**CPSC 335 - Algorithm Engineering**  
**Project 3 Report**  
**Spring 2025**  
**Due: 04/11**

Wayne Muse  
88601211  
CSUF Email**:** [waynemuse@csu.fullerton.edu](mailto:waynemuse@csu.fullerton.edu)

Submission for Project 3

# Introduction

This project involves the implementation and analysis of three algorithms:

1. **The Spread of Fire in a Forest**
2. **Delivery Route Planning**

# **Algorithm 1: Spead of Fire**

## Problem Description

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.

## Pseudo Code

Function spread\_of\_fire(forest):

rows = len(forest)  
cols = len(forest[0])  
queue = deque()  
healthy\_trees = 0

for each cell (r, c) in forest:

if forest[r][c] == 2:

queue.append((r, c, 0))

else if forest[r][c] == 1:

healthy\_trees += 1

if healthy\_trees == 0:

return 0

if queue is empty:

return -1

while queue is not empty:

(r, c, d) = queue.popLeft()

for each direction in [up, down, left, right]:

nr = r + direction[0], nc = c + direction[1]

if valid and forest[nr][nc] == 1:

forest[nr][nc] = 2

healthy\_trees -= 1

queue.append((nr, nc, d + 1))

days = d + 1

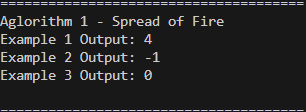
return days if healthy\_trees == 0 else -1

## Time Complexity Analysis

Initialize queue and count healthy trees: O(n\*m)  
BFS loop runs once for each tree: O(n\*m)  
4 directions checked per cell: O(4 × (n\*m))  
Total: O(n\*m)+O(n\*m)+4\*O(n\*m)

Thus, Big-O Efficiency: O(n\*m)

## Example Execution

****

# **Algorithm 2: Delivery Route**

## Problem Description

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).

* 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).
* 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.
* If there is no valid route that respects the stopover constraint, return -1.

## Pseudo Code

Function find\_cheapest\_delivery(n, routes, src, dst, k):

Initialize graph with n empty lists

for each route [u, v, price] in routes:

graph[u].append((v, price))

heap = [(0, src, 0)]

visited = {}

while heap is not empty:

(cost, node, stops) = heap.popMin()

if node == dst:

return cost

if stops > k or visited[node] <= stops:

continue

visited[node] = stops

for each (neighbor, price) in graph[node]:

heap.push((cost + price, neighbor, stops + 1))

return -1

## Time Complexity Analysis

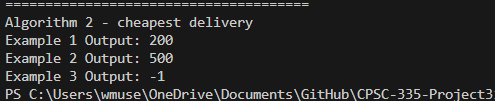
Graph construction: O(n)

push/pop operations: O(log n)

Total heap: O(n log n) + O(n)

Big O efficiency: O(n log n)

## Example Execution



# Project Components

The project submission includes the following files:

* **README.md** – Contains instructions for running the project.
* **main.py** – The main driver program to execute the algorithms.
* **alg1.py** – Implementation of the Spread of Fire
* **alg2.py** – Implementation of the Delivery
* **CPSC 335 Project3 Report** – project report