PRG (E.T.S. d'Enginyeria Informàtica)
Academic Year 2014-2015

Lab activity 1. The Recursion as an Strategy for Solving Problems: Hanoi Towers

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Contents

1 Context and previous work **1**

2 Basic concepts about recursion **1**

3 The problem **2**

4 Activities for solving the problem **3**
4.1 Activity 1: determining the problem structure and their parameters 3
4.2 Activity 2: case analysis 4
4.3 Activity 3: implementation in Java of the case analysis and code validation 5
4.4 Activity 4: estimation of the execution cost 6

1 Context and previous work

In the academic framework, this lab activity is the first one corresponding to Unit 1 (Recursion). The main objectives of the present activity are two: (1) reinforcing and practising the concepts introduced on recursive strategy and design, and (2) making the student realise of the utility of thinking recursive strategies for solving complex problems.

In order to reach these objectives during the lab activity, you have to face and solve different stages of the design of a recursive method in Java that solves the "Towers of Hanoi" problem, a classical example in Computer Science. To solve the proposed activities as well as helping and guiding you in the resolution, you have available a graphical interface which was designed for this activity. The interface is available in `PoliformaT` (`Recursos - Laboratorio - Práctica 1`). Once you finish the design of Hanoi, as a self-assessment of learning, you can find in `PoliformaT PRG: Exámenes` some extra exercises related to the work done during the lab session. The latter test is voluntary and does not affect the course grade.

It is recommended to carefully read the present report in order to obtain the maximum profit of the lab session, as well as working with the interface to solve the presented problem.

2 Basic concepts about recursion

Terms such as induction, recursion or inference are used in several fields, from informal to scientific environments, to name a problem solving strategy that consists of, given a finite set of solutions for a

problem in simple cases, obtain an hypothesis for obtaining the general solution for the problem (i.e., a method for solving that can be applied in any general case).

Human beings used naturally this inductive strategy, as can be seen in a simple problem such as: if you have the series of values 2, 4, 8, 16, 32, 64, what will be the next value after 64? After checking the values, your answer will be (surely) 128, and you will propose as following terms of the series values such as 256, 512, 1024, ...

You will easily propose what is the relation between each consecutive value of the series: $\forall i \geq 2$, the ith value is obtained from the $(i-1)$th value by using $u_i = 2 \cdot u_{i-1}$, i.e., the ith term is 2^i. You inferred this formula from the given values and you applied it to calculate the value following 64, i.e., 128, and consequently you found that the hidden function that gives the value of the series is the **exponential** 2^i, since $2^i = 2^1 \cdot 2^{i-1}$.

Check: you can check your understanding on the concept by completing with at least two values the following numeric series, as well as indicating the general expression you applied to obtain the ith value from the previous values and the hidden function. Finally, decide what type of method (recursive or iterative) would be more appropriate to obtain the ith value of each function.

- 1, 1, 2, 6, 24
- 1, 3, 7, 15, 31, 63, 127
- 0, 1, 3, 6, 10, 15
- 1, 1, 2, 3, 5, 8

Other interesting problems that are inductively solved are language learning in childhood, several mathematical functions (such as factorial and Fibonacci) and sets (such as the natural numbers). You have to get familiar with the induction principle for demonstrating properties. The following examples allow us to detail more exactly the nature of the inductive method:

1. The problem solution hypothesis for the general case is **explicitly** expressed in terms of the solution(s) for the same problem for simpler case(s)
2. The problem solution hypothesis for the general case should be validated or demonstrated by using a test process (to make the hypothesis a proved fact); this feature gives the name to the process (inductive process)
3. The most determining advantage is that the solution for a complex case of the problem is more easily expressed in inductive terms (based on the solution for simpler cases) than in deductive terms (by describing the steps that can be used to obtain the solution)

3 The problem

With the objective of making you propose recursive strategies in the future for solving problems, and their appropriateness with respect to their counterpart iterative strategies, in this lab activity you should design a Java method to solve the Hanoi Towers problem:

> There are three towers (`origin`, `destination`, and `temporary`) and a set of discs of different sizes. Initially, all the disks are staked in the `origin` tower in decremental diameter size (the largest disc is on the bottom). The problem consists of moving all the discs to the `destination` tower by following the next rules: only one disc can be moved at the same time and no discs of larger diameter can be on top of a disc of smaller diameter.

