CS3310 - Design and Analysis of Algorithms

Cal Poly Pomona

Project 2

Fall 2022

Description:

All Pairs of Shortest Paths

Name: Fidelis Prasetyo

Email: (fprasetyo@cpp.edu)

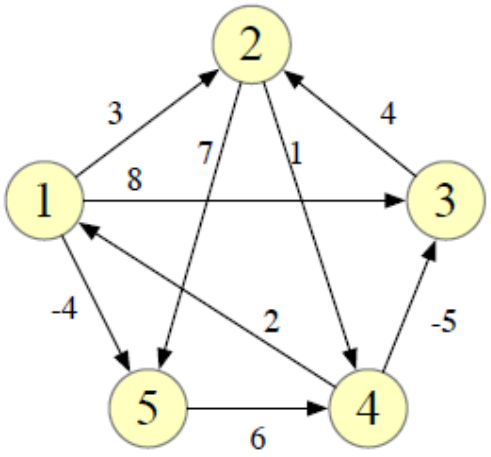
BroncoID: 015765555

Github: https://github.com/fidelisprasetyo

Methodology

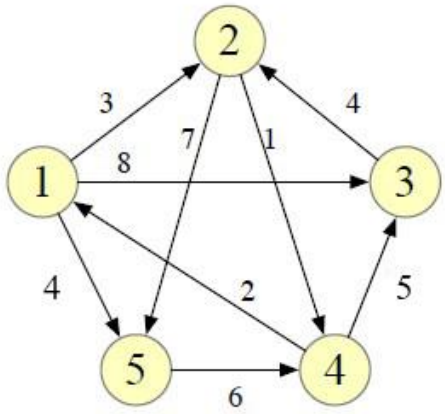
The test is divided into 2 parts. The first part is the sanity check to make sure that both algorithms are implemented correctly. The sanity check will be done with 2 different graphs:

- A directed weighted graph with negative weights (Floyd-Warshall algorithm)



	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

- A directed weighted graph with non-negative weights (Floyd-Warshall algorithm & Dijkstra algorithm)



	1	2	3	4	5
1	0	3	8	∞	4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	5	0	∞
5	∞	∞	∞	6	0

The second test is the runtimes observation to analyze the performance of the algorithms. The test strategies are as follows:

- Both algorithms will be tested with both dense and sparse connected graphs with different numbers of vertices for $n = 10, 20, 30, \ldots, 1000$.
- The graphs are generated randomly with non-negative weights ranging from 1 to 10.
- Dense graphs are directed graphs with $n(n-1)/2$ edges.
- Sparse graphs are undirected graphs with $n - 1$ edges.
- Each iteration of the graph is repeated 10 times to get accurate results.

Test Results

Sanity Check

- A directed weighted graph with negative weights.
    - Floyd-Warshall

```
---Directed Graph with negative weights:
FLOYD-WARSHALL ALGORITHM result (D5):
    0    1   -3    2   -4
    3    0   -4    1   -1
    7    4    0    5    3
    2   -1   -5    0   -2
    8    5    1    6    0
Shortest distances:
> Starting vertex = 0
  Dest:     0    1    2    3    4
  Dist:     0    1   -3    2   -4
> Starting vertex = 1
  Dest:     0    1    2    3    4
  Dist:     3    0   -4    1   -1
> Starting vertex = 2
  Dest:     0    1    2    3    4
  Dist:     7    4    0    5    3
> Starting vertex = 3
  Dest:     0    1    2    3    4
  Dist:     2   -1   -5    0   -2
> Starting vertex = 4
  Dest:     0    1    2    3    4
  Dist:     8    5    1    6    0
```

- A directed weighted graph with non-negative weights.
    - Dijkstra

```
---Directed Graph with non-negative weights:
DIJKSTRA ALGORITHM result:
> Starting vertex = 0
  Dest:     0    1    2    3    4
  Dist:     0    3    8    4    4
> Starting vertex = 1
  Dest:     0    1    2    3    4
  Dist:     3    0    6    1    7
> Starting vertex = 2
  Dest:     0    1    2    3    4
  Dist:     7    4    0    5   11
> Starting vertex = 3
  Dest:     0    1    2    3    4
  Dist:     2    5    5    0    6
> Starting vertex = 4
  Dest:     0    1    2    3    4
  Dist:     8   11   11    6    0
```

