COURSE NAME: Operating Systems

CONTEXT: Project Report

SECTION: 4J

SUBMITTED BY:

- Sunny Shaban Ali [22K-4149]
- Hamza Ahmed Khan [22K-4647]

SUBMITTED TO: Miss Mubashra Fayyaz

SUBMITTED ON: 13th May, 2024

Table of Contents

INTRODUCTION

Project Objective:

The primary goal of this project was to develop a multithreaded implementation of the A* search algorithm to determine the minimum cost path between two nodes within a graph. A* search is a renowned pathfinding algorithm that merges the advantages of Dijkstra's algorithm and Greedy Best-First Search.

Problem Description:

The task at hand involved working with a graph that serves as a map, where nodes represent different locations and edges symbolize paths connecting these locations. The project focused on the challenge of discovering the most cost-effective path starting from a designated start node and concluding at a specified goal node.

By employing multithreading, the implementation aimed to enhance the efficiency and performance of the A* search algorithm when dealing with graphs and pathfinding scenarios. Multithreading enables parallel processing of computational tasks, potentially accelerating the search process and optimizing the exploration of possible paths within the graph.

The utilization of multithreading in the A* search algorithm facilitates concurrent execution of path evaluation and exploration, leading to improved scalability and speed in finding the optimal path between the given nodes. This approach enhances the algorithm's capability to handle complex maps efficiently and effectively determine the minimum cost route.

The project's focus on integrating multithreading with the A* search algorithm underscores the significance of leveraging parallel computing

techniques to enhance the pathfinding process in graph-based scenarios. By combining the power of multithreading with the intelligence of the A* search algorithm, the project aimed to achieve faster and more robust pathfinding outcomes for various graph-based applications.

IMPLEMENTATION

Scenario Description:

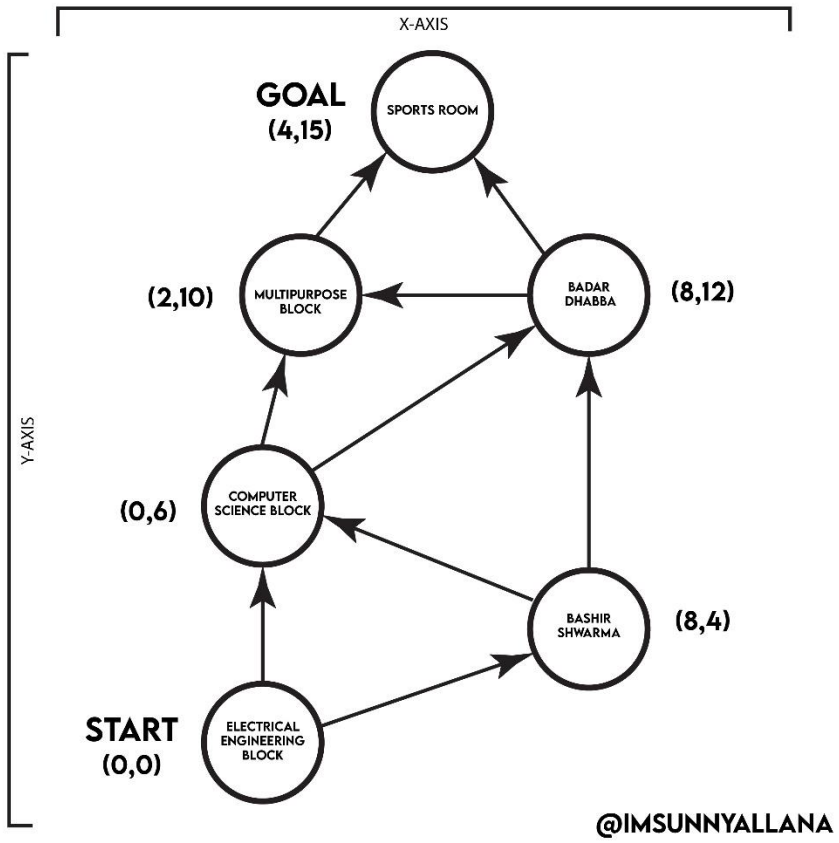
Within the simulated representation resembling FAST University, a meticulously crafted graph structure was devised to emulate the intricate layout of the educational institution. The journey embarked from the distinguished landmark of the Electrical Engineering (EE) Block, serving as the pivotal starting point for the traversal through the campus landscape.

