Desenho de Algoritmos
Programming Project I

An Analysis Tool for Railway Network Management

2LEIC03

(9 de Abril de 2023)

Ntsay Zacaria up202008863
Mario Branco up202008219
Treasure Vanilson Langa up202008876

INTRODUCTION

In this first programming project we were asked to implement an analysis tool that can support the management team of a railway network to make informed decisions about how to best allocate its resources, both financial as well as physical (e.g., trains).

This tool, which uses a realistic data set, allows management to decide on aspects such as how to best assign selected trains to lines, how to maximize its revenue when multiple levels of service are offered, or even, identify more sensitive sections of its network to failures so as to anticipate service disruption or at least to mitigate its nefarious effects.

IMPLEMENTATION

As the project objective states, the group needed to successfully use the greedy algorithmic approach in the context of maximum-flow problems in order to correctly perform the task that were required.

The first step was then to define the area to be tested and whether we would use the parser available or would we create the files ourselves containing the necessary information. We quickly realized that it would be more practical and instructive to create our own test files that are essentially the same that were given but in a more scaled down version.
Effectively, having the files ready and a well-defined idea of the area to be tested, easily, we have created the necessary functions to read text files, also creating the nodes and edges of our graph.

ALGORITHMS

- Maximum Number of trains that can arrive simultaneously at a given station - Edmonds-Karp per each station: O(V^2 E^2)
- Maximum Number of Trains between two trains - Edmonds-Karp: O(V E^2);
- Maximum Number of Trains between two trains (Reduced Connectivity) - we allow the user to choose reduction factor therefore the subgraph is generated dynamically - Edmonds-Karp: O(V E^2);

ALGORITHMS

- Edmonds-Karp pseudocode:

```
algorithm EdmondsKarp is
    input:
        graph   (graph[v] should be the list of edges coming out of vertex v in the
                 original graph and their corresponding constructed reverse edges
                 which are used for push-back flow.
                 Each edge should have a capacity, flow, source and sink as parameters,
                 as well as a pointer to the reverse edge.)
        s       (Source vertex)
        t       (Sink vertex)
    output:
        flow    (Value of maximum flow)

    flow := 0   (Initialize flow to zero)
```

```
    repeat
        (Run a breadth-first search (bfs) to find the shortest s-t path.
         We use 'pred' to store the edge taken to get to each vertex,
         so we can recover the path afterwards)
        q := queue()
        q.push(s)
        pred := array(graph.length)
        while not empty(q)
            cur := q.pop()
            for Edge e in graph[cur] do
                if pred[e.t] = null and e.t ≠ s and e.cap > e.flow then
                    pred[e.t] := e
                    q.push(e.t)
    if not (pred[t] = null) then
        (We found an augmenting path.
         See how much flow we can send)
        df := ∞
        for (e := pred[t]; e ≠ null; e := pred[e.s]) do
            df := min(df, e.cap - e.flow)
        (And update edges by that amount)
        for (e := pred[t]; e ≠ null; e := pred[e.s]) do
            e.flow  := e.flow + df
            e.rev.flow := e.rev.flow - df
        flow := flow + df

until pred[t] = null  (i.e., until no augmenting path was found)
return flow
```

ALGORITHMS

- Pairs of Stations that require the most amount of trains when taking full advantage of the network capacity - Floyd-Warshall (modified): O(V^3)

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

ALGORITHMS

- Top-k Transportation Needs using K as an optimization for a partial sort - Greedy Algorithm: O(V^2)

```
cnt = 0
tmp[K] = {} //desending array
for each pair(key, pld) in
    if(cnt < K)
        insert_sort(tmp[], pair)
    else
        if(pair.key > tmp[k].key)
            insert_sort(tmp[], pair)
```

ALGORITHMS

- Maximum Number of trains with minimum cost between two stations - Dijkstra (with PQ) + Edmonds Karp: O(V E^2)

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                           // Initialization
3
4      create vertex priority queue Q
5
6      for each vertex v in Graph.Vertices:
7          if v ≠ source
8              dist[v] ← INFINITY                 // Unknown distance from source to v
9              prev[v] ← UNDEFINED                // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
--
13
14     while Q is not empty:                      // The main loop
15         u ← Q.extract_min()                    // Remove and return best vertex
16         for each neighbor v of u:              // Go through all v neighbors of u
17             alt ← dist[u] + Graph.Edges(u, v)
18             if alt < dist[v]:
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist, prev
```