IS52038A Algorithms and Data Structures Term II Coursework part I (20%)
2015-2016

Deadline: Week 28, 16:00 Wednesday 2nd March 2016

You should review course topics on *algorithm analysis, linked lists, stacks and queues, trees, graphs, BST, heaps, recursions, sorting.*
You should read the assignments or questions carefully and pay particular attention to any submission instructions.
Note: programme source codes may be checked using anti-plagiarism programmes.

Assignment

Develop a game program in Java to simulate the movement of a Cheetah and Rabbit.

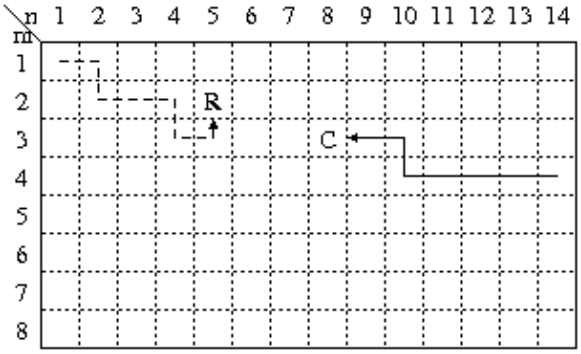

Figure 1: A display platform, C for cheetah and R for rabbit

To focus on the algorithmic ideas, we consider a simplified version of the problem as follows.
There are three main objects in the game, namely, the *platform*, *cheetah* and *rabbit*. As we can see from Figure 1, a platform consists of a two dimensional array of $m \times n$ positions, m for rows and n for columns. Each *position* is represented by a pair of indices (x, y), where x, y are integers representing the row index and column index respectively.
The Cheetah may be represented by a capital letter 'C' and Rabbit by 'R', which can move independently one step at a time. A *step* by the Cheetah or Rabbit is the smallest change of position, from the current position (x, y) to one of the 4 immediate neighbour positions (x', y') by moving in one of the 4 directions (Up, Down, Left, Right) (hereafter U, D, L, R, respectively). This can be denoted as $(x, y) \rightarrow (x', y')$.
A *walk* by the Cheetah or Rabbit consists of a number of steps. For example, in Figure 1, R is at position (2,5) and a walk from (1,1) consists of a sequence of changing positions: (1,1)→(1,2)→(2,2)→(2,3) →(2,4) →(3,4) →(3,5) →(2,5). This can be defined as a movement RDRRDRU. Similarly, C is at position (3,8) and a walk from (4,14) consists of another sequence of changing positions: (4,14) →(4,13) →(4,12) →(4,11) →(4,10) →(3,10) →(3,9) →(3,8). This can be defined as another movement LLLLULL.
Your tasks are:

1. Applying the 'divid and conquer' principles, define and write down a simplified version of at least 5 manageable subproblems.
2. Design appropriate data structures and algorithms for 3 most important visual functions of the game that you choose.
3. Analyse the time complexity of your algorithms. If Investigate the upperbound and lowerbound of the algorithms.
4. Write methods to implement the activities of the Cheetah and Rabbit following your own intuition and definitions.

5. Evaluate the programming experience and include up to 5 best screenshots to demonstrate the execution of your program.

While you are encouraged to be creative and stretch your programming ability, the following hints and examples are provided to stimulate your own imagination and keep your design and programming manageable. In fact, you may select and attempt a number of questions below if you find the original tasks difficult. However, you are expected to add certain details to the assignments during the algorithm design and implementation. Note that there may be dependencies or overlaps among the following questions.

