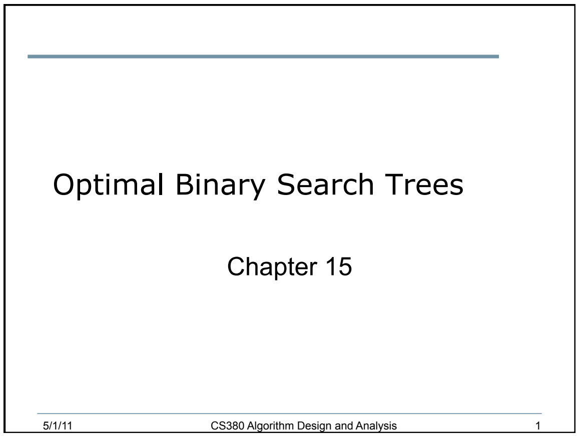
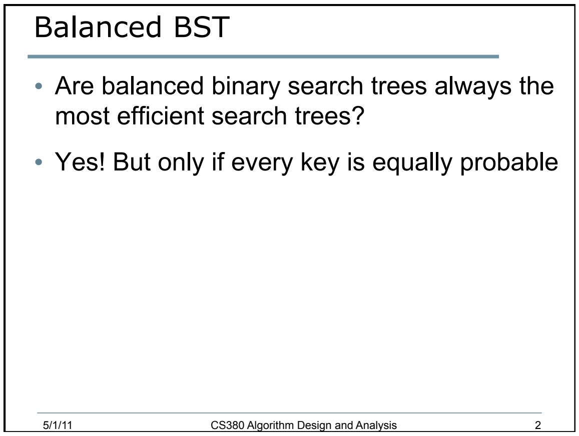


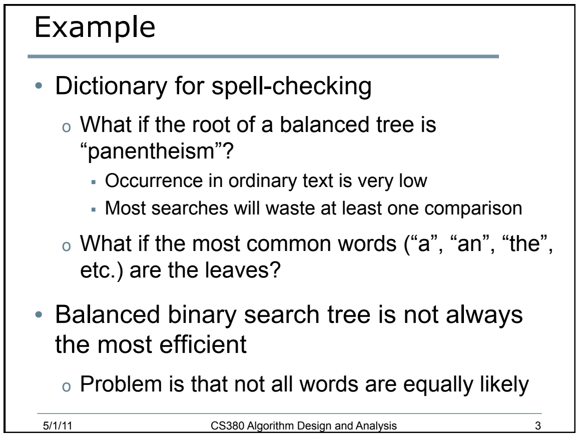
Optimal Binary Search Trees

Chapter 15



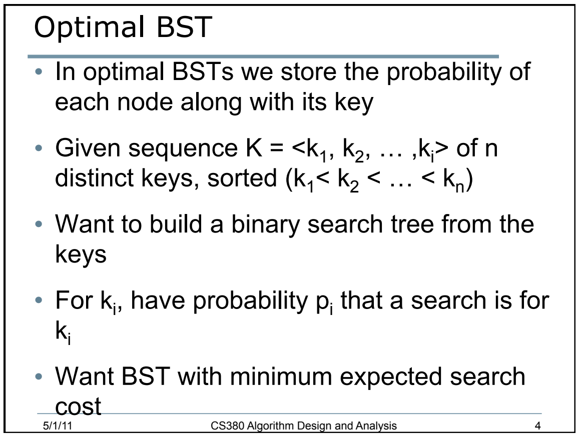
Balanced BST

- Are balanced binary search trees always the most efficient search trees?
- Yes! But only if every key is equally probable



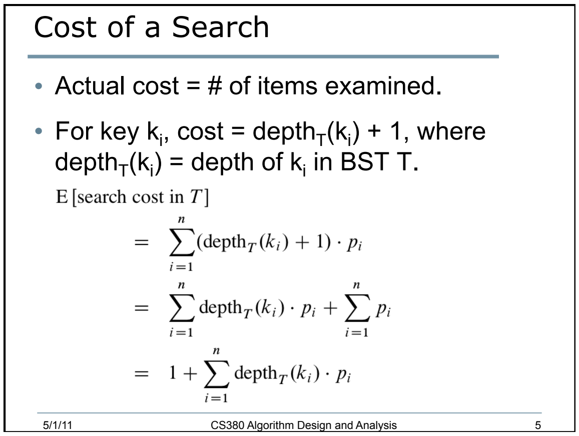
Example

- Dictionary for spell-checking
  - What if the root of a balanced tree is "panentheism"?
    - Occurrence in ordinary text is very low
    - Most searches will waste at least one comparison
  - What if the most common words ("a", "an", "the", etc.) are the leaves?
- Balanced binary search tree is not always the most efficient
  - Problem is that not all words are equally likely