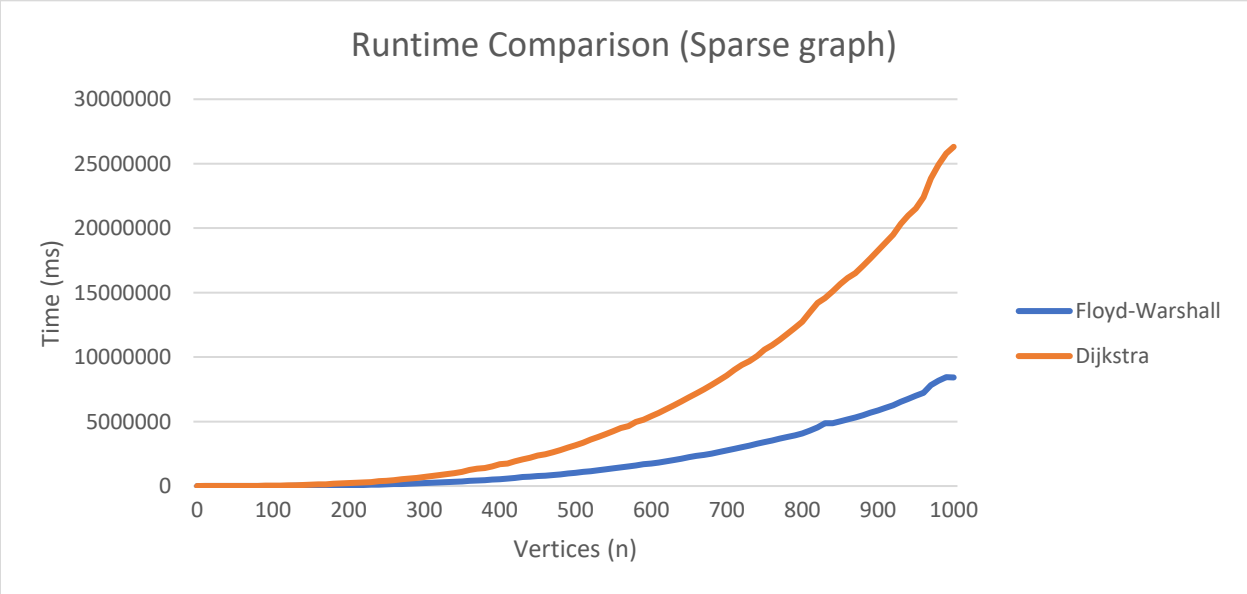
    - Floyd-Warshall

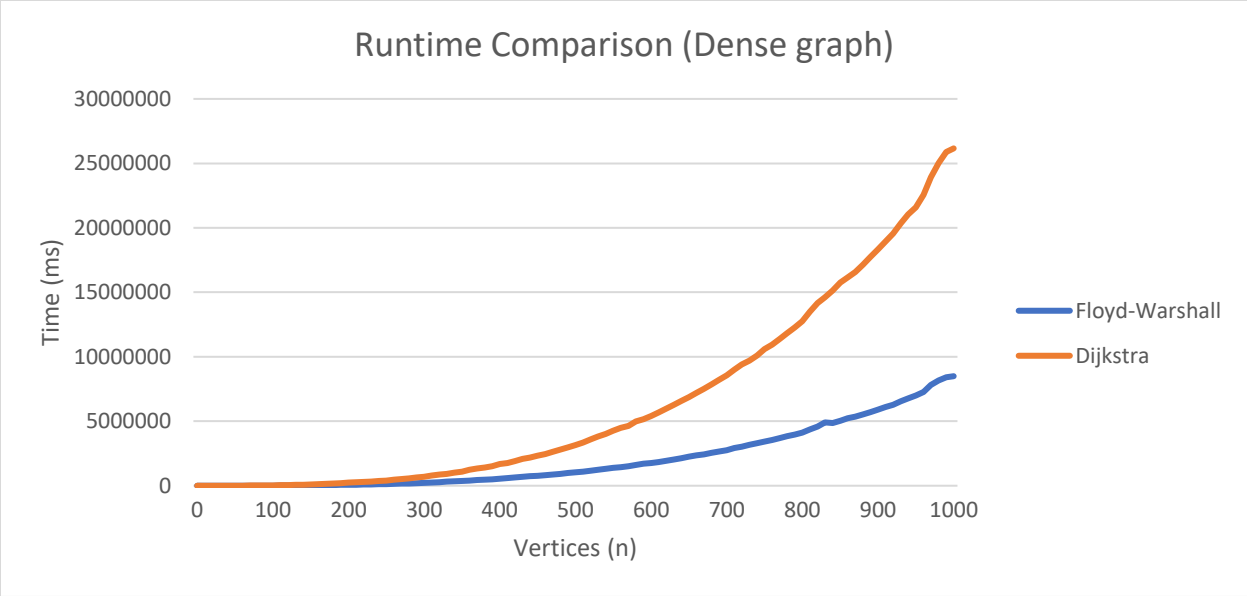
```
FLOYD-WARSHALL ALGORITHM result (D5):
    0    3    8    4    4
    3    0    6    1    7
    7    4    0    5   11
    2    5    5    0    6
    8   11   11    6    0
Shortest distances:
> Starting vertex = 0
  Dest:     0    1    2    3    4
  Dist:     0    3    8    4    4
> Starting vertex = 1
  Dest:     0    1    2    3    4
  Dist:     3    0    6    1    7
> Starting vertex = 2
  Dest:     0    1    2    3    4
  Dist:     7    4    0    5   11
> Starting vertex = 3
  Dest:     0    1    2    3    4
  Dist:     2    5    5    0    6
> Starting vertex = 4
  Dest:     0    1    2    3    4
  Dist:     8   11   11    6    0
```

