

Longest Common Subsequence

Kristian Guillaumier

Substrings and subsequences

- Substring:
 - **Contiguous sequence** (**in order**) of characters or symbols in a sequence/string.
- Subsequence:
 - **Non-contiguous** sequence (**also in order**) of characters or symbols in a sequence/string.
 - Equivalently, a subsequence is obtained by deleting zero or more symbols/characters in a sequence.

Some examples

- Substrings of HelloWorld:

- **Hel** HelloWorld
- **loWo** HelloWorld
- **Hello** HelloWorld
- **oWorld** HelloWorld

- Subsequences of HelloWorld:

- **Hll** HelloWorld
- **HWrd** HelloWorld
- **Hello** HelloWorld (every substring is also a subsequence, not vice-versa)
- **ε** HelloWorld (the empty string is a subsequence and a substring)
- **HelloWorld** HelloWorld (the string itself is a subsequence and a substring of itself)

The longest common subsequence

- We have two strings or sequences X and Y of **varying lengths**.
- $X = X[1..n]$ and $Y = Y[1..m]$ where **n** and **m** are the lengths of the strings.
- We must find **a longest subsequence** in X and Y.

- X: 3452345
 - Y: 4541534
- } In this example, the strings just happen to have the same length.
- One possible LCS(X,Y) is:
 - X: 3452345
 - Y: 4541534
 - $\text{LCS}(X,Y) = 4545$ and another one is 4534.
- } The LCS is not unique – its length is.

Dynamic programming

- Like divide and conquer, dynamic programming is a mathematical technique for ‘breaking down’ a problem into smaller parts (you should be familiar with the method).
- It can be used to solve optimisation problems.
- Here will use the **Longest Common Subsequence** (LCS) to illustrate DP.
- For example, finding common sequences in DNA sequences or online file backups.

A brute force method

- Create every subsequence in X and check if it is a subsequence of Y.
- Keep track of longest found so far.
- So, for the string X:3452345 generate all the subsequences:

ϵ , 3, 4, 5, ..., 34, 35, 32, ..., 345...

- ...and check each one in the other string Y.

...continued

- How many steps does it take to check if a string is a subsequence of another?
- E.g., to check if 544 is a subsequence of 4541534.
- At worse $|4541534| = 7$ steps.
- So, to check if a string is a subsequence in $Y[1..m]$, we need $O(m)$ time.

...continued

- How many checks need to be made (how many subsequences can be generated from X)?
- Answer is 2^n .
- To show this, let's use a bit string of length n (as long as the string X). Every bit set to 1 indicates a subsequence character selection in X .
- For example, ...

...continued

0000000 = 3452345 = ϵ

0000001 = 3452345

0000010 = 3452345

...

0000011 = 3452345

...

1111111 = 3452345

There are 2^n of these. So, the worst-case running time is $O(m2^n)$.

An idea for improvement

- **Step 1:** Find the **length** of the $\text{LCS}(X, Y)$.
- Note that we **are finding the length only not the actual LCS**.
- The length of $\text{LCS}(X, Y)$ is **unique**.
- **Step 2:** Find the actual $\text{LCS}(X, Y)$ knowing this length – this greatly simplifies the problem.
- Let's consider the prefixes of X and Y .
- We'll define:

$$c[i, j] = |\text{LCS}(X[1..i], Y[1..j])|$$

- For example, $c[2, 3]$ is the length of the LCS between the first 2 characters of X and of the first 3 characters of Y .

