Implementation of the Bellman-Ford Algorithm for calculating single source shortest distance on XEN10 FPGA Board

Digital Circuits Lab (EE-214)
Ritu Sharma(213079023)- 3rd-year WEL RA

Introduction

Bellman-Ford is a shortest path algorithm used to find the shortest paths from a single source vertex to all other vertices in a weighted graph. The Bellman-Ford algorithm works by iteratively relaxing the edges of the graph. The relaxation process involves updating the tentative distance of the vertices based on the current shortest paths found so far. The algorithm repeats this relaxation process for a certain number of iterations (equal to the number of vertices minus 1), ensuring that it gradually refines the distance estimates until the optimal shortest paths are found. In terms of time complexity, Bellman-Ford has a time complexity of O(V * E), where V is the number of vertices and E is the number of edges in the graph. This makes it less efficient than some other shortest path algorithms like Dijkstra's algorithm, but Bellman-Ford can handle graphs with negative-weight edges and is a more versatile solution in such cases.

Overall, Bellman-Ford is a fundamental algorithm for finding shortest paths in graphs and is widely used in networking, routing, and other applications where the graph may have varying edge weights.

Let $G = (V, E)$ denote the graph that consists of a set of vertices V and a set of edges E.
Let v denote the number of vertices
Let e denote the number of edges
Let edge(i, j) denote the edge from vertex i to vertex j
Let $w(i, j)$ denote the weight of edge(i, j)
Let $w(i)$ denote the weight of vertex i

```
Bellman-Ford($G(V, E)$)
for each vertex $x$ in $V$ do
    if $x$ is $source$ then
        $w(x) = 0$
    else
        $w(x) = \infty$
        predecessor$(x) = null$
    end if
end for
for $i = 1$ to $v - 1$ do
    for each edge$(i, j)$ in $E$ do
        if $w(i) + w(i, j) < w(j)$ then
            $w(j) = w(i) + w(i, j)$
            predecessor$(j) = i$
        end if
    end for
end for
```

Objective

Develop a high-level architecture for the Bellman-Ford algorithm on an FPGA board, taking into account the limitations of on-chip memory and computational resources. Divide the algorithm into four stages: memory read block, sorting block, computation block, and memory write block.

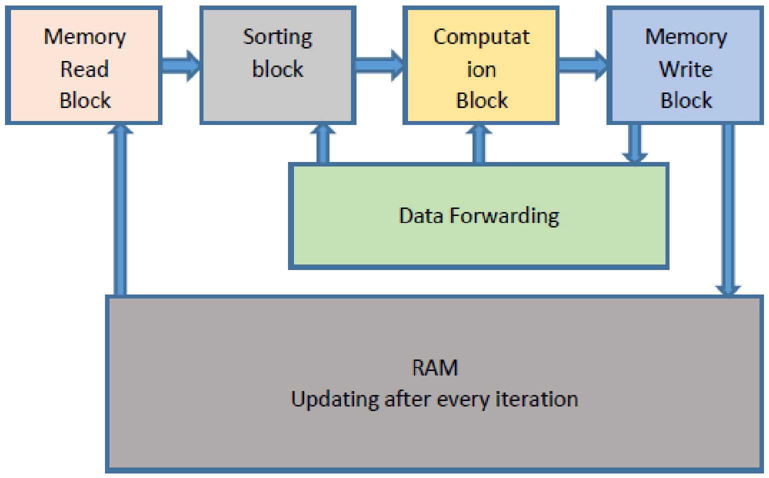


Fig 1.

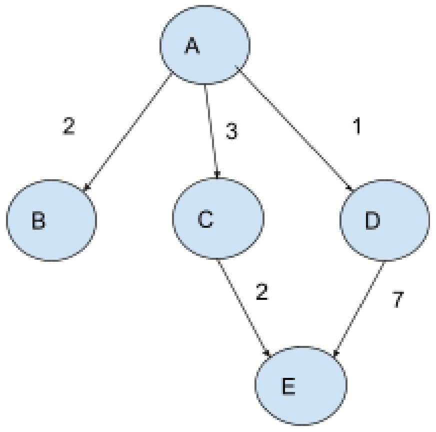
It is required to do **parallel processing** in this project to increase the throughput. Hence, you are supposed to take 4 words at a time from the memory. Each word contains information about the weight of the edge, source node number, and destination number.
Once you get your input you can start finding the shortest distance according to the **Bellman-Ford algorithm** which includes shorting and computational blocks.
Also, this project should be **pipelined** to further increase the throughput. The overall architecture is divided into 5 stages, so you need to implement 4 pipelined registers.
The 5 stages of the pipeline are Reading graph, memory read, sorting, computation, and memory write.

Reading graph - Take 4 words from the memory file.
Memory read - Take the previous shortest distance of the destination from the register file.
The register file includes your weight corresponding to the node and the predecessor node number.
Sorting - If the destination is the same for any 2 edges then find the shortest path among these 2 edges and make the longer distance path invalid.
Computation - Compare the new shortest distance with the previous distance of that destination node which you get in the memory read stage. I.e W(i,j) + W(i) < W(j) , where i is the source and j is the destination. If this condition is true then update the W(j) with W(i,j) + W(i).
Write back - If the W(j) is updated in the previous stage then we have to write it back to the register file.

