Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

Concepção e análise de algoritmos

Smart garbage collection

Assignment 4

Alunos:
Bernardo Barbosa, up201503477, up201503477@fe.up.pt
Duarte Carvalho, up201503661, up201503661@fe.up.pt
João Sá, up201506252, up201506252@fe.up.pt

8 de Abril, 2018

1 Abstract

The aim of this research is to study a smart garbage collection system, by assigning efficient routes to garbage trucks that start in a truck station and must collect garbage from containers when they need do be collected and then take it to a waste management station. The containers are equipped with weight systems that communicate their filled percentage, meaning that there is no need for periodic trips to collect garbage, instead the trucks leave the main station only when a container has reached a certain filled percentage.

2 Keywords

Smart, Garbage System, Graph Theory, Optimization, Routing, Path Finding

Conteúdo

3 Introduction

The objective of this program is to implement a garbage management system for a city equipped with garbage volume sensitive containers that communicate their filled percentage to the program, so that the the garbage trucks can be routed to collect the garbage with an efficient path. For this we made two iterations, listed and explained in next sections.

1º Iteration: Homogeneous fleet with limit capacity.

In this iteration, certain trucks can only collect certain containers, which means that when a container signals the station that it is ready for collection, the program must check the available trucks and select the adequate ones for collection, this implies that is possible that a container can't be collected as soon as it signals the station and there is the possibility of assigning multiple routes to different trucks.

2º Iteration: Heterogeneous fleet with limit capacity.

Finally, we considered the vehicles capacity, considering garbage type also, replicating a real world situation, meaning that now there's the possibility of a truck not being able to collect all containers that are in need to be collected, and it might be necessary to send more than one truck to collect one single type of garbage.

4 The problem

4.1 Input data and symbols

- Gi(N, E) - Directed Graph, containing Nodes (N) and Edges (E).
  - Ni - node with index i which represent the city elements.
    - Info - information about the node, that can be:
      - Location - node coordinates.
      - Volume (V) and Filled Percentage (F) - for Containers (C).
      - List of Trucks (T) - in case of Garages(Ga).
    - Dist (d) - auxiliary field used by dijkstra.
    - Path (p) - pointer to next node in path.
  - Ei - edge with index i that connect two nodes.
    - destination (Ni) - node which the edge connects to.
    - Weight (W) - weight associated to the edge.
- Container (C).
  - Volume (V) - volume of a container.
  - Filled Percentage (F) - percentage volume (V) occupied with garbage.
- Station (S) - place where trucks throw collected garbage.
- Garage (Ga) - trucks starting point.
  - Trucks (T) - list of trucks ready for duty.
- Truck (T) - vehicle that collect garbage.
  - Volume (V).
  - Garbage type (Gt) - the type of garbage that the vehicle can carry.
- Garbage type (Gt) - type of garbage.
  - paper.
  - glass.
  - plastic.
  - generic.

4.2 Restrictions

- $\forall C \in |C|, V(C[i]) > 0$
- $\forall C \in |C|, F(C[i]) \geq 0 \wedge F(C[i]) \leq 100$
- $\forall T \in |T|, i, x \in [1, |Ga|], \forall Ga[i] \in |Ga|, T \in Ga[i] \rightarrow T \notin Ga[x]$
- $\forall T \in |T|, i, x \in [1, |T|], g, y \in |Gt|, T[i](Gt) = g \rightarrow T[i](Gt) \neq y$

5 Garbage Collect Program

5.1 Data structures

In order to represent a generic graph, defined in "Graph.h", two template classes, Node and Edge, were implemented, in "Node.h" and "Edge.h" respectably, which hold just the necessary information for graph visualization, optimal path search calculation and graph connectivity evaluation.

- In node class, we implemented:
  - edges - vector of outgoing edges to other nodes
  - dist - distance to the node where the search has started.
  - path - pointer to the node that get us closer to the search solution.
  - visited - bool value to evaluate if node has been already used by dfs or bfs.
  - indegree - field used for topsort
  - processing - bool value to evaluate if node is being used by isDAG
- In edge class, we have:
  - dest - pointer to another node, which is the edge destination.
  - weight - weight associated with this edge, necessary for path search.

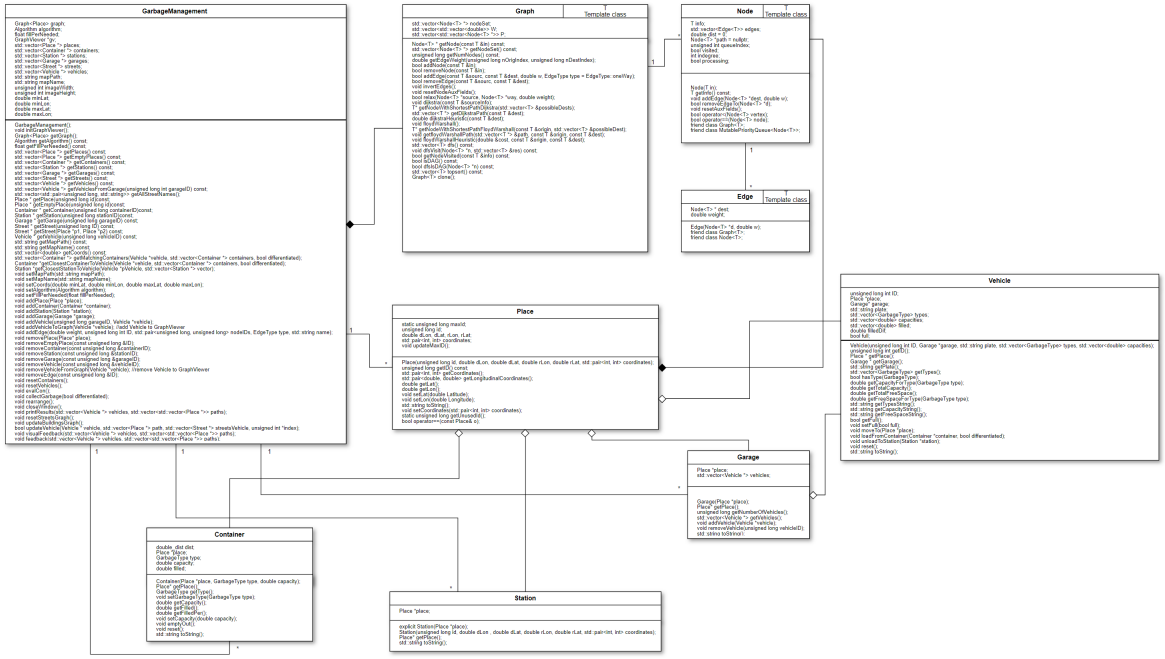
Furthermore, in graph class exists an unordered node set that is composed by pointers to all of the map nodes, containers, stations and garages. An unordered node set implies that the nodes have an id, but it's easier to do operations, like insert, remove or perform a search, on it.

