Chapter 5, Dynamic Programming:

- Introduction and sample problems (week-before Friday)
- Principles of DP (last week Monday)
- DP algorithms, solutions to sample problems (last week Wednesday
- Coding up DP algorithms (lab Thursday)
- Begin optimal BSTs (**Today**)
- Finish optimal BSTs/review for test (Wednesday)
- [Test 2, Thurs, Apr 4, *not* covering DP]

Today:

- The optimal BST definition
- The optimal-BST-building problem
- The dynamic programming solution

Coming up:

Due **Tues, Apr 2** *(end of day)*
Read Section 6.5
(No quiz on Section 6.5)

Due **Mon, Apr 8** *(end of day)*
Read Sections 7.(1 & 2)
Take quiz

Do **Optimal BST** *project (Due Mon, Apr 8)*

Why this problem?

- It connects dynamic programming with the quest for a better map.
- Its hardness is in the right places (building the table—hard; reconstructing solution—trivial).
- It is a representative of a bigger concept: What if we had more information—how would that change the problem.

Game plan:

- Understand the problem itself
- Understand the recursive characterization
- Understand the table-building algorithm

The **optimal binary search tree** problem:

- Assume we know all the keys $k_0, k_1, \ldots k_{n-1}$ ahead of time.
- Assume further that we know the probabilities $p_0, p_1, \ldots p_{n-1}$ of each key's lookup.
- Assume even further that we know the "miss probabilities" $q_0, q_1, \ldots q_n$ where q_i is the probability that an *extraneous key* falling between k_{i-1} and k_i will be looked up.
- We want to build a tree to minimize the *expected cost* of a look up, which is the *total weighted depth* of the tree:

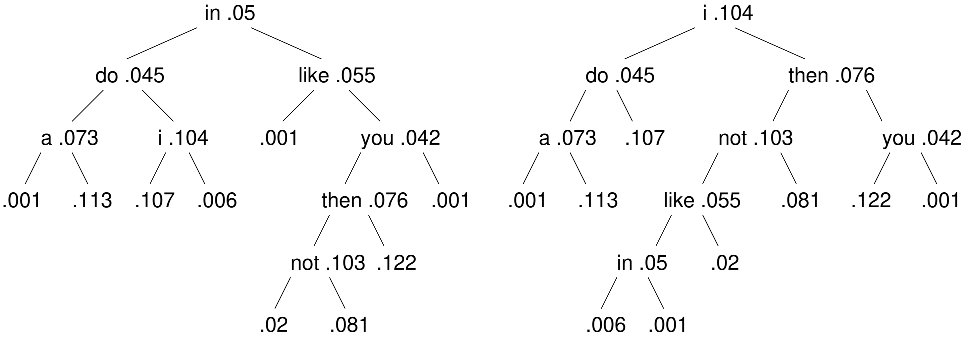
$$\sum_{i=0}^{n-1} p_i \; depth(k_i) + \sum_{i=0}^{n} q_i \; depth(m_i)$$

  where $depth(x)$ is the number of nodes to be inspected on the route from the root to node x, k_i stands for the node containing key k_i [notational abuse], and m_i is the dummy node between keys k_{i-1} and and k_i.
- Note that the rules of probability require $\sum_{i=0}^{n-1} p_i + \sum_{i=0}^{n} q_i = 1$

i	84	eat	24	ham	10	fox	7	rain	4
not	83	will	21	there	9	on	7	see	4
them	61	sam	19	train	9	tree	6	try	4
a	59	with	19	anywhere	8	say	5	boat	3
like	44	am	16	house	8	so	5	that	3
in	40	could	14	mouse	8	be	4	are	2
do	36	here	11	or	8	goat	4	good	2
you	34	the	11	box	7	let	4	thank	2
would	26	eggs	10	car	7	may	4	they	2
and	24	green	10	dark	7	me	4	if	1

Key or miss event	combined frequency
{ }	0
a	59
{ am and anywhere are be boat box car could dark }	92
do	36
{ eat eggs fox goat good green ham here house }	86
i	84
{ if let }	5
in	40
{ }	0
like	44
{ may me mouse }	16
not	83
{ on or rain same say see so thank that the }	65
then	61
{ there they train tree try will with would }	99
you	34
{ }	0

