**Dynamic Programming**

* Divide and Conquer divides the problem into independent subproblems, solve the problem recursively, and combines the solution
* Dynamic Programming is when subproblems are not independent and they share subsubproblems

1. **Knapsack Problem**

Brute Force: check all possible combination of items to find the most valuable

: give n items, there are 2n possible combinations

DP: Takes O(n\*W)

A black text on a white background

AI-generated content may be incorrect.

V = maximum value

k = items

w = knapsack size

bk = benefit/value of kth item

W = maximum knapsack size

Algorithm:

A white background with black text and numbers

AI-generated content may be incorrect.

Rows = elements/items

Columns = max weight

List the information we need:

A white background with black text

AI-generated content may be incorrect.

A table with numbers and numbers

AI-generated content may be incorrect.

**Building the DP table (bottom-up):**

* For each item and each capacity from 0 to W (max capacity),
* If the **item's weight > current capacity**, you **can't take** it → copy the value from the row above.
* If the **item's weight ≤ current capacity**, you decide between:
  + **Not taking** the item → value from the row above (same column).
  + **Taking** the item → value = item’s value + value from the row above, **shifted left** by the item's weight.
* **You pick the maximum** of these two options.

**Recovering which items were selected (traceback):**

* Start from the **bottom-right** of the table (max value).
* Compare the value with the value directly above it:
  + If **equal**, it means the current item **was not included** → move up.
  + If **different**, it means the current item **was included**:
    - Add it to your selection.
    - Move **up one row** and **left by the item's weight** (subtract weight from capacity).
* Repeat until reaching the first row or capacity 0.

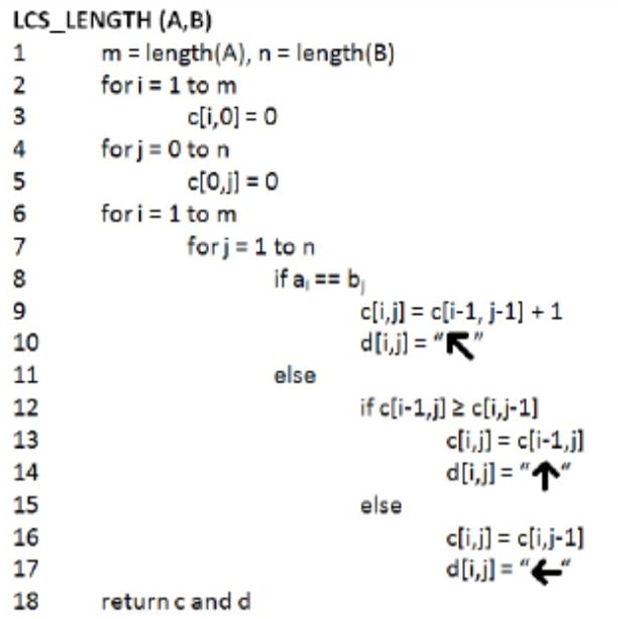
**Longest Common Subsequence**

Brute Force: generate all possible subsequences of A (length m) and check which of them are also subsequences of B (length n) and then retain the longest subsequence

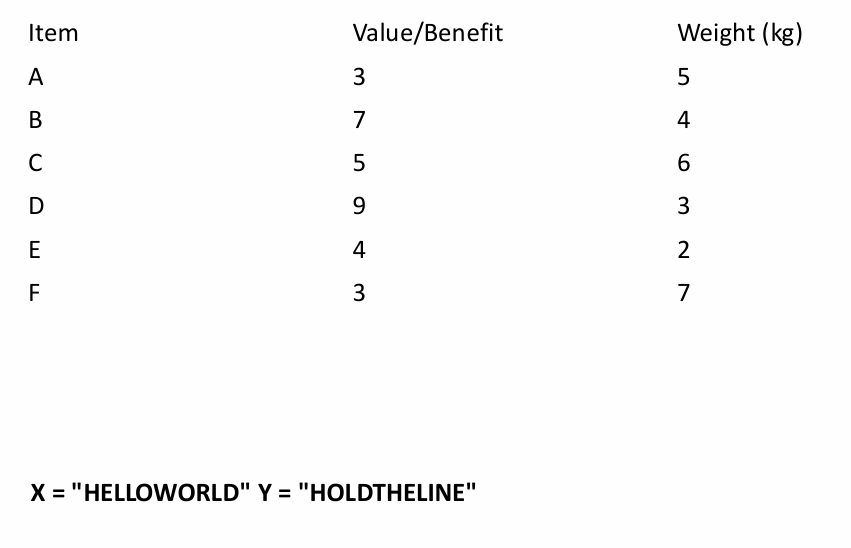
: all possible sequences of A = 2n – 1

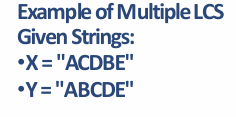
DP: the diagonals will be the matching stringsA screenshot of a graph

AI-generated content may be incorrect.



