CS 430 Lecture 16 Activities

Opening Questions

1. In the Optimal Binary Search Tree problem we are not just trying to balance the tree, instead we are trying to minimize what? We're trying to minimize expected search times.

2. What is different about the 0-1 knapsack problem as compared to the other problems we solved with dynamic programming?

Optimal Binary Search Tree

The Optimal Binary Search Tree problem is a special case of a BST where the data is static (no inserts or deletes) and we know the probability of each key in the data being searched for. We want to minimize total expected search time for the BST. Recall expectation

$$E(X) = \sum_{s \in S} X(s)p(s)$$

$$A = \frac{1}{3}, B = \frac{1}{2}, C = \frac{1}{6}$$

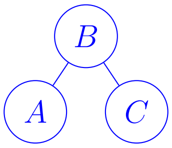


Figure 16.1: Example of Optimal Binary Search Tree

$$\begin{aligned} E(x) &= (1)\left(\frac{1}{2}\right) + (2)\left(\frac{1}{3}\right) + (2)\left(\frac{1}{6}\right) \\ &= \frac{1}{2} + \frac{2}{3} + \frac{2}{6} \\ &= \frac{3}{6} + \frac{4}{6} + \frac{2}{6} \\ &= \frac{9}{6} \\ &= \frac{3}{2} \end{aligned}$$

1. Apply the expectation formula to the Optimal Binary Search Tree problem with n keys and $C(i)$ is how deep item i is. $P(i)$ is probability of search for item i. Minimize $\sum_{keys} P(key)C(key)$ where C is the number of compares

2. The brute force approach (generate all possibilities, pick the optimal) would be to find all possible BSTs, find the expected search time of each and pick the minimum one. How many BSTs are there? Inserting each permutation of n keys might lead a different structure, and this would be $O(n!)$.

3. Step 1: Generically define the structure of the optimal solution to the Optimal Binary Search Tree problem. The optimal binary search tree with n keys and $C(i)$ is how deep item i is and $P(i)$ is probability of search for item i is: $k_1 \dots k_n$ where $k_1 < k_2 < \cdots < k_n$ and $p_1 \dots p_n$. Assume k_x as root yields BST with minimal expected search time.

$$\begin{aligned}\sum_{all\ \ keys} p(k_i)Depth(k_i) &= p(k_x)(1) + \sum_{other\ \ keys} p(k_i)Depth(k_i) \\ &= P(k_x)(1) + \sum_{keys<k_x} P(k_i)Depth(k_i) + \sum_{keys>k_x} P(k_i)Depth(k_i)\end{aligned}$$

If $A[i,n] =$ minimum expected search time for keys $k_1 \leftrightarrow k_n$ and probabilities $p_1 \dots p_n =$

$$\min_{1 \leftarrow i \rightarrow n} \{P(k_i)(1) + A[1, i-1] + A[i+1, n] +$$
$$(1)(All\ probabilities\ 1 \rightarrow (i-1)) + (1)(All\ probabilities\ (i+1) \rightarrow n)$$

$$A[i,n] = \min_{1 \leftrightarrow i \leftrightarrow n} \left\{ A[i, i-1] + A[i+1, n] + (1)\sum_{j=1}^{n} P(j) \right\}$$

4. Step 2: Recursively define the optimal solution. Assume $A(i,j)$ is the optimal answer for keys i to j. Make sure you include the base case. For any subproblem:

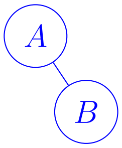
$$\begin{aligned}A[p,r] &= \min_{p \leftrightarrow i \leftrightarrow r} \left\{ A[p, i-1] + A[i+1, r] + \sum_{j=p}^{r} P(j) \right\} \\ A[k,k] &= (1)P(k)\end{aligned}$$

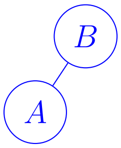
5. Use proof by contradiction to show that Optimal Binary Search Tree problem has optimal substructure, i.e. the optimal answer to problem must contain optimal answers to sub-problems.

6. Step 3: Compute solution using a table bottom up for the Optimal Binary Search Tree problem. Use your answer to question 4 above. Note the overlapping sub-problems as you go.

Table 16.1: Optimal Binary Search Tree Table

	A	B	C	D	E	F
A	.2	$.52_A$	$.72_B$			
B		.16	$.32_B$			
C			.08			
D				.22		
E					.21	
F						.13

The tree: 

would have the probability $(.2)(1) + (.16)(2) = .2 + .32 = .52$ while the tree 

would have the probability $(.16)(1) + (.2)(2) = .16 + .4 = .56$, so we choose the lower value of .52 with A as the root.

7. Step 4: Construct Optimal solution

A	B	C	D	E	F
.2	.16	.08	.22	.21	.13

Optimal Binary Search Tree: `http://www.cse.yorku.ca/~aaw/Gubarenko/BSTAnimation.html`