Edit Distance Problem

- Given two strings x and y what is the cheapest way to convert x into y
- By conversion we mean character edits:
  - Insert character c
  - Delete character c
  - Replace caracter c with c'
- Each one of these operations has a certain cost that is also a function of the character being inserted, deleted or replaced.
- The cost information is usually previously provided in a table
- Used on auto-correct string features and comparing DNA sequences
  spelling corrections

- Another problem encompassed by Edit Distance is the longest common subsequence problem, e.g.:

X string: HIEROGLYPHOLOGY
Y string: MICHAELANGELO

Question: What is the longest common subsequence?

Not substring! ⚠
- We are allowed to drop any set of letters from X.
- " " " " " " " " " " Y.
- We are not allowed to replace any characters
- I want X and Y in the end to be equal

- Longest common subsequence in the example:

$$\left.\begin{array}{l}\text{HIEROGLYPHOLOGY}\\ \text{MICHAELANGELO}\end{array}\right\}\ \text{HELLO}$$

- We can model the longest common subsequence problem as an edit distance problem in the following way:
  - cost of insert / delete = 1
  - cost of replace = $\begin{cases} 0, & \text{if } c = c' \\ \infty, & \text{otherwise} \end{cases}$
- We want to minimize the number of insertions and deletions
- Lets focus on the more general edit distance problem
- Ideally we should be able to use our "subproblems for strings" strategies, i.e. {suffixes, prefixes, substrings}
- But now we have two strings, which is annoying
- Lets see how we can solve this:

① subproblem = solve edit distance problem on $X[i:]$ and $Y[j:]$ $\forall i, j$

\# subproblems = n choices for X suffixes and n choices for Y suffixes $= n \times n = n^2 \Rightarrow \Theta(n^2)$

If x and y have different lengths then:

$\Theta(|x| \cdot |y|)$

② Guess / Try:

string x: i

string y: j

Question:
I want to convert x into y. What should I look into here?

- Lets focus on the first charackter.

Question:
What are the possible ways to deal with the first character of x?

Objective:
- I want the first character of x to become the first character of y

Possibility 1
- I can delete the first character of x

Possibility 2
- I can replace the first character of x with the first character of y

Possibility 3
- I can insert the first character of y immediately before the first character of x

∴ We need to guess/try one of 3 possibilities

- replace $x[i] \to y[j]$
- insert $y[j]$
- delete $X[i]$

③ Recurrence

$DP(i, j) = \min($

suffix of u, suffix of y

Option 1: "cost of replace $u[i] \rightarrow y[j]$" $+ DP(i+1, j+1)$,

Option 2: "cost of insert $y[j]$" $+ DP(i, j+1)$,

Option 3: "cost of delete $u[i]$" $+ DP(i+1, j)$

$)$

④ Recurse + memoize – nothing to be said here

⑤ $DP(0, 0)$

Question:

What is the running time?

#subproblems : $\Theta(|X| \times |Y|)$

time/subproblem : Each cost can be looked up on a table $\Rightarrow \Theta(1)$ time for the lookup.

Total Time $=$ # subproblems $\times$ time/subproblem

$= \Theta(|X| \times |Y|) \times \Theta(1)$

$= \Theta(|X| \times |Y|) \cancel{=} \Theta(n^2)$ if we assume $|X| = |Y| = n$

Knapsack Problem:

- List of n items each with a size s_i and a value v_i
- The sizes are integers
- Knapsack of size S

Objective:

- We want to maximize sum of values for a subset of items of total size $\leq S$

Question: How do we do this with DP?

Answer: With difficulty...

① Sub problems:

- We can look at this problem as a sequence of items, even though the order is not important
- This allows us to use the "subproblems for strings" strategies
  - suffix
  - prefix
  - substring
- Lets see with suffix $i:$ of items. This is helpful because we can choose items from the beginning
- What should I decide with the i^{th} item
  - Should item i be included or not?
- We also need to tell the recursion how much available space we will have in the knapsack if we decide to take the i...

∴ – suffix of items i:

– remaining capacity $X \leq S$

$\#$subproblems $= n \cdot S \Rightarrow \Theta(nS)$

② Guess / Try: is item i in subset or not?

