Project 3

Spring 2025 CPSC 335 - Algorithm Engineering

Instructor: Prof. Sampson Akwafuo

Algorithm 1: The Spread of Fire in a Forest

Scenario:
Imagine you are a forest ranger monitoring a forest that has recently experienced a wildfire. The forest is represented by a grid, where each cell can either be:

- 0: an empty area (no trees),
- 1: a healthy tree,
- 2: a burned tree (representing a tree affected by the wildfire).

The management needs to determine how many days it will take for all healthy trees to burn down, considering that every day, any healthy tree that is adjacent (up, down, left, or right) to a burned tree will also burn down. You need to calculate the minimum number of days it will take for all healthy trees to burn, or return **-1,** if it is impossible for some trees to burn (i.e., if there are healthy trees that are not adjacent to any burned trees).

Details:

1. A healthy tree (represented by 1) will burn down after one day, if it is adjacent to a burned tree (represented by 2).
2. An empty area (represented by 0) does not affect the burning process.
3. If there are no burned trees initially, it is impossible for any healthy trees to burn, so return -1.

Task:

Design an algorithm to simulate this process and return the minimum number of days it will take for all healthy trees to burn down, or **-1** if it is not possible.

Example 1:

Input:

```
forest = [
  [2,1,1],
  [1,1,0],
  [0,1,1]
```

]

Output:

4

Explanation:

- At day 1: the healthy trees at (0,1) and (1,0) burn.
- At day 2: the healthy trees at (1,1) and (2,0) burn.
- At day 3: the healthy tree at (2,1) burns.
- At day 4: the healthy tree at (2,2) burns.

All healthy trees burn in 4 days.

Example 2:

Input:

```
forest = [
   [2,1,1],
   [0,1,1],
   [1,0,0]
]
```

Output:

-1

Explanation:

• In this case, the healthy tree at (2,0) cannot burn because there are no adjacent burned trees.

Example 3:

Input:

```
forest = [
   [0,2]
]
```

Output:

0

Explanation:

• In this case, there are no healthy trees, so no time is needed for burning.

Algorithm 2: Delivery Route Planning

Scenario:

Assume that you are a logistics manager working for a company that needs to deliver packages across a network of distribution centers. The distribution centers are connected by a series of delivery routes (similar to flights). You are given a list of available delivery routes between different distribution centers, each with a specific cost. Your goal is to determine the cheapest delivery route from a starting distribution center (src) to a destination center (dst), but the delivery process can involve a maximum of k intermediate stops (transfers).

Details:

1. You are given a list of delivery routes between various distribution centers. Each route specifies:

    - The starting distribution center ($fromi$),
    - The destination distribution center (toi),
    - The cost of the delivery ($pricei$).

2. Your task is to calculate the minimum cost for delivering a package from the starting distribution center (src) to the destination distribution center (dst). You can make at most k stopovers (transfers) along the way.
3. If there is no valid route that respects the stopover constraint, return **-1**.

Task:

Design an algorithm to calculate the cheapest route from src to dst with at most k stopovers. Return the total delivery cost or return **-1** if no route exists.

Example 1:

Input:

```
routes = [
   [0, 1, 100],
   [1, 2, 100],
   [0, 2, 500]
]
src = 0
```

dst = 2
k = 1

Output:

200

Explanation:

The cheapest route from distribution center 0 to center 2 with at most 1 stop is:

- From center 0 to center 1 with a cost of 100, then from center 1 to center 2 with a cost of 100.
- The total cost is 200.

Example 2:

Input:

routes = [
  [0, 1, 100],
  [1, 2, 100],
  [0, 2, 500]
]
src = 0
dst = 2
k = 0

Output:

500

Explanation:

• The only valid route with 0 stopovers is from center 0 to center 2, which costs 500.

Example 3:

Input:

routes = [
  [0, 1, 100],
  [1, 2, 100],
  [0, 2, 500]
]
src = 0
dst = 3

k = 1

Output:

-1

Explanation:

• There is no route from distribution center 0 to center 3, so the result is **-1**.

Grading Rubric

The suggested grading rubric is given below:

Algorithm (50 points each)

a. Clear and complete Pseudocode = 10 points
b. Mathematical analysis and correct Big *O* efficiency class = 5 points
c. Inclusion of a Readme file = 5 points
d. Well commented codes = 5 points
e. Successful compilation of codes= 15 points
f. Produces accurate results = 10 points

Algorithm 2 (50 points)

a. Clear and complete Pseudocode = 10 points
b. Mathematical analysis and correct Big *O* efficiency class = 5 points
c. Inclusion of a Readme file = 5 points
d. Well commented codes = 5 points
e. Successful compilation of codes= 15 points
f. Produces accurate results = 5 points

Submitting your code

Submit your files to the Project 3 Assignment on Canvas. It allows for multiple submissions. Your codes should be submitted in their executable extensions (.py or .cpp), and report in PDF. All files should be submitted **separately**. Do not zip or use .rar.

Ensure your submissions are your own works. Be advised that your submissions may be checked for plagiarism using automated tools. Do not plagiarize. As stated in the syllabus, a submission

that involves academic dishonesty will receive a 0% score on that assignment. A repeat offense will result in an "F" in the class and the incident will be reported to the Office of Student Conduct.

Deadline

The project deadline is **Sunday, April 6, by 11:59 pm** on Canvas.
Penalty for late submission (within 48 hours) is as stated in the syllabus. Projects submitted more than 48 hours after the deadline will not be accepted.