Project

Multi-Stage Game Strategy Solver

Objective:
The goal of this project is to develop an computer-based strategy solver for multi-stage decision-making games. Students will implement various algorithms to evaluate optimal strategies, predict outcomes, and optimize decision-making in games like chess, tic-tac-toe, or a custom-defined turn-based game.

Problem Statement:

Many strategic games like chess, checkers require players to think multiple steps ahead to make optimal decisions. This project involves developing a program that can evaluate different possible game states and determine the best moves using various algorithmic approaches.

Students will implement key algorithms like **Minimax, Alpha-Beta Pruning, and Dynamic Programming** to create a robust game that can efficiently handle large decision trees and improve its performance.

Project Tasks & Mark Distribution:

1. Game Representation & State Management (15%)

- Define a structured representation of the game board.
- Implement a method to generate all possible moves for a given state.
- Ensure the game state updates correctly after each move.

2. Minimax Algorithm Implementation (25%)

- Implement the Minimax algorithm to evaluate all possible future game states.
- Develop a scoring function to assess the favorability of a given state.
- Test and verify correctness by playing against a basic AI or human player.

3. Alpha-Beta Pruning Optimization (20%)

- Optimize the Minimax algorithm using Alpha-Beta Pruning.
- Measure and compare execution time before and after pruning.
- Validate improved efficiency in large game trees.

4. Dynamic Programming for Sub-Problems (15%)

- Identify repetitive calculations in the game tree.
- Implement memoization or bottom-up approaches to store results.
- Improve the efficiency of decision-making using stored computations.

5. Difficulty Levels & Adaptive Learning (15%)

- Implement multiple difficulty levels (Easy, Medium, Hard) based on depth of search.
- Allow the program to adjust its strategy based on past moves and opponent behaviour.

6. Report & Code Documentation (10%)

- Write a detailed report explaining the implemented algorithms and design choices.
- Provide well-commented code with explanations for key functions.
- Discuss test results, performance improvements, and possible extensions.

Deliverables:

1. **Source Code** with a structured implementation of the game solver.
2. **Final Report** detailing the approach, algorithms used, and performance analysis.
3. **Demonstration Video** (if required) showcasing the program playing against a human opponent.

Evaluation Criteria:

- Correctness of the implemented algorithms.
- Efficiency and performance of program's decision-making.
- Code quality, readability, and documentation.
- Creativity in implementing adaptive strategies.

Bonus Marks: (Up to 5%)

- If students implement additional enhancements such as Machine Learning-based AI, Neural Networks, or Reinforcement Learning.

Conclusion:

By the end of this project, students will have hands-on experience in **game theory, algorithm optimization, and AI development**. This will help them understand real-world applications of decision-making algorithms in complex scenarios.

Project

Railway Network Optimization System

Objective:
The goal of this project is to develop an optimized railway network system that efficiently manages routes, minimizes travel time, and ensures maximum train flow between stations. Students will implement various graph algorithms to enhance route planning, minimize costs, and optimize rail traffic flow.

Problem Statement:

Railway networks are vast and complex, requiring optimal path selection and efficient train scheduling. The challenge is to find the shortest travel routes, reduce the number of active tracks, and maximize train flow between stations. This project involves applying **graph algorithms** to improve railway operations, ensuring faster and more efficient train management.

Students will implement key algorithms like **Dijkstra's Algorithm, Minimum Spanning Tree (MST), and Ford-Fulkerson's Max Flow Algorithm** to enhance the efficiency of the railway network.

Project Tasks & Mark Distribution:

1. Railway Network Representation (15%)

- Model the railway network as a **graph** where stations are nodes and tracks are edges.
- Implement an **adjacency matrix/list** to represent the connections efficiently.
- Ensure the system handles real-world constraints such as **one-way and bidirectional tracks**.

2. Shortest Path Calculation (25%)

- Implement **Dijkstra's Algorithm** to find the shortest route between any two stations.
- Compare execution times for different data structures (Adjacency Matrix vs List).
- Optimize the algorithm for large networks with **multiple route requests**.

3. Minimum Spanning Tree for Active Track Minimization (20%)

- Implement **Prim's or Kruskal's Algorithm** to determine the minimum set of tracks required for connectivity.
- Analyze the trade-off between active track minimization and redundancy in the network.
- Ensure the system supports scalability when new stations are added.

4. Maximum Train Flow Optimization (15%)

- Implement **Ford-Fulkerson Algorithm** to find the maximum number of trains that can flow between two stations simultaneously.
- Analyze and optimize bottlenecks in the network.
- Adjust the system for handling multiple peak-time schedules.