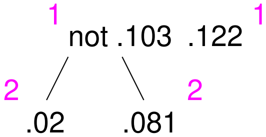
	0	1	2	3	4	5	6	7	
k_i	a	do	i	in	like	not	then	you	
p_i	.073	.045	.104	.05	.055	.103	.076	.042	
q_i	.001	.113	.107	.006	.001	.02	.081	.122	.001



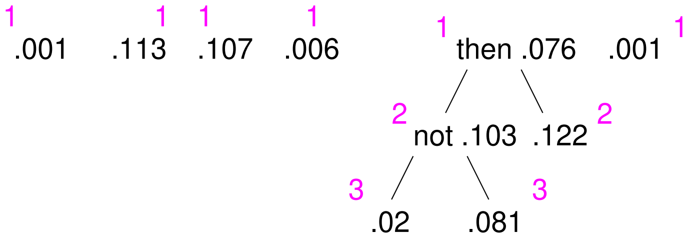
$1 \cdot .02 + 1 \cdot .081$
$= .101$

1 1
.02 .081

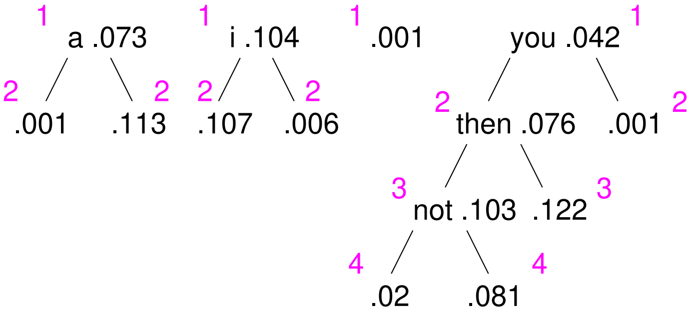
$$
\begin{aligned}
&2 \cdot .02 + 2 \cdot .081 \\
&+1 \cdot .103 + 1 \cdot .122 \\
&= .427
\end{aligned}
$$



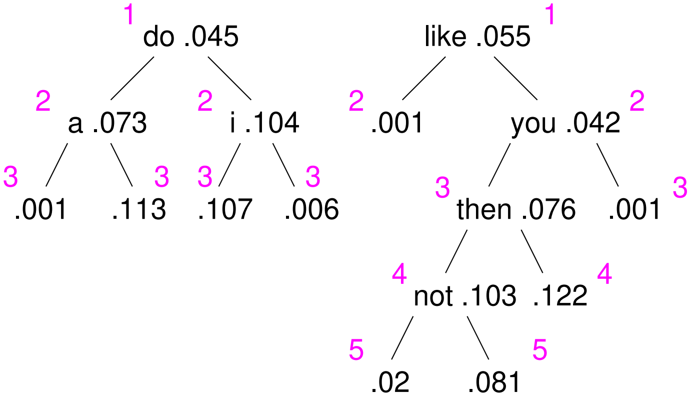
$$
\begin{aligned}
&3 \cdot .02 + 3 \cdot .081 \\
&+2 \cdot .103 + 2 \cdot .122 \\
&+1 \cdot .001 + 1 \cdot .133 + 1 \cdot .107 + 1 \cdot .006 + 1 \cdot .076 + 1 \cdot .001 \\
&= 1.057
\end{aligned}
$$



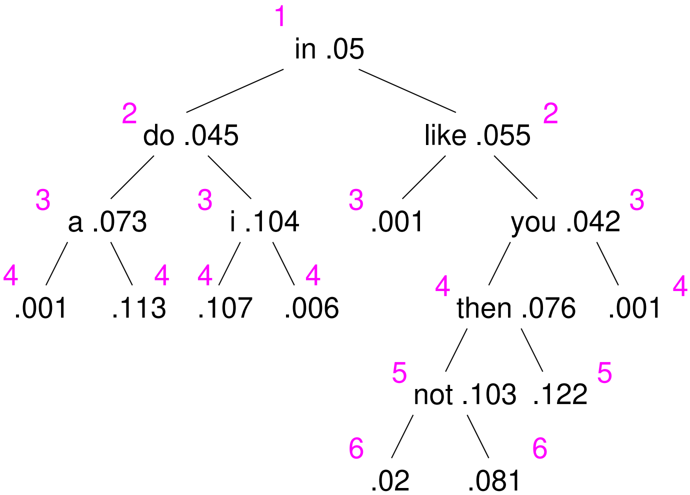
$$
\begin{aligned}
&4 \cdot .02 + 4 \cdot .081 \\
&+3 \cdot .103 + 3 \cdot .122 \\
&+2 \cdot .001 + 2 \cdot .133 + 2 \cdot .107 + 2 \cdot .006 + 2 \cdot .076 + 2 \cdot .001 \\
&+1 \cdot .073 + 1 \cdot .104 + 1 \cdot .001 + 1 \cdot .042 \\
&= 1.907
\end{aligned}
$$