5.2 Visualizing graph problem

For node template class, we used a set of classes to represent the different problem elements:

- Place - it's a normal node that simply holds a location. The next three classes extend this class.
  - coordinates - pair of x and y Cartesian coordinates.
  - dLon and dLat - place longitude and latitude in degrees.
  - rLon and rLat - place longitude and latitude in radians, used to calculate distance between two nodes.
  - id - its the node ID for identification
- Container - it represents the city containers that the garbage trucks must collect.
  - type - type of garbage that it's currently holding.
  - capacity - the volume it can hold.
  - filled - the percentage of volume it's currently occupied.
- Garage - it's the vehicles starting point, and were they are stored.
  - vehicles - vector of vehicles.

- Station - It's were the trucks throw the trash away.
- Vehicle - Collects garbage from containers and takes it to a station.
  - garage - pointer for the garage it belongs to.
  - types - vector with the types of garbage it can collect.
  - capacities - vector with the volume of each type of garbage it can collect.
  - filled - vector with the quantity of garbage it is currently transporting.



5.3 Garbage Collection Algorithm

The user will have previously decided which path finding algorithm to run. First, we place the vehicles in their respective garages and fill the containers with a randomly generated percentage of garbage.
After selecting which vehicles are empty and which containers are ready for pick-up, the program then enters a loop and randomly chooses and removes a vehicle from the set. Next, the algorithm calls the auxiliary function **getClosestContainerToVehicle** that will match the filled containers with the vehicle if the garbage is differentiated besides also checking if the vehicle has enough free space for a certain container.
The program then moves the vehicle to the container place and loads the garbage onto it, removing the container from the filled set, after doing this it will go into a loop where it calls the auxiliary function **getClosestContainerToVehicle** and repeating these steps until it returns a null pointer.
When the auxiliary function **getClosestContainerToVehicle** returns null pointer, the program will set the vehicle as full and call the **getClosestStationToVehicle** function, if it doesn't return null pointer, it will move the vehicle to the station and unload it, storing the vehicle and the path it took and repeating the loop until there are no more filled containers or empty vehicles.

5.3.1 Dijkstra Algortihm

Dijkstra's algorithm is an algorithm used to calculate the shortest distance between two nodes in a graph. It starts by setting every distance of every vertex to infinity except the starting vertex which is 0. The current vertex is inserted into a priority queue. The algorithm then gets the set of adjacent vertexes and iterates through them assigning their distance and putting them onto the queue. Once all of the adjacent vertexes are considered, the initial vertex is now "visited" and is extracted from the queue. If the target vertex has been reached, the algorithm is finished and the path is saved in each of the vertex, otherwise, repeat for the closest vertex (top of the queue).In the program's version of the algorithm, a vertex is passed as an argument and for the whole graph, the algorithm defines the shortest path from that vertex to every other one. The original algorithm (without queues) has a time complexity of $O(|N|^2)$ and the program's algorithm's complexity is of $O(|E| * \log(|N|))$ if $|E| > |N|$ and a space complexity of $O(|N|^2)$.

5.3.2 Floyd Warshall's Algorithm

Floyd Warshall's algorithm is an algorithm used to calculate the shortest distance between every node in a graph. It starts by creating two matrices of the size of $|N|^2$ and filling them, one with the edge's weights and the other with -1. It then iterates through the matrices and for each cell of a matrix the value of the edge's weight is updated if a lower one to the current vertex is found. The second matrix simply holds the current vertex that it is being iterated.

In the program's version, this algorithm is accompanied by a custom getPath function (getfloydWarshallPath) which returns the shortest path between two vertexes.

This algorithm has a time complexity of $O(|N|^3)$ and a spatial complexity of $O(|N|^2)$ due to both matrices.

6 Connectivity analysis

The program allows the user to make a connectivity analysis of the current selected graph.
It assumes a graph is directed, and tells the user if a graph is directed acyclic, which are the strongly connected components in the graph thanks to Kosaraju's algorithm and finally it prints if the graph itself is strongly connected using Depth-first search.

7 Difficulties and Conclusions

7.1 Difficulties

One of the difficulties that we had during the work was the lack of precision of the maps obtained in OpenStreetMap, because the lack of some nodes and some connections did not allow us to test the smart garbage collection in a greater radius in the several maps implemented.

Another difficulty found, even if overcome, was the time required to test the Floyd-Warshall algorithm due to its pre-processing complexity.

7.2 Conclusions

In addition to helping us better understand each algorithm, this project gave us an idea of how many things work around us and how these things are designed in small and large cities, such as smart garbage collection.