Hints and examples

1. Define the entities of the three objects: the platform, Cheetah and Rabbit as precisely as possible. For example, you may decide that the display platform can be of any shape: square, round or triangle, or of different colour backgrounds, or that the Cheetah or Rabbit can 'run', as well as 'walk'.
   Note: In the early stages, you should write down as many features as possible in order to understand the problem(s), to give you choices later on, and to appreciate the potential of the program. However, you should keep 3–5 most representative features at the top of your list for implementation.
2. Define a step which can move from the current position to one of the 4 immediate neighbouring positions in one of the 4 directions (Up, Down, Left, Right) by a single character instruction from the alphabet (U,D,L,R).
3. Display the C or R on the platform, and one step of their movement.
4. Write a method that returns a pair of random integers as the next position of the C or R.
5. Write a method that adds one position to an walk.
   For example, the current walk is [(3,2), (3,1), (3,2), (4,2)], the next position is (4,3), the new walk will become [(3,2), (3,1), (3,2), (4,2), (4,3)].
6. Write a method to display a walk of the C or R from a starting position.
7. Define a walk by the C or R as a number of intermediate positions which can be stored in a linked list, a stack, or a queue.
   Hint: When the walking route is in a stack, the stack operations such as `push`, `pop`, can be applied following FILO principles.
   When the route is in a queue, the queue operations such as `enqueue`, `dequeue` can be applied following FIFO principles.
8. Write a method that searches for a given walk from a list of (past) routes.

IS52038A Algorithms and Data Structures Term II Coursework part II (20%)
2015-2016

Deadline: Week 28, 16:00 Wednesday 2nd March 2016

You should review course topics on *decision trees, game trees, tries, quadtrees, hashing, searching, divide and conquer, dynamic programming, optimisation, time complexity.*
You should read the assignments or questions carefully and pay particular attention to any submission instructions.
Note: programme source codes may be checked using anti-plagiarism programmes.

Assignment

Your primary task in this part is to continue your previous programming experience and add more advanced features to the Cheetah and Rabbit simulation program. Again, you are encouraged to be creative and use your own imagination. However, you may refer to and select a number of the following questions if you find the original task difficult.
Note: if the total full marks of your selected questions exceeds the available total for this part, the solutions with the best marks will be used to count your coursework grade.

1. Extend the previous function `step` in 4 directions by the Cheetah or Rabbit to 8 directions (U, D, L, R, UL, UR, DL, DR) for each of the 8 immediate neighbouring postions accordingly.
2. Write a method to display a histogram to show the number of walks/routes against the length of the walks/routes.
3. Store the walks in a binary search tree, according to the length of the walks. Suppose all the walks are of an unique length.
4. Define a walk with different characteristics, which include but are not restricted to the following:
   (a) random walk: a sequence of random steps, where each `nextPosition` can be selected from the 8 directions (U, D, L, R, UL, UR, DL, DR) with an equal probability.
   (b) the shortest walk: the walk with the minimum number of steps from a *start* position to an *end* position.
   (c) chase: a walk aiming to keep the minimum distance from the current position of the Rabbit.
   (d) escape: a walk aiming to keep the maximum distance from the current position of the Cheetah.
5. Use graphs to model a `walk`.
6. Store walks in a trie structure, assume each trie is identified by a starting position followed by a string from alphabet (U, D, L, R, UL, UR, DL, DR). For example, [(1,1),RDRR] represents a walk [(1,1), (1,2), (2,2), (2,3), (2,4)].
   Hint: A subproblem is to write a method to add such a new position to a trie.
7. Write a method to display all the walks so far.
8. Write a method to find the route to a given destination with the maximum or minimum length.
9. Write a method to sort the existing walks by length.
10. Simulate any interactions between the Cheetah and Rabbit. For example, when the rabbit 'sees' the cheetah,
    (a) she 'runs' instead of 'walks' normally
    (b) she makes a 'sharp' turn to avoid the Cheetah
    (c) she stops for a few seconds before running away
    (d) she runs on a 'best' route to escape.
11. Simulate a 'fast and clever' cheetah who can always catch the rabbit.
    Hint: write a method which takes a route, and search for a 'best possible' (shortest) route, and to find the nearest meeting point of the two routes.

Submission Requirements

The requirements apply to both coursework parts I and II.

General

Your solutions should be typed and printed. Be concise and to the point. Write in your own words based on your own understanding and avoid copying from others. Any use of others' work should be put in quotes and identified at the point of use and declared in the `References` section at the end of your coursework.

Implementation

The programming submission should consist of the *source codes* and a *Document* in print.

