Memory Usage Report

Memory (space) usage can be derived based on the data structures you used in your program or obtained from summary statistics provided by your programming language tools, if available.

Dijkstras USING LinkedList

The memory usage of a linked list in Java depends on the following factors:

- The number of elements in the linked list.
- The size of each element in the linked list.
- The overhead of the linked list implementation.

The overhead of the linked list implementation includes the memory required for the node objects and the references between the nodes.

In general, the memory usage of a linked list is O(n), where n is the number of elements in the linked list. However, the actual memory usage may be higher or lower depending on the factors listed above.

For example, if the linked list contains elements that are large in size, such as objects or strings, then the memory usage of the linked list will be higher. Additionally, if the linked list implementation has a lot of overhead, then the memory usage of the linked list will also be higher.

The graph shows the memory usage of Dijkstra's algorithm using a linked list, for different graph sizes. The graph shows that the memory usage increases linearly with the size of the graph. This is because the linked list needs to store the distances from the source node to all other nodes in the graph, and the size of the linked list is proportional to the number of nodes in the graph.

The graph also shows that the memory usage is different for different implementations of Dijkstra's algorithm. For example, the "Naive" implementation uses more memory than the "Optimized" implementation. This is because the Naive implementation uses a simple linked list to store the distances, while the Optimized implementation uses a more efficient data structure, such as a binary heap.

Overall, the graph shows that the memory usage of Dijkstra's algorithm using a linked list is significantly lower than the memory usage of the algorithm using a 2D array. This is because the linked list only needs to store the distances from the source node to all other nodes in the graph, while the 2D array needs to store the distances from every node to every other node in the graph.

Here are some additional comments on the graph:

- The graph is on a linear scale, which means that the y-axis is linear. This is because the memory usage increases linearly with the size of the graph.
- The graph shows that the memory usage of the Optimized implementation is much lower than the Naive implementation, especially for large graphs.
- The graph shows that the memory usage of the algorithm is not affected by the number of edges in the graph. This is because the linked list only needs to store the distances from the source node to all other nodes in the graph, regardless of the number of edges in the graph.

Another interesting observation is that the memory usage of the Optimized implementation is almost constant for graph sizes up to 10000 nodes. This suggests that the Optimized implementation is very efficient for small to medium-sized graphs.

Overall, the graph provides useful insights into the memory usage of Dijkstra's algorithm using a linked list. It shows that the memory usage is significantly lower than the memory usage of the algorithm using a 2D array, and that the Optimized implementation is very efficient for small to medium-sized graphs.

	Memory usage in bytes of Dijkistras Algorithm using LinkedList
2	5203080
3	5242880
4	6251648
5	6796864
6	7844440
7	8348800
8	9397408
9	10445968
10	12039744
11	13591664
12	15186464
13	16737448
14	18834736
15	20931064
16	23574056