

Laboratory Assignment

Subtask II



LABORATORY GROUP: A1 – 07

COMPONENTS:

AMANDA SÁNCHEZ GARCÍA

FERNANDO VELASCO ALBA

GITHUB REPOSITORY: A1-07

01/11/2017

## INDEX

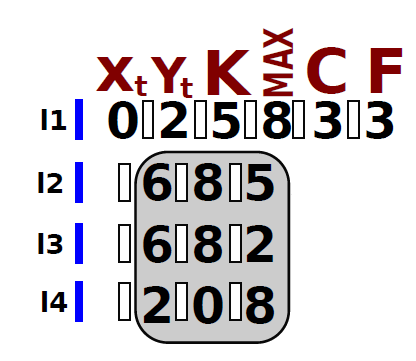
1. EXPLANATION OF THE PROBLEM
2. OUR IMPLEMENTATION OF THE PROBLEM
   1. PROGRAMMING LANGUAGE CHOSEN
   2. CLASSES IMPLEMENTED
   3. MAIN METHODS
3. TESTING

### explaNATION OF THE PROBLEM

The main goal of this laboratory assignment consists of defining, designing and developing an agent program to find the sequence of actions to be taken by a tractor to ensure that all the sand in a field is evenly distributed on the ground. So, all the boxes will have an equal amount of sand K.

The first things to be done are:

* Implement an internal representation of the field.
* Create a field.
* Reading and writing a field from/to a file.
* Generate all possible actions from a field with the tractor in the (Xy, Yt) box.
* Get a new field after applying an action to a given one.

It is necessary to take into account the format of the file where the provided information is going to be:

When we have all this implemented, we need to create two data structures: the node and the frontier of the tree.

In the node, we need:

* Access to the parent
* State
* Cost
* Action
* Depth
* Value (a random value)

We need an ordered list to create the frontier of the tree, where the nodes are sorted from lower to higher value of “Value”.

With all this, we need to define the state space of the problem, the initial state and a goal function.

### OUR IMPLEMENTATION OF THE PROBLEM

### pROGRAMMING LANGUAGE CHOSEN

### The programming language that we are going to use is Java. We have decided to use it because it is the programming language that we know best, also Java is a very complete language so we will have available all the data structures we are going to need.

### CLASSES IMPLEMENTED

### The field is going to be represented as a bidimensional array which boundaries are defined through the file. The file defines the position (x,y) of the tractor in the field, the desired quantity of sand in each square (k), the maximum sand that can be placed in a square, and the number of columns and rows of the field.

### We have implemented seven classes:

### **State.** Class where the current state is defined. That is, the field is defined with its number of rows and columns, and the maximum and allowed sand in each square. It is also defined the position of the tractor (X, Y) and the current sand.

### In this class are found the methods to move the tractor and sand along the field, and generate the successors of the tree.

### **FileHandler.** This class is used to manage the file, that is, to have the possibility to read it and write on it. It checks that the file is read correctly, taking into account the format, throwing the corresponding errors.

### **InputExceptions.** Class to check that the format of the file is the correct one (checking that there are only positive integers, the correct use of blank spaces or other error).

### **Node.** Here every node of the tree is defined. Every node has cost, depth, value and action, and they have associated and state an are referenced to a parent.

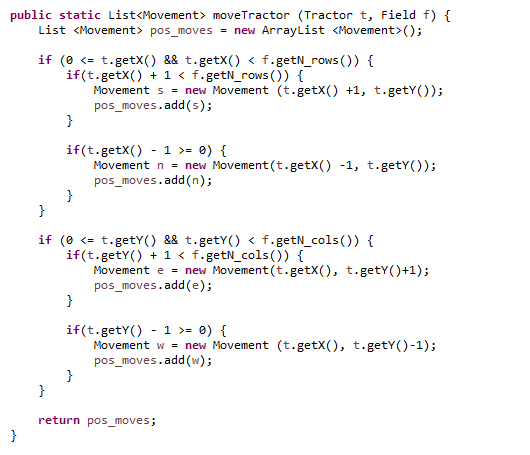
### **Action.** Class where an action is defined. The main attributes of this class are the next movement of the tractor and the quantity of sand to be redistributed in each possible move (north, east, west and south).

### **Movement.** Where every movement of the tractor is stored.

### **Problem.** Main class of the program. Here the field is created and the file is read, and invokes methods of other classes. Also, it creates the queue of the tree and apply every action.