Graph.mef file example - This information you have to give to your code

Wieght of edges , Source , Destination	**Comment**
010 001 010	A (001) to B (010) with distance 2 units
011 001 011	A (001) to C (010) with distance 3 units
001 001 100	A (001) to D (010) with distance 1 units
010 011 101	C (010) to E (101) with distance 2 units
111 100 101	D (100) to E (101) with distance 7 units

Note: Here A, B, C,D, E corresponds to node number 1,2,3,4,5 respectively.



You need to make one more memory, lets say, register file to store the distance from the starting node to other nodes

Register file (initialization)

	Distance from starting node, predecessor	**Comment**
000	1000 000	We know 0 is an invalid node from the graph
001	0000 000	001 is the starting node so distance is 0 unit
010	1000 000	MSB is 1 mean initially distance is infinite
011	1000 000	MSB is 1 mean initially distance is infinite
100	1000 000	MSB is 1 mean initially distance is infinite
101	1000 000	MSB is 1 mean initially distance is infinite

Register file (Final)

	Distance from starting node, predecessor	**Comment**
000	1000 000	We know 0 is an invalid node from the graph
001	0000 000	001 is the starting node so distance is 0 unit
010	0010 001	MSB = 0, distance of 010 from 001 is 2 units
011	0011 001	MSB = 0, distance of 011 from 001 is 3 units
100	0001 001	MSB = 0, distance of 100 from 001 is 1 units
101	0101 011	MSB = 0, distance of 100 from 001 is 1 units but here predecessor is c (011)

Weekly milestones

Pre-requisite:

- Understand the Bellman-Ford algorithm and its application in finding the shortest path between two nodes in a weighted graph.
- Read the paper " Accelerating Large-Scale Single-Source Shortest Path on FPGA" [1] as this project architecture is based on it.

Week 1

- Fix the architecture which includes
    - Max no of nodes and edges
- Implement the architecture in VHDL, starting with small modules and sub-modules
    - Memory files
    - Shorting Block
    - Computational Block
    - Pipeline Register
    - Forwarding Block

Week 2

- Once the modules and sub-modules are working correctly, integrate them into the main module using a hierarchical design.
- Validate the implementation using test benches and simulations in ModelSim Altera.

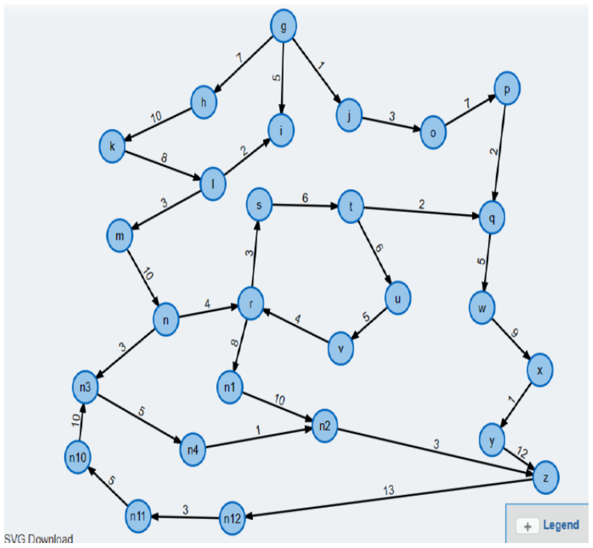
Week 3

- Once the simulation results are satisfactory, program the FPGA board with the VHDL code and use JTAG-UART to communicate with the system.
- Provide the input to the Bellman-Ford algorithm using switch pins as node address inputs and output to LEDs to represent the weight of the node whose address is given as input. Collect the output from the algorithm and cross-check it with the expected output to validate the implementation on the XEN10 board [3].
- Write code for traceback of path in python/C/C++.

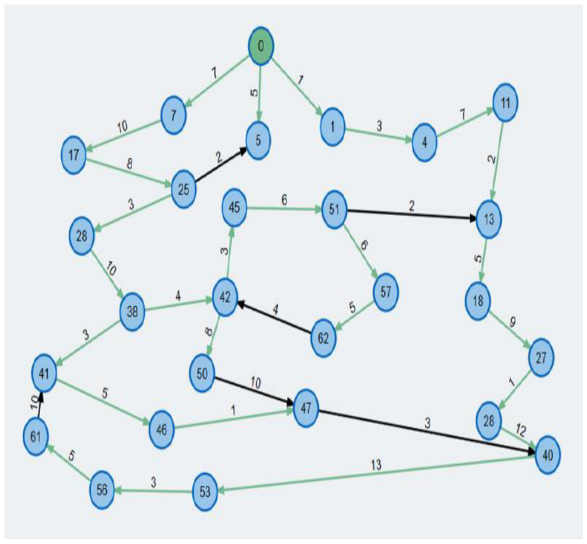
Bonus

- Trace back the shortest distance path from the predecessor values on the FPGA itself and display using LED matrix.

Input and expected output



Input Graph


Output Graph

The graph consists of a total of 27 nodes and 32 edges. For the output graph node 0 represents the source, colored in green and the value inside the circle denotes the final weights of respective nodes upon completion of the Bellman-Ford algorithm.

The input graph is stored in .mef file and details of the output is stored in a register file. See the above toy example for better understanding.

References

[1] Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
[2] https://www.cs.albany.edu/~cchelmis/pubs/raw15.pdf
[3]https://algorithms.discrete.ma.tum.de/graph-algorithms/spp-bellman-ford/index_en.html