The subsequent nodes within the graph were meticulously delineated to reflect key locations within the campus environment:

1. **EE_BLOCK (Starting Point):** Signifying the initial departure point, the EE Block stood as an emblematic representation of the commencing location within the university premises.
2. **BADAR_DHABBA:** Positioned strategically in the graph, this node denoted a notable spot within the campus surroundings, following the EE Block in the traversal sequence.
3. **BASHIR_SHWARMA:** Another distinctive node intricately integrated into the graph, symbolizing a specific point of interest within the campus fabric.

4. **CS_BLOCK:** A pivotal node encapsulating the essence of the Computer Science Block, strategically placed within the graph to denote a significant landmark within the campus layout.
5. **SPORTS_ROOM (Destination):** Serving as the final destination node, the Sports Room epitomized the ultimate goal of the traversal, representing a recreational facility within the campus enclave.

The fundamental essence of this scenario revolved around orchestrating a seamless navigation trajectory from the EE Block, the origin point, towards the Sports Room, the ultimate terminus. The core objective of this journey encompassed the meticulous optimization of the traversal path to minimize cost or distance, thereby exemplifying the intricate interplay between pathfinding algorithms and efficient route planning within a complex campus setting.



Implementation in a Nutsell:

- The program employs the pthread library to create and manage multiple threads.
- NUM_THREADS threads are created to execute the A* search algorithm concurrently.
- Each thread executes the aStarSearch function.
- The graph is represented as a vector of nodes, where each node contains its ID and a list of neighboring nodes.
- Node indices are stored in a map for efficient node lookup.
- A mutex (mtx) is used to synchronize access to shared resources, such as the priority queue (pq) and the exit flag (exitFlag).
- Before accessing shared data structures, threads acquire the mutex lock. After completing their operations, they release the lock.
- Threads access the priority queue (pq) to retrieve and insert paths.
- Mutex locks are employed before accessing the priority queue to prevent data races and ensure thread safety.
- The exit flag (exitFlag) is used to signal termination.
- When the priority queue is empty, threads check the exit flag. If set, they exit the loop; otherwise, they continue waiting for tasks.
- Each thread computes the cost of traversing a path based on the actual cost from the start node (startNode) and the heuristic cost to the goal node (goalNode).
- When a thread finds a path reaching the goal node, it updates the minCostPath if necessary and sets the exit flag to terminate other threads.

TRY IT FOR YOURSELF

Project Structure:

AStar-Threaded-Search/

```
│
├── documentFiles/
│   ├── projectProposal.pdf
│   └── projectReport.pdf
│
└── projectFiles/
    ├── Makefile
    ├── testcase
    └── src/
        ├── singleThreaded.cpp
        └── multiThreaded.cpp
```

- **documentFiles/:** Contains documents related to the project, including the project proposal and project report.
  - projectProposal.pdf: Document outlining the initial proposal for the project.
  - projectReport.pdf: Final report detailing the project's implementation, results, and analysis.
- **projectFiles/:** Contains files related to the project's code and build process.
  - Makefile: Makefile for compiling the project.

- testcase: File containing test cases for evaluating the project's functionality.
- src/: Directory containing the source code for the project.
  - singleThreaded.cpp: Implementation of the A* search algorithm using a single thread.
  - multiThreaded.cpp: Implementation of the A* search algorithm using multiple threads for parallelization.

Steps to Produce Output:

- **Cloning Repository using Git**:
  - Make sure you have Git installed
  - Open the terminal or command prompt on your system.
  - Run the following command to clone the repository from GitHub:

```
@sunnyallana →/workspaces/AStar-Threaded-Search (main) $ git clone https://github.com/sunnyallana/AStar-Threaded-Search.git
Cloning into 'AStar-Threaded-Search'...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 39 (delta 6), reused 29 (delta 2), pack-reused 0
Receiving objects: 100% (39/39), 842.67 KiB | 4.26 MiB/s, done.
Resolving deltas: 100% (6/6), done.
```

