**Name:** Kelechi Ogudu

**USC\_ID:** 5413285132

**Title:** Lab 1



**Output file generation:**

1. To compile code, one need only use the Makefile and the command **make FindShortestPath** in the terminal.
2. Next the file is passed through the command line to compile .o file - **./FindShortestPath <filename.txt>**
3. Within the main file **FindShortestPath.cpp** I parse the input **<filename.txt>**

and concat **output-<filename.txt>** to generate output file.

**Space Constraint Improvement**

The format of the input file is set up as an Adjacency matrix, which stores information about each node and every other node in the graph **S(n) = O(|V|2)**. To reduce this constraint, I convert the file into an adjacency list, which only stores information about the edges **S(n) = O(|V| + |E|**). This reduction is done in hindsight because G(V, E) in most instances is sparse, which means the number of edges is much less than the number of edges.

**Faster Negative Cycle Detection Algorithm**

**Introduction**

For my lab I have chosen to implement an optimized version of Bellman Ford developed by Saranya C.R and Shobhalatha G. in their paper, “*A Faster Negative Cycle Detection Algorithm (FNCDA)”.* The FNCDA uses a modified *timeout* and *walk to root* strategies to terminate once a solution has been found or a negative cycle has been detected in **O(|E|2 )**. *Timeout* is a technique employed by most labelling algorithms as a termination strategy after a certain amount of labelling operations have been performed in the absence of a negative cycle. In the case of FNCDA – the number of times a vertex becomes the scanning vertex is considered (ns). If ns > 2 then a negative cycle is present and the pointers to the previous iteration are returned. On the other hand, the *Walk to root* strategy is negative cycle detection technique that words by considers if each new edge (v, w) forms a cycle or not. It does this efficiently by considering the directed subtree toward v – if w is present in the subtree then there is a cycle present; otherwise there isn’t.

**Nomenclature**

1. **Π[v]** - Shortest path which maintains predecessor of vertex v.
2. **d[v]** - Distance form source to v.
3. **ns[v]** - vector containing the number of times a vertex has been scanned.
4. **pd[v]** - Distance of a vertex in the previous iteration.
5. **Labeled\_New** – Set containing all newly labeled vertices.
6. **Labeled\_Old** – Set containing all vertices in the previous iteration.

**Implementation**

1. The algorithm starts by initializing d[v], pd[v], Π[v] to ∞ and ns[v] to 0.
2. We the set the distance of the source node s to 0, d[s] = 0.
3. A s is then assigned to a variable named source and the sets Labeled\_New and Labeled\_Old are made empty.
4. “The for loop runs for 2\*|V|, each iteration of the loop checks for a negative cycle with the if statement if(ns[source] > 2 || d[s] < 0). Otherwise, algorithm starts labeling the other vertices only if it’s distance in the previous iteration or its previous distance is greater than the newly evaluated distance. At the end of every iteration the vertex with the minimum of all the distances is made the source for the next iteration. Once the set Labeled\_Old becomes empty at the end of any iteration the algorithm terminates as it has obtained the solution” Saranya C.R et al.

**Experimental Results**

The algorithm employs the early termination property of Bellman Ford by using the timeout strategy, however it shines brightest when negative cycles are present, which gives the algorithm an average time complexity of O(n) and a worst case time complexity of O(n2), which is ideal for sparse graphs.

**N3\_neg.txt**

0 -1 inf

inf 0 -1

-1 inf 0

**Output-N3\_neg.txt**

Negative Loop Detected

0->1->2->0

**N7.txt**

0 inf 100 10 inf 32 inf

4 0 inf inf 17 inf 5

5 inf 0 30 inf 42 inf

inf 23 3 0 14 inf inf

inf 10 inf 26 0 2 inf

inf inf 9 13 3 0 inf

inf 6 inf inf 12 12 0

**output-N7.txt**

0, 33, 13, 10, 24, 26, 38

0

0->3->1

0->3->2

0->3

0->3->4

0->3->4->5

0->3->1->6

Iteration: 4

**N10.txt**

0 inf 3 inf 2 inf inf 1 inf inf

4 0 3 inf inf 3 8 inf 2 inf

2 inf 0 inf 5 inf 4 8 inf inf

5 inf inf 0 inf 4 inf inf 7 4

inf 3 8 inf 0 inf inf 3 inf inf

inf 1 inf inf 4 0 inf inf inf inf

inf inf 5 3 inf inf 0 inf inf 1

inf inf 7 inf 2 inf inf 0 inf inf

inf 2 inf inf inf 6 7 1 0 inf

inf inf 4 inf inf 3 1 2 inf 0

**output-N10.txt**

0, 5, 3, 10, 2, 8, 7, 1, 7, 8

0

0->4->1

0->2

0->2->6->3

0->4

0->4->1->5

0->2->6

0->7

0->4->1->8

0->2->6->9

Iteration: 5

**N7\_multi\_cost.txt**

[0,0,0] inf [20,20,60] [2,5,3] inf [14,14,4] inf

[1,2,1] [0,0,0] inf inf [2,4,11] inf [1,2,2]

[2,1,2] inf [0,0,0] [11,9,10] inf [20,12,10] inf

inf [3,10,10] [1,1,1] [0,0,0] [3,5,6] inf inf

inf [4,3,3] inf [6,12,8] [0,0,0] [0,1,1] inf

inf inf [1,4,4] [4,4,5] [1,1,1] [0,0,0] inf

inf [1,2,3] inf inf [-3,3,6] [4,4,4] [0,0,0]

**output-N7\_multi\_cost.txt**

0, 33, 13, 10, 24, 26, 38

0

0->3->1

0->3->2

0->3

0->3->4

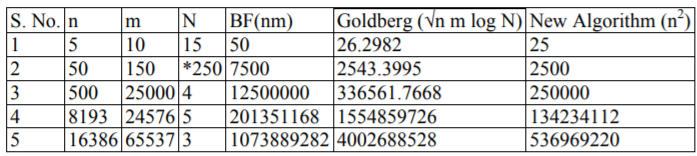
0->3->4->5

0->3->1->6

Iteration: 4

**Benchmarking**

Saranya C.R et al. provide a benchmark of their algorithm against Bellman Ford **O(nm)** and A.V. Goldberg **O()**. They observe that for graphs where, m is greater than or equal to 2n, the new algorithm is faster. The group benched marked their algorithm against arbitrarily chosen values for n, m and N, where N is the absolute value of the most negative edge weight.



**References**

Global Journal of Mathematical Sciences: Theory and Practical. ISSN 0974-3200 Volume 3, Number 4 (2011), pp. 307-315, Saranya C.R and Shobhalatha G.