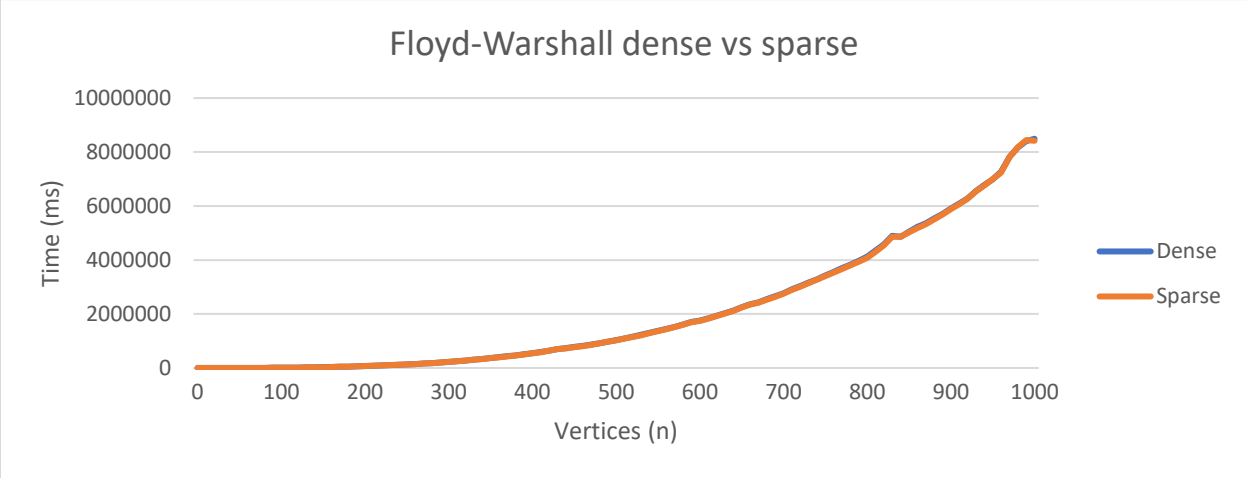
*Dest = Vertex destination

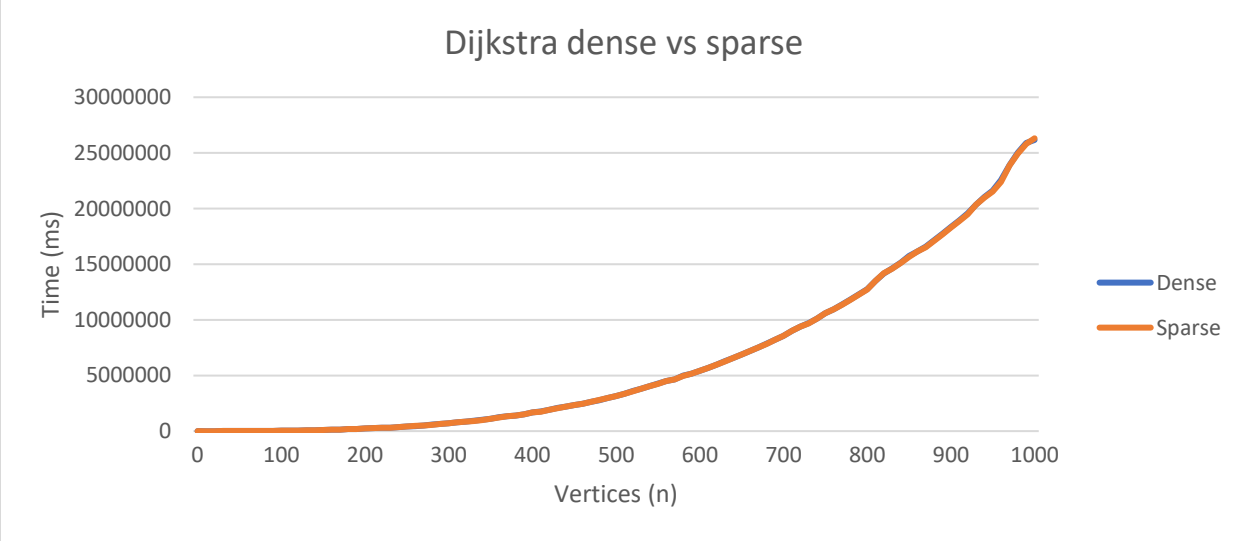
*Dist = Distance

Performance Test




Runtime Comparison (Dense graph)
30000000
25000000
20000000
15000000
10000000
5000000
0
Time (ms)
0
100
200
300
400
500
600
700
800
900
1000
Vertices (n)
Floyd-Warshall
Dijkstra


Floyd-Warshall dense vs sparse
10000000
8000000
6000000
4000000
2000000
0
Time (ms)