$\#$Guesses $= 2$ choices (yes/no) $\Rightarrow \Theta(1)$ time

③ Recurrence:

$DP(i, X) = \max($

Option 1: "Don't include the item"

$DP(i+1, X)$,

Option 2: "Include item i"

$DP(i+1, X - s_i) + v_i$

$)$

Total Time $= \#$subproblems $\times$ time/subproblem

$= \Theta(n \cdot S) \times \Theta(1) = \Theta(n \cdot S)$

Question:

$\boxed{\text{Is this polynomial time ?}}$

↳ Not polynomial time!

↳ Polynomial time ⇒ polynomial of the size of the input

size of the input here is n and S

Question: Why? Any ideas?

– We would need $\log_2 S$ bits to encode all possible input sizes

– S is exponential in $\log S$ ⇒ exponential time algorithm !!

1 bit ⇒ 2^1 combinations

2 bits ⇒ 2^2 "

- The algorithm seems polynomial but it is not.
- Well if $\underline{S}$ is small then the algorithm can be seen as polynomial
- Otherwise if $\underline{S}$ is large then we have exponential time
- We call this behaviour $\boxed{\text{pseudo polynomial time}}$
- In an abuse of notation:

$$\text{polynomial time} < \text{pseudo polynomial time} < \text{exponential time}$$

15.4 Longest Common Subsequence Problem [Cormen 2001]

Question: How "similar" two ~~strings~~ sequences are ?

- Different measures of similarity:
  - Substring
  - Number of changes to turn one string into another
  - Subsequences

- Subsequence of a sequence is just the given sequence with zero or more elements left out. Formally:

  Definition: Given a sequence $X = \langle x_1, x_2, \dots, x_m \rangle$ another sequence $Z = \langle z_1, z_2, \dots, z_k \rangle$ is a subsequence of X if there exists a strictly increasing sequence $\langle i_1, i_2, \dots, i_k \rangle$ of indices of X such that for all $j = 1, 2, \dots, k$ we have $x_{i_j} = z_j$

- E.g. $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, \dots B, D, A, B \rangle$ with corresponding index sequence $\langle 2, 3, 5, 7 \rangle$

- Given two sequences $\underline{X}$ and $\underline{Y}$ we say that a sequence $\underline{Z}$ is a common subsequence of $\underline{X}$ and $\underline{Y}$ if $\underline{Z}$ is a subsequence of both $\underline{X}$ and $\underline{Y}$
  - E.g.:
    - $X = \langle A, B, C, B, D, A, B \rangle$
    - $Y = \langle B, D, C, A, B, A \rangle$

    A common subsequence: $\langle B, C, A \rangle$

    Longest common subsequences:
    - $\langle B, C, B, A \rangle$
    - $\langle B, D, A, B \rangle$

15.4.1 Characterizing a longest common sequence

- Brute-force approach:
  - Enumerate all possible subsequences of $\underline{X}$ and check each subsequence to see if it is also a subsequence of $\underline{Y}$, keeping track of the longest subsequence found
  - Each subsequence of $\underline{X}$ corresponds to a subset of the indices $\{1, 2, \ldots, m\}$ of $\underline{X}$
  - $\boxed{\text{Question:}}$ $\boxed{\text{How many possible subsequences of } \underline{X} \text{ exist?}}$

    $X = \langle x_1, x_2, \ldots, x_m \rangle$

    - x_1 may or may not be in the sequence (yes/no answer)
    - x_2 may or may not be in the sequence (yes/no answer)
    - ...
    - x_m may or may not be in the sequence (yes/no answer)

$\therefore_1$ 2 possible choices for x_1
2 possible " " x_2
⋮
2 " " " x_m

Implies that there are: $\underbrace{2 \times 2 \times \ldots 2}_{m \text{ times}} = 2^m$

$\therefore_2$ Exponential #subsequences for $\underline{X}$ (2^m)

$\therefore_3$ Exponential #subsequences for $\underline{Y}$ (2^k)

$\therefore_4$ For each subsequence of $\underline{X}$:

For each subsequence of $\underline{Y}$:

Check if $x \stackrel{?}{=} y$

I.e $2^m \times 2^k = 2^{mk} = 2^{2m}$ if $|Y| = |X|$

Theorem 15.1 – Optimal substructure of an LCS

- Let $X = \langle x_1, x_2, \dots, x_m \rangle$
  $Y = \langle y_1, y_2, \dots, y_n \rangle$

  be sequences
- Let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be an LCS of X and Y
- Then:

  Prefix AB

  1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}
  2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y
  3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}

Notes:

- Given a sequence $X = \langle x_1, x_2, \dots, x_i \rangle$ we define the ith prefix of X, for $i = 0, 1, \dots, m$ as $X_i = \langle x_1, x_2, \dots, x_i \rangle$
- E.g. $X = \langle A, B, C, B, D, A, B \rangle$ then $X_4 = \langle A, B, C, B \rangle$ and X_0 is the empty subsequence

Proof:

(1). If $z_k \neq x_m$ ~~then we could append $x_m = y_n$~~ then we could append $x_m = y_n$ to Z to obtain a common subsequence of X and Y of length $k+1$

- This would be a contradiction that Z is an LCS. Thus we must have that $z_k = x_m = y_n$
- Now, the prefix Z_{k-1} is a length $(k-1)$ common subsequence of X_{m-1} and Y_{n-1}. We wish to show that it is an LCS.

  Y_{n-1}
- Suppose for the purpose of contradiction that there is a common subsequence W of X_{m-1} and Y_{n-1} with length greater than $k-1$
- Then appending $x_m = y_n$ to W produces a common subsequence of X and Y whose length is greater than k, which is a contradiction

(2) If $z_k \neq x_m$ then Z is a common subsequence of X_{m-1} and Y. If there were a common subsequence W of X_{m-1} and Y with length greater than k then W would also be a common subsequence of X_m and Y, contradicting the assumption that Z is an LCS of X and Y

(3) The proof is symmetric to (2)

■ (end of proof)

15.4.2 - Recursive Solution

- Theorem 15.1 implies that there are either one or two subproblems to examine
- If $x_m = y_n$ we must find LCS of X_{m-1} and Y_{n-1}. Appending $x_m = y_n$ to this LCS yields an LCS of X and Y
- If $x_m \neq y_n$ then we must solve two subproblems:

  Subproblem 1: Finding an LCS of X_{m-1} and Y
  Subproblem 2: Finding an LCS of X and Y_{n-1}

  Whichever of these two LCSs is longer is an LCS of X and Y.
- Define $c[i,j]$ to be the length of an LCS of sequences X_i and Y_j. If either $i=0$ or $j=0$ one of the sequences has length 0 so the LCS has length 0:

$$c[i,j] = \begin{cases} 0 & , \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1] & , \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & , \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

- #subproblems:
  - We need to check each prefix of X — there are m
  - We need to check each prefix of Y — there are n
  - Total number of subproblems $m \cdot n$
- Total time = #subproblems x time/subproblem $= \Theta(m \cdot n) \cdot \Theta(1) = O(m \cdot n)$

15.4.3 Computing the Length of an LCS

- For once lets see how we can calculate an iterative procedure

- LCS-Length(X, Y)

```
  m = X.length
  n = Y.length
  let { b[1..m, 1..n]
      { c[0..m, 0..n]   be new tables   /* Table b points to the optimal subproblem solution */

  for i = 1 to m
      c[i, 0] = 0
  for j = 0 to n
      c[0, j] = 0

  for i = 1 to m
      for j = 1 to n
          if xi == yj
              c[i, j] = c[i-1, j-1] + 1
              b[i, j] = "↖"
          elseif c[i-1, j] >= c[i, j-1]
              c[i, j] = c[i-1, j]
              b[i, j] = "↑"
          else c[i, j] = c[i, j-1]
              b[i, j] = "←"
// c[m, n] contains the length of an LCS of X and Y
```

<u>Example for strings:</u>

$X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

i \ j	0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A
0 x_i	0	0	0	0	0	0	0
1 A	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖
2 B	0	1 ↖	1 ←	1 ←	1 ↑	2 ↖	2 ←
3 C	0	1 ↑	1 ↑	2 ↖	2 ←	2 ↑	2 ↑
4 B	0	1 ↖	1 ↑	2 ↑	2 ↑	3 ↖	3 ←
5 D	0	1 ↑	2 ↖	2 ↑	2 ↑	3 ↑	3 ↑
6 A	0	1 ↑	2 ↑	2 ↑	3 ↖	3 ↑	4 ↖
7 B	0	1 ↖	2 ↑	2 ↑	3 ↑	4 ↖	4 ↑

- To reconstruct the elements of an LCS follow the $b[i,j]$ from the lower right corner. Each ↖ on the shaded sequence corresponds to an entry for which $x_i = y_j$ is a member of an LCS

An LCS = $\langle B, C, B, A \rangle$