Optimal BST

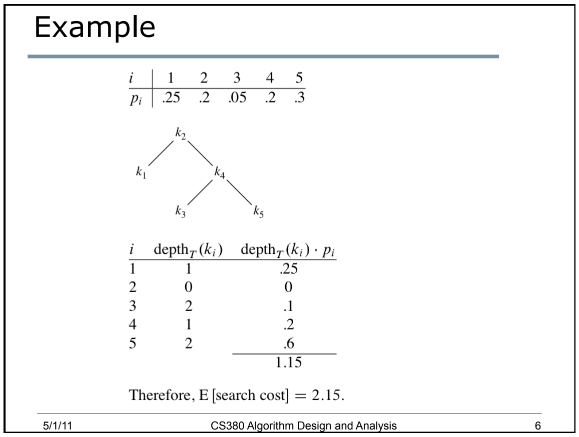
- In optimal BSTs we store the probability of each node along with its key
- Given sequence K = <k_1, k_2, … ,k_i> of n distinct keys, sorted (k_1< k_2 < … < k_n)
- Want to build a binary search tree from the keys
- For k_i, have probability p_i that a search is for k_i
- Want BST with minimum expected search cost



Cost of a Search

- Actual cost = # of items examined.
- For key k_i, cost = $\text{depth}_T(k_i)$ + 1, where $\text{depth}_T(k_i)$ = depth of k_i in BST T.

$$
\begin{aligned}
&\mathrm{E}\,[\text{search cost in } T] \\
&\quad = \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i \\
&\quad = \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=1}^{n} p_i \\
&\quad = 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i
\end{aligned}
$$

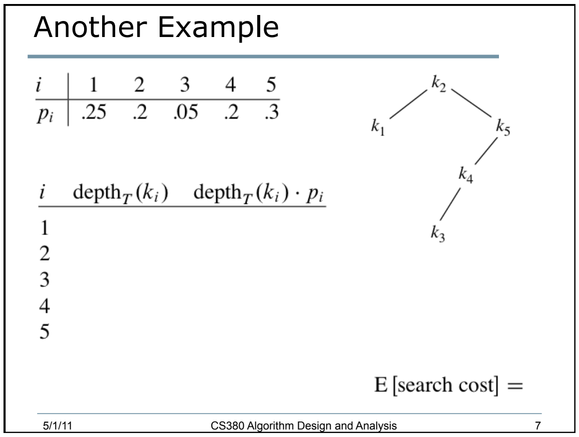


Example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	2	.1
4	1	.2
5	2	.6
		1.15

Therefore, E [search cost] = 2.15.



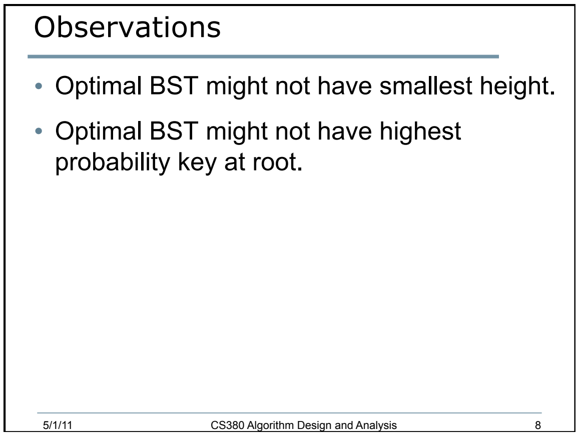
Another Example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

k_2
k_1
k_5
k_4
k_3

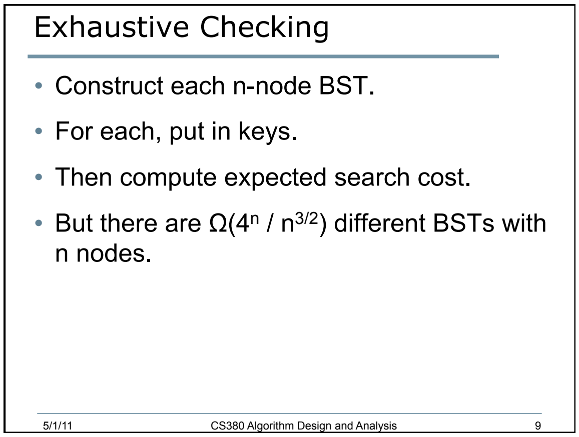
i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1		
2		
3		
4		
5		

