Informatics Institute of Technology

Department of Computing

Module: 5SENG002C.2 Algorithms

<u>Coursework 01</u>

Network Flow

Module Leader: Sudharshan Welihinda

UoW ID - Student ID	Student Name
W1761767 - 2018130	Luqman Rumaiz

Algorithm

- The Algorithm to find the Maximum Flow implemented was **Dinic's Algorithm**.

  In Dinic's Algorithm a Level Graph has to be constructed through each Iteration till a Path cannot be Augmented, for this the **Breadth First Search Algorithm** is used.

  To Augment a Path the **Depth First Search Algorithm** is used. It visits each Adjacent Edge for each Vertex in this Algorithm, until the Target is reached.

Data Structures

- A Vertex Array was used as an **Adjacency List** and the Adjacent Edges are a List of Edges that are an Attribute of the Vertex Class.
  - The reason behind using an Adjacency List over an Adjacency Matrix is because a Matrix takes much more space as a second Dimension would be involved and in scenarios where their are not many Edges it would have to look through a lot of Values taking more time, whereas an Adjacency List is much more simple and compact as it stores only Adjacent Edges to each Vertex without having to look through many empty Edges in case where there are not many Edges.

.

- A Queue was used for the **Breadth First Search Algorithm**.
  - Queues are more appropriate than Stacks. As usually for **Breadth First Search** we look through each Vertex and search through the Adjacent Edges for the Parent first, this is perfect for Queues as it takes the least recent Item. Husban (2020)

- An Integer Array was used for the **Level Graph**. It is best as we know the number of Vertices we can just initialize the Level Graph and access the Level of Vertices through the Index of a Vertex, Arrays are 40% Faster than ArrayLists and are more appropriate in this case. (assylias , 2013)

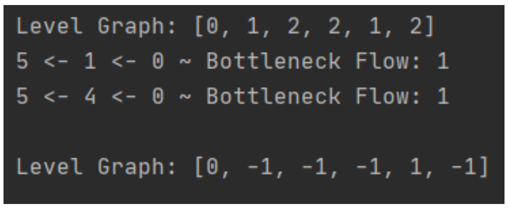
- An Integer Array was used to store the last visited Adjacent Edge for each Vertex, as mentioned earlier for **Level Graph** even for this Array we know the number of Adjacent Edges after initializing the Network Flow.

Depth First Search did not require a Data Structure as it was provided with a Vertex and from that Vertex it looks through its Adjacent Edges and the Adjacent Edges for the Second Vertex involved in that Adjacent Edge, it does this till the Target is reached to Augment a Path. This is done with just For Loops.

Smallest Benchmark with Explanation

Number of Nodes: 6 | Number of Edges: 9 | Source: 0 | Target: 5

1(0,1) | 4(0,4) | 1(1,2) | 2(1,3) | 2(2,4) | 1(3,4) | 4(1,5) | 1(4,5)



```
Level Graph: [0, 1, 2, 2, 1, 2]
5 <- 1 <- 0 ~ Bottleneck Flow: 1
5 <- 4 <- 0 ~ Bottleneck Flow: 1

Level Graph: [0, -1, -1, -1, 1, -1]
```

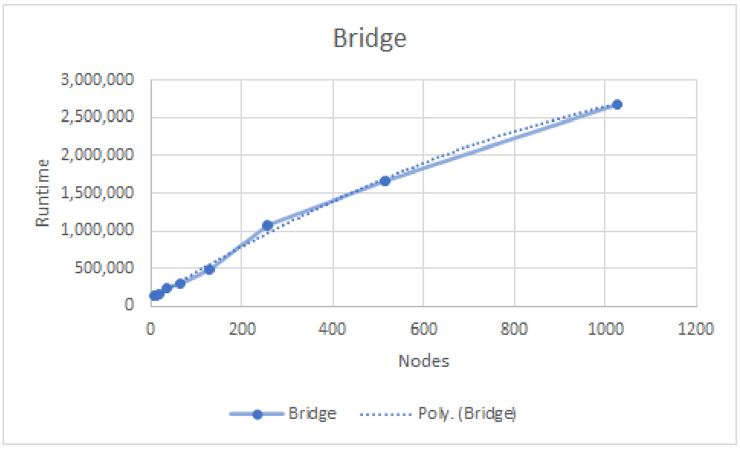
- As you can see first BFS constructs a Level Graph then DFS augments two Paths with a Bottleneck Flow of all together 2, then the Level Graph is reconstructed but execution stops as the blocking part is reached since the Level of the Target is less than 0.

