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, up201708805, up201708805@fe.up.pt

8 de Abril, 2018

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

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, take it to a waste management station and then return to the starting truck 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 when a certain number of containers as reached a certain filled percentage.

2 Keywords

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

Conteúdo

1 Abstract **1**

2 Keywords **1**

3 Introduction **3**

4 The problem **4**
4.1 Input data and symbols 4
4.2 Restrictions 5

5 Solution description **6**
5.1 Data structures 6
5.2 Visualizing graph problem 6

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 three iterations, listed and explained in next sections.

1º Iteration: Trucks collect all types of garbage with unlimited capacity.

In the first stage of out research we considered that the trucks have no limit in terms of capacity and can collect any type of garbage, which means that a single truck can go over all the containers that need to be collected.

Notice that a path between all interest points (truck station, containers and waste management station) must exist. So a two way connection between each element must exist. Besides, as previously mentioned, the trucks just leave the station when a certain number of containers as reached a certain filled percentage because there is a cost associated with each trip to collect garbage.

2º Iteration: Heterogeneous fleet with no limit capacity.

In this iteration, certain trucks can only collect certain containers, which means that when a containers 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.

3º 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(Ni, Ei) - Directed Graph, containing Nodes (Ni) and Edges (Ei).
  - Ni - nodes 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 - edges 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 Solution description

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
  - maxId -
- Container - it represents the city containers that the garbage trucks must collect.
  - dist -
  - 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 -

The run method from the SocketListener class creates a Handler instance everytime it receives a message and adds it to the thread pool.

```
//run = listen
public void run() {
    boolean listen = true;
    System.out.println("Listening on: " + this.mcAddress + " port: "
        + this.mcPort + ".");

    while (listen) {
        byte[] buf = new byte[Constants.HEADER_MAXSIZE +
            Constants.CHUNK_MAXSIZE];
        DatagramPacket packet = new DatagramPacket(buf, buf.length);

        try {
            this.mcSocket.receive(packet);
            Message msgReceived =
                Message.decodeMessage(packet.getData());

            if(!msgReceived.getHeader().getSenderId().equals(Peer.getInstance().getId())){
                Handler handler = createHandler(msgReceived);
                scheduler.execute(new Thread(handler));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    closeSocket();
}

private Handler createHandler(Message msg) {
    return new Handler(msg);
}
```

Now for a snippet of the Peer code that initializes the multicast threads.

```
public Peer(Version version, String id, InetAddress mcAddr, int mcPort,
            InetAddress mdbAddr, int mdbPort, InetAddress mdrAddr,
                int mdrPort) throws IOException,
                ClassNotFoundException {
    this.scheduler = new ScheduledThreadPoolExecutor(4); //3 threads
        for listening, 1 for protocol

    this.socket = new MulticastSocket();

    this.version = version;
    this.id = id;

    // load deletion log
    this.deletionLog = DeletionLog.load(id);

    this.disk = this.loadDisk();

    this.mcListener = new MCListener(mcAddr, mcPort);
    this.mdbListener = new MDBListener(mdbAddr, mdbPort);
    this.mdrListener = new MDRListener(mdrAddr, mdrPort);
```

```
        scheduler.execute(new Thread(mcListener));
        scheduler.execute(new Thread(mdbListener));
        scheduler.execute(new Thread(mdrListener));
```