CMPT 435 - Spring 2019 - Dr. Labouseur

Assignment Four – Dynamic Programming & Greedy Algorithms

Christopher Ravosa
christopher.ravosa1@marist.edu

May 16, 2019

1 Running Time of Bellman-Ford SSSP

The Bellman-Ford SSSP algorithm is an SSSP algorithm which allows the shortest path to be discovered in the event that the graph being evaluated contains weighted edges. This makes Bellman-Ford unique since other popular path-finding algorithms, like Dijkstra's, are unable to deal with weighted edges. Bellman-Ford can also determine if there is a "negative-weight cycle", meaning there is a path in which every pass makes the distance to the destination smaller. This is an endless cycle that would lead to negative infinite. In this case, no solution to the path exists. In short, the Bellman-Ford algorithm determines shortest paths from the source to the other vertices, as well as the path to said vertices.

Bellman-Ford begins by initializing the distance of all vertices from the source to infinite (or the maximum integer value). It then relaxes each edge of the graph |V |- 1 times. Relaxation is a O(1) runtime method which determines if there exists a shorter path to each vertex than the one that each vertex is currently marked with. Lastly, it checks for a negative weight cycle with a O(E) for-loop.

The Bellman-Ford algorithm's runtime is O(|V| * |E|); where 'V' equals the number of vertices in the graph, and 'E' equals the number of edges. This is because the initialization method takes roughly O(V) time, the relaxation method runs | V | - 1 times with a runtime of approximately O(E), and the negative-weight cycle check takes O(E) time.

2 Running Time of Fractional Knapsack

The 0 - 1 knapsack problem has restrictive rules which allow the taking of all of an item or none of it. As opposed to the 0-1 knapsack problem, fractional knapsack allows the taking of chunks of items instead of wholes. Fractional knapsack runs in $O(n \cdot \log_2 n)$ time, where 'n' is the number of items in the problem. Fractional knapsack's worst-case runtime is $O(n \cdot \log_2 n)$ because the algorithm must evaluate the costs of each item. Traversing all the items takes $O(n)$ time. At most, the algorithm can take from each item its entire height. The cost of evaluating the total height of each item is roughly equal to $\log_2 n$. Therefore, if the knapsack being used in the algorithm can fit all of every item, it will run through an entire item ($\log_2 n$), as many times as there are items (n). Pairing these two together yields the total run time, $O(n \cdot \log_2 n)$.