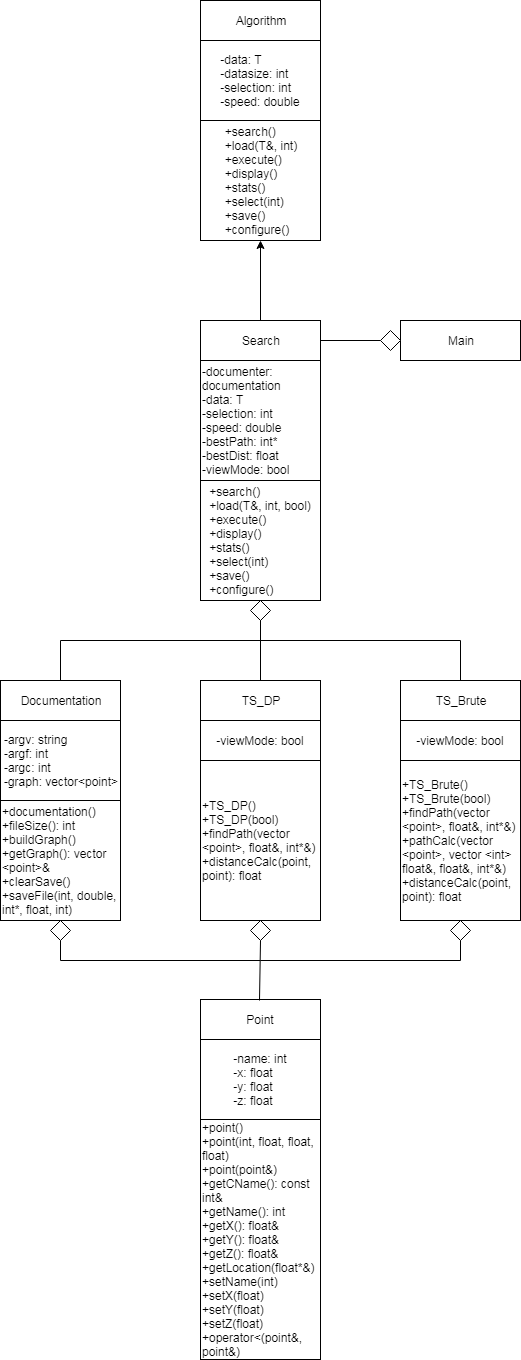
|  |  |  |
| --- | --- | --- |
| Nodes | Brute Force Average Time | Dynamic Programming Average Time |
| 1 | 0 | 0 |
| 2 | 0.001 | 0.001 |
| 3 | 0.001 | 0.001 |
| 4 | 0.001 | 0.001 |
| 5 | 0.001 | 0.001 |
| 6 | 0.002 | 0.002 |
| 7 | 0.004 | 0.003 |
| 8 | 0.021 | 0.006 |
| 9 | 0.079 | 0.012 |
| 10 | 0.6 | 0.1 |
| 11 | 8.01 | 1.13 |
| 12 | 93.8704 | 13.3698 |
| 13 | 1197.89 | 181.434 |

Both Brute Force (BF) and Dynamic Programming (DP) behave in similar times with few nodes, but a major deviation occurs when there are greater than 7 Nodes. BF rapidly increases at a rate of Big O(N!) time. While DP is better, it also begins to rapidly increase, though at a rate of Big O(2^N) time.

For my Dynamic Programming, I chose to implement Tabulation to solve the sub-problems. I did this by creating a 2-Dimensional matrix of type float and filling each slot with the distance calculation of the two nodes corresponding to the elements of the matrix. For example, the distance between Point 1 and Point 3 is stored in the table at location [1][3]. When the program goes through the various paths, instead of calculating the distance between each Point for a given path, it instead searches into the matrix at the location corresponding to those elements. Because the distance has been stored this way, there is no searching involved with the matrix, since the matrix is used like a Hash Table, but a Hashing Function is not needed to store it. Because it has been set up in this way, the element is accessed directly, making it a constant time operation to access the data.

System Architecture:



One of the main design decisions I made when creating my system was to only include one class into my main class. That class is the Search interface. Search inherits from the pure virtual Algorithm Class. For this lab, no other classes were needed to inherit from the Algorithm class, but the possibilities are available. The Documentation class creates a graph using the Point class and it is built at the same time as the Search class constructor. The graph is stored within the documentation class and can be copied over if needed. The graph within the Search class can be overwritten using the load function in main() so the random Point generator can be used rather than the manual inputs in the positions.txt document.

The two Travelling Salesman algorithms used are TS\_Brute (Travelling Salesman Brute Force) and TS\_DP (Travelling Salesman Dynamic Programming). They are both included in the Search class. They only need to be created and used within one function, the execute function. Each algorithm could have been used to store the best path and distance, but it would require the Search class to retrieve the information in order to display the information and to send it to the documentation to save it on the file. Instead, I had the paths and distance information within the parameters within Search and passed them by reference. Since the data was directly manipulated in the algorithms, it is not needed to retrieve the information from it after it has been run. Additionally, the algorithms are needed to be stored as parameters and can just be used in the execute function so that it goes out of scope at the end of the function.

The Point class is the only class that is needed to be included multiple times. Both algorithms and the documentation class need to include it in order to know how to manipulate the functions and parameters in the Point class. While I would have preferred to not have it be included multiple times, I relented since it is the container for the information gained from the random generator or the manual input from the positions.txt file. It could have been stored within a vector or even an array, which would have likely required an include from the standard library but putting the information into a custom container class allowed me more flexibility on how I could manipulate the data after I had issues with other container classes in the Travelling Salesman algorithms based on how I designed them.

On designs for the future, the documentation class can be used for additional classes outside of the Search class if need be. Since the Search class runs algorithms based on a selection, additional Travelling Salesman algorithms, or really any algorithm that requires the parameters in the Search class, can be added with minimal friction to the Search class, and would not affect any other class. Additionally, if another algorithm besides Search is needed, another class can be easily created to inherit from the Algorithm or even the Search class, if the required functionality is similar enough.