5. Dynamic Route Adjustment for Delays (15%)

- Implement a mechanism to adjust routes dynamically when a station/track is temporarily closed.
- Use **Bellman-Ford Algorithm** to handle negative delays (such as train rescheduling).
- Ensure efficiency by precomputing alternative routes.

6. Report & Code Documentation (10%)

- Write a detailed report explaining the implemented algorithms and design choices.
- Provide well-commented code with explanations for key functions.
- Discuss test results, performance improvements, and possible extensions.

Deliverables:

1. **Source Code** with a structured implementation of the railway optimization system.
2. **Final Report** detailing the approach, algorithms used, and performance analysis.
3. **Demonstration Video** (if required) showcasing the system handling different scenarios.

Evaluation Criteria:

- Correctness of the implemented algorithms.
- Efficiency and performance of the railway optimization system.
- Code quality, readability, and documentation.
- Scalability and adaptability of the system to handle dynamic changes.

Bonus Marks: (Up to 5%)

- If students implement additional enhancements such as real-time train scheduling, passenger congestion analysis, or predictive maintenance features.

Conclusion:

By the end of this project, students will gain hands-on experience in **graph algorithms, optimization techniques, and real-world transportation system management**. This will help them understand practical applications of algorithmic solutions in large-scale networks.

Project

Dynamic Memory Allocator (Heap Management System)

Objective:
The goal of this project is to develop an efficient dynamic memory allocation system that manages heap memory efficiently by minimizing fragmentation and optimizing allocation and deallocation processes. Students will implement various memory management algorithms to ensure efficient space utilization.

Problem Statement:

Operating systems and applications require efficient memory allocation mechanisms to optimize performance and reduce wastage. The challenge is to dynamically allocate and deallocate memory blocks efficiently while minimizing fragmentation. This project involves implementing **dynamic memory allocation strategies** to manage heap memory in an optimized way.

Students will implement key algorithms like **First-Fit, Best-Fit, Next-Fit, and Buddy Memory Allocation** to enhance memory management efficiency.

Project Tasks & Mark Distribution:

1. Heap Memory Representation (15%)

- Design a structure to represent the heap memory.
- Implement linked list or array-based representation for managing free and allocated blocks.
- Ensure efficient initialization and handling of memory blocks.

2. First-Fit, Best-Fit, and Next-Fit Allocation (25%)

- Implement **First-Fit** to allocate the first available memory block.
- Implement **Best-Fit** to allocate the smallest available block that fits the request.
- Implement **Next-Fit** to continue searching for memory from the last allocated position.
- Compare the performance of these strategies in terms of speed and fragmentation.

3. Buddy Memory Allocation (20%)

- Implement **Buddy System Allocation** to split and merge memory blocks dynamically.
- Ensure efficient memory splitting and coalescing for better memory utilization.
- Compare performance with other allocation techniques.

4. Memory Deallocation and Garbage Collection (15%)

- Implement a mechanism to free allocated memory blocks.
- Handle memory fragmentation and merging of adjacent free blocks.
- Simulate garbage collection to optimize memory usage.

5. Performance Analysis and Fragmentation Reduction (15%)

- Measure external and internal fragmentation for each allocation strategy.
- Optimize memory allocation to reduce fragmentation and improve performance.
- Analyze time complexity and efficiency of different methods.

6. Report & Code Documentation (10%)

- Write a detailed report explaining the implemented algorithms and design choices.
- Provide well-commented code with explanations for key functions.
- Discuss test results, performance improvements, and possible extensions.

Deliverables:

1. **Source Code** with a structured implementation of the dynamic memory allocator.
2. **Final Report** detailing the approach, algorithms used, and performance analysis.
3. **Demonstration Video** (if required) showcasing memory allocation and deallocation.

Evaluation Criteria:

- Correctness of the implemented memory allocation algorithms.
- Efficiency and performance of the memory management system.
- Code quality, readability, and documentation.
- Ability to handle dynamic allocation requests effectively.

Bonus Marks: (Up to 5%)

- If students implement additional enhancements such as **paging-based allocation, defragmentation techniques, or memory compaction strategies**.

Conclusion:

By the end of this project, students will gain hands-on experience in **memory management, dynamic allocation strategies, and fragmentation reduction techniques**. This will help them understand the importance of efficient memory usage in operating systems and embedded systems.

