COMP-SCI 201L (FS17) - **Section 2**

Lab 8 (10/31/17)

Halloween Game: Computing the Path with Maximum Total Reward in a Two-Dimensional Grid

Task Description

In this Halloween, you are going to complete a game challenge using your coding skills and your intelligence. Suppose there is an $m \times n$ grid (m rows, n columns). In each cell, there is a Halloween pumpkin in it. When you reach a cell, the pumpkin in that cell will give you a certain amount of reward (guaranteed to be a non-negative integer). Your path starts at the top left cell and ends at the bottom right cell. In each step, you can choose either to move to the right cell, or move down. **You can never go up or left**. Suppose all the reward values are known at the beginning. Your task is to find a specific path which gives you the maximum total reward. To make it simpler, you only need to output the maximum reward you could get in the given grid. For example, in Figure 1, which is a 4 × 4 grid, the maximum reward is 52. The path which could get the maximum reward is shown in red line.

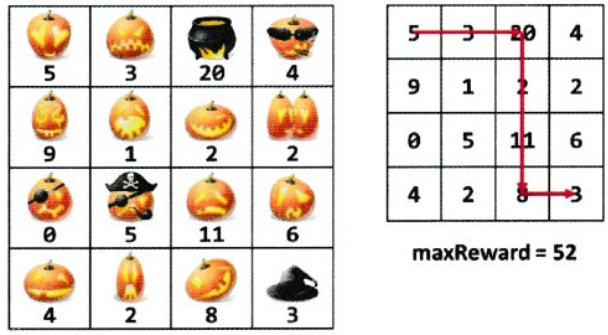


Figure 1. 4 × 4 grid whose maximum total reward is 52. The paths which generates the maximum total reward is marked in red solid line.

Recursion

Recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem. The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.

A recursive function definition has one or more *base cases*, meaning input(s) for which the function produces a result trivially (without recurring), and one or more *recursive cases*, meaning input(s) for which the program recurs (calls itself). For example, the factorial function can be defined recursively by the equations 0! = 1 and, for all $n > 0$, $n! = n(n - 1)!$. Neither equation by itself constitutes a complete definition; the first is the base case, and the second is the recursive case. Because the base case breaks the chain of recursion, it is sometimes also called the *terminating case*. Let's take the Fibonacci number as an example to articulate the steps of writing a recursive function (see Table 2 for details).

Table 2. Example of the Fibonacci number, demonstrating the steps of writing a recursive function.

Step	Purpose	Result
1	Define the problem (subproblem).	Let $F(n)$ be the n^{th} number in the Fibonacci sequence.
2	Find the base case(s).	$F(1) = 0$ $F(2) = 1$
3	Find the recursive relation(s).	For all $n > 2$, $F(n) = F(n - 2) + F(n - 1)$

4	Complete the code.	`unsigned int F(unsigned int n) {`
		`  if (n == 1) return 0;`
		`  if (n == 2) return 1;`
		`  return F(n - 2) + F(n - 1);`
		`}`

The two greatest advantages of using recursive relation are simpler mathematics (no need to completely solve the math equations) and less lines of code (compared with the iterative methods).

However, is recursive function always good? The answer is **no**. For example, if we use the recursive function generated above to calculate $F(6)$, the specific calculations are illustrated in Figure 3. We can see that in order to calculate $F(6)$, we have to repeatedly calculate $F(4)$ twice and calculate $F(3)$ for three times. If the number n goes up to a large number, the number of repeated calculations will also increase dramatically, which will slow down the execution of your program.

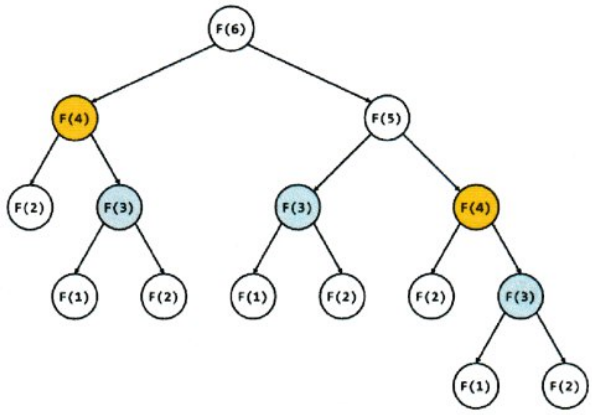


Figure 3. Using recursive function to calculate $F(6)$, we have to calculate $F(4)$ twice (orange) and calculate $F(3)$ three times (blue).

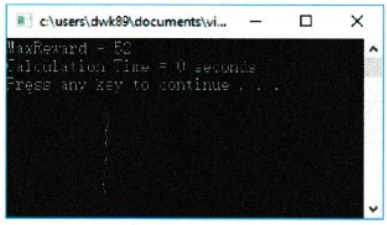
Difference between Recursion and Induction

Recursion is kind of *top-down* method, which first attempts to solve the biggest problem. If same problems with smaller size are required, then it solves the smaller problems until it reaches the base cases.

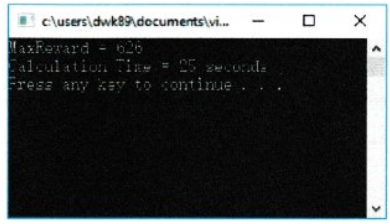
Induction is kind of *bottom-up* method, which first solves all the subproblems (no matter necessary or not for the biggest problem), then solves the biggest problem. You will learn a lot of recursion and induction in your algorithm class. In this lab, we just focus on the simplest recursive method mentioned in Table 2.

Programming Instruction

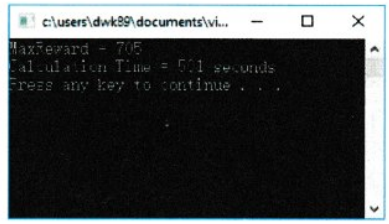
1. Start a new empty Visual C++ project in MS Visual Studio.
2. Download files of `Grid.h`, `Grid.cpp`, and `Lab_8.cpp` from Blackboard.
3. In `Grid.h`, everything is defined except the function **`MaxReward()`**. You need to define the function **`MaxReward()`** in `class Grid`. **You must explain the meaning of every parameter in you function definition (comment)**. You will get zero if your comment of explanation is absent.
4. In `Grid.cpp`, every member/non-member function is implemented except the function **`MaxReward()`**. You need to implement the function **`MaxReward()` using recursive relation**. **In your code comments, you must show the definition of subproblem, base cases, and induction relations**. You will get zero if the above information is absent. **Also, you will get zero if your code is not recursive**.
5. In the `main()` program, the input part is done for you. All you need to do is to output the maximum reward you could get from the input `Grid`. **You need to output the value to standard console (cout), not to a file (not fout)**.
6. **Take the screenshots** of all three input files, see below for the correct output and estimated running time.



`input1.txt`



`input2.txt`



`input3.txt`

Submission and Grading

1. Submit your work **no later than Thursday, Nov 2nd, 2017 @ 11:59 PM**.
2. Submit your files of `Grid.h`, `Grid.cpp`, and `Lab_8.cpp`. **Screenshots** for the 3 input files need to be submitted as well.
3. Your code will be graded using another input file (named **`input.txt`**) to see if it can generate the expected output. The grader will also look at your code manually to see if your code quality meets our other requirements.
4. **You will get zero if required documentation is absent in your code!**