CAL POLY POMONA COMPUTER SCIENCE DEPARTMENT

The Optimal Binary Search Tree Problem

Project Report by Khamille Sarmiento

Professor Srinivas
CS 331 Fall 2014

Project 1: Optimal Binary Search Tree

Introduction:

The purpose of this project was to determine the optimal arrangement of a binary search tree given some number of elements and their corresponding probabilities of being searched, as well as the average number of comparisons to find an element. Binary search trees of the same elements can have multiple valid arrangements, but when given each element's probability of being accessed, the tree's shape can be optimized for an improved average number of comparisons necessary to access the element.

Dynamic Programming and Memory Functions:

Dynamic programming is a method of optimization which allows a program to save already calculated values and use them without recalculation. Dynamic programming is a good way to implement recursive problems because it avoids unnecessary recursive calls which would otherwise slow down a large program. Dynamic programming calculates a solution in a bottom-up fashion, meaning a program will recurse down to the smaller cases or the base case, and then move up towards the solution.

Dynamic programming is possible when a memory function is implemented into a program. Memory functions make recalculating solutions unnecessary because they save found solutions into another data structure which can be used to look up already calculated values.

Approach:

To conclude a binary search tree is optimal, we try to find the smallest average number of comparisons necessary to find an element. An optimal binary search tree would have the most accessed element as its root, and its left and right subtrees would be optimized as well.

My approach to solving this problem was written using Python 3.4. My program contains two 2d arrays: one that served as a backing array and another that served as the optimal root array. The most important function in my program is the recursive

memory function A(i,j). This memory function A was written using pseudocode provided by the professor:

```
function A(i,j)
1. If a[i,j]=null
   1.1 Compute A(i,j) using the recurrence relation, producing result
   1.2 Set a[i,j] to result
2. Else set result to a[i,j]
3. Return result
```

The most complex step in the pseudocode was step 1.1 which included two base cases:

- When a[i][j] is empty, this means I've reached the diagonal of my backing array.
- When i = j, this means I've reached the index which should contain an original probability.

The last case was an else statement that contained the recurrence relation explained in my textbook.

Reflection:

This program was fun to write because it was my first larger program that I've written in Python. It is interesting to learn a new language.

This program was confusing to write because it was difficult for me to understand how all the pieces worked together: how to use the memory function A(i,j), where the recurrence relation fit into the program, and how the tree(i,j) function in the textbook would help me print the arrangement of the tree using parenthesis and commas. Once I understood how all these pieces worked together, I was able to appreciate the efficiency of dynamic programming in this case.

One thing that I can improve on is my understanding of recursive functions. I know how to step through a recursive function and I understand how they work, but it is always difficult for me to write them. The tree function psuedocode from my textbook really helped me to understand how recursion works well to perform the printing task.