Veermata Jijabai Technological Institute, Mumbai 400019

Experiment No.: 05

Aim : Implement DIJKSTRA'S ALGORITHM using OpenMP

Name: Kiran K Patil

Enrolment No.: 211070904

Branch: Computer Engineering

Batch: IV

Theory :

Introduction to Dijkstra's Algorithm

Ever wondered how does Google Maps find the shortest and fastest distance between two places?

Well the answer is Dijkstra's Algorithm. Dijkstra's Algorithm was conceived by computer scientist Edsger W. Dijkstra in 1956. It is a single source shortest paths algorithm. It means that it finds the shortest paths from a single source vertex to all other vertices in a graph. It is a greedy algorithm and works for both directed and undirected, positively weighted graphs (a graph is called positively weighted if all of its edges have only positive weights).

Working of Dijkstra's Algorithm

Dijkstra's Algorithm requires a graph and source vertex to work. The algorithm is purely based on greedy approach and thus finds the locally optimal choice(local minima in this case) at each step of the algorithm.

In this algorithm each vertex will have two properties defined for it

1. Visited property: -

- This property represents whether the vertex has been visited or not.
- We are using this property so that we don't revisit a vertex.
- A vertex is marked visited only after the shortest path to it has been found.

2. Path property: -

- This property stores the value of the current minimum path to the vertex. Current minimum path means the shortest way in which we have reached this vertex till now.
- This property is updated whenever any neighbour of the vertex is visited.
- The path property is important as it will store the final answer for each vertex.

Algorithm

1. Mark the source node with a current distance of 0 and the rest with infinity.
2. Set the non-visited node with the smallest current distance as the current node, lets say C.
3. For each neighbour N of the current node C: add the current distance of C with the weight of the edge connecting C-N. If it is smaller than the current distance of N, set it as the new current distance of N.
4. Mark the current node C as visited.
5. Go to step 2 if there are any nodes are unvisited.

Dijkstra's Algorithm Using OpenMP

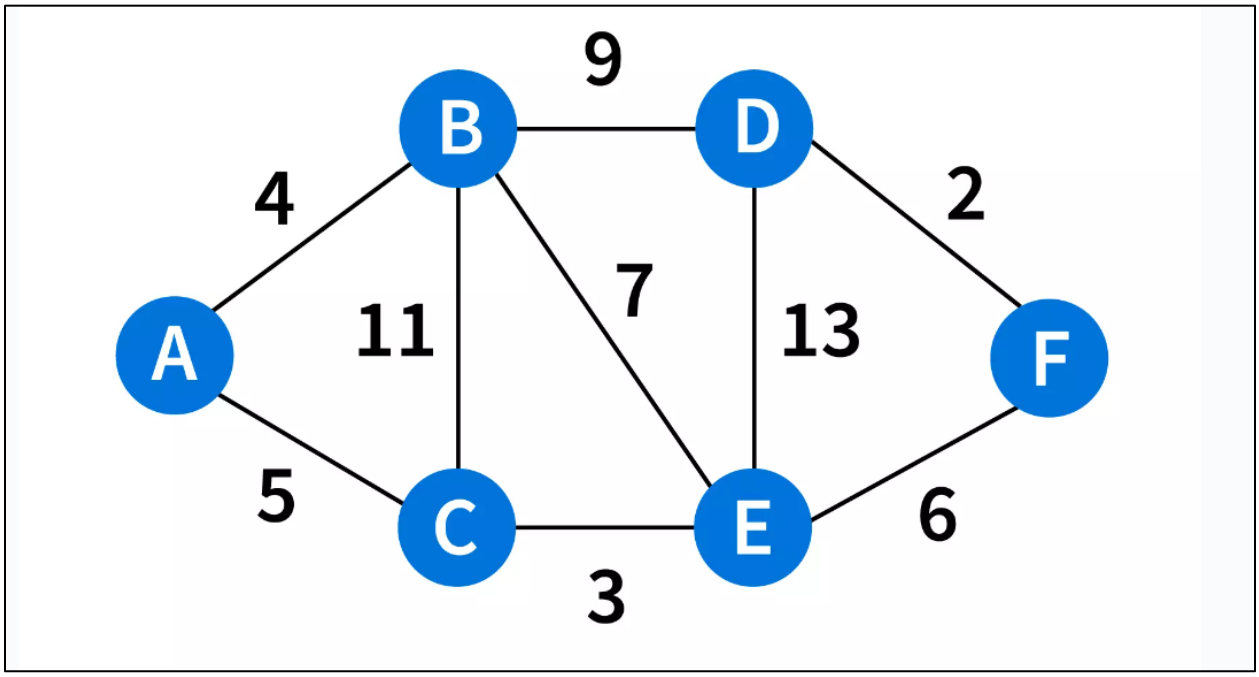
- This is an implementation of Dijkstra's Algorithm using OpenMP in C for a graph with 6 vertices.

- The **minDistance** function is parallelized using the parallel for directive of OpenMP, which distributes the iterations of the loop among the available threads.

- The **dijkstra** function has a loop that iterates V-1 times, where V is the number of vertices in the graph. In each iteration, it finds the vertex with the minimum distance from the source vertex that has not been visited yet, and marks it as visited. Then, it updates the distances of its neighbours if a shorter path is found.

