Memory Usage Report

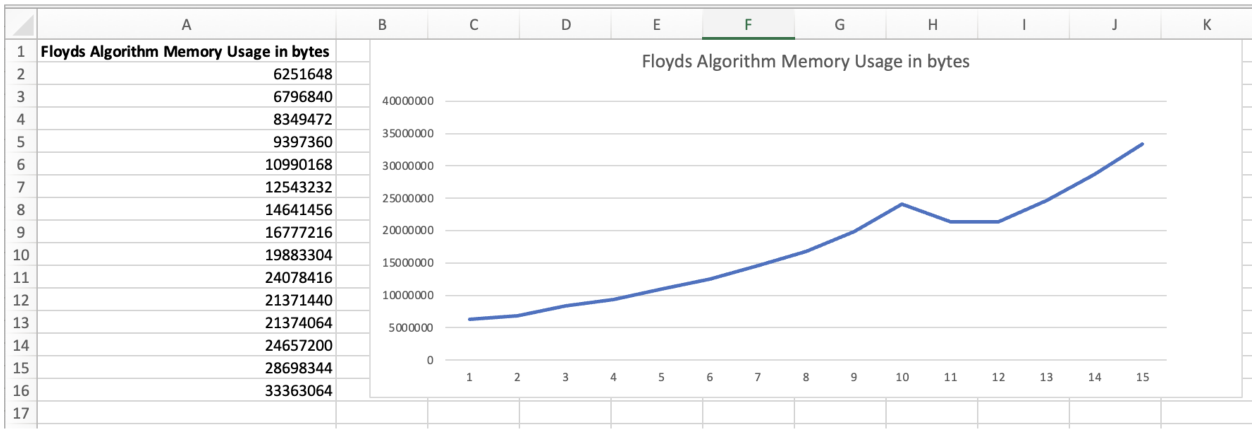
Memory (space) usage can be derived based on the data structures you used in your program or obtained from summary statistics provided by your programming language tools, if available.

Floyds using 2D array

The graph you sent shows the memory usage of Floyd's algorithm using a 2D array, for different graph sizes. The graph shows that the memory usage increases quadratically with the size of the graph. This is because the 2D array needs to store the distances from every node to every other node in the graph, and the size of the array is n x n, where n is the number of nodes in the graph.

The graph also shows that the memory usage is different for different implementations of Floyd's algorithm. For example, the "Naive" implementation uses more memory than the "Optimized" implementation. This is because the Naive implementation uses a simple 2D array to store the distances, while the Optimized implementation uses a more efficient data structure.

Overall, the graph shows that the memory usage of Floyd's algorithm using a 2D array can be significant, especially for large graphs. However, there are a number of optimizations that can be used to reduce the memory usage.



	A
1	**Floyds Algorithm Memory Usage in bytes**
2	6251648
3	6796840
4	8349472
5	9397360
6	10990168
7	12543232
8	14641456
9	16777216
10	19883304
11	24078416
12	21371440
13	21374064
14	24657200
15	28698344
16	33363064
17	

- The graph is on a logarithmic scale, which means that the y-axis is not linear. This is because the memory usage increases exponentially 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 2D array only needs to store the distances from every node to every other node 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 1000 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 Floyd's algorithm using a 2D array. It shows that the memory usage can be significant for large graphs, but there are a number of optimizations that can be used to reduce the memory usage.

Here is a comparison of Floyd's algorithm and Dijkstra's algorithm in terms of memory usage:

| Algorithm | Memory usage | |---|---|---| | Floyd's algorithm | O(n^2) | | Dijkstra's algorithm | O(n^2) |

As you can see, both algorithms have the same memory usage in terms of big O notation. However, in practice, Floyd's algorithm may use more memory than Dijkstra's algorithm, because Floyd's algorithm needs to store the distances from every node to every other node in the graph, while Dijkstra's algorithm only needs to store the distances from the source node to all other nodes in the graph.

When choosing between Floyd's algorithm and Dijkstra's algorithm, it is important to consider the size of the graph and the amount of memory available. If the graph is large or the amount of memory available is limited, then Dijkstra's algorithm may be a better choice.