  - Change directory to cloned directory's project files

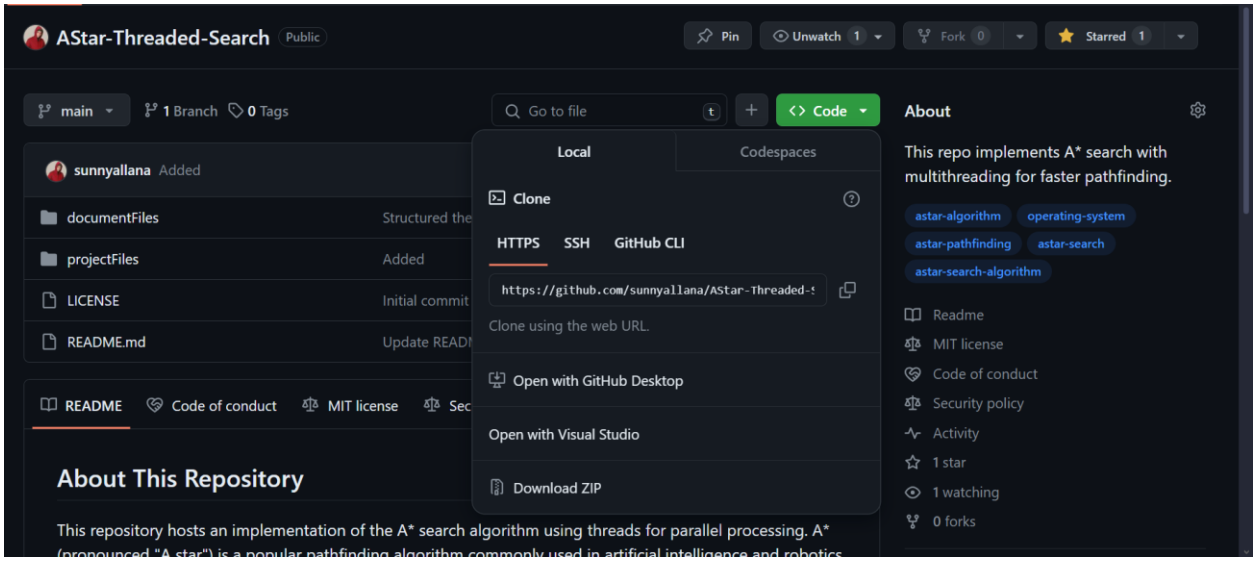
```
@sunnyallana →/workspaces $ cd AStar-Threaded-Search/projectFiles/
@sunnyallana →/workspaces/AStar-Threaded-Search/projectFiles (main) $
```

- Execute make all and make run_multi to see output of multithreaded.cpp or make run_single to see output of singleThreaded.cpp

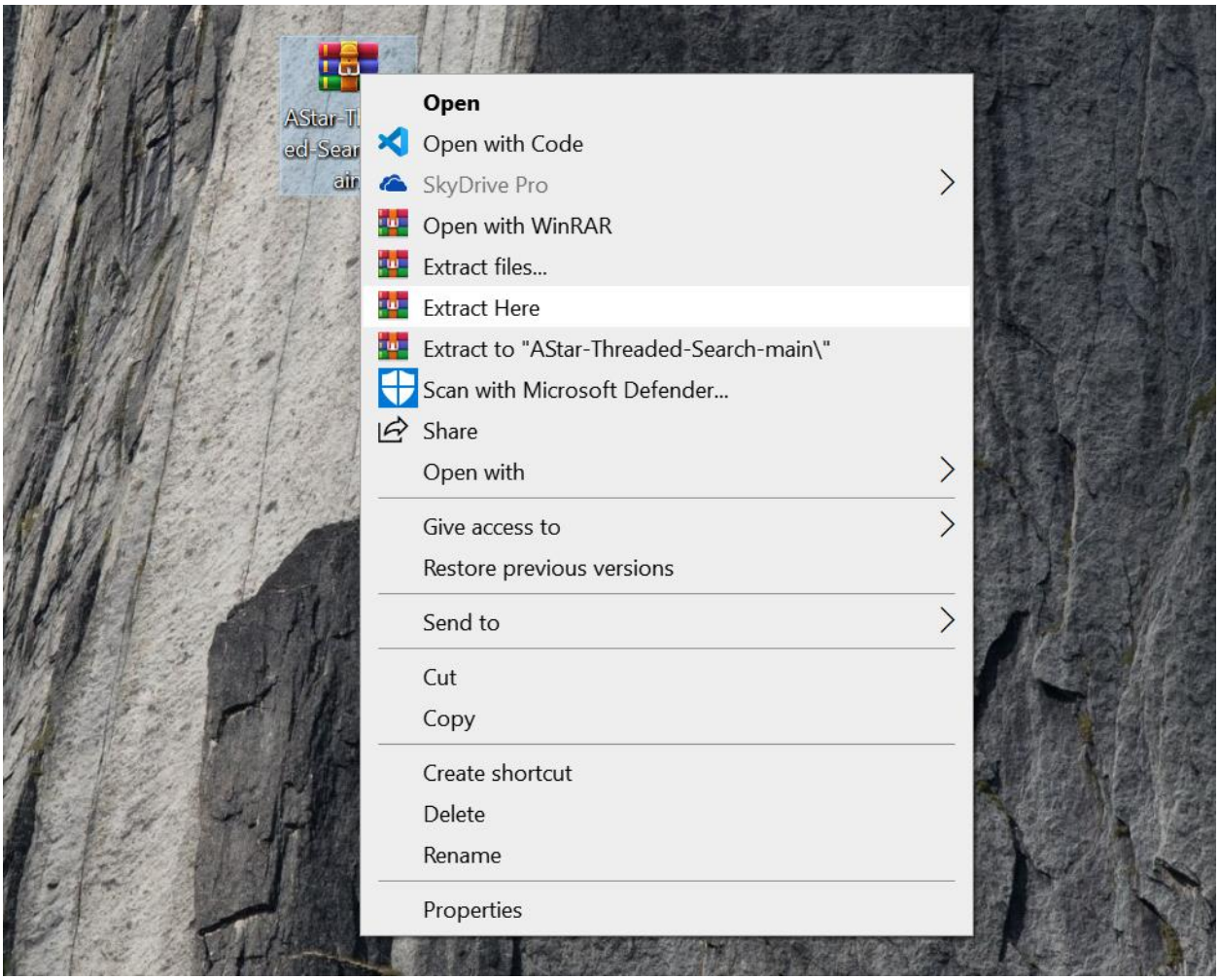
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

