**PARALLEL AND DISTRIBUTED COMPUTING**

**PROJECT REPORT**

***Graph Algorithms: BFS and A Search*\***

**GROUP MEMBERS:**

**Abdul Ahad 22k4353**

**Abrar Malik 22k4429**

**Ashar Usmani 22k4581**

In this section, we will explore the process of implementing two core graph algorithms: **Breadth-First Search (BFS)** and *A Search*\*. We will delve into the underlying logic of both algorithms, including their data structures, computational steps, and the challenges encountered during their development.

For **BFS**, we explore how the algorithm explores all vertices at the current level before moving to the next level, making it suitable for finding the shortest path in an unweighted graph. The implementation challenges stemmed from ensuring an efficient traversal of the graph, especially when considering parallel processing using OpenMP.

For the *A Search*\* algorithm, we discuss the incorporation of a heuristic to improve search efficiency. The A\* algorithm combines features of Dijkstra’s algorithm and greedy best-first search, which allows it to perform better in terms of computational complexity in specific use cases. We highlight how different heuristics impacted the algorithm's performance, and how tuning the heuristics affected the outcomes.

The challenges involved in both implementations included managing memory efficiently, handling parallelization with OpenMP, and ensuring that the algorithms returned correct and optimal results in all scenarios. Additionally, handling large graphs, managing concurrency, and debugging parallel executions posed difficulties.

This section presents a comprehensive analysis of the **numerical results** obtained from running the BFS, A\* and Greedy BFS algorithms across various test cases. We evaluate their performance using **execution time**, **memory usage**, and **scalability** as key performance metrics.

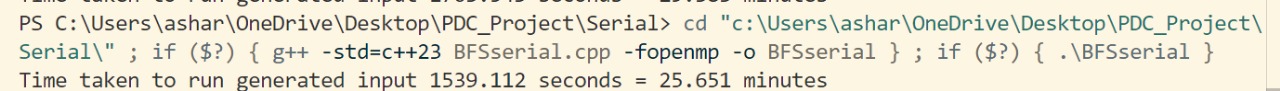
We provide a series of experiments to show how both algorithms performed with varying graph sizes (nodes and edges). For each test case, the execution time and memory usage are compared to evaluate the efficiency of each algorithm. Additionally, we measure the scalability of the parallelized versions of BFS, Greedy BFS and A\*, noting how performance improves (or potentially degrades) with the number of processors or threads used.

A key part of this section will also involve comparing the results of **parallel vs. serial executions**, demonstrating the impact of parallelism on performance. This includes a comparison of **OpenMP-parallelized BFS** against a serial version to analyze the trade-offs in terms of computation time and memory consumption.

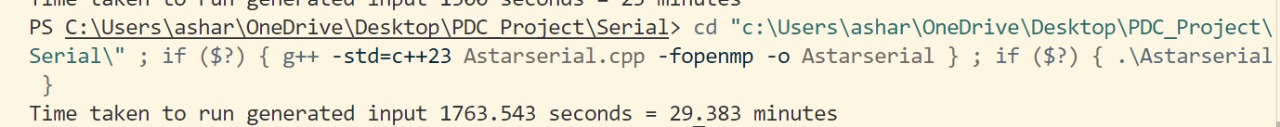
Across all algorithms the speedup is quite proportional to the number of cores used allocated to the system, although the speedup does gradually decrease as the cores are increased due to the parallel, this is caused by dependencies.

The Serial run times are attached below:

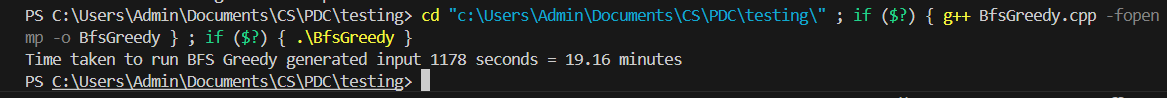
* A\*



* BFS



* BFS Greedy



\*When run on local machine

Overall the parallelization was successful and the run-times were significantly reduced in comparison to the original serial run times.