Team Members:

- **Li Hao**
- **Zainab Fariha**
- **Zheng Xing**

Parallel Dijkstra Algorithm

Introduction:

This Document explains the parallel implementation of Dijkstra’s shortest path algorithm for a weighted graph given as adjacency matrix.

Dijkstra Algorithm:

The Dijkstra algorithm (or Dijkstra shortest path algorithm) is a graph search algorithm to find the shortest path between nodes in a graph. It is used to solve the single source shortest path problem for the graphs with non-negative weight. It is widely used in network routing protocols, road networks etc.

Dijkstra algorithm finds a shortest path tree from a single source node, by building a set of nodes that have minimum distance from the source.

Application Domain of Dijkstra Algorithm:

- Geographic Information System (GIS), which needs to determine the shortest path between point A to point B on a road map. Example: Google maps.
- IP routing to Open shortest Path first.
- Telephone networks
- Game Development
- Artificial Intelligence

Explanation of Dijkstra Algorithm:

- **Dijkstra Algorithm- 1st round:**
  1. So, starting at A the shortest path to A is 0, so we would write 0 in the table. Since we have finished considering A as a destination, we would put a box around it.
  2. Now we would write the distance of each node that can be reached from A i.e B, C, D.
  3. Since F, G, H aren't directly connected to A so we would put highest distance i.e. infinity for them.

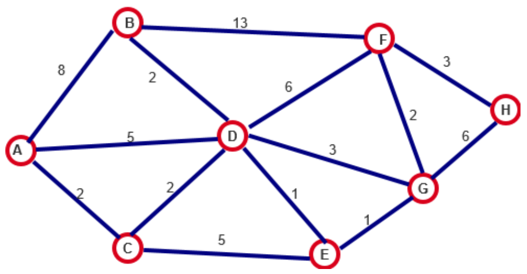
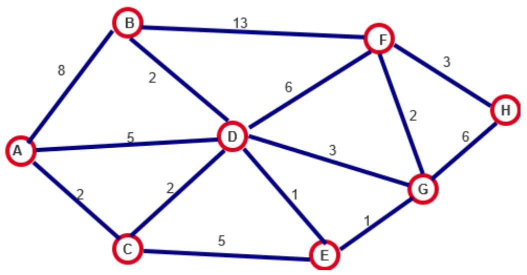


Fig1 8-node Simple network

Vertex	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞

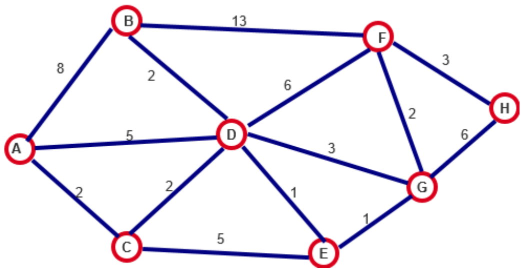
Table1: The routing table for node A

- **Dijkstra Algorithm- 2nd round:**
  1. Now looking at the first row the shortest distance is from A to C i.e. 2. So, we would copy that into the second row.
  2. Now I can't go to B from C so we copy it as it is.
  3. The distance from node C to D is 2 and from A to C is 2 that totals to 4 do we put 4 into the column of D. Repeat the same for E



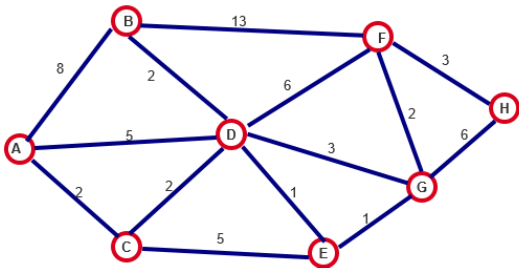
Vertex	A	B	C	D	E	F	G	H
A	0_A	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞

- **Dijkstra Algorithm- 3rd round:**
  1. The minimum distance is now is for vertex D i.e. 4. So, D is the next vertex to consider.
  2. The distance to B from D is 6 i.e. smaller than previous 8 so we replace 8 by 6. Similarly calculate the weight for E, F, G.



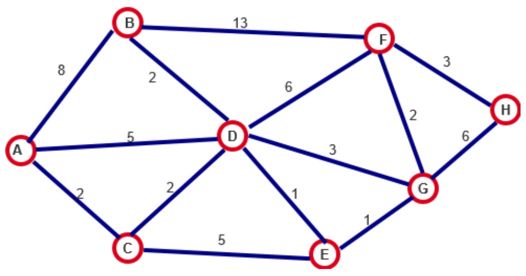
Vertex	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞
D		6_D		$\boxed{4_C}$	5_D	10_D	7_D	∞

- **Dijkstra Algorithm- 4th round:**
  1. The smallest weight now is 5, so we have copied that from the previous row, and repeat the process for all the nodes that are not in the box.



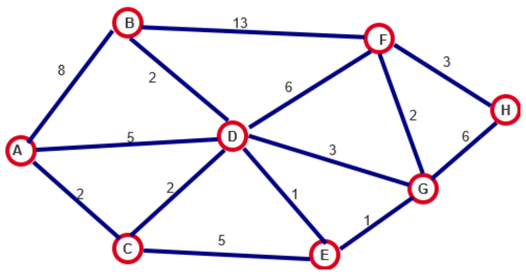
Vertex	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞
D		6_D		$\boxed{4_C}$	5_D	10_D	7_D	∞
E		6_D			$\boxed{5_D}$	10_D	6_E	∞

- **Dijkstra Algorithm- 5th round:**
  1. Now there are two smallest values 6D and 6E so we can choose any one of them to copy to the new row.