$\text{E}\,[\text{search cost}] =$



Observations

- Optimal BST might not have smallest height.
- Optimal BST might not have highest probability key at root.



Exhaustive Checking

- Construct each n-node BST.
- For each, put in keys.
- Then compute expected search cost.
- But there are $\Omega(4^n / n^{3/2})$ different BSTs with n nodes.

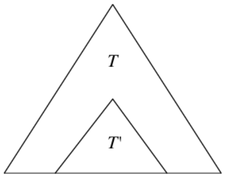
Solution

- Dynamic Programming

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution bottom-up
4. Construct an optimal solution from the computed information

Step 1: Optimal Solution

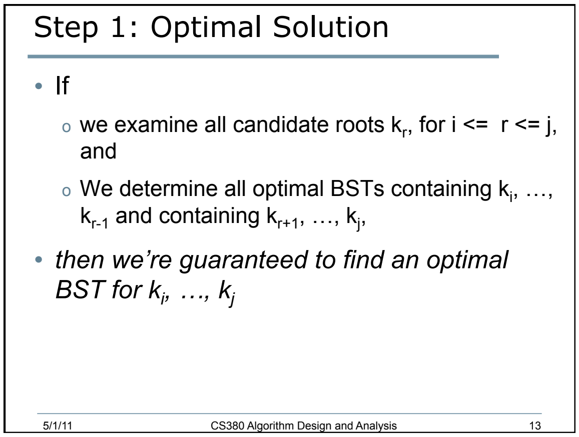
- Consider any subtree of a BST. It contains keys in a contiguous range k_i, …, k_j for some 1 <= i <= j <= n.



If T is an optimal BST and T contains subtree T` with keys $k_i$, …, $k_j$, then T` must be an optimal BST for keys k_i, … k_j.

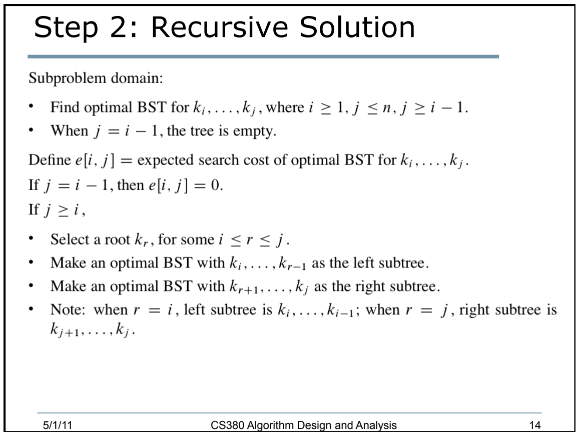
Step 1: Optimal Solution

- Use optimal substructure to construct an optimal solution to the problem from optimal solutions to subproblems:
- Given keys k_i, …, k_j (the problem).
- One of them, k_r, where i <= r <= j, must be the root.
- Left subtree of k_r contains k_i, …, k_{r-1}.
- Right subtree of k_r contains k_{r+1}, …, k_j.



Step 1: Optimal Solution

- If
  - we examine all candidate roots k_r, for i <= r <= j, and
  - We determine all optimal BSTs containing k_i, …, k_{r-1} and containing k_{r+1}, …, k_j,
- *then we're guaranteed to find an optimal BST for k_i, …, k_j*



Step 2: Recursive Solution

Subproblem domain:

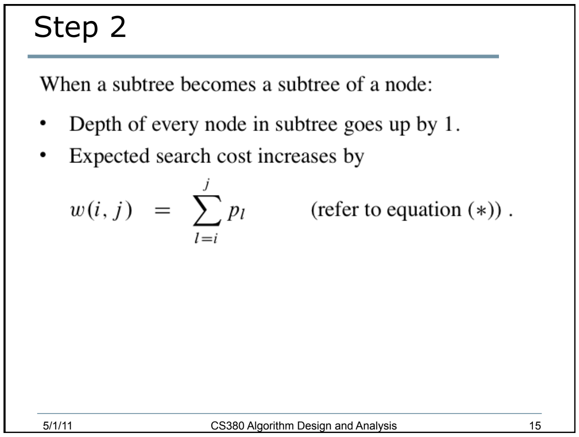
- Find optimal BST for $k_i, \ldots, k_j$, where $i \geq 1$, $j \leq n$, $j \geq i - 1$.
- When $j = i - 1$, the tree is empty.

Define $e[i, j]$ = expected search cost of optimal BST for $k_i, \ldots, k_j$.

If $j = i - 1$, then $e[i, j] = 0$.

If $j \geq i$,

- Select a root k_r, for some $i \leq r \leq j$.
- Make an optimal BST with $k_i, \ldots, k_{r-1}$ as the left subtree.
- Make an optimal BST with $k_{r+1}, \ldots, k_j$ as the right subtree.
- Note: when $r = i$, left subtree is $k_i, \ldots, k_{i-1}$; when $r = j$, right subtree is $k_{j+1}, \ldots, k_j$.

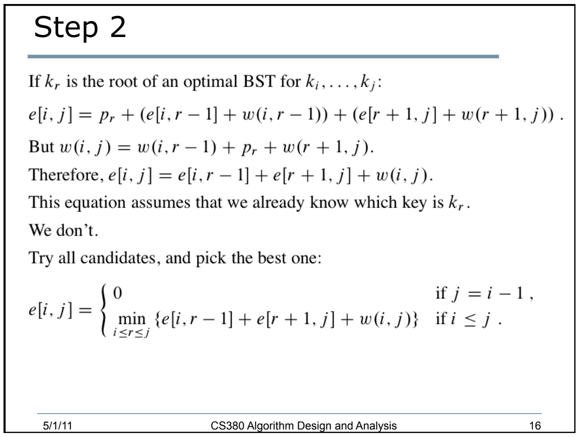


Step 2

When a subtree becomes a subtree of a node:

- Depth of every node in subtree goes up by 1.
- Expected search cost increases by

$$w(i, j) = \sum_{l=i}^{j} p_l \qquad \text{(refer to equation } (*)) .$$



Step 2

If k_r is the root of an optimal BST for $k_i, \ldots, k_j$:

$e[i,j] = p_r + (e[i,r-1] + w(i,r-1)) + (e[r+1,j] + w(r+1,j))$.

But $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$.

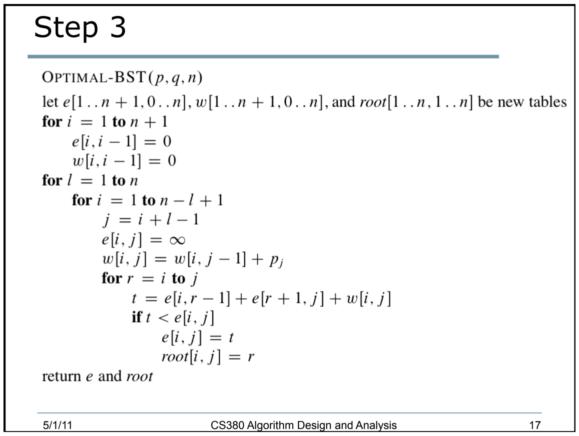
Therefore, $e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$.

This equation assumes that we already know which key is k_r.

We don't.

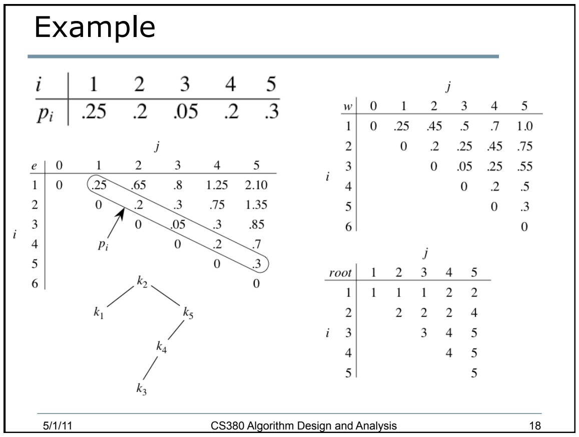
Try all candidates, and pick the best one:

$$e[i,j] = \begin{cases} 0 & \text{if } j = i-1 \,, \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j \,. \end{cases}$$



