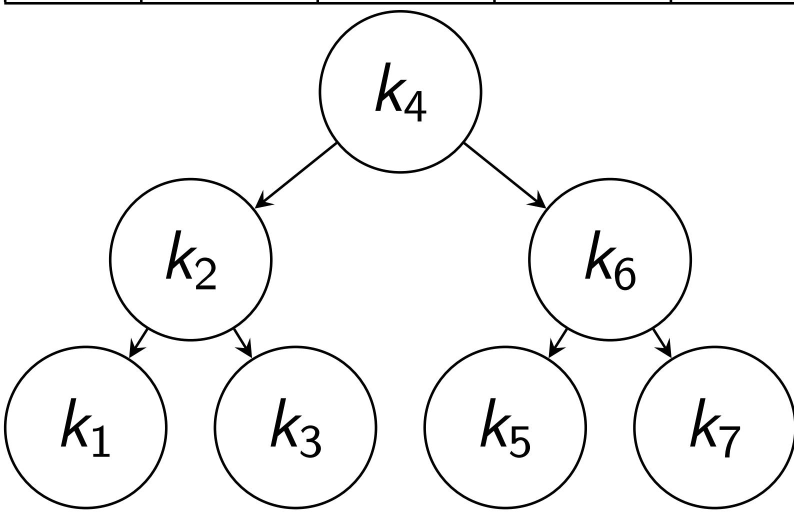
CompSci 161
Winter 2023 Lecture 14:
Dynamic Programming V:
Optimal [Offline] Binary Search Trees

Offline Optimal Binary Search Trees

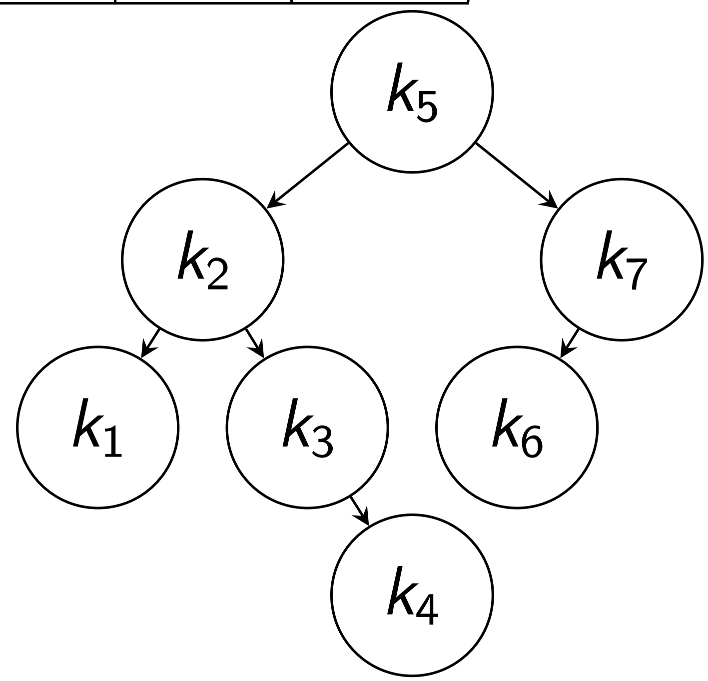
- In ICS 46, you saw "online" search trees
  - Additions happened one at a time
  - Resolve addition before next request
  - Had to maintain "balance"
  - Did not know probability distribution of requests.
- Today we will look at "offline" search trees
  - Know full set of keys at beginning
  - Know probability distribution of requests
  - Want to minimize expected lookup time
  - Even if that means bad lookup for some

Examples of Binary Search Trees

i	1	2	3	4	5	6	7
p_i	.13	.21	.11	.01	.22	.08	.24



E[lookup] $= 2.69$



E[lookup] $= 2.12$

Problem Statement

- Input: n probabilities, $p_1 \ldots p_n$
- p_i is probability of looking up ith key.
- Goal: build binary search tree.
  - Minimize expected lookup cost.

Check for understanding

- Suppose we have d_i (depth of each node)
- Root has $d_i = 1$, its children have $d_i = 2$, etc.
- What is the expected lookup cost of this tree?

$\sum p_i \cdot d_i$

Creating the Dyn Prog Algorithm

Define Tree(i, j): *cost* of opt tree keys i through j

or expected remaining node accesses given we're accessing this [sub]tree

- Base cases:

if $i=j$, return P_i

if $i>j$ return 0

- Which key(s) can be the root of a binary search tree consisting of keys i through j?

$i \le r \le j$ (r is root)

- Cost of BST, rooted at r, has keys i through j?

$\text{Tree}(i, r-1) + \text{Tree}(r+1, j)$

$\cancel{+ \sum_{k=i}^{r-1} P_k + P_r + \sum_{k=r+1}^{j} P_k}$

$+ \sum_{k=i}^{j} P_k$

:) O(1) lookup by PSet 4 Q1

First make recursive solution

Tree(i, j) :

if $j < i$ **then**

  return 0

else if $j = i$ **then**

  return p_i

else

  $r = i$

  min = Tree$(i, r-1)$ + Tree$(r+1, j)$ + $\sum p_k$

included in recursive case anyway.

$\sum_{k=i}^{j} p_k$

O(1) lookup

for r = i+1 ... j

Cost = Tree(i, r-1) + Tree(r+1, j) + Σ

if cost < min

min = cost

Ea case O(n), ∃ O(n²) cases, Total: O(n³)

Iterative Version: Topological Order

- Caution: some recursive calls to *higher* values.
- We can't iterate increasing i and j together.
- Tree$[i, j]$ will make calls to:
  - Tree$[i,\ r-1]$ for $i \leq r \leq j$
  - Tree$[r+1,\ j]$ for $i \leq r \leq j$

- For example, Tree$[2, 5]$ will call:
  - Tree$[2, 1]$ and Tree$[3, 5]$ $(r = 2)$
  - Tree$[2, 2]$ and Tree$[4, 5]$ $(r = 3)$
  - Tree$[2, 3]$ and Tree$[5, 5]$ $(r = 4)$
  - Tree$[2, 4]$ and Tree$[6, 5]$ $(r = 5)$

Table looks like

$\delta=0$ $\delta=1$ $\delta=2$

	k_1	k_2	k_3	k_4	k_5	k_6	k_7
k_1	.13						
k_2		.21					
k_3			.11				
k_4				.01			
k_5					.22		
k_6						.08	
k_7							.24

Fill in Tree(i,j) where $j = i + \delta$

Iterative Version: Memoize the Data

for $i \leftarrow 1 \ldots n$ **do**
  $\text{Tree}[i, i-1] \leftarrow 0$
  $\text{Tree}[i, i] \leftarrow p_i$

for $\delta = 1 \ldots n-1$
  for $i = 1 \ldots n-\delta$
    $j = i + \delta$
    // fill in Tree(i,j)

How to get the tree itself?

	k_1	k_2	k_3	k_4	k_5	k_6	k_7
k_1	0.13	0.47	0.69	0.72	1.28	1.52	2.12
k_2		0.21	0.43	0.46	1	1.17	1.73
k_3			0.11	0.13	0.47	0.63	1.19
k_4				0.01	0.24	0.4	0.95
k_5					0.22	0.38	0.92
k_6						0.08	0.4
k_7							0.24

To be continued...