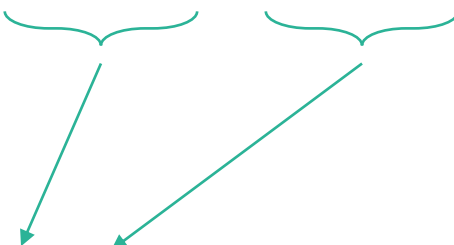
...continued

- Note that using:

$$c[i,j] = |\text{LCS}(X[1..i], Y[1..j])|$$

- The length of the entire LCS would be given when $i=n$ and $j=m$:

$$c[n,m] = |\text{LCS}(X[1..n], Y[1..m])|$$


$$c[n,m] = |\text{LCS}(X, Y)|$$

Theorem

$c[i,j]$ is

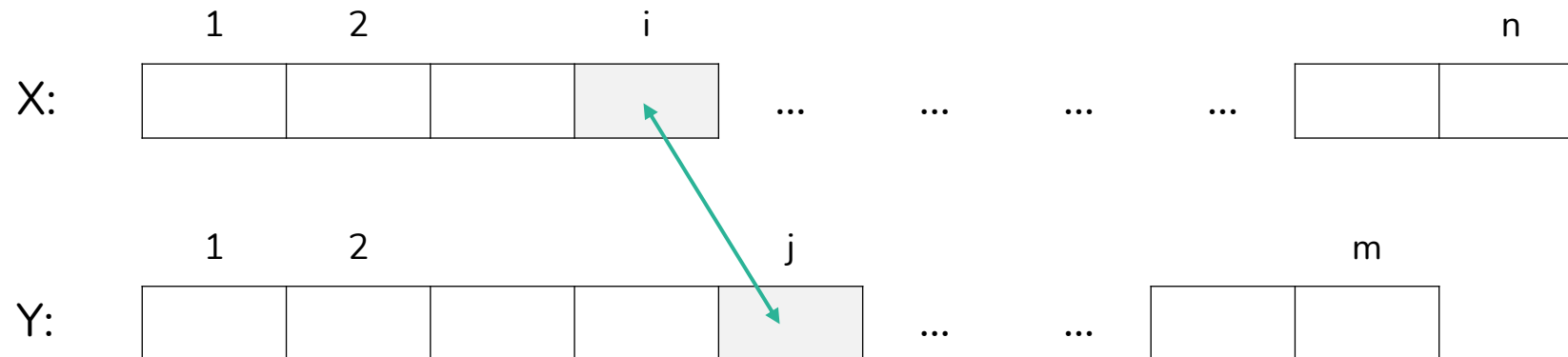
$= c[i-1,j-1] + 1$ if $x[i] = y[j]$

$= \max(c[i, j-1], c[i-1, j])$ otherwise

Remember that we did something like this in edit string distance.

		0	1	2	3	4	5
		ε	w	h	a	l	e
0	ε						
1	h						
2	e						
3	l						
4	l						
5	o						

Proof for case $X[i]=Y[j]$



When these characters $x[i]$ and $y[j]$ are the same.

...continued

- Theorem for case 1:

- $c[i, j] = c[i-1, j-1] + 1$ for $X[i]$ is equal to $Y[j]$.

- Let:

- $k = c[i, j]$ be the length of the LCS in the prefix of X and prefix of Y
- $Z[1..k] = \text{LCS}(X[1..i], Y[1..j])$

k is the length of the LCS in the prefix of X and prefix of Y



Z is the actual LCS in the prefix of X and prefix of Y



Example

$X = 3452345$

$Y = 43141534$

$i = 2$

$j = 4$

Remember that $X[i]$ must be the same as $Y[j]$ in this case: $X[2] = Y[4]$, “4” = “4”.

$c[2,4]$

$= |\text{LCS}(X[1..2], Y[1..4])|$

$= |\text{LCS}(\text{“34”}, \text{“4314”})|$

$= |\text{“34”}|$

$= 2$

So, the length of the LCS up to i in X and j in Y is 2,
and the actual LCS Z is “34”

...continued

X = 3452345
Y = 43141534
i = 2
j = 4

$c[2,4] = |\text{LCS}(X[1..2], Y[1..4])|$
 $= |\text{LCS}("34", "4314")|$
 $= |"34"|$
 $= 2$

- Remember we let:
 - $c[i,j] = k$
 - $Z[1..k] = \text{LCS}(X[1..i], Y[1..j])$
- We observe that:
 - $Z[k] = X[i] = Y[j]$
 - "4" = "4" = "4"

} k^{th} character in the LCS Z, i^{th} character in X, and j^{th} character in Y must be the same!
- Thus:
 - $Z[1..k-1]$ is a common subsequence (CS) of $X[1..i-1]$ and $Y[1..j-1]$.
 - In our example, "3" is a CS of "3" and "431" which is true.
- We will now show that not only $Z[1..k-1]$ is a CS but also an LCS.

