CISC 471 HW 2

Hershil Devnani (20001045)

February 5, 2021

1 Programming

1. Why you think your algorithm is correct (whether you program worked on the sample data or not).

Find Euclerian Cycle in a Graph: I believe the algorithm is correct since the algorithm uses the methods describes in the text and videos to finding a Euclerian cycle from a graph. The algorithm first create a cycle by choosing an arbitrary starting vertex in the graph. That cycle is then taken and iteratively reconstructed until a Euclerian cycle is found. It does so by looking for any unexplored branching present in the current cycle, and restarting the beginning of the cycle from that node, but traversing the original path of the cycle first. This way, we modify the original cycle to end at the node from which we have more potential steps we can take and proceed further in the graph, making sure that we only traverse each edge only once. The algorithm implements the pseudo code from both the video and text book, as well as performing correct cycle formations using the random large sample sets provided by Rosalind.

Generate Contigs from a Collection of Reads: I believe that this algorithm is correct since what the algorithm is essential doing is performing a search on the graph until a vertex is reached that either has no further neighbour or has a branch, i.e. multiple neighbours. This way, the algorithm is able to generate contigs based on a given graph, with the core idea being to select a node and build a maximal non-branching path. Further, the algorithm was implemented by referencing pseudo code from the textbook and videos as well as on Rosalind. The algorithm was also test on many data sets, as well as smaller sample data sets that could be verified manually to ensure correct output of generated contigs.

2. Provide an estimate of the time and space complexity of your algorithm.

Find Euclerian Cycle in a Graph:

Time Complexity: $O(E)$
Space Complexity: $O(V + E)$

The first part of finding a Euclerian cycle from a graph is generating the first base cycle. The first cycle generates may in fact be the Euclerian cycle, which gives us a worst case run-time of the initial cycle gen ration to be proportionate to the number of edges in the graph, since that number of computations we make to move though a Euclerian cycle is the number of edges in the cycle. The second part of finding an Euclerian cycle, is if the initial cycle generated has any vertices that have edges that were left unexplored, we select that node as the starting point for our next iteration, traverse the cycle using the same path as before, but continue exploring the new path once we arrive back at our starting node. Since we already have part of the original cycle, and are essentially rotating our cycle in order to continue exploring, our maximum worst case computation is proportional to the number of edges in the graph. Since these are separate loops, each loop gives us a time complexity of $O(E)$, leading to $O(E + E)$, which can just be reduced to $O(E)$ since complexity is relative.

In terms of space complexity, we have to store the information about both vertices and edges that we have traversed during the construction of the Euclerian cycle. Since when we backtrack and iteratively reconstruct our cycle, we still retain the information of the cycle, so we don't always have to reallocate space for every edge and vertex, only for the new vertices and edges we add to the cycle. Each repetitive restart of the cycle, by selecting a node in the cycle that has unexplored edges we only rotates the cycle, so we only add more edges and vertices, we never have to start from scratch with an entirely new sequence. The number of vertices is independent of the number of edges, so we need to define our memory requirement by the number of vertices as well as the number of edges. Since each vertex has a variable number of incoming and outgoing edges we have to factor both the vertices and edges. As a result, the space complexity of this algorithm is $O(V + E$.

Generate Contigs from a Collection of Reads

Time Complexity: $O(V + E)$
Space Complexity: $O(V^2 + E^2)$

Generating contigs from a collection of reads requires us to exhaustively search the given graph for non-branching paths. This requires us to traverse each vertex in the graph in addition to traversing the graph as many times as there are edges in the graph. For every node in the graph, we have to perform a search operation, which is $O(V)$. Then, we have to contually take a look at all the outgoing edges of the vertex until we reach a point where the number of outgoing edges being to branch, which is a totla numer of operation proportional to the number of edges in the graph, at worst case being $O(E)$. to As a result, the time complexity of this algorithm is $O(V + E$.

In terms of space complexity, similarly, we have to store and keep vertices and edges as well as the sub-graphs for found contigs, as a result the complexity at worst would be $O(V^2 + E^2)$.

2 Theory

2.1 Lesson 3.3

Construct a 4-universal string.

0000110010111101

2.2 Peaceful Placement of Queens

In the game of chess a queen can move any number squares along a horizontal, vertical or diagonal path. A peaceful placement of n queens on an n $\times$ n chessboard places the queens so that no two queens are in each other's path. Consider a n $\times$ n chess board.

1. What is the smallest n such that n be peacefully placed?

The smallest n such that n can be placed peacefully is n = 1. However, realistically, for any practical purposes of the problem, the smallest n such n can be placed peacefully is n = 4. The reason for this being with n = 2, on a 2x2 grid, placing any single queens on any position, the queen will be able to move to every remaining spot on the board, and so there is no peaceful configuration available if we try to add the second queens. In the case of n = 3, placing two queens on the board in any configuration will not leave any remaining positions on the board that would not cause a conflict, and thus the third queens cannot be placed on the board.

2. Write a recursive algorithm that either places the n Queen's or determines that no such placement is possible.

Please see **n-queens-pseudocode.py** for pseudo code.

3. Modify the algorithm so that it counts all peaceful placements.

Please see **n-queens-pseudocode.py** for pseudo code.