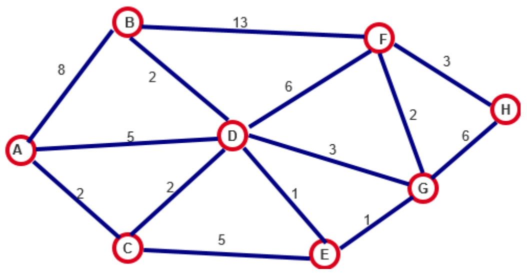
Vertex	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞
D		6_D		$\boxed{4_C}$	5_D	10_D	7_D	∞
E		6_D			$\boxed{5_D}$	10_D	6_E	∞
B		$\boxed{6_D}$				10_D	6_E	∞

- **Dijkstra Algorithm- 6th round:**



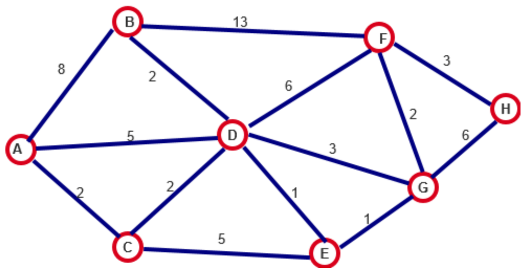
Vertex	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞
D		6_D		$\boxed{4_C}$	5_D	10_D	7_D	∞
E		6_D			$\boxed{5_D}$	10_D	6_E	∞
B		$\boxed{6_D}$				10_D	6_E	∞
G						8_G	$\boxed{6_E}$	12_G

- **Dijkstra Algorithm- 7th round:**



Vertex	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞
D		6_D		$\boxed{4_C}$	5_D	10_D	7_D	∞
E		6_D			$\boxed{5_D}$	10_D	6_E	∞
B		6_D				10_D	6_E	∞
G						8_G	$\boxed{6_E}$	12_G
F						$\boxed{8_G}$		11_F

- **Dijkstra Algorithm- 8th round:**



Vertex	A	B	C	D	E	F	G	H
A	$\boxed{0_A}$	8_A	2_A	5_A	∞	∞	∞	∞
C		8_A	$\boxed{2_A}$	4_C	7_C	∞	∞	∞
D		6_D		$\boxed{4_C}$	5_D	10_D	7_D	∞
E		6_D			$\boxed{5_D}$	10_D	6_E	∞
B		$\boxed{6_D}$				10_D	6_E	∞
G						8_G	$\boxed{6_E}$	12_G
F						$\boxed{8_G}$		11_F
H								$\boxed{11_F}$

Now all the boxes show the shortest path from A to all the other vertices in the boxes.

Our Implementations

- **Sequential Dijkstra's Algorithm Pseudo Code:**
  Below is the pseudo code for our implementation of the serialized version of the Dijkstra Algorithm.

```
Create a cluster cl[V]
Given a source vertex s
While (there exist a vertex that is not in the cluster cl[V])
{
FOR (all the vertices outside the cluster)
Calculate the distance from non-member vertex
to s through the cluster
END
** O(V) **
Select the vertex with the shortest path and add it to
the cluster
** O(V) **
}
```

Runtime $O(V^2)$

In order to obtain the routing table we need O(V) rounds of iterations for all the vertices. In each round we will update the value for O(V) vertices and select the smallest weighted vertex, so the run time for each round is O(V) and the total runtime would be O(V) * O(V) i.e. O(V2).

Disadvantage:

For a large network the latency would be very high, this would pose as a disadvantage especially for the real time applications.

- <u>**Parallel Dijkstra Algorithm**</u>

  1. We tried to reduce the latency for Dijkstra algorithm by trying to parallelize it.

  2. The approach we took is:

  a) Each core identifies its closest vertex to the source vertex and perform parallel prefix to select the globally closest vertex.

  b) It will then broadcast the result to all the cores so that they can update their list of vertices accordingly.

Pseudo-code:

```
Create a cluster cl[V]
Given a source vertex s
Each core handles a subgroup of V/P vertices
While (there exist a vertex that is not in the cluster cl[V])
{
FOR (vertices in my subgroup but outside the cluster)
Calculate the distance from non-member vertex to s
through the cluster;
Select the vertex with the shortest path as the local
closest vertex;
END
** Each processor work in parallel O(V/P) **
Use the parallel prefix to find the global closest vertex
among all the local closest vertices from each core.
** Parallel prefix O(V/P) **
}
```

Run Time Complexity:

P is the number of cores used. In order to obtain the routing table we need O(V) rounds of iteration i.e. until all the vertices are included in the cluster. The value of O(V) vertices would be updated in parallel using P cores running at the same time independently. Then we would do the parallel prefix to get the global closest vertex.

So, the total time complexity in each round id $\mathbf{O(V^2/P + V^2/P) = O(2V^2/P)}$.

<u>Comparison Between the Sequential and parallel:</u>

Data SIZE (no. of Vertices /no. of edges)	SEQUENTIAL (ms)	Parallel (ms) with 4 threads
5/20	0	2
10/90	0	7
100/9900	0	75
1000/999000	35	622
2000/3998000	64	2104
3000/8997000	129	2424
4000/15996000	299	11505

Improvements:

As can be seen from the table we are not able to obtain the running time improvement through our implementation. So, the future implementation that we can suggest of is to decrease the run time of the algorithm.

References:

- Wikipedia
- An implementation of parallelizing Dijkstra by Zilong Ye