CS202, Spring 2025
Homework 2 - Heaps

Due: 23:59, 23/03/2025

Before you start your homework, please read the following instructions carefully:

FAILURE TO FULFIL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- Make sure to abide by the Honor Code for Assignments.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as `studentID_name_surname_hw2.zip`.
- Your ZIP archive should contain only the following files:
  - Your `.cpp` and `.h` files, as well as a file named `PackageDelivery.cpp`, which does not include the `main()` function, along with `PackageDelivery.h`, a **README** (explanation of your code).
  - Do not forget to put your name, student id, and section number in all of these files. Add a header (see below) to the beginning of each file:

```
/**
Author : Name & Surname
ID: 12345678
Section : 1
Homework : 2
Description : description of your code
*/
```

  - Do not add unnecessary files like the auxiliary files generated from your preferred IDE.
  - Please do not use Turkish letters or spaces in your file and folder names.
- Your code must be compiled.
- Your code must be complete & well-structured, and you must include comments to explain the functions as well.
- **You ARE NOT ALLOWED to use any data structure or algorithm-related function from the C++ standard template library (STL) or any other external libraries.**

- We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using the examples in the paper. We recommend you write your own driver files to make extra tests. However, you **MUST NOT** submit these test codes (we will use our own test code).
- We will test your code using the Google Test library. Therefore, the output message for each operation **MUST** match the format shown in the output of the example code. Additionally, do not submit a `main()` function in your `.cpp` file.
- Your code must run on the `dijkstra.cs.bilkent.edu.tr` server.
- For any questions related to the homework, contact your

  TA: `hesam.matinpouya@bilkent.edu.tr`.
- However, do not ask simple questions whose answers are obvious, as I have tried to write the whole scenarios and cases and explain them in the paper.

LATE SUBMISSION: Late submissions are not allowed and will receive a ZERO grade.

Drone Fleet Management Simulation

1 Introduction

In modern logistics, fleet management systems ensure that products are delivered on time and optimized. A delivery drone fleet is no exception. Managing a fleet of drones involves challenges such as optimizing task assignments based on drone capabilities and delivery priorities. Additionally, the system must minimize the average task completion time while efficiently utilizing the drones' limited battery life. This homework focuses on simulating a drone delivery system that manages a fleet of drones tasked with delivering packages across a city. The drones start from a central hub and must pick up and deliver packages based on priority and proximity. Each drone has limited battery life and can only complete delivery if it has sufficient energy and the package is within its operational capabilities.

2 System Overview and Key Components

The system simulates the operation of a fleet of drones delivering packages to various locations in a city. Tasks are assigned based on package priority, the drone's proximity to the destination, and the drone's available battery life. If a drone cannot complete a task due to insufficient battery, it is temporarily removed from service using a cooldown mechanism, allowing other drones a chance to complete the task.

Drones

Each drone is characterized by:

- **ID:** A unique identifier.
- **Battery Life:** The maximum energy available before recharging is required.
- **Speed:** The drone's base speed (meters per second).

In our simulation, drones are prioritized by their remaining battery life; if battery levels are equal, the drone with the lower ID is selected. All drones start from the hub located at $(0, 0)$. If a drone goes to cool down process, it will be taken to the hub by a external carrier.

Packages

Each package to be delivered has:

- **ID:** A unique identifier.
- **Destination:** Coordinates (x, y) where it must be delivered.
- **Priority:** A numerical value indicating delivery urgency (higher means more urgent).
- **Weight:** The package weight, affecting the drone's effective speed.

Tasks

A task involves a drone delivering a package from the hub to its destination. Tasks are prioritized by package urgency and estimated completion time. Once a task is complete, the drone becomes available for another assignment.

Cooldown Mechanism

If a drone fails to complete a task (due to insufficient battery), it is placed in a cooldown period (measured in discrete time steps or ticks) before it can be assigned a new task. This prevents repeatedly assigning the same underperforming drone.

3 Task Execution and Battery Considerations

Once assigned to a task, a drone travels toward the package's destination. However, its effective speed is adjusted based on two factors:

1. **Package Weight:** Heavier packages reduce the drone's speed.
2. **Remaining Battery Life:** As battery levels drop, the drone's performance degrades.