...continued

- Claim:
 - Let $Z[1..k]$ be a the LCS having length k of the strings $X[1..i]$ and $Y[1..j]$.
 - $Z[1..k-1]$ is not only a CS in $X[1..i-1]$ and $Y[1..j-1]$ but also an LCS.
 - Suppose a sequence w exists which is a longer CS in $X[1..i-1]$ and $Y[1..j-1]$:
 - i.e., $|w| > k-1$
 - Argument:
 - Concatenate that sequence w with $Z[k]$ giving $w+Z[k]$.
 - If $|w| > k-1$
 - Then $|w + Z[k]|$ would be greater than k – contradicts our initial hypothesis.
- Remember that $Z[k]$ is the last character in the LCS of $X[1..i]$ and $Y[1..j]$.

...continued

- Since we showed that $Z[1..k-1]$ is an LCS of $X[1..i-1]$ and $Y[1..j-1]$.
- Then $k-1$ is the length of the LCS of $X[1..i-1]$ and $Y[1..j-1]$.
- Thus...
 - $c[i-1, j-1] = k-1$
 - And k is $c[i, j]$, so...
 - $c[i, j] = c[i-1, j-1] + 1$.
- Proven first case.

...continued

- We have shown that if $Z = \text{LCS}(X, Y)$
- Then any prefix of $Z = \text{LCS}$ (in a prefix of X , a prefix of Y).

This is what we would call the “optimal substructure” property in DP. We’ll talk about this later.

...continued

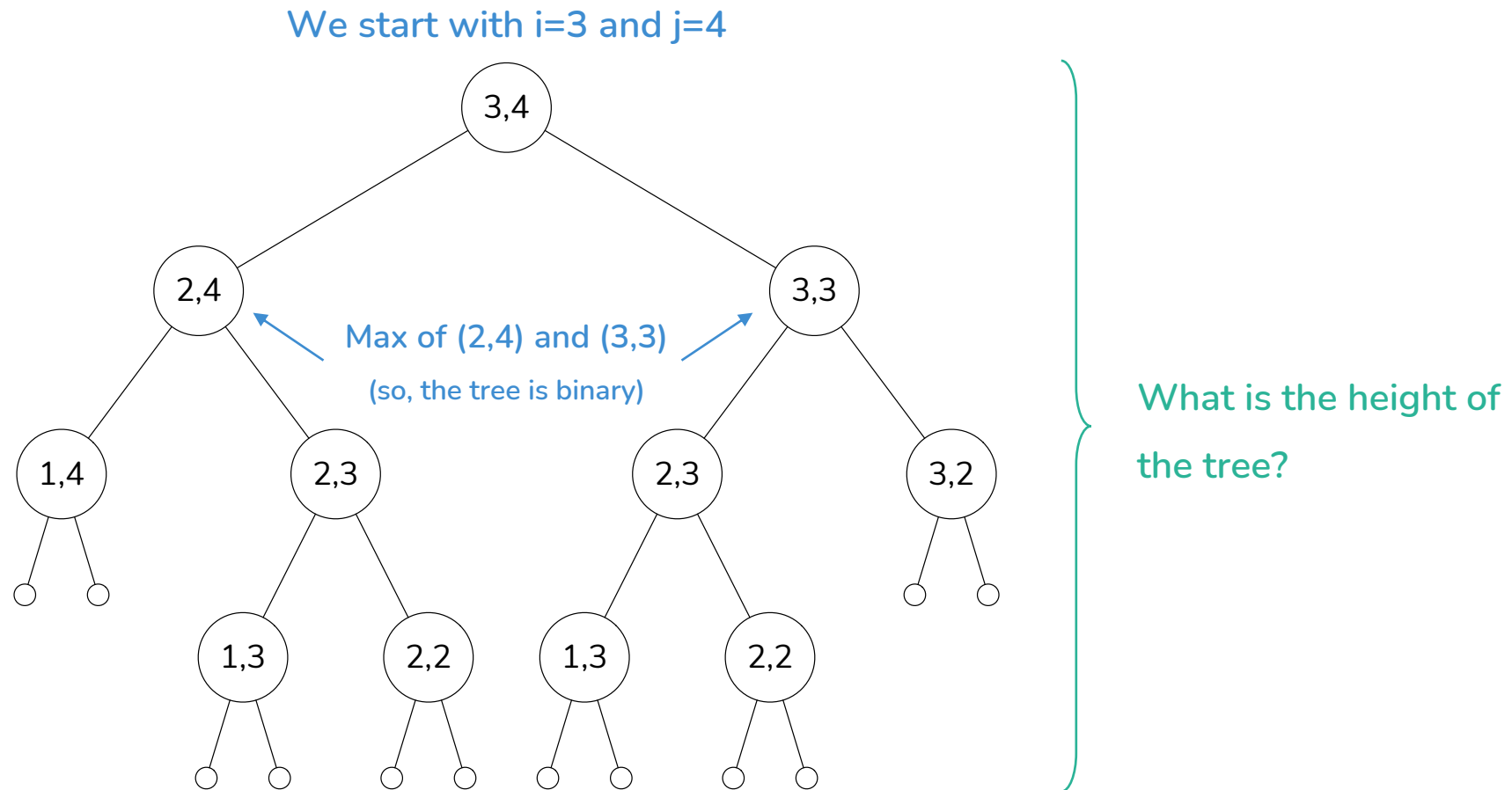
- Case 2:
 - $c[i,j] = \max(c[i, j-1], c[i-1, j])$ for $X[i] \neq Y[j]$
- After showing case 1, this is clearly true and will be left as an exercise for the student – work out an example.