Step 3

```
OPTIMAL-BST$(p, q, n)$
let $e[1..n+1, 0..n]$, $w[1..n+1, 0..n]$, and $root[1..n, 1..n]$ be new tables
for $i = 1$ to $n+1$
    $e[i, i-1] = 0$
    $w[i, i-1] = 0$
for $l = 1$ to $n$
    for $i = 1$ to $n - l + 1$
        $j = i + l - 1$
        $e[i, j] = \infty$
        $w[i, j] = w[i, j-1] + p_j$
        for $r = i$ to $j$
            $t = e[i, r-1] + e[r+1, j] + w[i, j]$
            if $t < e[i, j]$
                $e[i, j] = t$
                $root[i, j] = r$
return $e$ and $root$
```



Example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

j

	e	0	1	2	3	4	5
	1	0	.25	.65	.8	1.25	2.10
	2		0	.2	.3	.75	1.35
i	3			0	.05	.3	.85
	4		p_i		0	.2	.7
	5					0	.3
	6						0

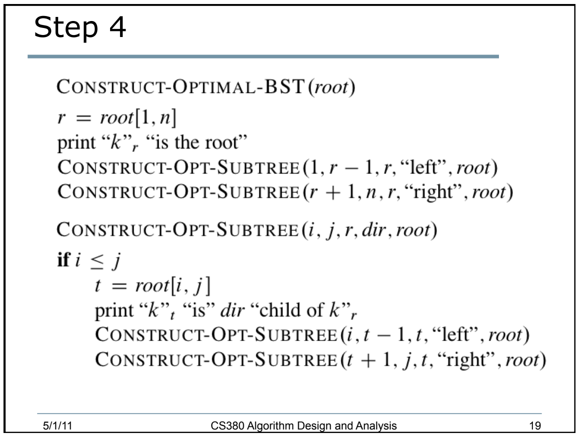
k_2
k_1
k_5
k_4
k_3

j

	w	0	1	2	3	4	5
	1	0	.25	.45	.5	.7	1.0
	2		0	.2	.25	.45	.75
i	3			0	.05	.25	.55
	4				0	.2	.5
	5					0	.3
	6						0

j

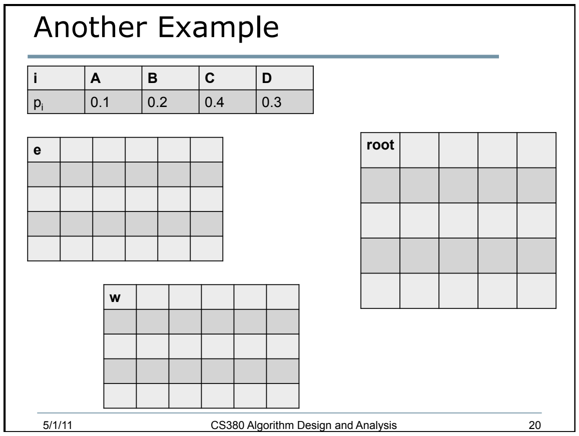
	$root$	1	2	3	4	5
	1	1	1	1	2	2
	2		2	2	2	4
i	3			3	4	5
	4				4	5
	5					5



Step 4

```
CONSTRUCT-OPTIMAL-BST($root$)
$r = root[1, n]$
print "$k$"$_r$ "is the root"
CONSTRUCT-OPT-SUBTREE($1, r-1, r$, "left", $root$)
CONSTRUCT-OPT-SUBTREE($r+1, n, r$, "right", $root$)

CONSTRUCT-OPT-SUBTREE($i, j, r, dir, root$)
if $i \le j$
    $t = root[i, j]$
    print "$k$"$_t$ "is" $dir$ "child of $k$"$_r$
    CONSTRUCT-OPT-SUBTREE($i, t-1, t$, "left", $root$)
    CONSTRUCT-OPT-SUBTREE($t+1, j, t$, "right", $root$)
```



Another Example

i	A	B	C	D
p_i	0.1	0.2	0.4	0.3

e					

root				

w					