0
100
200
300
400
500
600
700
800
900
1000
Vertices (n)
Dense
Sparse


Dijkstra dense vs sparse
30000000
25000000
20000000
15000000
10000000
5000000
0
Time (ms)
0
100
200
300
400
500
600
700
800
900
1000
Vertices (n)
Dense
Sparse

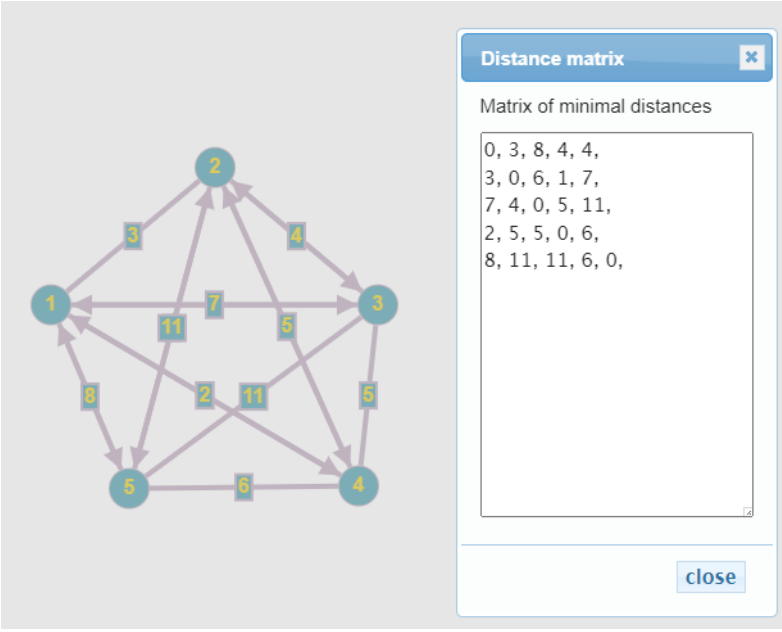
Data Analysis

The first sanity test uses the same graph from the Dynamic Programming slides. The result according to the slide is as below:

$$D^5$$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

By comparing the sanity test and the slide, we can conclude that the implemented Floyd-Warshall algorithm is working correctly since both produce the same result. For the second test, we can see from the test result above that both algorithms also give the same result. To make sure that the results are indeed correct, I compared them with the result from an online website (https://graphonline.ru/en/) that calculates all shortest paths of the same graph, their result is as follows:



The result from the implemented algorithms is the same as above, therefore, we can conclude that both algorithms are working correctly with non-negative weighted graphs.

In order to analyze the performance test correctly, it is necessary to understand the design choice for this program's implementation first. This program uses adjacency matrix to

represent a graph which uses a 2d vector with rows and columns acting as the graph vertices, and the values of the matrix as the distances/ weights between the vertices. Adjacency matrix generally takes more space $O(V^2)$ than adjacency list with $V = number\ of\ vertices$; however, adjacency matrix provides a quick way to access, insert, and remove edges ($O(1)$). Therefore, it is logical to choose the adjacency matrix graph representation over the adjacency list for this project.

Based on the test results above, we can see that the Dijkstra algorithm grows faster in terms of runtime; meanwhile, Floyd-Warshall algorithm growth tends to be more linear. Theoretically, time complexity of Floyd-Warshall algorithm to find all shortest paths is $O(V^3)$, since there are 3 n-sized loop comparisons performed. Comparing the theoretical runtime and empirical runtime, it is safe to say that the test results are close enough to the theoretical runtime. Dijkstra algorithm, on the other hand, has theoretical runtime of $O(V^2)$ for this implementation with adjacency matrix. However, Dijkstra algorithm only calculates single source shortest paths, thus, to obtain all shortest paths, the algorithm must be repeated V more times. Therefore, the total theoretical runtime is $O(V^3)$. The empirical runtime from this test, however, does not yield the same result. Dijkstra algorithm, while having the same time complexity as Floyd-Warshall algorithm, grows much faster in terms of runtime. However, this is expected to happen as asymptotic approach doesn't take account of overheads which tend to happen in practical applications. In this implementation of Dijkstra algorithm, to get the nearest vertex from the current vertex, we must loop through a vector that contains the distances from all other vertices. In other words, this is another $O(V)$ operation. Since this operation has a lower order term, the asymptotic approach ignores this altogether; however, in practical applications, this kind of operation adds up when the n is not large enough to make them negligible. Now if we compare it with the implemented Floyd-Warshall algorithm, we can see that not much happens outside the main loop of the algorithm. Thus, it has minimal overhead compared to Dijkstra.

Lastly, there are almost no difference in terms of runtime between sparse and dense graphs for both algorithms. This is also expected as this program uses the adjacency matrix as the graph representation. It doesn't matter how many edges in the graph since the program will iterate through all the element of the matrix regardless of the sparseness/ denseness.

Strength and Constraints

Strength:

- Both algorithms are working as intended.
- Since adjacency matrix is used as the graph representation, it is easy to build the graph.
- The results are easy to read.
- Performance of both algorithms are good enough.

Constraints:

- The actual performance of Dijkstra algorithm does not reflect the theoretical performance as the number of vertices (n) used in the test is not big enough.
- Adjacency matrix takes a lot of memory $O(V^2)$
- Can only obtain the shortest distances but not the actual paths

Source Code & Supporting Files

The source code, this pdf file, and data.csv, which contains the raw data of the second test, can be obtained from this GitHub repository:

https://github.com/fidelisprasetyo/shortest-paths-algorithms