Lab Task 6

Beam Search and Simulated Annealing

Task 1: Factory Floor Optimization with Beam Search

A factory has multiple workstations, each capable of handling different tasks. Your goal is to assign tasks to workstations in a way that minimizes the total time required to complete all tasks (makespan). Since the number of possible assignments is large, you will use Local Beam Search to efficiently explore the best task assignments.

Input:

- A list of N tasks, each with a specific duration. Example: tasks = [10, 15, 5, 8, 12, 7, 14, 6]
- A set of M workstations. Example: M = 3
- Beam width (number of best states to keep): Example: Beam Width = 3

Output:

- A schedule showing which task is assigned to which workstation.
- The total time for each workstation.
- A **Gantt chart** visualizing task assignments over time.

Constraints:

1. Each workstation can only process one task at a time.
2. The goal is to minimize the makespan (the maximum load across all workstations).
3. The heuristic function should evaluate states based on makespan.
4. Keep only the k-best states at each step (k = beam width).

Write a Python program that implements the Beam search algorithm to solve the problem.

Heuristic Function (h(n))

Use the makespan (minimizes the total time required to complete all tasks) as a heuristic

Task 2: Optimizing University Course Scheduling with Simulated Annealing

A university needs to schedule courses in a way that minimizes conflicts and balances classroom usage. Courses must be assigned to specific classrooms and time slots while ensuring no student or professor has conflicting classes. Since scheduling is an NP-hard problem, we will use Simulated Annealing to search for an optimal solution.

Input:

- A list of `N` courses.
- A set of `M` classrooms.
- `T` available time slots.
- Course constraints:
  - Some courses **cannot** be scheduled at the same time.
  - Some courses require **specific rooms** (e.g., labs).
  - Each professor can only teach **one course per slot**.

Output:

- A **schedule** that assigns each course to a **classroom and time slot**.
- A **conflict-free** timetable.
- A **visualization** of the schedule.

Constraints:

1. **No conflicts**:
   - No professor can be assigned to multiple courses at the same time.
   - No student can have overlapping classes.
2. **Classroom capacity** must not be exceeded.
3. The algorithm should **minimize empty time slots** (improve schedule efficiency).
4. **Simulated Annealing** will be used to optimize the schedule.
5. **Cooling schedule:** Start with a high temperature and **gradually reduce it** using a factor (e.g., `0.98`).

Heuristic Function ($h(n)$)

$h(n)$ = Number of **conflicts in the schedule**

Example Input:

```
courses = ["CS101", "MATH201", "PHYS301", "ENG102", "BIO104"]

time_slots = 6

classrooms = 3

professors = {"CS101": "Dr. Gilles De Rais", "MATH201": "Dr. Gilgamesh",
"PHYS301": "Dr. Dunban", ...}

students = {"CS101": ["S1", "S2"], "MATH201": ["S3", "S4"], ...}
```

Example Output :

```
CS101 → Room 2 at Time Slot 1
MATH201 → Room 1 at Time Slot 3
PHYS301 → Room 3 at Time Slot 2
…

Conflicts remaining: 0
```

I also need a visualization of the timetable