4 Activities for solving the problem

To recursively solve the problem and express the solution as a recursive Java method, you have to face and solve (with the help of the graphical interface) the four activities that are described in this section. Each activity covers a stage of the recursive design you should implement. In order to use the graphical interface, you previously have to:

1. Open a console and create the directory `prg` in your `$HOME`:

   ```
   cd $HOME
   mkdir prg
   ```

2. Make the subdirectory for the current session (`~/prg/pract1`).

   ```
   cd $HOME/prg
   mkdir pract1
   ```

3. Copy the executable code of the GUI, `HanoiGui.jar`, available in PoliformaT (Recursos - Laboratorio - Práctica 1) into `$HOME/prg/pract1`.

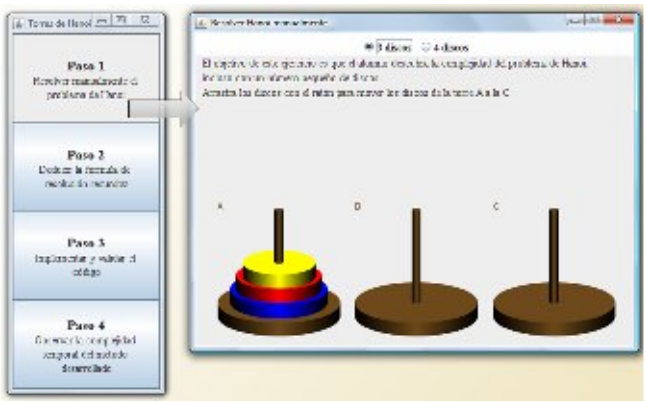
4. Execute the command `java -jar HanoiGui.jar` from the Linux console, and then the following graphical menu will appear on the screen:

   You have to click in each of their steps (*Pasos*) from 1 to 4 in order to solve the activities that are presented below; do not change to the next activity until you solve the current one.

4.1 Activity 1: determining the problem structure and their parameters

The first activity consists of solving the first step (*Paso 1*) of the graphical interface, which appears in the following figure: manually solve the Hanoi Towers problem for 3 and 4 discs.



This manual work will allow you to analyse the problem features and, consequently, make you choose between a recursive or an iterative solution for the problem. You should also realise about the function, amount and types of the formal parameters of the method that you will design, as well as the data type of the return value and, above from all, which parameter of the method is the one which makes easier or harder to solve the problem.

After covering this first stage of the recursive design you should be able to answer the following questions with no problems:

Questions on Activity 1 in order to design a Java method `hanoi()` for solving the problem:

- How many formal parameters should it have? Which are their data types? What names would you use to identify the parameters easily? Which parameter establishes the difficulty degree for the Hanoi Towers problem?
- What is the data type of the result of this problem? Fill in the following table to fix these concepts.

Nature	Return data type	Name	Parameter list
`static`		`hanoi()`	

- How will you invoke the `hanoi()` method for solving the instances of the problem stated in the first step of the GUI?

  Remind the profile of the method you defined in the previous question.

- Although you could describe the sequence of movements you made to solve the problem for three and four discs, would you be able to propose an iterative procedure for solving the problem whatever be the number of discs? Would you be able to propose it for 1, 2, 3, and 4 discs? You can suppose that you have available a set of instructions where the instruction `moveDisc()` is included; `moveDisc()` moves the top disc in the tower given as `origin` tower to the one given as `destination` tower (preventing violations of the Hanoi Towers rules).

