Middle East Technical University

Department of Computer Engineering

CENG 242

Programming Language Concepts
Spring 2023
Programming Exam 2

Due: 10 April 2023 23:55
Submission: **via ODTUClass**

1 Problem Scenario

In this programming exam, you will be using Haskell to build some planning functions for a dungeon crawler game, where you lead a group of adventurers who happen upon a cave that they might wish to explore. You need to decide whether you want to explore the cave, and if so, what path to follow and what strategy to use in order to get as much gold as possible with as little effort as possible. You will implement 5 Haskell functions to help you decide what to do.

1.1 General Specifications

- The signatures of the functions, their explanations and specifications are given in the following section. Read them carefully.
- Make sure that your implementations comply with the function signatures.
- You may define helper function(s) as needed.
- Data types that we will be using are already defined for you in the homework files, as given below:

  – Our first (and primary) data structure for this homework is a tree, which is a recursively defined data structure. The tree that we will used is polymorphic, meaning that our trees can hold different types of data. Our tree is arbitrary-arity, different nodes in the tree may have different numbers of children; the children of a node is represented as a list of trees. Its definition is given below.

```
data Tree a b = EmptyTree | Leaf a b | Node a b [Tree a b] deriving (Show, Eq)
```

– Our second data type is the Chamber. We will use it to represent the different kinds of rooms that may appear in a dungeon. It has no type parameters and no value constructor with more than zero parameters, so it only has 4 distinct values. It is defined as:

```
data Chamber = Cavern |
               NarrowPassage |
               UndergroundRiver |
               SlipperyRocks deriving (Show, Eq)
```

– Next we have three interrelated data types that will be used for describing what kinds of things exists in our dungeon. First we have the Enemy type, which represents hostile entities in the dungeon that we must fight. An enemy has a name, an amount of damage that it will deal, and an amount of gold that it will drop upon being defeated. It is defined as:

```
data Enemy = Enemy String Integer Integer deriving (Show, Eq)
```

– Second we have the Loot type, which represents any valuable items we may encounter. It may be a certain sum of gold, which would increase your total gold, or it may be a healing potion that heals your party for a certain amount of hp when found:

```
data Loot = Gold Integer | Potion Integer deriving (Show, Eq)
```

– Third, we have the Encounter type, which wraps the previous two types and unifies them. An encounter is either a Fight, in which case it would hold an enemy that must be fought; or a Treasure, giving you some free loot:

```
data Encounter = Fight Enemy | Treasure Loot deriving (Show, Eq)
```

– Finally, we have a type ***type synonym*** which we will use to represent our dungeon maps. It is a tree whose each node holds one value of type Chamber, and one value of type [Encounter]. These nodes will represent the rooms in our dungeon, and the encounters that await within them:

```
type Dungeon = Tree Chamber [Encounter]
```

– Here is an example of the trees that we will be using:

```
dungeonMap = Node NarrowPassage [] [
    Node UndergroundRiver [Fight (Enemy "Orc" 10 5)] [
        Leaf SlipperyRocks [Fight (Enemy "Necromancer" 30 100)]],
    Node Cavern [Fight (Enemy "Goblins" 5 2), Treasure (Gold 15)] [
        Leaf NarrowPassage [Treasure (Gold 10), Treasure (Potion 5)],
        Leaf UndergroundRiver [Fight (Enemy "Lich" 25 50)],
        Leaf Cavern [Fight (Enemy "Goblins" 5 2),
                     Fight (Enemy "Imp" 4 4),
                     Treasure (Potion 15),
                     Fight (Enemy "Bear" 15 9)]]]
```

2 Functions

2.1 traversePath (10 pts.)

This function takes an integer (the starting hp), a tree and a list of integers that describe a downward path through the tree. Each integer in the path defines what path to take down from a node (children are numbered starting from 0). For example, [3,0,1] would mean go to child number 3, then to child number 0, then to child number 1. This function will calculate the amount of hp left, and of gold collected, at the end of the path. The function signature is:

```
traversePath :: Integer -> Dungeon -> [Int] -> (Integer, Integer)
```

Here is an example call to the function:

```
ghci> traversePath 20 dungeonMap [1,0]
(20,27)
```

2.2 findMaximumGain (20 pts.)

