Edit Distance between two strings is the minimum number of insertions, deletions, and substitutions of single characters to turn one string into the other.

\[ \text{FOOD} \rightarrow \text{MOOD} \rightarrow \text{MOND} \rightarrow \text{MOYED} \rightarrow \text{MONEY} \]

E.D. = 4

Levenshtein (coding theory), Ulam (biological sequences), Vintsyuk (speech recognition)
Input: two string A[1..m], B[1..n]
Output: Their edit distance.

Imagine edits by stacking strings. Each column is an edit.

What should we do for the last column? Use backtracking to try all options.
The rest of the columns should represent the edit distance of what remains!

So recursive subproblems are prefixes... encoded as their final indices (i.e., their lengths).

\[ \text{Edit}(i,j): \text{edit distance for } A[1..i] + B[1..j] \]

We want to compute \( \text{Edit}(m,n) \).
Recurrence: Suppose $\omega > 0$, $j > 0$. If our last column is an...

Insertion:

$\text{Edit}(i, j) = 1 + \text{Edit}(i, j - 1)$

Deletion:

$\text{Edit}(i, j) = 1 + \text{Edit}(i - 1, j)$

Substitution (?):

$\text{Edit}(i, j) = \text{Edit}(i - 1, j - 1) + \text{Edit}(i - 1, j) - \text{edit}(i, j - 1)$
\[ [A[i] \neq B[j]] \]

indicator variable

= 1 if proposition is true

0 o.w.

\[ \text{Edit}(i,j) \text{ uses min of those three expressions.} \]

\[ \text{Edit}(0,j) = j \]

\[ \text{Edit}(i,0) = i \]

\[ \text{Edit}(0,0) = 0 \]
$Edit(i, j) = \begin{cases} 
\omega & \text{if } j = 0 \\
\omega & \text{if } i = 0 \\
\min \left\{ 1 + Edit(i, j - 1), \\
1 + Edit(i - 1, j), \\
[AC_i \neq BC_j] + Edit(i - 1, j - 1) \right\} & \text{otherwise} 
\end{cases}$
Dynamic Programming

Subproblems: $0 \leq i \leq m$
$0 \leq j \leq n$

Data structure: 2D Array
$Edit[0..m, 0..n]$

Dependencies:

Eval order:
Space: $O(mn)$, Time: $O(1) \cdot O(mn) = O(mn)$
Can trace reasons for each optimal choice to see sequence of edits in O(m+n) additional time.

**Algorithm**

```
0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9

A   1 - 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8
L   2 - 1 - 0 - 1 - 2 - 3 - 4 - 5 - 6 - 7
T   3 - 2 - 1 - 1 - 2 - 3 - 4 - 5 - 6 - 7
R   4 - 3 - 2 - 2 - 2 - 4 - 3 - 4 - 5 - 6
U   5 - 4 - 3 - 3 - 3 - 3 - 5 - 6 - 5 - 6
I   6 - 5 - 4 - 4 - 4 - 5 - 6 - 3 - 4 - 5 - 6
S   7 - 6 - 5 - 5 - 5 - 4 - 4 - 5 - 5 - 6 - 6
T   8 - 7 - 6 - 6 - 6 - 6 - 4 - 5 - 6 - 6 - 6
I   9 - 8 - 7 - 7 - 7 - 7 - 6 - 6 - 5 - 6 - 6
C   10 - 9 - 8 - 8 - 8 - 8 - 7 - 6 - 6 - 6 - 6
```
Optimal Binary Search Trees

Input: Array of keys $A[1..n]$, that are sorted

Frequencies $f[1..n]$.

Key $A[i]$ will be sought after $f[i]$ times.

How to minimize total search time in a BST over $A$.

($A$ is sorted. Not $f$)
For a given BST $T$: 

$V_1, V_2, ..., V_n$ in order 

$V_i$ store $A[\hat{u}]$

$$\text{Cost}(T, f[1..n]) = \sum_{\hat{u} \in [1..n]} \text{\# ancestors of } V_i \text{ in } T$$

$\hat{u} = 1$ (root has 1 ancestor, itself)

$T$ has a root, a left subtree, right subtree...
$r$: root key

$\text{Cost}(T, f) = \sum_{\omega=1}^{n} \text{Cost}(\text{left}(T), f[1..\omega - 1]) + \sum_{\omega=1}^{n} \text{Cost}(\text{right}(T), f[\omega + 1..n])$

$\text{Cost}(T, f[1..0]) = 0$

Idea: Guess root key $r$ and recurse on smaller and greater keys.
OptCost(\(\omega, k\)) : optimal cost of any tree over \(A(\omega \ldots k)\).

\[
\text{OptCost}(\omega, k) = \begin{cases} 
0 & \text{if } \omega > k \\
\min_{i=\omega}^{k} \left\{ \text{cost of touch root over all searches} \right\} + \min_{\omega \leq r < k} \{ \text{OptCost}(\omega, r-1) + \text{OptCost}(r+1, k) \} & \text{guess for root key} \\
\end{cases}
\]