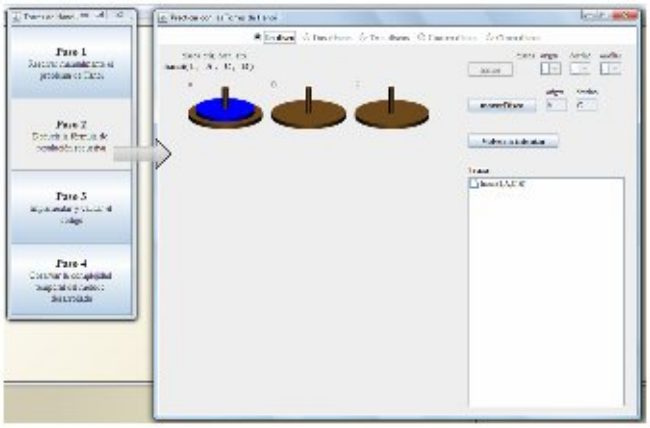
Maybe you will not be able to solve the last question, but it is normal. In fact, one of the conclusions that must be extracted from this activity is that a step-by-step (iterative) description for solving the Hanoi Towers problem is too hard given the set of rules. You can observe this difficulty in the following proposal of iterative solution for the Hanoi Towers problem:

```
int h = disks;
String a = origin, c = destination, b = temporary;
String k;
int no = 1;
while ( no != 0 ) {
    while ( h != 2 ) { h--; k = b; b = c; c = k; no = 2 * no; }
    moveDisc(a, b); moveDisc(a, c); moveDisc(b, c);
    while ( (no % 2) != 0 ) { h++; k = a; a = b; b = k; no = no/2; }
    if ( no != 0 ) {
        moveDisc(a, b);
        k = a; a = c; c = b; b = k; no++;
    }
}
```

4.2 Activity 2: case analysis

After solving the first activity you should think about changing your initial strategy: instead of facing the general resolution of the problem, the most natural alternative is to face the solution of the Hanoi Towers problem from the simplest case (only one disc must be moved) to the most complex case, taking into account that the complexity is defined by the number of discs to be moved. With this new vision you should be able to formulate the solution of one case in terms of the solution(s) of previous (simpler) cases, i.e., with one disc less. Actually, this is the main advantage of the inductive process with respect to the deductive process when trying to solve a complex enough problem.

Activity 2 tries to make you think recursively: you must solve the problem for 1, 2, 3, 4, and 5 discs by using the game of step 2 (*Paso 2*) of the graphical interface, which is shown in the following figure:



In this game you can play on the level i ($1 \leq i \leq 4$) only if you solved the game for level $i-1$. Only two commands are available: (1) `moveDisc()`, which moves the disc with lowest diameter from the tower given as origin to the one give as destination, and (2) `hanoi()`, that solves the problem in the level $i-1$ assuming you provide the correct values for towers `origin`, `destination`, and `temporary`. In order to make you learn quickly, the following questions are proposed:

- Independently of the amount of discs i ($2 \leq i \leq 5$) that are in the origin tower, which is the action you have to perform with the $i-1$ discs that are on the top of the disc of largest diameter (bottom disc) in order to move the $i-1$ discs to another tower? Which tower? You should express it by using one of the two available commands.
- When there is just one disc in the origin tower (the largest one). To which tower should it be moved? Express the movement by using one of the available commands.
- After having the largest disc on the correct place, how would you move the $i-1$ remaining discs on it in order to form the resulting Hanoi Tower? Express it again by using one of the two commands, and remember where the discs are placed, where they should be moved, and which tower is free in the current situation.

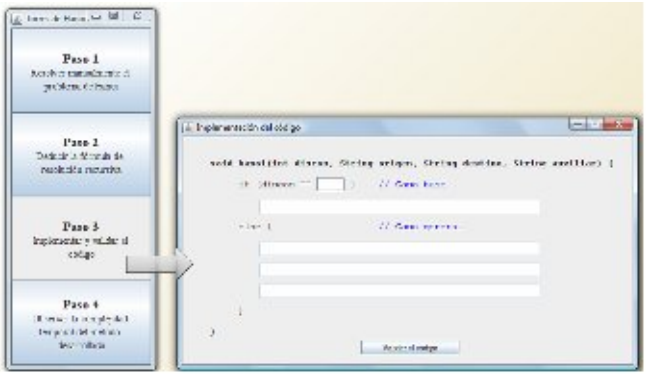
Level 5 must be solved in a reasonable amount of time, otherwise we recommend to you go back to activity 1. Anyway, the following questions will allow you to check if you understood the recursive formulation for solving the Hanoi Towers problem. Moreover, try to describe the trivial and general cases you found in the case analysis you used when playing, along with their associated instructions.