This function takes an integer (the starting hp) and a tree, and returns the maximum amount of gold that can be collected by going down the tree along any downward path. The path that you find needn't reach a leaf (you can descend down a path and then stop there without going all the way down). Note that you are looking for a linear path, i.e. no backtracking. You will return the amount of gold that can be collected along that path. Here is the function signature:

```
findMaximumGain :: Integer -> Dungeon -> Integer
```

And it can be called as:

```
ghci> findMaximumGain 15 dungeonMap
32
```

2.3 findViablePaths (25 pts.)

This function takes an integer (the starting hp) and a tree, and returns another tree that contains only those paths that can be fully traversed with the given starting hp. You will only keep the portions of the paths that you can pass through with the given hp, and cut off the rest. Note that some internal nodes may become leaves after removing the paths, make sure you convers them to `Leaf` structures. There shouldn't be any `Nodes`s that have an empty list of children. The function signature is given as:

```
findViablePaths :: Integer -> Dungeon -> Dungeon
```

It is called like so:

```
ghci> findViablePaths 15 dungeonMap
Node NarrowPassage [] [
    Leaf UndergroundRiver [Fight (Enemy "Orc" 10 5)],
    Node Cavern [Fight (Enemy "Goblins" 5 2),Treasure (Gold 15)] [
        Leaf NarrowPassage [Treasure (Gold 10),Treasure (Potion 5)],
        Leaf Cavern [Fight (Enemy "Goblins" 5 2),
                     Fight (Enemy "Imp" 4 4),
                     Treasure (Potion 15),
                     Fight (Enemy "Bear" 15 9)]]]
```

2.4 mostDistantPair (15 pts.)

This function takes an integer (the starting hp) and a tree, and finds the pair of nodes that are furthest away from each other among the nodes that you can visit with the given hp (not necessarily consecutively, i.e. you should be looking for the pair inside the viable paths). The distance between two nodes is defined as the number of edges in the path that connects them. That is, a node has distance 0 to itself, distance 1 to its parent and its children, distance 2 to its grandparent and grandchildren, and so on; two children of the same node would also have distance 2. You should return a pair containing the distance between the two most distant nodes, and the path that connects these nodes. Here is the function signature:

```
mostDistantPair :: Integer -> Dungeon -> (Integer, Dungeon)
```

Example usage:

```
ghci> mostDistantPair 11 dungeonMap
(3,
Node NarrowPassage [] [
    Leaf UndergroundRiver [Fight (Enemy "Orc" 10 5)]
    Node Cavern [Fight (Enemy "Goblins" 5 2), Treasure (Gold 15)] [
        Leaf NarrowPassage [Treasure (Gold 10), Treasure (Potion 5)]]])
```

2.5 mostEfficientSubtree (30 pts.)

This function finds the most efficient subtree in a given dungeon tree. The efficiency of a subtree is simply its total gold divided by its total hp. (i.e. how much gold you can collect for the amount of damage that you would take). Each subtree spans all the descendants of its starting node, i.e. your subtree can have any node in the tree as its root, but it cannot remove (or add) any nodes from among its children. Note that the most efficient subtree can be the whole tree, or just a leaf, or anything in between. It is `EmptyTree` only when then given tree is the `EmptyTree`. This function's signature is:

```
mostEfficientSubtree :: Dungeon -> Dungeon
```

And here is a call with our example tree:

```
ghci> mostEfficientSubtree dungeonMap
Leaf NarrowPassage [Treasure (Gold 10),Treasure (Potion 5)]
```

Note that if a subtree has either 0 or negative hp cost, its efficiency is infinite. If there are multiple such subtrees (or any other kind of tie between the efficiencies of subtrees), returning any one of such subtrees is accepted. (i.e. if there is a tie, you can return any of the subtrees that tied)

3 Regulations

1. **Implementation and Submission:** The template file named "PE2.hs" is available in the Virtual Programming Lab (VPL) activity called "PE2" on OdtuClass. At this point, you have two options:

   - You can download the template file, complete the implementation and test it with the given sample I/O (and also your own test cases) on your local machine. Then submit the same file through this activity.
   - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submit a file.

   If you work on your own machine, make sure that your implementation can be compiled and tested in the VPL environment after you submit it.
   There is no limitation on online trials or submitted files through OdtuClass. The last one you submitted will be graded.

2. **Cheating: We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

3. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to consider the invalid expressions.

- **Important Note:** The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your ***actual*** grade after the deadline.