|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| **Parallel and distributed computing**  **Report** |
| |  |  |  | | --- | --- | --- | | 1. Yahya Naeem (20K-1065) | Mehma Waseem (20K-0261) | Sabah Mawani (20K-0393) | |

# ***Shortest Path Finder(Paarallel vs Serial)***

# ***Introduction :***

This software enables the user to find shortest route from the given routes using longitude ,latitude and altitude from the file . The shortest route has the minimum value among all the routes .

# ***Background of project:***

We researched from following sites/Apps .

* Google Maps (For finding paths ) .
* Google for longitude ,latitude and altitude value .
* <http://answers.google.com/answers/threadview?id=326655>
* <https://github.com/OpenMP/Examples/tree/v4.5.0/sources>
* Classroom Slides .

# ***Statement of Problem :***

How can we find minimum distance between starting point and ending point ? This is necessary as the world evolved people moved toward the time saving approach . This software find the Minimum distance path between two point (Shah-Latif and Nazimabad) using the longitude ,Latitude and altitude values . With the help of formula

It finds out the distances between the points in route and ultimately distance of route . Then it compares it with the other routes and draws out the route with minimum distance .

# ***Scope of study :***

We studied the longitude ,latitude and altitude relation to find out the distance by using the formula . Also the role of openmp and its clauses like shared , private and some others were studied during the project . We went through google maps to find paths between the fixed points .

# ***Methodology :***

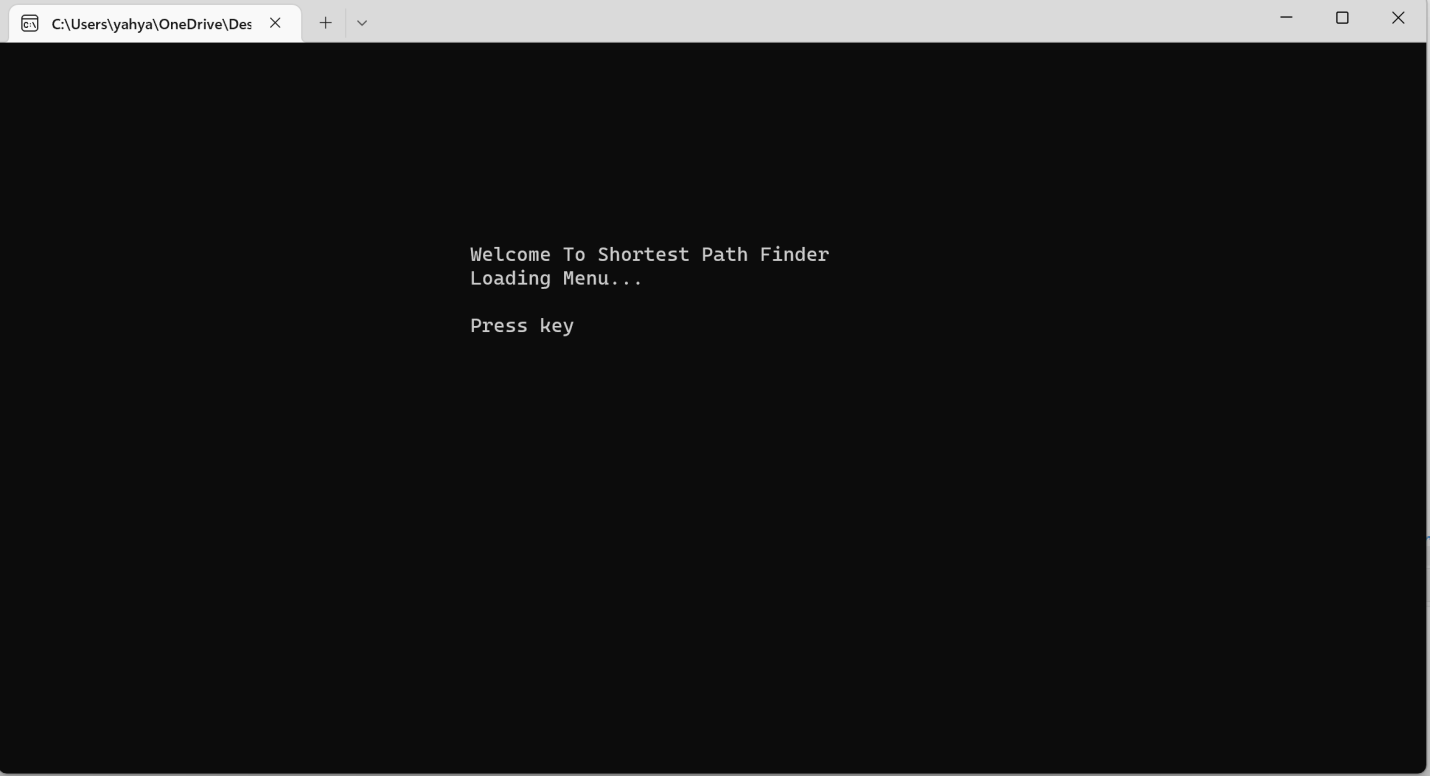
The code first accesses the csv file, where we’ve stored all our data in relation to each path, and the location of each node. The paths are formed between the same starting and ending point (Shah-latif - Nazimabad ). Each path has the number of locations between the starting and ending point stored within the file. These are accessed and then stored in a jagged array structure which is then used to calculate the minimum distance path both serially and parallely.