Project
Parallel Job Scheduling System

Total Marks: 100 (+10% Bonus)

---

1. Introduction
In modern computing systems, efficiently scheduling parallel jobs across multiple workers (e.g., CPU cores, servers) is critical for optimizing resource utilization and performance. This project challenges students to design and implement a parallel job scheduling system that handles dynamically arriving jobs with varying priorities, durations, and resource requirements. The system must maximize throughput, minimize waiting time, and ensure fairness.

---

2. Problem Statement

Design a parallel job scheduler that:

- Accepts jobs with **arrival time**, **duration**, **priority**, and **required workers**.
- Assigns jobs to a fixed number of workers (e.g., 4 workers).
- Implements **multiple scheduling algorithms** for comparison.
- Ensures thread-safe execution and avoids resource contention.
- Evaluates performance using metrics like average waiting time and throughput.

---

3. Tasks and Marks Distribution

Task 1: Problem Analysis and Literature Review (10%)

- Research parallel scheduling algorithms (e.g., Round Robin, SJF, Priority-based).
- Identify challenges in concurrency, starvation, and scalability.
- Submit a 2-page summary of key findings.

Task 2: System Design (15%)

- Define components: Job Queue, Worker Pool, Dispatcher, and Metrics Collector.
- Design data structures for jobs (e.g., priority queues) and workers.
- Create UML diagrams or flowcharts for the scheduler's workflow.

Task 3: Scheduler Implementation (30%)

- Implement **3 scheduling algorithms** (e.g., FCFS, SJF, Priority-based).

- Use threading/multiprocessing to simulate parallel workers.
- Handle synchronization (e.g., locks, semaphores) to prevent race conditions.

Task 4: Testing and Evaluation (25%)

- Generate test datasets (varying job sizes/priorities).
- Compare algorithms using metrics (waiting time, throughput).
- Identify scenarios where each algorithm performs best.

Task 5: Documentation and Report (20%)

- Write a report explaining design choices, challenges, and results.
- Include code documentation (comments, README).

Task 6: Bonus Features (10% Extra)

- Dynamic worker scaling.
- Implement advanced algorithms (e.g., Multilevel Feedback Queue).
- Visualization of job execution timelines.

---

4. Deliverables

1. **Code**: Well-structured, commented source code.
2. **Report**: PDF detailing design, results, and analysis.
3. **Presentation** (Optional): 5-minute demo video or slide

---

5. Evaluation Criteria

Task	Marks	Key Evaluation Points
Task 1	10%	Depth of research, clarity of summary
Task 2	15%	Design coherence, UML/flowchart quality
Task 3	30%	Correctness, concurrency handling, algorithm diversity
Task 4	25%	Test coverage, metric analysis, critical evaluation
Task 5	10%	Report quality, code documentation
Bonus	+10%	Complexity of added features

---

6. Conclusion: By the end of this project, students will gain hands-on experience in **Parallel job sequencing, how systems are designed**. This will help them understand the importance of system designing and operating system.

Other projects

5. Custom File Compression & Decompression System

- Algorithms Used: Huffman Coding, Run-Length Encoding, Burrows-Wheeler Transform
- Task: Implement a file compression system that reduces file size using efficient encoding techniques and decompresses without data loss.

6. Graph-Based Task Dependency Manager

- Algorithms Used: Topological Sorting, DFS/BFS, Strongly Connected Components (Tarjan's Algorithm)
- Task: Simulate a task dependency scheduler that determines execution order for dependent tasks, such as project planning.

7. Secure Data Transmission Using Cryptographic Algorithms

- Algorithms Used: RSA Algorithm, Diffie-Hellman Key Exchange, Caesar Cipher
- Task: Implement a secure data encryption and decryption system using classical cryptographic techniques.

8. Airline Ticket Pricing and Seat Allocation System

- Algorithms Used: Dynamic Programming (Knapsack for Seat Pricing), Greedy Algorithm (Seat Allocation)
- Task: Optimize ticket pricing and seat allocation based on availability and revenue maximization.

9. Custom Database Indexing System

- Algorithms Used: B-Trees, Hashing, AVL Trees
- Task: Implement an efficient indexing system for searching and storing large datasets in a custom-built database.

10. Metro Train Route Optimization

- Algorithms Used: Floyd-Warshall (All-Pairs Shortest Path), Minimum Spanning Tree (Kruskal/Prim), Bellman-Ford (Negative Cycles)
- Task: Develop an optimized metro train system that finds the fastest routes and minimizes infrastructure costs.

10. Warehouse Storage and Retrieval System

- Algorithms Used: 0/1 Knapsack (Storage Optimization), Dijkstra's (Shortest Retrieval Path), Sorting Algorithms
- Task: Optimize item placement in a warehouse for minimal retrieval time and efficient space utilization.