Performance Analysis

These Benchmarks are taken without printing additional information

Name	Nodes	Edges	Trial 01	Trial 02	Trial 03	Average
bridge_1	6	9	139,300	136,299	154,500	**143,366**
bridge_2	10	17	144,300	163,400	140,600	**149,433**
bridge_3	18	33	158,900	161,100	176,200	**165,400**
bridge_4	34	65	238,900	263,000	229,800	**243,900**
bridge_5	66	129	287,900	314,000	285,401	**295,767**
bridge_6	130	257	403,701	482,199	588,500	**491,467**
bridge_7	258	513	1,006,800	1,113,900	1,073,000	**1,064,567**
bridge_8	514	1025	1,525,100	1,663,100	1,799,500	**1,662,567**
bridge_9	1026	2049	3,015,500	2,335,300	2,711,100	**2,687,300**
ladder_1	6	9	161,100	177,200	154,700	**164,333**
ladder_2	12	21	217,500	243,100	217,800	**226,133**
ladder_3	24	45	555,600	484,700	487,900	**509,400**
ladder_4	48	93	1,160,700	1,247,300	1,210,800	**1,206,267**
ladder_5	96	189	2,011,800	1,866,900	1,548,600	**1,809,100**
ladder_6	192	381	2,414,800	2,961,200	3,020,300	**2,798,767**
ladder_7	384	765	5,220,800	5,311,900	5,284,800	**5,272,500**
ladder_8	768	1533	17,352,100	18,387,800	18,008,400	**17,916,100**
ladder_9	1536	3069	33,053,300	32,846,000	36,616,300	**34,171,867**

Table 1 - Benchmarks



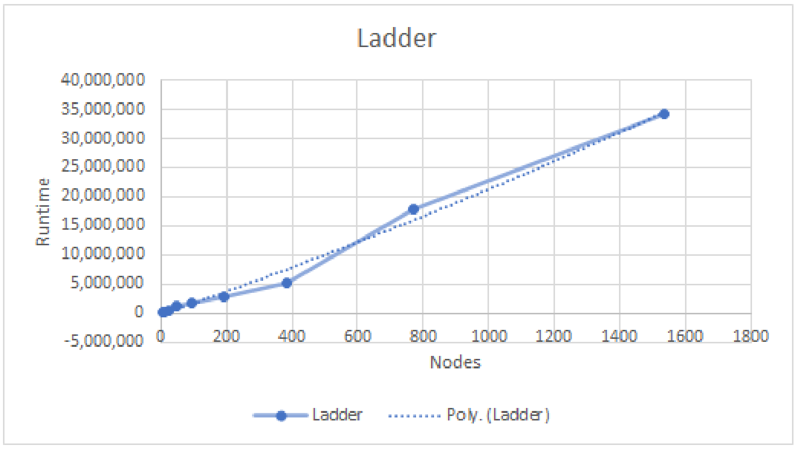


Figure 1 - Bridge Dataset Graph

Figure 2 - Ladder Dataset Graph

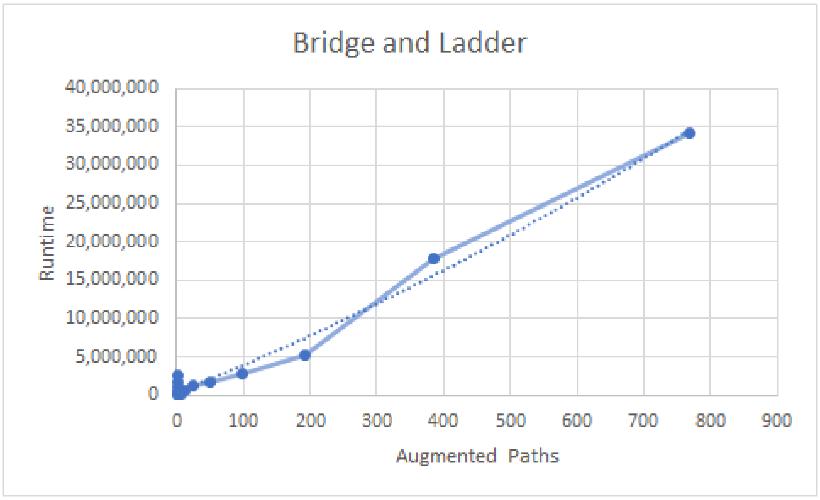


Figure 3 - Bridge and Ladder Dataset Graph

- As seen in the Data provided for the Bridge Dataset the number of Vertices are about 1026 at its peak and the Average Runtime is about 2,687,300 ns. If we compare this to the Data provided for the Ladder Dataset we can see that even for 384 Vertices the Average Runtime is 5,272,500 ns.
- The Reason behind the significant time increase despite the lesser number of Vertices is because if you look at Figure 3 you can see that there is a significant increase of Augmented Paths thereby the Runtime has increased along with that significantly. However this is not the Bridge Dataset rather it is the Ladder Dataset, as the Bridge Dataset looks like an actual Bridge Point A to Point B. Whereas Ladders have several Points in between to reach the top of the Ladder. Therefore the time taken is based on the amount of Augmented Paths.
- Overall, to run BFS to construct the Level Graph it would take O(E) time, and Sending Flow using DFS would take O(VE) time. And it would take O(V) time to loop till a Blocking Path is reached. Based on this the running time of this Algorithm is $O(V^2E)$. (Nishant Singh, 2021)

List of References

- Nishant Singh (02 Feb, 2021) *Dinic's algorithm for Maximum Flow,* Available at: *https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/* (Accessed: 07 April, 2021).

- Husban (07 Jul, 2020) *Difference between Stack and Queue Data Structures,* Available at: *https://www.geeksforgeeks.org/difference-between-stack-and-queue-data-structures/* (Accessed: 08 April, 2021).

- assylias (15 May, 2013) *Array or List in Java. Which is faster?,* Available at: *https://stackoverflow.com/questions/716597/array-or-list-in-java-which-is-faster#:~:text=Conclusion%3A%20set%20operations%20on%20arrays,hundreds%20of%20millions%20of%20times!* (Accessed: 08 April, 2021).