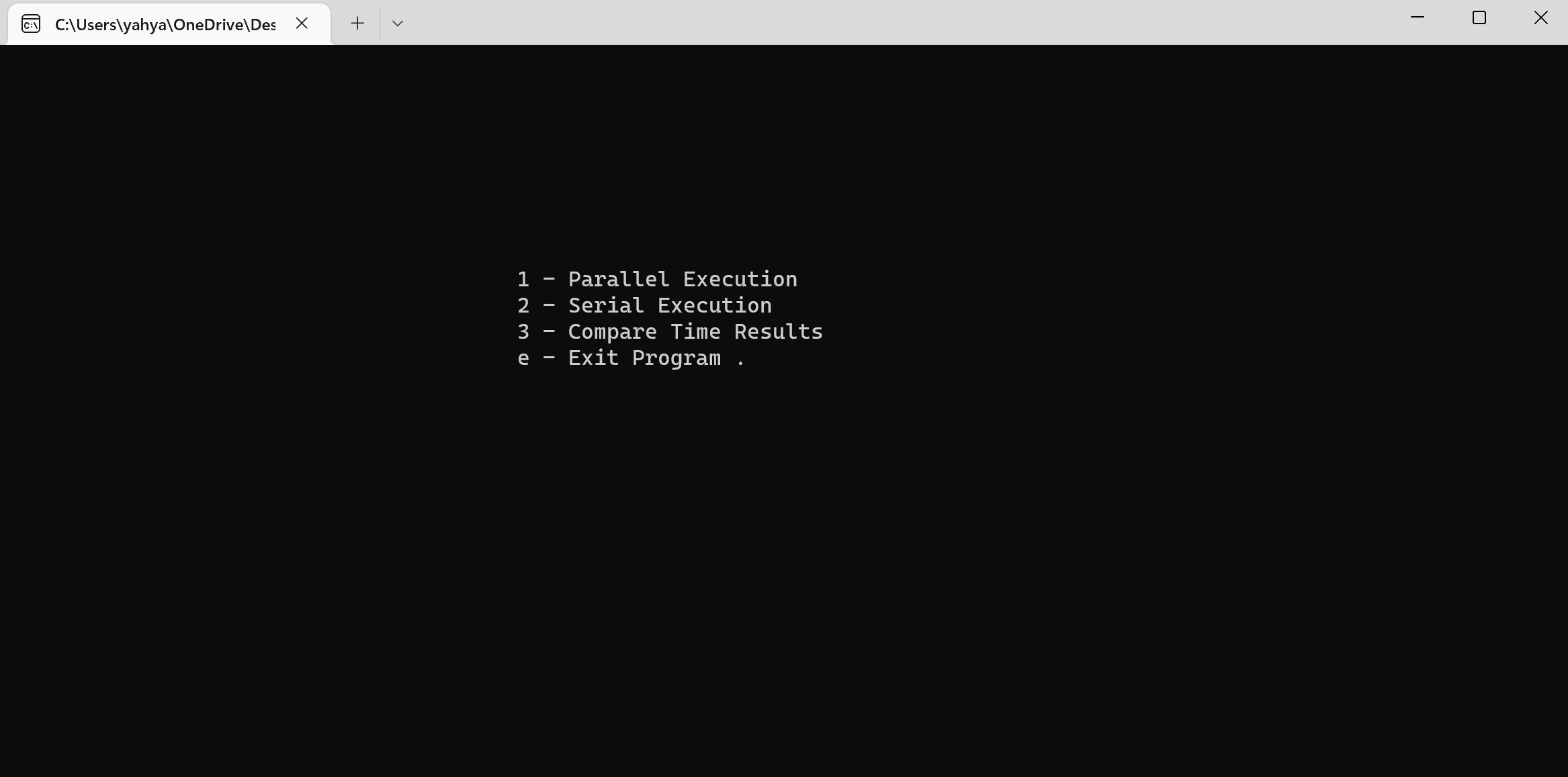
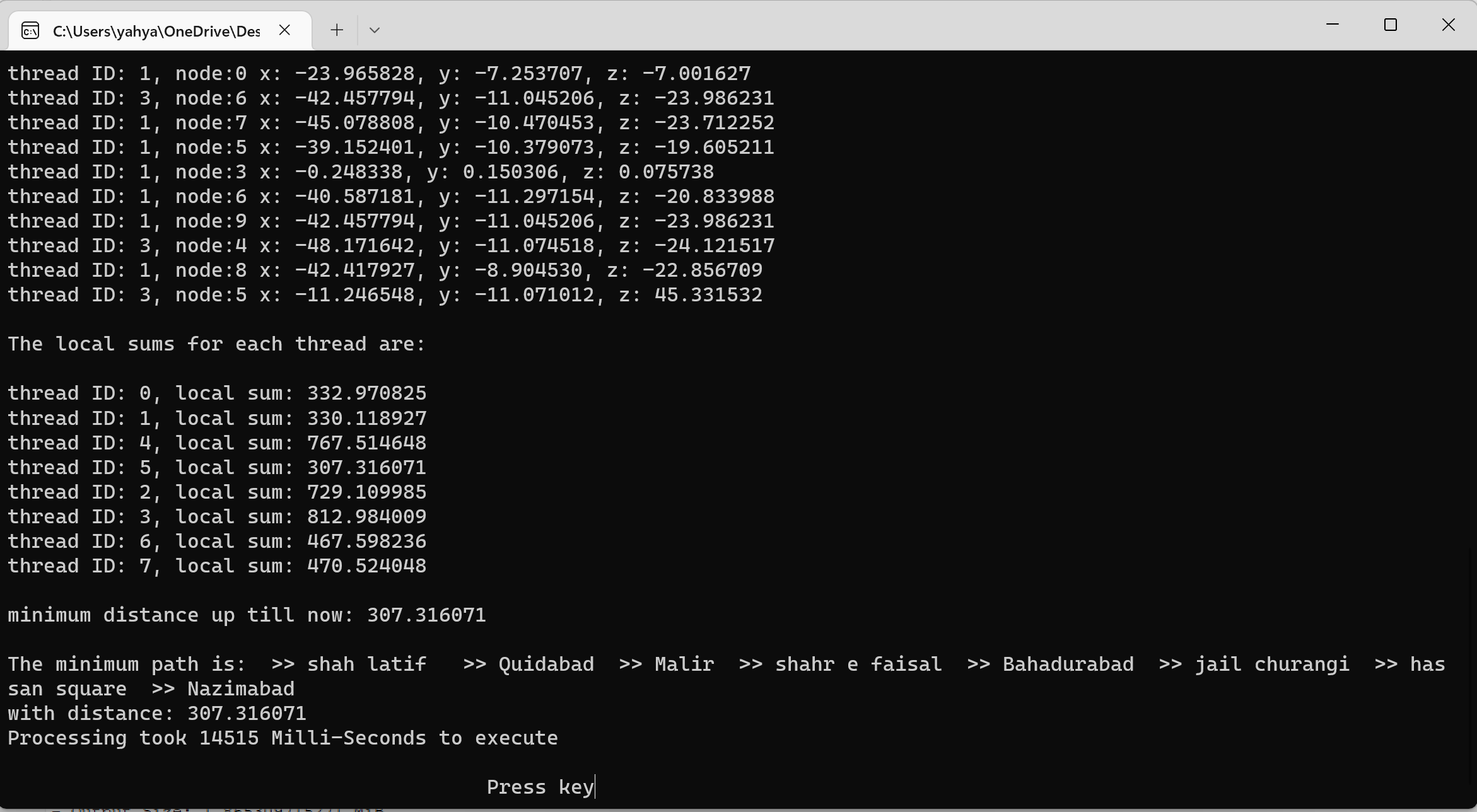
For the serial approach we’ve calculated the x, y and z values using the formulae given above, these are then used to calculate the distance between each pair of adjacent points. The sum of these distances is then found and stored in an array, after which the minimum is found and displayed. The time taken is also recorded and presented at the end.

