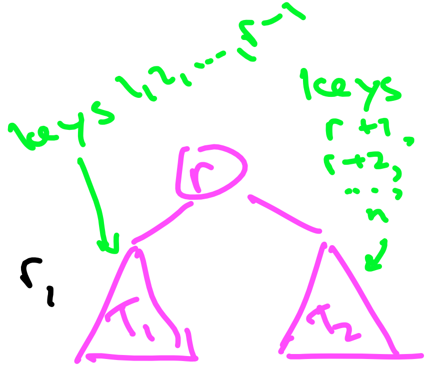
Algorithms: Design and Analysis, Part II

Dynamic Programming

Optimal BSTs: A Dynamic Programming Algorithm

Optimal Substructure

Optimal Substructure Lemma: If T is an optimal BST for the keys $\{1,2,\ldots,n\}$ with root r, then its subtrees T_1 and T_2 are optimal BSTs for the keys $\{1,2,\ldots,r-1\}$ and $\{r+1,\ldots,n\}$, respectively.



Note: items in a subproblem are either a prefix <u>or</u> a suffix of the original problem.

Relevant Subproblems

Question: Let $\{1,2,3,\ldots,n\}$ = original items. For which subsets $S \subseteq \{1,2,\ldots,n\}$ might we need to compute the optimal BST for S?

(A) prefixes ($S = \{1,2,\ldots,i\}$ for every i)

(B) prefixes and suffixes ($S = \{1,2,\ldots,i\}$ and $\{i,\ldots,n\}$ for every i)

(C) contiguous intervals ($S = \{i,i+1,\ldots,j-1,j\}$ for every $i \le j$)

(D) all subsets S

Tim Roughgarden

The Recurrence

Notation: For $1 \le i \le j \le n$, let C_{ij} = weighted search cost of an optimal BST for the items $\{i, i+1, \dots, j-1, j\}$

[with probabilities $p_i, p_{i+1}, \dots, p_j$]

Recurrence: For every $1 \le i \le j \le n$:

$$C_{ij} = \min_{r=i}^{j} \left\{ \sum_{k=i}^{j} p_k + C_{i,r-1} + C_{r+1,j} \right\}$$

recall formula $C(T) = \sum_k p_k + C(T_1) + C(T_2)$ from last video

interpret $C_{xy} = 0$ if $x > y$

Correctness: optimal substructure narrows candidates down to $(j-i+1)$ possibilities, recurrence picks the best by brute force.

Tim Roughgarden

The Algorithm

Important: Solve smallest subproblems (with fewest number $(j-i+1)$ of items) first.

Let A = 2-D array. [$A[i,j]$ represents opt BST value for items $\{i,\dots,j\}$]

For $s = 0$ to $(n-1)$ [s represents $(j-i)$]

    For $i = 1$ to n [so $i+s$ plays role of j]

$$A[i,i+s] = \min_{r=i}^{i+s} \left\{ \sum_{k=i}^{i+s} p_k + A[i,r-1] + A[r+1,i+s] \right\}$$

- interpret as 0 if 1st index > 2nd index
- available for $O(1)$-time lookup

Return $A[1,n]$.

Pictorially

$A[i,i] = p_i$

Running Time

- $\Theta(n^2)$ subproblems
- $\Theta(j-i)$ time to compute $A[i,j]$

$\Rightarrow \Theta(n^3)$ time overall

Fun Fact: [Knuth '71, Yao '80] optimized version of this DP algorithm correctly fills up entire table in only $\Theta(n^2)$ time [$\Theta(1)$ on average per subproblem]

[idea: piggyback on work done in previous subproblems to avoid trying all possible roots]