- The parallel for directive is used again to parallelize the loop that updates the distances of the neighbours of the current vertex. Note that we only parallelize the inner loop that iterates over the neighbours of the current vertex, not the outer loop that iterates over the vertices.

Example:



Output :

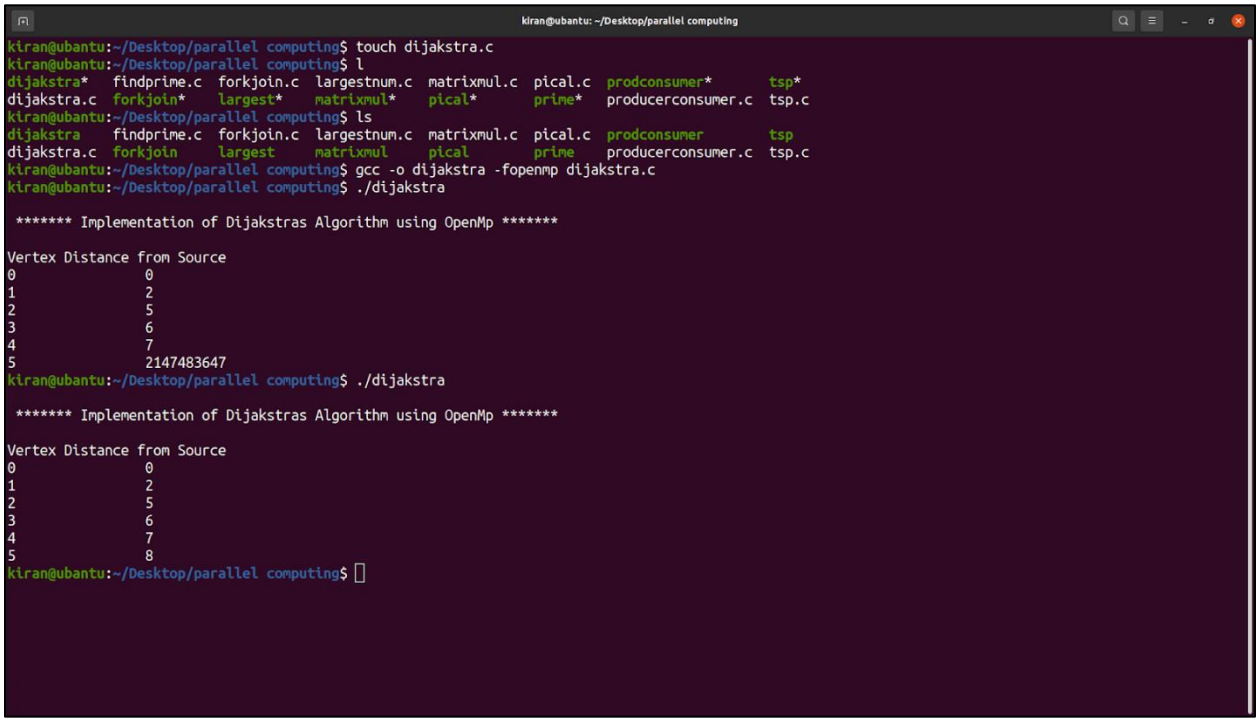
Vertex	Distance from Source
A	0
B	4
C	5
D	13
E	8
F	14

Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <omp.h>

#define V 6  // Number of vertices in the graph

int minDistance(int dist[], int visited[])
{
  int min = INT_MAX, min_index;

  #pragma omp parallel for
  for (int i = 0; i < V; i++)
  {
      if (visited[i] == 0 && dist[i] <= min)
      {
          min = dist[i];
          min_index = i;
      }
  }

  return min_index;
}

void dijkstra(int graph[V][V], int src)
{
  int dist[V];
  int visited[V];

  #pragma omp parallel for
  for (int i = 0; i < V; i++)
  {
      dist[i] = INT_MAX;
      visited[i] = 0;
  }

  dist[src] = 0;

  for (int count = 0; count < V - 1; count++)
  {
      int u = minDistance(dist, visited);
      visited[u] = 1;
```

```
        #pragma omp parallel for
        for (int v = 0; v < V; v++)
        {
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]
+ graph[u][v] < dist[v])
            {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    printf("\n ******* Implementation of Dijakstras Algorithm using OpenMp *
****** \n\n");
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; i++)
    {
        printf("%d \t\t %d\n", i, dist[i]);
    }
}

int main()
{
    int graph[V][V] = {{0, 2, 0, 6, 0, 0},
                       {2, 0, 3, 8, 5, 0},
                       {0, 3, 0, 0, 7, 0},
                       {6, 8, 0, 0, 9, 10},
                       {0, 5, 7, 9, 0, 1},
                       {0, 0, 0, 10, 1, 0}};

    dijkstra(graph, 0);

    return 0;
}
```

Output :



Conclusion:

- Thus, we implemented Dijkstra's algorithm using OpenMP, a parallel programming model for shared memory multiprocessing. The program finds the shortest path from a given source vertex to all other vertices in a weighted graph with V vertices. The program uses OpenMP parallelization to speed up the computation of the algorithm. The output of the program shows the distance of each vertex from the source vertex.