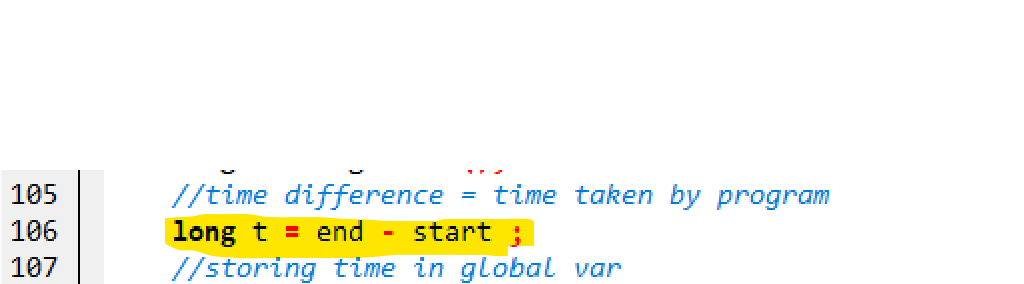
For the parallel approach we’ve used the concept of , , ,etc. In order to calculate the various values. The time taken by each thread, and its local results are shown as well. We’ve also used the concept of nesting parallel regions, in order to make the code more efficient.

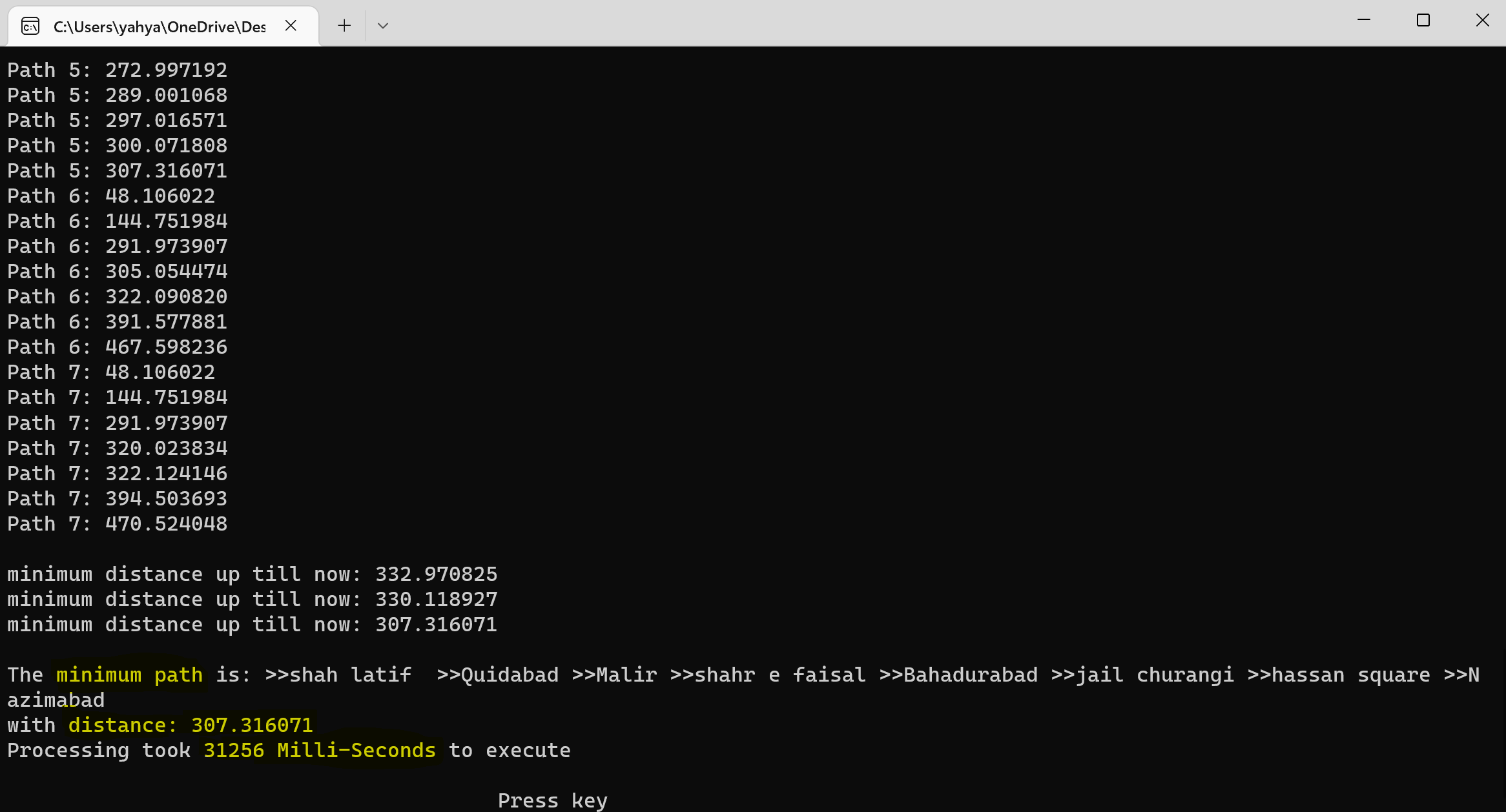
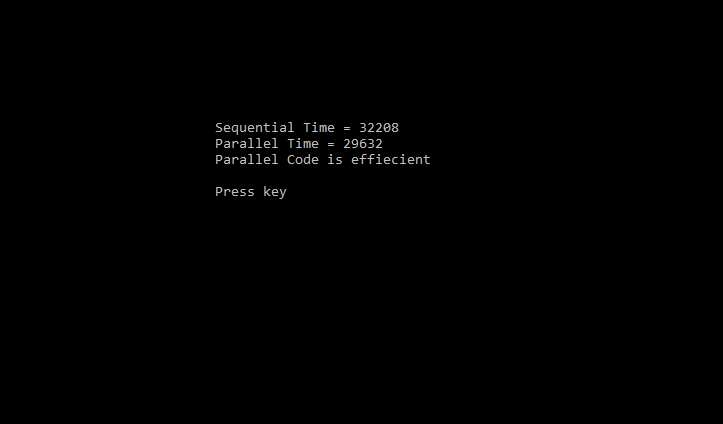
# ***Programming Design :***

