Project 3 Written Report

CPSC 335 - Algorithm Engineering

Spring 2025

Names:

Anthony Le | antlecsuf@csu.fullerton.edu

Ryan Nishikawa | ryannishikawa48@csu.fullerton.edu

Dylan Tran | dylanht341@csu.fullerton.edu

Jasmine Youssef | JasmineYoussef@csu.fullerton.edu

Algorithm 1: The Spread of Fire in a Forest

The Spread of Fire in a Forest problem is:

Input: a matrix of 2's, 1's, and 0's where 2 represents a burning tree, 2 represents a healthy tree, and 0 represents an empty area

Output: the number of days it takes for every healthy tree to burn down, or -1 if it is impossible for every healthy tree to burn down

Pseudocode:

```
dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}

def minDaysToBurn(forest):

  if forest is empty:

    return -1

  rows = number of rows in forest

  cols = number of columns in forest

  create queue to store coordinates of burning trees

  healthyTrees = 0

  for i from 0 to rows-1:
```

```
        for j from 0 to cols-1:
            if forest[i][j] == 2:
                push to queue
            if forest[i][j] == 1:
                increment healthyTrees
    if queue is empty:
        return -1
    days = 0
    burnedTrees = 0
while queue is not empty:
    size = number of elements in queue
    for i from 0 to size-1:
        [x, y] = front element of queue
        remove front element of queue
        for each [dx, dy] in dirs:
            newX = x + dx
            newY = y + dy
            if newX is between 0 and rows-1 && newY is between 0 and cols-1 && forest[newX][newY] == 1:
                set forest[newX][newY] equal to 2
                push {newX, newY} to queue
                increment burnedTrees
    if queue is not empty:
        increment days
if burnedTrees != healthyTrees:
    return -1
return days
```

Efficiency Analysis:

"for i from 0 to rows-1" and "for j from 0 to cols-1" run at most (rows × cols) times.

The while loop processes each cell only once. For each burning tree, it checks at most 4 neighboring cells, resulting in a constant amount of work per cell. Thus, the overall work done by the while loop is also proportional to (rows × cols).

Therefore, the efficiency class of this algorithm is O(rows × cols), which is linear in the size of the input forest.

Algorithm 2: Delivery Route Planning

Problem Description

You are given a set of delivery routes between distribution centers, each with a cost. The objective is to determine the cheapest route from a starting distribution center (src) to a destination center (dst) with at most k stopovers (intermediate transfers). Return the minimum delivery cost or -1 if no valid route exists.

Sample Input and Output

Example 1:

routes = [

[0, 1, 100],

[1, 2, 100],

[0, 2, 500]

]

src = 0

dst = 2

k = 1

Output: 200

Explanation: 0 -> 1 -> 2 costs 100 + 100 = 200

Example 2:

routes = [

[0, 1, 100],

[1, 2, 100],

[0, 2, 500]

]

src = 0

dst = 2

k = 0

Output: 500

Explanation: Only direct path 0 -> 2 is valid

Example 3:

routes = [

[0, 1, 100],

[1, 2, 100],

[0, 2, 500]

]

src = 0

dst = 3

k = 1

Output: -1

Explanation: No valid route to center 3

Pseudocode:

```
START
    INPUT routes, src, dst, k
    CREATE graph from routes
    INITIALIZE minHeap with (cost=0, node=src, stops=0)
    WHILE minHeap is not empty:
        POP (cost, current_node, stops) from minHeap
        IF current_node == dst:
            RETURN cost
        IF stops > k:
            CONTINUE
        FOR each neighbor of current_node in graph:
            PUSH (cost + edge_cost, neighbor, stops + 1) to minHeap
    RETURN -1
END
```

Mathematical Analysis and Efficiency

- **Time Complexity**:

- Let n be the number of centers and e be the number of routes.
- Graph construction: O(e)
- Min-heap operations: O(k * e * log n)

Space Complexity: O(n + e) for graph, O(n * k) for min-heap.

Efficiency Class: O(k * e * log n), efficient for sparse graphs and limited k.