### Main methods

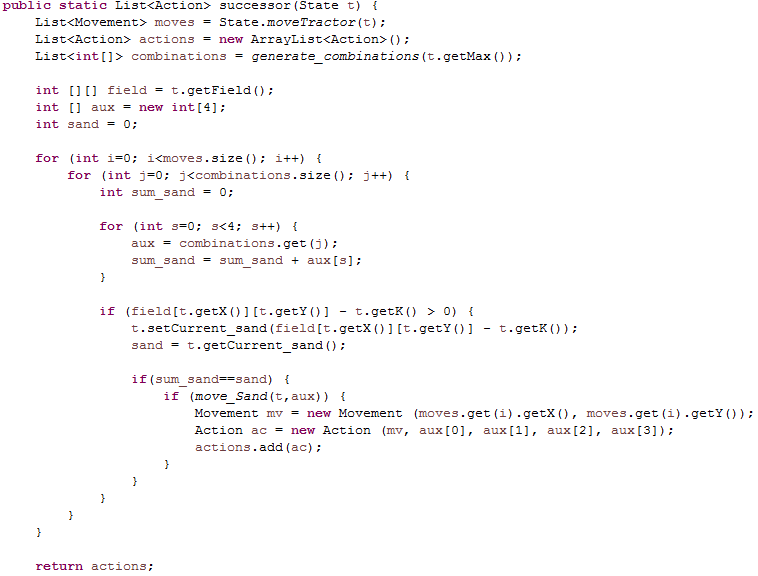
### Move tractor



The method moveTractor is located in the class State. The main goal of the method is to generate a list with all the possible movements the tractor can perform. The next movement of the tractor depends on the current position of it. This is, the tractor can only moved within the field. If the next position is out of the boundaries of the field, it will not be considered as a movement.

Diagonal movements are not considered, so the tractor can only moved to the right or to the left, and to the north or to the south.

* succesor



Another important method is in State class and it is called successor This method is crucial because the distribution of the sand is decided through the actions considered in this method.

In order to generate all the possible actions, we have to take into account the number of possible combinations. For example, if the maximum quantity of sand for each box is 8 we will use 8888 possible combinations, checking later that a higher number will not be used.

Let’s explain this graphically:

|  |  |  |  |
| --- | --- | --- | --- |
| N | S | E | W |
| 8 | 8 | 8 | 8 |

The maximum quantity of sand to be moved for each position is that one, but we have to assure that the next situation is not going to happen:

|  |  |  |  |
| --- | --- | --- | --- |
| N | S | E | W |
| 8 | 8 | 7 | 9 |

As we have 8888 possible combinations, this combination can take place, so we have to control that 9 quantities of sand are not going to be moved.

We have to check the possible combinations of sand movements for each new position. To select the correct combinations, we have to compare the total amount of sand to be redistributed. This figure has to be equal to the quantity of sand removed from the initial position of the tractor. The sand removed from this position is the initial amount of sand minus the desired amount of sand in each box, called K.

If this condition takes place, we generate a new action. An action is composed of the next movement of the tractor and the sand for each adjacent box.

Once the action is created, we add it to the list of all possible actions.

### frontier queuehttps://i.gyazo.com/0ea0b212d6c60eda25e31e28898576e0.png https://i.gyazo.com/4111fac11d1ff0a9eedc9addd67ef06c.png

This method is located in the Problem class. A structure called frontier is created to be used in the tree. Following the search tree algorithm, the nodes are placed in the frontier in ascending order of their value (which is randomly generated). In this method it is also computed the time needed to insert every node in the frontier (maximum, minimum and average time).

There is another method similar to this one but implements the frontier as a list instead of a queue.

### IS goal

### https://i.gyazo.com/45f01e280450c699ec79721c3ff54b3e.png

Method that also belongs to the Problem class. Checks whether the node is a goal state or not (true or false).

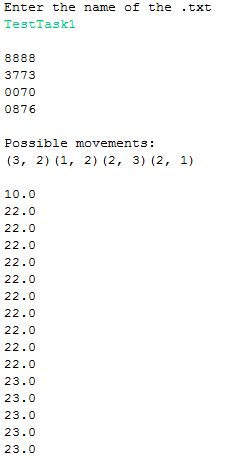
### apply action

### https://i.gyazo.com/6ebc4742debe470241f80243926c8b22.png

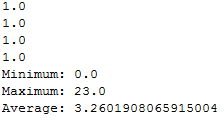
Another method from the Problem class, in which every action of the state space is applied. That is, moving the quantity of sand allowed (looking at the restrictions of the problem) to the north, south, east or west.

### Testing

When we test what is the output of the program this is what we get:



…



The name of the file is asked in the beginning, and then it prints the field, the possible movements, the time of every node entered in the frontier and in the end, appears the minimum, maximum and average time.