entries is rearrangeable is as follows:

We construct a bipartite graph G with a set R of nodes on one side and a set C of nodes on the other side. Let each node i in set R be a row in the matrix M and let each node j in set C be a column in the matrix M. Now, only connect a node i in R with a node j in C if an element $a_{i,j}$ in the matrix M is equal to 1. In order for the matrix M to be rearrangeable, there must exist a perfect matching in the bipartite graph G.

To find if there exists a perfect matching in the bipartite graph G, we construct a maximum flow problem; to construct this flow network, we place a source node s on the side of the set of nodes R and a sink node t on the side of the set of nodes C. We connect all nodes in R to the source node s and connect all nodes in C to the sink node t; all of these edges are of weight capacity 1 and we call this new graph G'. To find a perfect matching, we use the *Ford-Fulkerson Algorithm* to find all paths in the G'. Along each path we verify that all nodes are indeed matched by marking the nodes along each augmented path, and if by the end of the algorithm, the maximum flow is found to be $|C|$, there should be a perfect matching for the original graph G. If there is a perfect matching, then we say that the matrix M is rearrangeable; otherwise, we say that it is not rearrangeable.

The runtime of this algorithm is polynomial as constructing the complete maximum flow problem G takes $O(n^2 + 4n + 2)$ time to add all nodes and edges. Then, executing the *Ford-Fulkerson Algorithm* takes $O(max\_flow*E)$ and, in the context of this problem, the maximum flow in the worst case could be n, so we have n times the number of edges E (worst case is n^2+2n), so the running time of *Ford-Fulkerson* is $O(n*(n^2+2n))$. So, the final running time of this algorithm is $\approx O(n^3)$, which is polynomial-time.

Jason Lai
204995126
Dis 1B

Homework 8

2 Problem 7.23

Give an algorithm that takes a flow network G and classifies each of its nodes as being upstream, downstream, or central. The running time of your algorithm should be within a constant factor of the time required to compute a single maximum flow.

An algorithm that takes a flow network G and classifies each of its nodes as being upstream, downstream, or central and runs in a constant factor of the time required to compute a single maximum flow is as follows:

First, we define a separate, auxiliary algorithm A that computes a minimum cut (S, T), where S is the set of all nodes belonging to a specific equivalence class: upstream, downstream, or central. For this algorithm, we run the *Ford-Fulkerson Algorithm* on the flow network G and get the residual graph G'. From the source node s, we run a depth-first search and call the vertex set that it yields S. We say that $(S, V\text{-}S)$ is the desired minimum cut, which is true as the depth-first search reveals all nodes that flow in the upstream direction from the source node s. This also provides a valid minimum cut for $(S, V\text{-}S)$, as we can see that due to running depth-first search on a residual graph, we can affirm that down to all leaf nodes from the resulting directed depth-first search, this is a cut that divides the residual graph between nodes in S and nodes in $(V\text{-}S)$. The running time of this is polynomial as *Ford-Fulkerson* and depth-first search are both polynomial-time algorithms.

Using this algorithm A, we run A first on the residual graph G', where we take the source node to be s; this gives all upstream nodes in the set S_{upstream}. We then run A again on the residual graph G', but instead take the source node to be the sink node t. This gives us all of the downstream nodes in a set $S_{\text{downstream}}$. Once all upstream and downstream nodes have been found, it's obvious to see that we can obtain all central nodes in a set S_{central} by finding $(V\text{-}S_{\text{upstream}}\text{-}S_{\text{downstream}})$. We then return S_{upstream}, $S_{\text{downstream}}$, and S_{central} as the desired classified nodes.

We can see that this algorithm runs in a constant factor of the time required to compute a single maximum flow, as for each execution of the auxiliary algorithm A, we run *Ford-Fulkerson* and depth-first search once each, so the complexity of A is $O((max\_flow\text{*E}) + \text{V} + \text{E})$. Since the complete algorithm runs A twice and computes a difference of sets, which can be done in $O(V)$ time, the total running time of the algorithm is $O(2(max\_flow\text{*E}) + 3\text{V} + 2\text{E}) \approx O(2(max\_flow\text{*E}))$, which is indeed, a constant factor 2 of the time required to compute a single maximum flow as desired.

3 Problem 7.34

(a) Suppose you're given a set of n wireless devices, with positions represented by an (x, y) coordinate pair for each. Design an algorithm that determines whether it is possible to choose a back-up set for each device (i.e., k other devices, each within d meters), with the further property that, for some parameter b, no device appears in the back-up set of more than b other devices. The algorithm should output the back-up sets themselves, provided they can be found.
(b) Give an algorithm that determines whether it is possible to choose a back-up set for each device subject to this more detailed condition, still requiring that no device should appear in the back-up set of more than b other devices. Again, the algorithm should output the back-up sets themselves, provided they can be found.

a) An algorithm that determines whether it is possible to choose a back-up set for each device, with the further property that, for some parameter b, no device appears in the back-up set of more than b other devices, and returns the back-up set itself is as follows:

First, we construct a bipartite graph G with a set of nodes on one side that is the complete set S_1 of n wireless devices, and a second set S_2 of the same n devices on the other side. For each node i in S_1, we connect an edge to a node j in S_2 if it the devices that these two nodes represent meet the condition that they are within d meters of each other; we don't connect nodes in S_1 to their counterparts in S_2. We give all of these edges weight capacity 1. We then construct a flow network by adding a source node s and a sink node t, and connect all of the nodes in S_2 to the sink node t with edges of weight capacity 1. Then, for each node i in S_1, we compute the maximum flow where i is the source node using the *Ford-Fulkerson Algorithm*; if the maximum flow for a node exceeds the parameter b, then we say that there is no such back-up set that satisfies the constraint for that device. We repeat this process for every node in S_1. Since the *Ford-Fulkerson Algorithm* can find all paths from the source to the sink, it is a simple matter to get the nodes for each path between the node i in S_1 and the sink t, and return this set as the back-up set for the device i.

b) To implement this further constraint, we take the algorithm from part **(a)** and output the back-up sets for each device. We then sort the back-up sets in order of distance d from the device that that back-up set is for from least to greatest. We then use the power decay function f and compare the $f(d_j)$ of each sorted back-up list to the power p_j for each device j. If the comparisons do not yield a valid order matching the paradigm $f(d_j) \geq p_j$ for each $d_1 \leq d_2 \leq ... \leq d_k$ and $p_1 \geq p_2 \geq ... \geq p_k$, then we say no such back-up set exists for a given device i. No time complexity constraint is imposed on either part **(a)** nor **(b)**, but both algorithms should still run in polynomial-time.