1. The Java source codes (`.java`)
2. The Document (`.pdf` is preferred, but `.doc`, and `.docx` are acceptable) including the following sections:
   (a) Algorithms
   (b) Implementation
   (c) Demonstration
   (d) Discussion

Further requirements for each section are as follows:

1. The Java source codes:

   All classes placed in *ONE* directory which can be referred to by your own name in form of *firstnameFAMILYNAMEcw[1—2]*, where *firstname* is your first name in lowercase, *FAMILYNAME* is your family name in capital letters, `[1|2]` is a single digit of the coursework number.

   If you think your name is too common to be uniquely identified, extend the directory name by adding your student ID number. For example, `robertWHITEcw1`, `emmaBROWN123456789cw2` are valid, but `RobertWhitecw1` are invalid.

   '`menu.java`' Use and modify accordingly the *menu.java* program provided (see Appendix) as your main class to integrate and start your individual programs for coursework questions. We shall check your source codes by running *menu.class* only.

   Penalty a ZERO mark may be awarded if
   (a) your program(s) cannot be run from the coursework directory by a simple command '*java menu*';
   (b) your source code(s) does not compile;
   (c) your program(s) does not do what you claim it should do;
   (d) your program(s) crashes within the first *three* interactive execution steps;
   (e) your program(s) works for the first time of execution only;
   (f) there is no single comment in your source code.

2. Document:
   The document printed should include the following sections, be concise and not exceed 1000 words:
   (a) Approches (algorithms and implementation): You should use a flow chart to explain the functionality of the main blocks of your program and any internal relationships.
   You should also report how you have implemented the program blocks referring to the diagram(s).
   (b) Demonstration: You should report the platforms on which your program is developed and run. You should also include up to 5 screenshots to show the general performance of your program.
   (c) Discussion: Your primary goal in this section is to conduct a self-evaluation for your implementation as accurate as possible. You should comment on the limitation or success of your implementation process.

Algorithms: Your own algorithms should be highlighted. If you use other people's algorithms, you should explicitly give references or the source. *Note*: you will be treated as a plagiarist if you use other's work without declaration. Your own implementation should be highlighted. If you use part of other people's source code, you should attach a print of their original program, and mark clearly with a colour pen the sections that you have used in your program on both copies of their program and your own program.

Implementation: You should introduce any approaches that you have used during the implementation. For example, if you have chosen to implement certain program blocks earlier than others, you may justify yourself by discussing the advantages (or shotcomings) of the strategy. If you have used particular data structures, such as an linked list, you may discuss or comment on the approach. If you have noticed certain issues during testing of your program, you may evaluate the major testing methods.

Demonstration: You should include a few of best screenshots (up to 5) to show the performance of your programme(s). Here the 'best' is subjective. You may explain in what way, you think they are good, or the 'best'.

Discussion: You should focus on your own experience of java programming, and outline the limitation or success of your programs, and discuss what you have learnt from the coursework.

Recommended tools

You may like to organise your ideas and and draft the algorithms using a free design tool *cmap* (`http://cmap.ihmc.us`), *OpenOffice draw*, or *Dia* to produce comprehensive diagrams or flow charts.

Appendix

```
import java.lang.*;
import java.io.*;
class menu {       // Modify the display content to suit your purposes...
private static final String TITLE
"\n 2010226 coursework \n"+
"   by firstname-FAMILYNAME\n\n"+
"\t********************\n"+
"\t1. no attempt \n"+
"\t2. Question 2 \n"+
"\t3. Question 3 \n"+
"\t4. no attempt \n"+
"\t0. Exit \n"+
"\t********************\n"+
"Please input a single digit (0-4):\n";
menu() {
int selected=-1;
while (selected!=0) {
System.out.println(TITLE);
BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
try { selected = Integer.parseInt(in.readLine());
                            switch(selected) {
                                 case 1:  q1();
                                   break;
                                 case 2:  q2();
                                   break;
                                 case 3:  q3();
                                   break;
                                 case 4:  q4();
                                   break;
                            }
                 }
   catch(Exception ex) {
         }
}
               System.out.println("Bye!");
}
private void q1() {   // Modify the types of the methods to suit your
purposes...
        System.out.println("in q1");
}
private void q2() {
System.out.println("in q2");
}
private int q3() {
System.out.println("in q3");
return 1;
}
private boolean q4() {
System.out.println("in q4");
return true;
}
   public static void main(String[] args) {
new menu();
   }
}
```