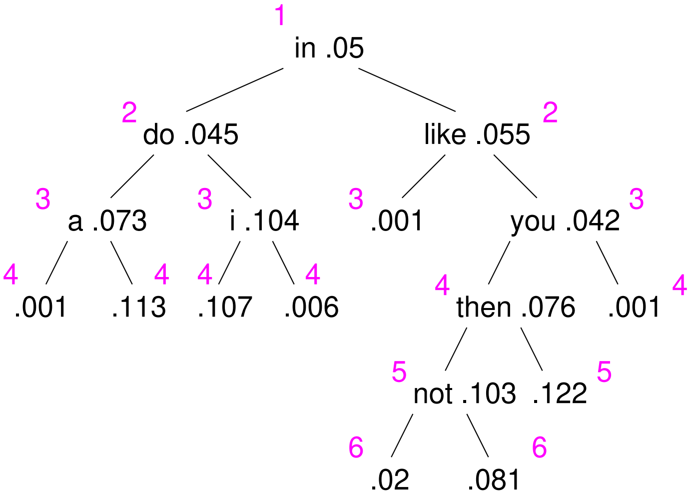
$$
\begin{aligned}
&5 \cdot .02 + 5 \cdot .081 \\
&+4 \cdot .103 + 4 \cdot .122 \\
&+3 \cdot .001 + 3 \cdot .133 + 3 \cdot .107 + 3 \cdot .006 + 3 \cdot .076 + 3 \cdot .001 \\
&+2 \cdot .073 + 2 \cdot .104 + 2 \cdot .001 + 2 \cdot .042 \\
&+1 \cdot .045 + 1 \cdot .055 \\
&= 2.857
\end{aligned}
$$



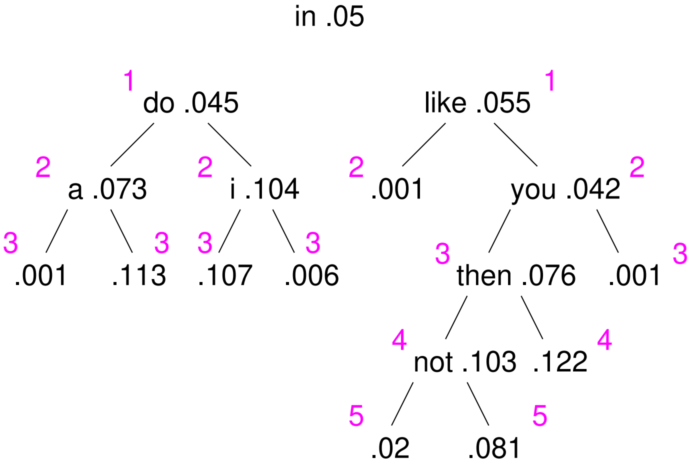
$$
\begin{aligned}
&6 \cdot .02 + 6 \cdot .081 \\
&+5 \cdot .103 + 5 \cdot .122 \\
&+4 \cdot .001 + 4 \cdot .133 + 4 \cdot .107 + 4 \cdot .006 + 4 \cdot .076 + 4 \cdot .001 \\
&+3 \cdot .073 + 3 \cdot .104 + 3 \cdot .001 + 3 \cdot .042 \\
&+2 \cdot .045 + 2 \cdot .055 \\
&+1 \cdot .05 \\
&= 3.857
\end{aligned}
$$