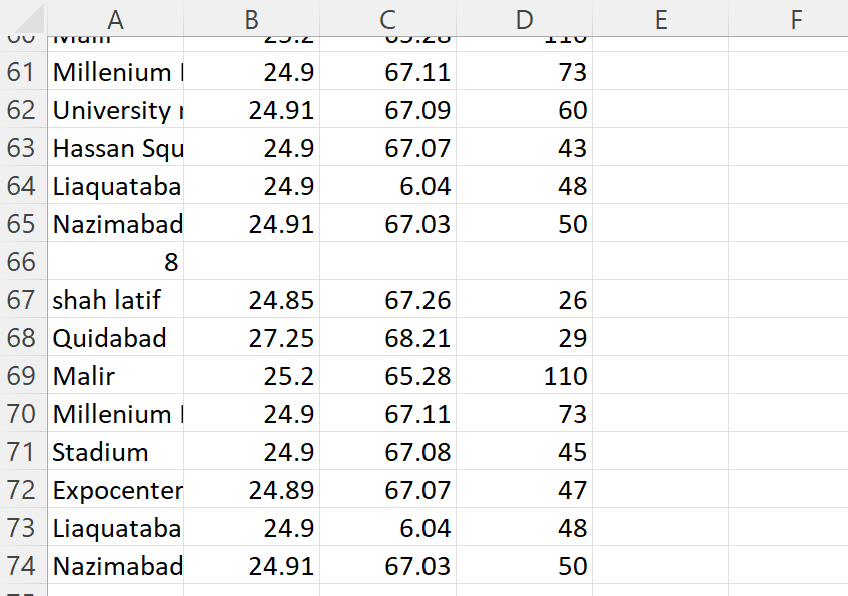
The project start with a Program Name and proceeds to the menu as key is pressed .



The user menu has following options on console .  
  
Each option Is associated with a key . The user enters his preference no and moves to that section .   
>>>Firstly Consider for **Parallel Execution .** The parallel code is executed with 8 threads that evaluates 8 paths on backend . Each thread is scheduled dynamically with chunk size of 2 to make it efficient . The threads not only calculates the shortest paths but also show their Local results , Shortest Path and Distance . The output screen is as follows .  


The program ends by showing time in Milli-seconds , which is calculated by the difference as .  
 **Time = Time\_after\_execution - Time\_before\_execution**   
At backend time calculation is .  
    
Now on pressing key User returns to the menu and ,  
>>> Now to the second choice , **Sequential Execution .**

In sequential execution the data file is read and output is evaluated sequentially with the help of loops . the path no’s and their sums are shown . The shortest path is calculated by comparisons and minimum path with it’s Route is printed . The time taken is also printed out .  
The console appears as follows .  
  
On pressing key the user is returned back to the menu to execute third option . which is   
>>> **Compare time results .**  
 It shows both the time taken in sequential and parallel execution and also tells the user that which one of the code execution was efficient by comparisons .  
Output is as follows .  


The data is stored in .CSV format containing routes of multiple paths . A screenshot from data file is added below .  


# ***Conclusions :***

So by the results we can conclude that parallel execution of codes takes lesser time , making it efficient regardless of overheads as compared to the sequential one’s .