@sunnyallana →/workspaces/AStar-Threaded-Search/projectFiles (main) $ make all
g++ -o single src/singleThreaded.cpp -lpthread
g++ -o multi src/multiThreaded.cpp -lpthread
@sunnyallana →/workspaces/AStar-Threaded-Search/projectFiles (main) $ make run_multi
g++ -o multi src/multiThreaded.cpp -lpthread
./multi
Minimum Cost Path: EE_BLOCK -> CS_BLOCK -> MULTIPURPOSE_BLOCK -> SPORTS_ROOM
Cost: 41
@sunnyallana →/workspaces/AStar-Threaded-Search/projectFiles (main) $
```

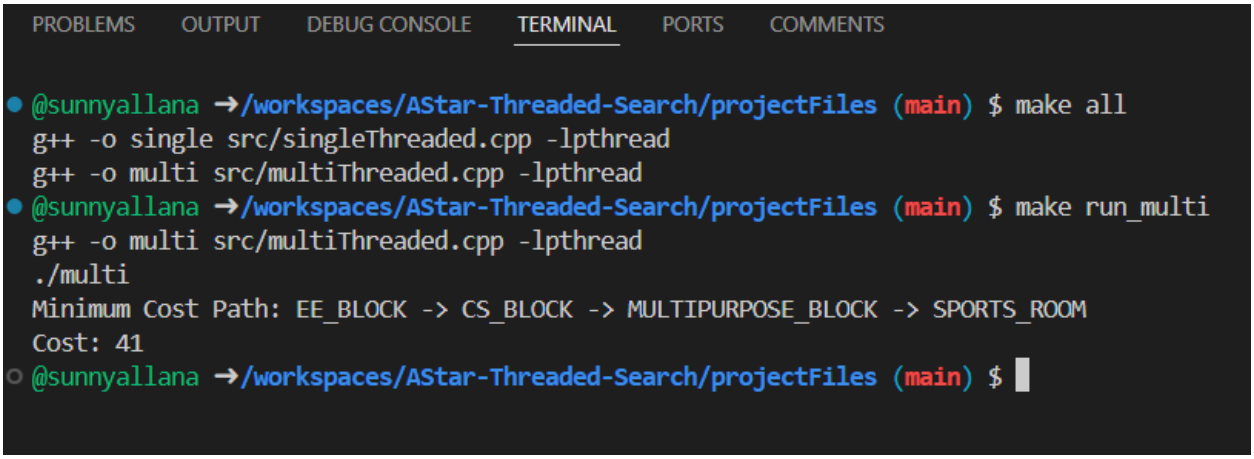
- **Cloning Repository without Git**:
  - Visit AStarThreadedSearch
  - Click on Code and then click on Download Zip



- Unzip the downloaded file



- Open the terminal or command prompt on your system
- Navigate to the projectFiles directory of the unzipped AStarThreadedSearch-main
- Execute make all and make run_multi to see output of multithreaded.cpp or make run_single to see output of singleThreaded.cpp



CONCLUSION

In conclusion, the implementation of the multithreaded A* search algorithm has proven to be a significant advancement in optimizing pathfinding efficiency. By applying this algorithm to the navigation problem from the EE Block to the Sports Room within our university, we have demonstrated a notable reduction in search time while ensuring the discovery of an optimal path.

Google Drive: Click Here

GitHub: Click Here