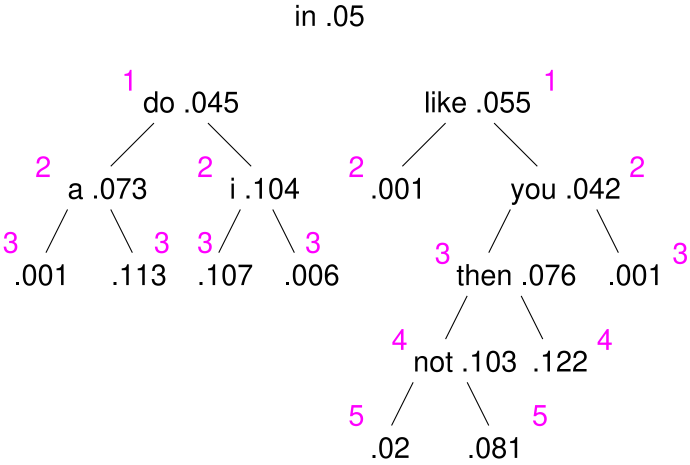
$$4 \cdot .001 + 3 \cdot .073 + 4 \cdot .133 + 2 \cdot .045 + 4 \cdot .107 + 3 \cdot .104 + 4 \cdot .006$$
$$+1 \cdot .05$$
$$+3 \cdot .001 + 2 \cdot .055 + 6 \cdot .02 + 6 \cdot .081 + 4 \cdot .076 + 5 \cdot .122 + 3 \cdot .042 + 4 \cdot .001$$
$$= 3.857$$



$$
\begin{aligned}
&3 \cdot .001 + 2 \cdot .073 + 3 \cdot .133 + 1 \cdot .045 + 3 \cdot .107 + 2 \cdot .104 + 3 \cdot .006 \\
&+ .001 + .073 + .133 + .045 + .107 + .104 + .006 \\
&+ .05 \\
&+ 2 \cdot .001 + 1 \cdot .055 + 5 \cdot .02 + 5 \cdot .081 + 3 \cdot .076 + 4 \cdot .122 + 2 \cdot .042 + 3 \cdot .001 \\
&+ .001 + .055 + .02 + .081 + .076 + .122 + .042 + .001 \\
&= 3.857
\end{aligned}
$$



$$
\begin{aligned}
&3 \cdot .001 + 2 \cdot .073 + 3 \cdot .133 + 1 \cdot .045 + 3 \cdot .107 + 2 \cdot .104 + 3 \cdot .006 \\
&+2 \cdot .001 + 1 \cdot .055 + 5 \cdot .02 + 5 \cdot .081 + 3 \cdot .076 + 4 \cdot .122 + 2 \cdot .042 + 3 \cdot .001 \\
&+.001 + .073 + .133 + .045 + .107 + .104 + .006 \\
&+.05 \\
&+.001 + .055 + .02 + .081 + .076 + .122 + .042 + .001 \\
&= 3.857
\end{aligned}
$$



Total weighted depth for a given tree (expected lookup cost):

$$\underbrace{\sum_{i=0}^{n-1} p_i depth(k_i)}_{\text{keys}} + \underbrace{\sum_{i=0}^{n} q_i \ depth(m_i)}_{\text{misses}}$$

Let $depth_{k_a}(k_i)$ be the depth of the node with k_i in the subtree rooted at node with k_1. For example, if k_r is the root of the entire tree and k_a is a child of the root, then

$$depth_{k_r}(k_i) = depth_{k_a}(k_i) + 1$$

Then we can rewrite the total weighted depth as

$$\underbrace{\sum_{i=0}^{r-1} p_i \ depth_{k_r}(k_i) + \sum_{i=0}^{r} q_i \ depth_{k_r}(m_i)}_{\text{left subtree total weighted depth (absolute)}} + p_r + \underbrace{\sum_{i=r+1}^{n-1} p_i \ depth_{k_r}(k_i) + \sum_{i=r+1}^{n} q_i \ depth_{k_r}(m_i)}_{\text{right subtree total weighted depth (absolute)}}$$

