CS3230 Programming Assignment 1

Background

Idle games are getting very popular nowadays, especially on mobile platforms. These are games where you perform some simple action, like tapping the screen, in order to earn some form of currency. Over time, you can buy upgrades that increase the amount of currency you get per action, and units that generate currency automatically for you over time.

However, many idle games require you to perform manual actions to reach a certain amount of currency before you can start purchasing units. There are usually multiple ways to achieve this, some ways longer than others. A game developer has designed a new idle game with an achievement system and has decided that they want to award an achievement to the player if they reach this amount using the minimum number of taps.

In order to implement the achievement, they first need to calculate what is the minimum number of taps required. The developer realised that you are taking a class on algorithm design and has thus tasked you to help him design and implement an algorithm to solve this problem.

Details

The design of the idle game is as follows. The currency is dollars($) and you get dollars every time you tap the screen. You start with $0 and your tapping starts at level 0 and generates $\$value_0$ per tap.

At any point, you can upgrade your tapping to level i for a cost of $\$cost_i$ and from then on every tap will generate $\$value_i$ per tap. The amount of money you have cannot go below $0 so you must have the required amount of money before you can upgrade. You are, however, allowed to skip levels, so for example from level 0 you can immediately level up to level 2 by paying $\$cost_2$ and you do not have to pay $\$cost_1$.

The developer has also designed the game such that the higher the level, the higher the cost and value you get per tap. Mathematically, this means that if $i > j$ then $cost_i > cost_j$ and $value_i > value_j$. Given this information, you need to calculate what is the minimum number of taps you need to get some required amount. For the purposes of this problem, upgrading a level does not count as a tap.

Implementation

The game developer has already implemented the rest of the game and only needs you to implement a function to calculate the value described above. This function will take in 4 parameters:

- maxLevel - The maximum level your tapping can be upgraded to
- amountRequired - The amount of money that needs to be reached
- values - A list of (maxLevel + 1) integers where values[i] is $value_i$ as described above
- costs - A list of (maxLevel + 1) integers where costs[i] is $cost_i$ as described above

You are allowed to submit in either Java or C++11. Templates for both languages have been provided in the IVLE Workbin in the same folder as this task statement. The templates have implemented the I/O routines for you so you are strongly encouraged to use them. In addition, you are allowed to add other functions into your program to modularise your code but you must submit only **ONE** source file.

For testing purposes, the program will read in the parameters listed above from standard input in the exact order listed above for each test case with each parameter on a separate line. There can be multiple test cases within one run and the first integer gives the number of test cases. The program will output one line for each test case, which contains one integer that is the answer for that test case.

For example, if a game is designed such that maxLevel is 2, amountRequired is 15, values = [2, 4, 9] and costs = [0, 5, 8], then the input it expects will look like this:

```
1
2
15
2 4 9
0 5 8
```

Example

For the example game listed above, if you just keep tapping and upgrade by 1 level every time you reach the required cost, you will follow these series of steps.

1. Tap 3 times to reach $6
2. Pay $5 to upgrade to level 1 with $1 left
3. Tap 2 times to reach $9
4. Pay $8 to upgrade to level 2 with $1 left
5. Tap 2 times to reach $19

This will require 3 + 2 + 2 = 7 taps.

You also do not have to upgrade to the maximum level in order to reach the amount required. For example, you can follow these series of steps.

1. Tap 3 times to reach $6
2. Pay $5 to upgrade to level 1 with $1 left
3. Tap 4 times to reach $16

However, this will still require 3 + 4 = 7 taps.

The sequence of steps with the least number of taps is as follows.

1. Tap 4 times to reach $8
2. Pay $8 to upgrade to level 2 with $0 left
3. Tap 2 times to get $18

This set of moves only requires 4 + 2 = 6 taps and this is the least you can use so your function should return 6.

Submission

The assignment is split into multiple parts with a separate score for each part. You are encouraged to work on all the parts in order since it will help you design your algorithm step by step.

You will submit your solutions on `http://algorithmics.comp.nus.edu.sg/~mooshak/` using a username and password that will be emailed to you. You are NOT allowed to share accounts and you must submit your own code. Your codes will be checked for plagiarism and any instances of plagiarism found will be subject to disciplinary action.

To submit your solutions, select the contest ”CS3230 PA1” and login. You will see parts A and B which represent the parts described below. Submit your solution and it will be run against a set of 10 test cases. Your code must solve all test cases correctly and within 1 second for a part in order to get the credit for that part. You will see the result of your submission immediately after submitting and you can submit as many times as you want before the deadline without any penalty for wrong answers.

Scoring

Part A (2%)

In the first part, you are required to simply design any algorithm that solves the problem to show that you understand the problem. You are strongly encouraged to use recursion to solve this problem to help you in solving part 2. The test cases for this part will satisfy the following restrictions:

- $1 \leq maxLevel, amountRequired \leq 10$
- $0 = cost_0 < cost_1 < ... < cost_{maxLevel} \leq 10$
- $1 \leq value_0 < value_1 < ... < value_{maxLevel} \leq 10$

Part B (6%)

In the second part, you are required to design a Dynamic Programming algorithm that is efficient enough to solve the problem with time complexity $O(maxLevel^2 value_{maxLevel})$. You should be able to make use of your solution in part 1 to solve this part by adding a memoisation table. (Hint: the amountRequired should not contribute to your time complexity). The test cases for this part will satisfy the following restrictions:

- $1 \leq maxLevel \leq 100$
- $1 \leq amountRequired \leq 1000000$
- $0 = cost_0 < cost_1 < ... < cost_{maxLevel} \leq 10000$
- $1 \leq value_0 < value_1 < ... < value_{maxLevel} \leq 1000$

Bonus

There is in fact an algorithm that solves this problem with time complexity $O(maxLevel)$. If you are interested you may challenge yourself and think about or research about the solution.

Deadline

The deadline for submission for this assignment is 8th Oct 2018 2359h The portal will automatically close at this time and no further submissions will be accepted.