Effective Speed Calculation

The effective speed is calculated using:

$$\text{Effective Speed} = \text{Base Speed} \times \left(1 - \text{WEIGHT\_FACTOR} \times \frac{\text{package weight}}{\text{MAX\_PACKAGE\_WEIGHT}}\right)$$

which is then multiplied by

$$\left(1 - \text{BATTERY\_FACTOR} \times \left(1 - \frac{\text{battery life}}{\text{max battery}}\right)\right).$$

The travel time is computed as:

$$\text{Time Required} = \frac{\text{Distance}}{\text{Effective Speed}}$$

If the required time exceeds the remaining battery, the drone cannot complete the task and is placed into a cooldown period.

Tip: All calculated floating point numbers (Effective speed, Distance, Required time) should cast to one decimal precision.

Constants Used

- **MAX_PACKAGE_WEIGHT** is 10.
- **WEIGHT_FACTOR** is 0.2.
- **BATTERY_FACTOR** is 0.3.
- **COOLDOWN_PERIOD** is 5 tick.
- **MAX_BATTERY** is 300.

4 Task Assignment, Prioritization, and Optimization

Tasks are encapsulated in a `Task` class and sorted by package priority (and, when equal, by estimated completion time). Drones are sorted based on their remaining battery life (with lower IDs used as tie-breakers).

Time is divided into discrete steps (ticks). At each tick:

- **Cooldown Update:** Assume that the battery life amount would be enough to handle the package delivery after cool down. Set battery life to 300.
- **Task Assignment:** The highest-priority task is assigned to the best available drone.
- **Feasibility Check:** The simulation calculates if the drone can complete the task based on its effective speed and remaining battery.
- **Execution or Cooldown:** If the drone completes the task, its battery is updated and it is reinserted into the available pool; if not, it enters a cooldown state.

This approach ensures that high-priority tasks are handled by drones capable of completing them, optimizing overall performance and reducing the average task completion time.

5 Simulation Input and Output

5.1 Input Example

Input data will be read from two input .txt files. The content structure of each .txt file is illustrated below.

```
// drone.txt //ID, battery life, speed
4
1 100 2.5
2 120 3.0
3 150 4.0
4 100 2.8
```

```
// package.txt //ID, (x,y),  Weight, Prority
4
1 150 50 5 60
2 70 80 8 30
3 200 100 4 15
4 200 100 6 90
```

5.2 Expected Output

The simulation output indicates which drones successfully complete their deliveries and which fail due to battery constraints.

```
Drone 3 Package 4 at tick 0 (delivery time: 77.1, battery life: 72.9)
Drone 2 Package 1 at tick 1 (delivery time: 71.8, battery life: 48.2)
Drone 1 Package 2 at tick 2 (delivery time: 66.4, battery life: 33.6)
Drone 4 Package 3 at tick 3 cool down
Drone 4 Package 3 at tick 8 (delivery time: 89.4, battery life: 210.6)
```

Tip: In the current simulation, when a package is assigned to a drone, it's immediately removed from the pending task list. So if the drone (like Drone 4 at tick 3) is found to have insufficient battery to complete the delivery, the drone will get back to the hub after cool down period it will assign again to the package.

Tip: You have to measure the distance by Euclidean distance formula.

Tip: You MUST use heap to implement (not sorting array).

Tip: In completed deliveries assume that always the new package assigned to the drone is in the drone current location. Drone doesn't need to get back to the hub to get package. You have just calculate the distance between the new position of the drone and the newly assigned package destination to deliver.

6 C++ Class for Package Delivery Simulation

Below is an example of a C++ class named `PackageDelivery` that encapsulates the simulation.

```
class PackageDelivery {
public:
    PackageDelivery ( const string droneFile, const string packageFile
        );
    // Simulation function prototype
    void simulation();
};
```

This class serves as a template for your simulation implementation in C++, where you will read input from `drone.txt` and `package.txt` and run your simulation accordingly.