Again, let k_r be the root of the entire tree and k_a and k_b be the root's children. Then

$$\underbrace{\sum_{i=0}^{r-1} p_i(depth_{k_a}(k_i)+1) + \sum_{i=0}^{r} q_i(depth_{k_a}(m_i)+1)}_{\text{left subtree total weighted depth (absolute)}} + p_r + \underbrace{\sum_{i=r+1}^{n-1} p_i(depth_{k_b}(k_i)+1) + \sum_{i=r+1}^{n} q_i(depth_{k_r}(m_i)+1)}_{\text{right subtree total weighted depth (absolute)}}$$

Convert to "relative depth":

$$\underbrace{\sum_{i=0}^{n-1} p_i + \sum_{i=0}^{n} q_i}_{\text{total probability}} + \underbrace{\sum_{i=0}^{r-1} p_i \; depth_{k_a}(k_i) + \sum_{i=0}^{r} q_i \; depth_{k_a}(m_i)}_{\text{left subtree total weighted depth (relative)}} + \underbrace{\sum_{i=r+1}^{n-1} p_i \; depth_{k_b}(k_i) + \sum_{i=r+1}^{n} q_i \; depth_{k_r}(m_i)}_{\text{right subtree total weighted depth (relative)}}$$

Let $TWD(k)$ be the total weighted depth of the tree rooted at k (relative to k) and $TP(k)$ be the total probability of the tree rooted at k. Then

$$TWD(k_r) = TP(k_r) + TWD(k_a) + TWD(k_b)$$

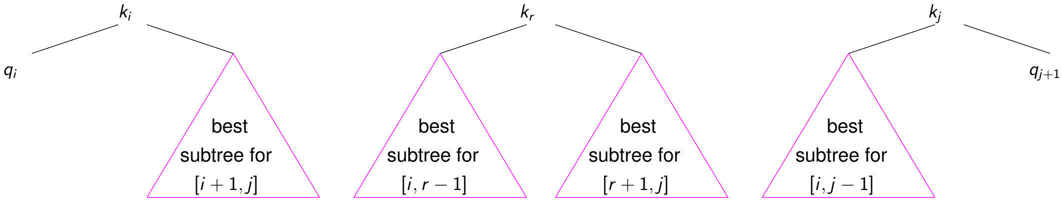
Let $P[i][j]$ be the total probabilities of the keys and misses in the range $[i, j]$:

$$P[i][j] = \sum_{k=i}^{j} p_k + \sum_{k=i}^{j+1} q_k$$

Let $C[i][j]$ be the least total weighted depth of any BST composed from keys in the range $[i, j]$. The recursive characterization is

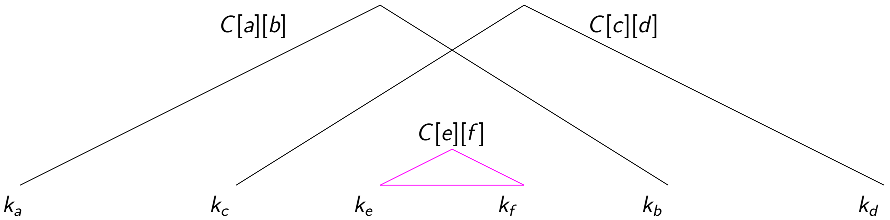
$$C[i][j] = \begin{cases} 2q_i + p_i + 2q_{i+1} & \text{if } i = j \\ \\ P[i][j] + \min \left\{ \begin{array}{l} q_i + C[i+1][j] \\ C[i][r-1] + C[r+1][j] \text{ for } \; r \in (i, j) \\ C[i][j-1] + q_{j+1} \end{array} \right\} & \text{if } i < j \end{cases}$$

$$C[i][j] = \begin{cases} 2q_i + p_i + 2q_{i+1} & \text{if } i = j \\ \\ P[i][j] + \min \left\{ \begin{array}{l} q_i + C[i+1][j] \\ C[i][r-1] + C[r+1][j] \text{ for } \; r \in (i,j) \\ C[i][j-1] + q_{j+1} \end{array} \right\} & \text{if } i < j \end{cases}$$



$q_i + C[i+1][j]$ $\qquad$ $C[i][r-1] + C[r+1][j]$ $\qquad$ $C[i][j-1] + q_{j+1}$