Activity 2 questions:

- Select successively the buttons of 5 and 4 discs of the game; what difference do you observe among the states of the towers A, B, and C? Is this difference the same when the number of discs varies between 4 and 3, 3 and 2, and 2 and 1? In case of positive answer, tell if the difference is congruent with the recursive strategy you defined to solve the problem.
- Repeat the same process of the previous question with the button that shows the number of discs, but watch what happens when, for each number of active discs, you click in the trace folder (`Traza`) which is on the rightmost lower corner of the window game. Is that congruent with respect to the recursive strategy you defined to solve the problem? Does this make you think that definitely an iterative solution for this problem is nearly impossible?

4.3 Activity 3: implementation in Java of the case analysis and code validation

Now take the third step (*Paso 3*) of the graphical interface, which is shown in the following figure, in order to express in Java the recursive strategy you defined in the activity 2. Validate your code by using the button that appears below the code. If the code is correct, you can informally check its correction by watching in the GUI an execution trace for 3 discs.

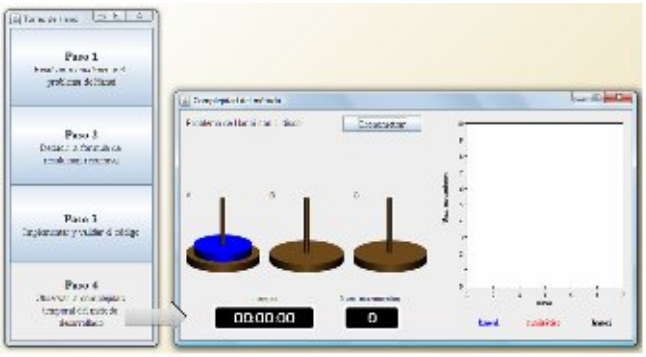


If you look carefully the trace for 3 discs, then you can answer the following questions about how the JVM uses the call stack:

- Which `moveDisc()` instruction is executed the first one, the one corresponding to the trivial case or to the general case? Argue your answer.
- How many call-stack registers (represented as *post-it*) must be stacked **before** executing this first movement? Is it equal to the number of calls that have been executed until that moment? Argue your answer.
- How many *post-it* need to be stacked **after** executing the first movement?
- Which `moveDisc()` instruction is executed the second one, the one corresponding to the trivial case or to the general case? Argue your answer and ask yourself again the two previous questions but in relation to this case.
- Which is the last movement that is executed, the one corresponding to the trivial case or to the general case? Argue your answer and return again to the second and third questions for this case.
- Based on your previous answers, could you demonstrate that the execution of the method finishes in a finite number of calls? You can use the concept of limiting function or the induction principle for the demonstration.

4.4 Activity 4: estimation of the execution cost

After checking that your model finishes its execution in a finite number of calls, this last activity allows you to establish how long will it take depending on the number of discs N you used in the first call. In other words, if the temporal cost function is proportional to N (linear), to N^2 (quadratic), ... You have to use the step 4 (*Paso 4*) of the interface, which is shown in the following figure, and annotate the number of movements that are used in each execution of the method. From the series of values you will obtain, you should induce the mathematical function that is associated to the *temporal complexity* of the method.



When you find the solution you will discover why the Hanoi Towers problem is known as *the end of the world problem* and you will be able to solve the following question:

> Says the legend that in Brahma's temple in Benares there exist three diamond needles that stack 64 gold discs. Restless, the priests move the discs from one needle to another, always obeying Brahma's law: no disc can be on a smaller disc and only one disc can be moved at a time. In the beginning, all the 64 discs where put on the first needle and formed a Brahma's tower. The legend says that when the smallest disc will be placed on the third tower and a new Brahma's tower is formed, the end of the world will come. If the legend is true and we suppose that the monks are fast enough to move a disc each 20 milliseconds, when will the world finish?