Istanbul Technical University
Faculty of Computer and Informatics
Computer Engineering Department

BLG 336E
The LaTeX
Report

Analysis of Algorithms II, Project 2

Leminur Çelik - 150190085

April 18th, 2023

Contents

1 Description of Code

1.1 Pseudo-code

1.1.1 Graph Implementation

I implement the code which creates (n x n) adjacency matrix as a graph. If absolute difference of pount numbers in between two cities is less than or equal to the decided threshold times the mean of their pount numbers, then there is an edge between two vertices.

Algorithm 1 Set Edges

```
function SetEdges(*nodes, numberVertices)
    i ← 0
    while i < numberVertices do
        while j < numberVertices do
            if i ← j then
                graph[i][j] ← 0
            end if
            plow ← nodes[i].GET-POUNT − nodes[j].GET-POUNT()
            if plow ≤ ((nodes[i].GET-POUNT + nodes[j].GET-POUNT)/2)*threshold then
                adjMatrix[i][j] ← plow
                adjMatrix[j][i] ← plow
            end if
        end while
    end while
end function
```

1.1.2 Prim's Algorithm Implementation

I created a vector called visited with the number of vertices. If vertices are visited, the value of that vertex will be true in the visited vector. Moreover, I created a 2D vector called key whose number of rows is equal to the number of bakery and the number of columns is equal to the number of cities. Key vector holds the edge weight from a node. Every bakery has a vector called visited vertices. The branches they opened are added. Prim's algorithm is run step by step for each bakery to produce the list of branches.

Algorithm 2 Prim's Algorithm

```
function PrimsAlgorithm(*nodes, bakeries)
    visited ← vector of bool with number of vertices elements, initialized to false
    key ← 2D vector of int row with number of bakeries elements
    i ← 0
    while i ≠ bakeryNumber do
        Set visited vertices vector of a bakeries
        Set bakery indices as visited
        Set finish of bakeries as false
        Declare the size of the column of 2D vector called key
        i ← i + 1
    end while
    i ← 0
    while i ≤ numberVertices − 1 do
        checker ← 0
        j ← 0
        while j ≤ bakeryNumber do
            if that bakery cannot open a new branch continue then
                chekcer ← checker + 1
                index ← visited vertices vector's back value of bakeries
                u ← MaxKey(visited, index, key[j])
                if u ← −1 then
                    Set that bakery's finish as true
                    Continue the loop
                end if
                visited[u] ← true
                Put the value u in visited vertices vector of that bakery
            end if
            j ← j + 1
        end while
        if checker ← 0 then
            Exit the loop
        end if
    end while
end function
```

Algorithm 3 Find Maximum Key

```
function MaxKey(visited, index, key)
    maxIndex ← −1
    i ← 0
    while i < numberVertices do
        if graph[index][i] ≠ 0 and (visited[index] ← false) and key[i] < graph[index][i] then
            key[i] = graph[index][i]
        end if
        largest ← 0
        i ← 0
        while i < numberVertices do
            if key[i] > largest and visited[i] ← false then
                largest ← key[i]
                maxIndex ← i
            end if
        end while
    end while
    return maxIndex
end function
```

1.2 Time Complexity

The graph has n vertices, the time complexity to build an adjacency matrix is $O(n^2)$. Finding the adjacent vertices of selected vertex requires checking all elements in the row. This takes linear time O(n). Summing over all the n iterations, the total running time is $O(n^2)$. In the Prim Algorithm function, the for loop loops approximately n times and the inner for loop loops m times, where m is the number of bakeries resulting in $n * m$. In the Max Key function, the for loop loops n times and other for loop loops n times, so their sum is 2n. This means that time complexity of Max Key function is O(n). Since the number of bakery will be equal to at least 1 and at most cities, time complexity is O(n) in the best case and $O(n^2)$ in the worst case.

2 Relationship between Number of Cities and Memory Complexity

2.1 Space Complexity

The adjacency matrix of a graph requires $O(V^2)$ memory, where V is the number of vertices. The space complexity of Prim's Algorithm can be expressed as $O(V * B)$, where V is the number of vertices and B is the number of bakeries, because when finding the greater edge weight from the relationship of a vertex to other vertices, a 2D vector is required that holds the edge weights of each bakery with other cities. Increasing the number of cities means increasing the number of vertices. Creating an adjacency matrix is proportional to the square of the number of vertices. Since the number of bakery will be equal to at least 1 and at most cities, memory complexity is O(V) in the best case and $O(V^2)$ in the worst case.

2.2 Run-Time

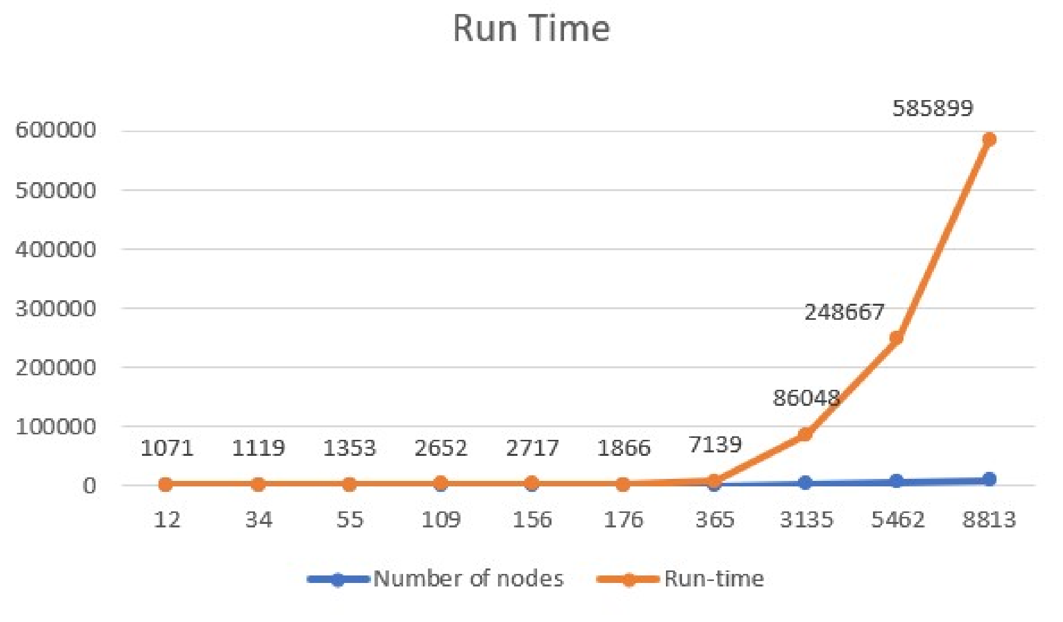

Figure 1: The graph for run-time in microseconds for all cases