Practice Exercise:



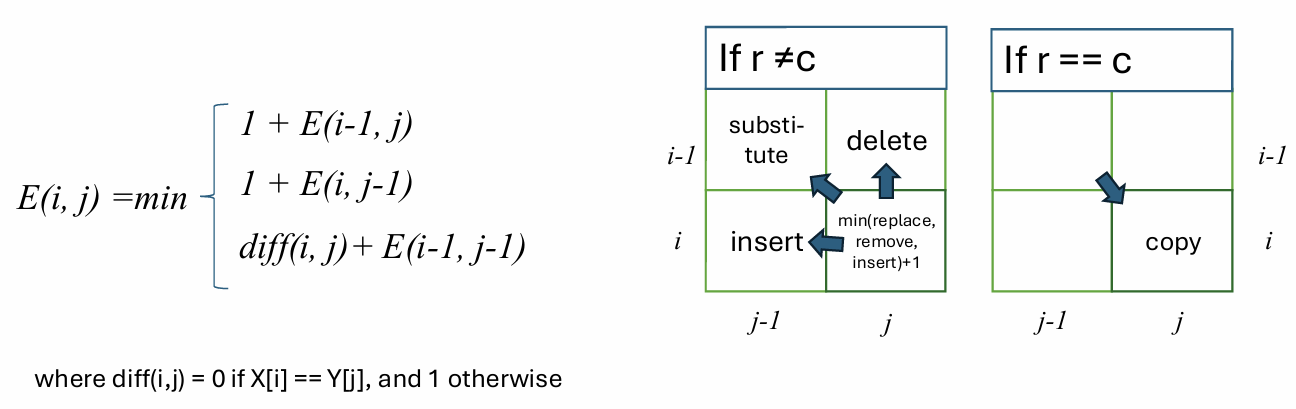


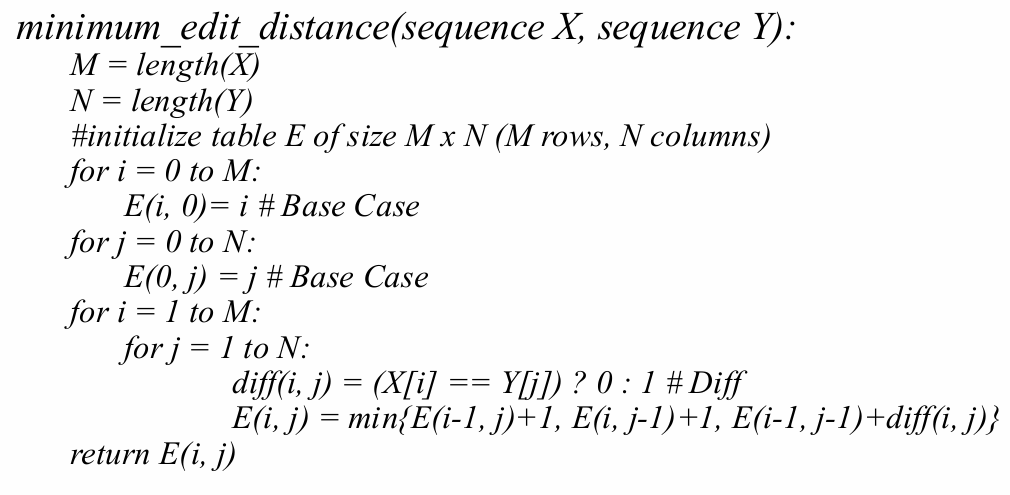
**Minimum Edit Distance**

Brute Force: generate all possible alignments for 2 sequences and compute cost

DP: deletion, insertion, replacement, copy



****

****

**Longest Increasing Subsequence**

Brute Force: Find all increasing subsequences of A and return the greatest length

: O(2n) for checking each subsequence and if it’s is increasing

A white background with black text

AI-generated content may be incorrect.

DP O(n2) : LIS of the shorter sequence is also part of the LIS of longer sequence

A screenshot of a computer

AI-generated content may be incorrect.

A black text with black text

AI-generated content may be incorrect.