CS 5800: Algorithms Virgil Pavlu

Homework 6: Dynamic Programming part II

v1

Problems

1. (no credit). Using the class notes (without the book), finish the Optimal BST and completely write the solution (including the DP recipe elements 1,2A,2B,3,4,5). Then compare your writeup with the book chapter.

2. Jars on a ladder problem. Given a ladder of n rungs and k identical glass jars, one has to design an experiment of dropping jars from certain rungs, in order to find the highest rung (HS) on the ladder from which a jar doesn't break if dropped.

   Idea: With only one jar (k=1), we can't risk breaking the jar without getting an answer. So we start from the lowest rung on the ladder, and move up. When the jar breaks, the previous rung is the answer; if we are unlucky, we have to do all n rungs, thus n trials. Now lets think of k=log(n): with log(n) or more jars, we have enough jars to do binary search, even if jars are broken at every rung. So in this case we need log(n) trials. Note that we can't do binary search with less than log(n) jars, as we risk breaking all jars before arriving at an answer in the worst case.

   Your task is to calculate $q = MinT(n,k)$= the minimum number of dropping trials any such experiment has to make, to solve the problem even in the worst/unluckiest case (i.e., not running out of jars to drop before arriving at an answer). $MinT$ stands for Minumum number of Trials.

   A(5 points). Explain the optimal solution structure and write a recursion for $MinT(n,k)$.

   B(5 points). Write the alternative/dual recursion for $MaxR(k,q)$ = the Highest Ladder Size n doable with k jars and maximum q trials. Explain how $MinT(n,k)$ can be computed from the table $MaxR(k,q)$. $MaxR$ stands for the Maximum number of Rungs.

   C(10 points). For one of these two recursions (not both, take your pick) write the bottom-up non-recursive computation pseudocode. *Hint: the recursion $MinT(n,k)$ is a bit more difficult and takes more computation steps, but once the table is computed, the rest is easier on points E-F below. The recursion in $MaxR(q,k)$ is perhaps easier, but trickier afterwards: make sure you compute all cells necessary to get $MinT(n,k)$— see point B.*

   D(10 points). Redo the computation this time top-down recursive, using memoization.

   E(10 points). Trace the solution. While computing bottom-up, use an auxiliary structure that can be used to determine the optimal sequence of drops for a given input n,k. The procedure TRACE(n,k) should output the ladder rungs to drop jars, considering the dynamic outcomes of previous drops. *Hint: its recursive. Somewhere in the procedure there should be an **if statement** like "if the trial at rung x breaks the jar... else ..."*

F(extra credit, 20 points). Output the entire decision tree from part E) using JSON to express the tree, for the following test cases : (n=9,k=2); (n=11, k=3); (n=10000,k=9). Turn in your program that produces the optimum decision tree for given n and k using JSON as described below. Turn in a zip folder that contains: (1) all files required by your program (2) instructions how to run your program (3) the three decision trees in files t-9-2.txt, t-11-3.txt and t-10000-9.txt (4) the answers to all other questions of this homework.

To represent an algorithm, i.e., an experimental plan, for finding the highest safe rung for fixed n,k, and q we use a restricted programming language that is powerful enough to express what we need. We use the programming language of binary decision trees which satisfy the rules of a binary search tree. The nodes represent questions such as 7 (representing the question: does the jar break at rung 7?). The edges represent yes/no answers. We use the following simple syntax for decision trees based on JSON. The reason we use JSON notation is that you can get parsers from the web and it is a widely used notation. A decision tree is either a leaf or a compound decision tree represented by an array with exactly 3 elements.

```
// h = highest safe rung or leaf
{ "decision_tree" : [1,{"h":0},[2,{"h":1},[3,{"h":2},{"h":3}]]] }
```

The grammar and object structure would be in an EBNF-like notation:

```
DTH = "{" "\"decision_tree\"" ":" <dt> DT.
DT = Compound | Leaf.
Compound = "[" <q> int "," <yes> DT "," <no> DT "]".
Leaf = "{" "\"h\" " ":" <leaf> int "}".
```

This approach is useful for many algorithmic problems: define a simple computational model in which to define the algorithm. The decision trees must satisfy certain rules to be correct.

A decision tree t in DT for HSR(n,k,q) must satisfy the following properties:

(a) the BST (Binary Search Tree Property): For any left subtree: the root is one larger than the largest node in the subtree and for any right subtree the root is equal to the smallest (i.e., leftmost) node in the subtree.

(b) there are at most k yes from the root to any leaf.

(c) the longest root-leaf path has q edges.

(d) each rung 1..n-1 appears exactly once as internal node of the tree.

(e) each rung 0..n-1 appears exactly once as a leaf.

If all properties are satisfied, we say the predicate correct(t,n,k,q) holds. HSR(n,k,q) returns a decision tree t so that correct(t,n,k,q).

To test your trees, you can download the JSON HSR-validator from the url:
`https://github.com/czxttkl/ValidateJsonDecisionTree`

G(extra credit, 20 points). Solve a variant of this problem for $q = \text{MinT(n,k)}$ that optimizes the average case instead of the worst case: now we are not concerned with the worst case q, but with the average q. Will make the assumption that all cases are equally likely (the probability of the answer being a particular rung is the same for all rungs). You will have to redo points A,C,E specifically for this variant.

3. (20 pts) Problem 15-2. Hint: try to use the LCS problem as a procedure.

4. (30 pts) Exercise 15.4-6.

5. (Extra credit 20 pts) Suppose that you are the curator of a large zoo. You have just received a grant of $\$m$ to purchase new animals at an auction you will be attending in Africa. There will be an essentially unlimited supply of n different types of animals, where each animal of type i has an associated cost c_i. In order to obtain the best possible selection of animals for your zoo, you have assigned a value v_i to each animal type, where v_i may be quite different than c_i. (For instance, even though panda bears are quite rare and thus expensive, if your zoo already has quite a few panda bears, you might associate a relatively low value to them.) Using a business model, you have determined that the best selection of animals will correspond to that selection which maximizes your perceived profit (total value minus total cost); in other words, you wish to maximize the sum of the profits associated with the animals purchased.

   Devise an efficient algorithm to select your purchases in this manner. You may assume that m is a positive integer and that c_i and v_i are positive integers for all i. Be sure to analyze the running time and space requirements of your algorithm.

6. (20 points) Problem 15-4.

7. (20 points) Problem 15-10.