CS 430 Programming Languages
Spring 2016
Module 17 Programming Assignment

JAMES MADISON UNIVERSITY.

Introduction

This PA uses lists to solve a simple optimization problem.

Consider the following hopscotch game. The game is laid out in a single line. Each square is marked with a digit from 1 to 9, with repetitions possible. A player starts in the first square and hops to other squares, adding up the numbers in the squares along the way. A player in square *k* must hop to either square *k*+2 or *k*+3 (that is, a hop must skip one or two squares). The objective is to obtain the maximum possible sum by the end of the layout.

For example, consider the following game layout.

3	1	4	2	1	5	6	7	2

Among the possible paths through the game is [3,4,1,6,2], [3,4,5,7], [3,2,6,2], and so forth. But the path with the highest total is [3,4,5,7].

Requirements

A game layout will be represented by a list of numbers from 1 to 9. For example, the layout above would be represented as [3,1,4,2,1,5,6,7,2]. Solution paths are represented as lists of numbers like those in the example above. You must write one predicate:

> `hopscotch(X,Y)`—This predicate succeeds if `X` is a game layout list and `Y` is a solution list for that layout. For example, `hopscotch([3,1,4,2,1,5,6,7,2], [3,4,5,7]).` succeeds but `hopscotch([3,1,4,2,1,5,6,7,2], [3,2,6,2])` fails.

You must download the file `mod17Basis.pl` and use it as the basis for your program. It contains a `test_hopscotch` predicate that you must not change, and a `test` predicate that you can augment if you like. When I grade your programs I will run the `test` predicate to test your code.

Hints

The easiest way to solve this problem is brute force: generate a list with all the possible paths through the game layout and then find a member of this list (a path) with the greatest sum. This will work until the list of paths gets so big that it overflows the stack, which happens on my computer with about 40 squares in the game. But this is an acceptable solution. There are other algorithms that avoid this problem—you can try to make one if you like.

You will need to make some helper predicates. Writing tests for them would be a good idea.

Deliverables

The deliverable for this assignment is a Prolog source file that must be named `mod17PA.pl` and submitted on Canvas. Your name must appear in a comment at the top of the file. This file must be submitted by the due date on Canvas.