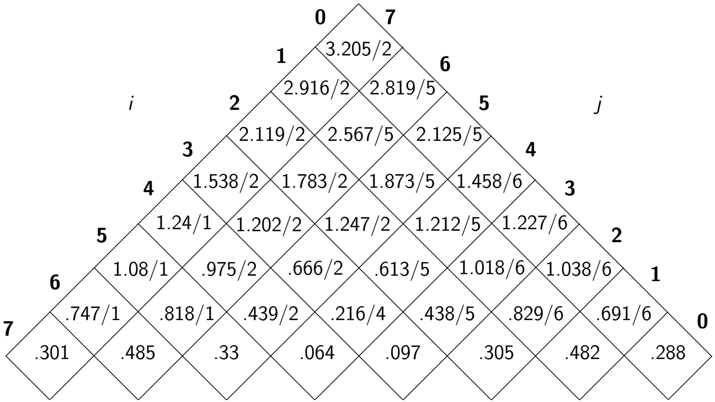
$$C[i][j] = \begin{cases} 2q_i + p_i + 2q_{i+1} & \text{if } i = j \\ \\ P[i][j] + \min \left\{ \begin{array}{l} q_i + C[i+1][j] \\ C[i][r-1] + C[r+1][j] \text{ for } \; r \in (i,j) \\ C[i][j-1] + q_{j+1} \end{array} \right\} & \text{if } i < j \end{cases}$$



$$C[i][j] = \begin{cases} 2q_i + p_i + 2q_{i+1} & \text{if } i = j \\ \\ P[i][j] + \min \left\{ \begin{array}{l} q_i + C[i+1][j] \\ C[i][r-1] + C[r+1][j] \text{ for } \ r \in (i,j) \\ C[i][j-1] + q_{j+1} \end{array} \right\} & \text{if } i < j \end{cases}$$

$$P[i][j] = \begin{cases} q_i + p_i + q_{i+1} & \text{if } i = j \\ \\ \left\{ \begin{array}{ll} & q_i + p_i + P[i+1][j] \\ \text{or} & P[i][r-1] + p_r + P[r+1][j] \text{ for } \ r \in (i,j) \\ \text{or} & P[i][j-1] + p_j + q_{j+1} \end{array} \right\} & \text{if } i < j \end{cases}$$

	0	1	2	3	4	5	6	7	
k_i	a	do	i	in	like	not	then	you	
p_i	.073	.045	.104	.05	.055	.103	.076	.042	
q_i	.001	.113	.107	.006	.001	.02	.081	.122	.001



```
# For each candiate root r between i and j exclusive
for r in range(i+1,j):
    # The cost of making key r the root
    current_subtree_cost = (total_weighted_depths[i][r-1] +
                            total_weighted_depths[r+1][j])
    # If its cost is better than best so far, it's the new best so far
    if current_subtree_cost < least_subtree_cost :
        least_subtree_cost = current_subtree_cost
        best_root = r

# The cost of making key j the root
current_subtree_cost = total_weighted_depths[i][j-1] + miss_probs[j+1]
# If its cost is better than best-so-far, it's the new best-so-far
if current_subtree_cost < least_subtree_cost :
    least_subtree_cost = current_subtree_cost
    best_root = j
# Record the best option and corresponding cost in the tables
total_weighted_depths[i][j] = total_probs[i][j] + least_subtree_cost
decisions[i][j] = best_root
```

From its similarity to the algorithm for optimal matrix multiplication, we recognize the running time for building the tables as $\Theta(n^3)$. See Exercise 6.47 for details.

The value $C[0][n-1]$ in `total_weighted_depths[0][n-1]` gives us the cost of the best tree for the given keys with their probabilities. As with other dynamic programming problems, a more useful result is the tree itself. Exercise 6.48 asks you to write a function that reconstructs the optimal binary search tree using a populated decision table, but for Project 6.2 we have an alternate strategy. Instead of reconstructing the tree after building the table, we build the actual optimal subtrees along with the table. Instead of a table of decisions as in the algorithm above, we maintain a table such that in position (i,j) we store the root of the best subtree for keys k_i through k_j.

Coming up:

Due **Tues, Apr 2** *(end of day)*
Read Section 6.5
(No quiz on Section 6.5)

Due **Mon, Apr 8** *(end of day)*
Read Sections 7.(1 & 2)
Take quiz

Do **Optimal BST** *project (Due Mon, Apr 8)*