Lab Five

Joseph McDonough
Joseph.McDonough1@marist.edu

December 4, 2020

1 Results

1.1 Fractional Knapsack

A file, titled spice, is a text file that has the name of 4 different spices and 5 different knapsack sizes. Each spice has a name, a total price, and a quantity while the knapsacks just have a capacity. The goal is get the most value as possible into each knapsack.

Upon filling the knapsacks starting with the most valuable items, these are the results:

Table 1.1: The 4 Graphs

Knapsack Capacity	**Actual Run-Time**	**Expected Run-Time**
1	2	$\mathrm{O}(n \log_2 n) \rightarrow O(1 \log_2 1) \rightarrow O(0)$
6	13	$\mathrm{O}(n \log_2 n) \rightarrow O(6 \log_2 6) \rightarrow O(16)$
10	21	$\mathrm{O}(n \log_2 n) \rightarrow O(10 \log_2 10) \rightarrow O(33)$
20	43	$\mathrm{O}(n \log_2 n) \rightarrow O(20 \log_2 20) \rightarrow O(86)$
21	44	$\mathrm{O}(n \log_2 n) \rightarrow O(21 \log_2 21) \rightarrow O(93)$

Following an analysis of Big O, the comparison count for filling all the knapsacks are within the order of magnitude and can be deemed appropriate results for knapsacks with such capacities and the predefined amount of spices.

1.2 Single Source Shortest Path

A file, titled graphs2, is a text file that has the makings of 5 different graphs. Each graph has a set of vertices and edges. The edges have weights such that the shortest path between two given vertices is the path with the lowest weight

Upon running the shortest path algorithm from the source vertex to all connected edges, these were the results for how long it took to determine such a path:

Table 1.2: The 4 Graphs

Graph Name	Actual Run-Time	Expected Run-Time
CLRS	45	O(V*E) $\rightarrow O(5*10) \rightarrow O(50)$
Directed v1	79	O(V*E) $\rightarrow O(7*12) \rightarrow O(84)$
Directed v2	79	O(V*E) $\rightarrow O(7*12) \rightarrow O(84)$
Directed v3	79	O(V*E) $\rightarrow O(7*12) \rightarrow O(84)$

Following an analysis of Big O, the comparison count for all the searches are within the order of magnitude and can be deemed appropriate results for graphs with said vertices and edges.

2 Asymptotic Analysis

2.1 Fractional Knapsack

Fractional knapsack is a greedy algorithm. Greedy algorithms make whatever choice is best in the moment and hopes that this choice will lead to the best choice overall. To do so, they tend to work recursively. Fractional knapsack is no different. The items that are to be inserted into the knapsack need to be ordered from most to least valuable. This is so the algorithm can ensure it is getting the most valuable items at each iteration that it can. This sorting runs at $\log_2 n$ time.

In order to fill each knapsack, a while loop is used to take the most valuable item remaining and place it in the knapsack. The process of putting the item in the knapsack works at O(1), so in the grand scheme of complexity, it is practically ignored. But the amount of times that an item must be put into the knapsack is dependent on the size of the knapsack, therefore that process has a complexity of O(n). Overall, the process of filling up the knapsack is O(1 * n), or just simply O(n).

Ultimately, fractional knapsack works by taking in a series of items, then sorts them from most valuable to least valuable (O($\log_2 n$)). Upon completion of sorting, the knapsacks get filled (O(n)). Making the entire process run in O($n \log_2 n$) time.

2.2 Single Source Shortest Path

Bellman Ford Single Source Shortest Path algorithm is designed for use on directed graphs with weights. The shortest path is that with the lowest weight. And since edges can have varying weights, including negative values, the shortest path may not be the most direct. This algorithm is designed to identify the shortest path, regardless of how many "stops" it takes.

To begin the quest for the shortest path, each vertex is given an initialization value. The algorithm defines this value to be infinity, as it wants to be the highest possible number such that the first weight it comes across must be lower. This is done for all the vertices in the graph, making this process take O(V) time where V is the amount of vertices.

Once all the vertices are initialized, the algorithm goes through each edge, via a for-loop, comparing the current lowest distance between two points, and the distance at the current vertex. In the end, this process will allow the algorithm to identify the shortest path between the source vertex and any connected vertex on the graph. This for-loop function that evaluates the distance does so on each edge in the graph, resulting in a run-time of O(E) where E is the amount of edges.

The Bellman Ford Single Source Shortest Path works in two parts, the first sets an initial distance of infinity on each vertex (O(V)) and then goes through each edge on the graph (O(E)) comparing the current weighted distance between the source and said vertex, and the current weight, determining if there is a shorter path to be had. The combination of these two processes have a final run-time of O($V * E$).