optimal binary search tree
also known as OBST

build a BST on given words (or values)
v_1 \ v_2 \ \ldots \ \ v_n \ \text{each with probabilities}
p_1 \ p_2 \ \ldots \ p_n

probabilities: p_1+p_2+ \ldots +p_n =1 \text{ (not strictly necessary)}

goal:
optimize expected search time

\[ \sum p_i \cdot \text{depth}(v_i) \]
example

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>0.3</td>
<td>0.2</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

expected depth:

$0.1 \times 1 + 0.3 \times 0 + 0.2 \times 2 + 0.3 \times 3 + 0.1 \times 1 = 1.5$

expected depth:

$0.1 \times 2 + 0.3 \times 1 + 0.2 \times 2 + 0.3 \times 0 + 0.1 \times 1 = 1.0$
usual approach

try all possible $v_k$ as root

get optimal left and right subtrees recursively

important note: if

is optimal for $abc$, then the extra contribution of that subtree in

is $p_1+p_2+p_3$ (because their depth has increased by one)
subproblem/recurrence

$m[i,j]$ is the expected depth of a node in the OBST for nodes $v_i, v_{i+1}, ..., v_j$

$m[i,i-1] = 0$ -- no nodes
$m[i,i] = 0$ -- one node

$m[i,j] = \min_{i \leq k \leq j} (m[i,k-1] + m[k+1,j] + (p_i + ... + p_{k-1}) + (p_{k+1} + ... + p_j))$
partially worked example

\[
\begin{array}{cccccc}
\text{p}_1 & \text{p}_2 & \text{p}_3 & \text{p}_4 & \text{p}_5 \\
0.1 & 0.3 & 0.2 & 0.3 & 0.1 \\
\end{array}
\]

\[
\begin{array}{cccccc}
m[i,j] & 0 & 1 & 2 & 3 & 4 & 5 \\
1 & 0 & 0 & 0.1 & 0.3 & 0.8 \\
2 & 0 & 0 & 0.2 & 0.6 \\
3 & 0 & 0 & 0.2 & 0.3 \\
4 & 0 & 0 & 0.1 \\
5 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
r[i,j] & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 2 \\
2 & 2 & 3 \\
3 & 4 & 4 \\
4 & 4 \\
\end{array}
\]

\(r[i,j]\) is the minimum \(k\) used by \(m[i,j]\)
before we write code

\[
m[i,j] = \text{MIN}_{i \leq k \leq j}( m[i,k-1] + m[k+1,j] + (p_i + ... + p_{k-1}) + (p_{k+1} + ... + p_j) )
\]

may be hard to calculate because of the sum within the MIN

let \( s[i,j] = p_i + p_{i+1} + ... + p_j \)  
(maybe pre-compute it)

now
\[
m[i,j] = \text{MIN}_{i \leq k \leq j}( m[i,k-1] + m[k+1,j] + s[i,j] - p_k )
\]
code – version 1

for \( i = 1 \) to \( n \)
\[
m[i, i] = m[i, i-1] = 0
\]
\[
s[i, i] = p_i
\]
for \( d = 1 \) to \( n-1 \)
for \( i = 1 \) to \( n-d \)
\[
\quad j = i + d
\]
\[
\quad s[i, j] = s[i, j-1] + p_j
\]
\[
m[i, j] = \text{MIN}_{(i \leq k \leq j)} (m[i, k-1] + m[k+1, j] + s[i, j] - p_k)
\]
fuller code

```plaintext
for i = 1 to n
    m[i,i]=m[i,i-1]=0
    s[i,i]=p_i
for d=1 to n-1
    for i=1 to n-d
        j=i+d
        s[i,j]=s[i,j-1]+p_j
        m[i,j]=m[i,i-1]+m[i+1,j]+s[i,j]-p_i
        r[i,j]=i
        for k = i+1 to j
            exp=m[i,k-1]+m[i+1,j]+s[i,j]-p_k
            if exp<m[i,j] then
                m[i,j]=exp
                r[i,j]=k
return m[1,n]
```

looks like O(n^3) time O(n^2) space

r is used to reconstruct tree