The algorithm

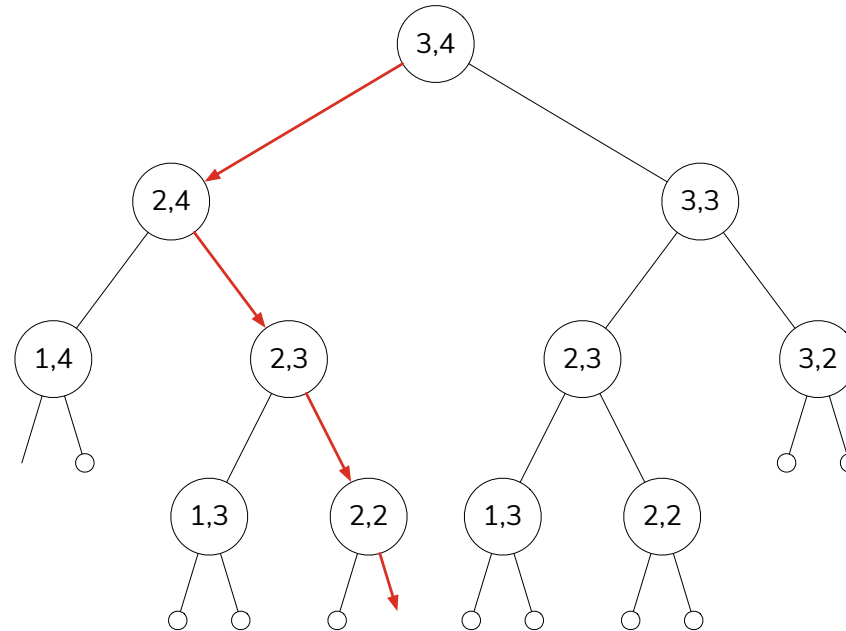
```
c(i,j) {  
    if (X[i] == Y[j])  
        return c(i-1, j-1) + 1  
    else  
        return max( c(i-1,j) , c(i, j-1) )  
}
```

- Worst case is when $X[i] \neq Y[j]$ for all i, j (the LCS will be the empty string).
- We must compute 2 sub problems to get the max.

Recursion tree analysis

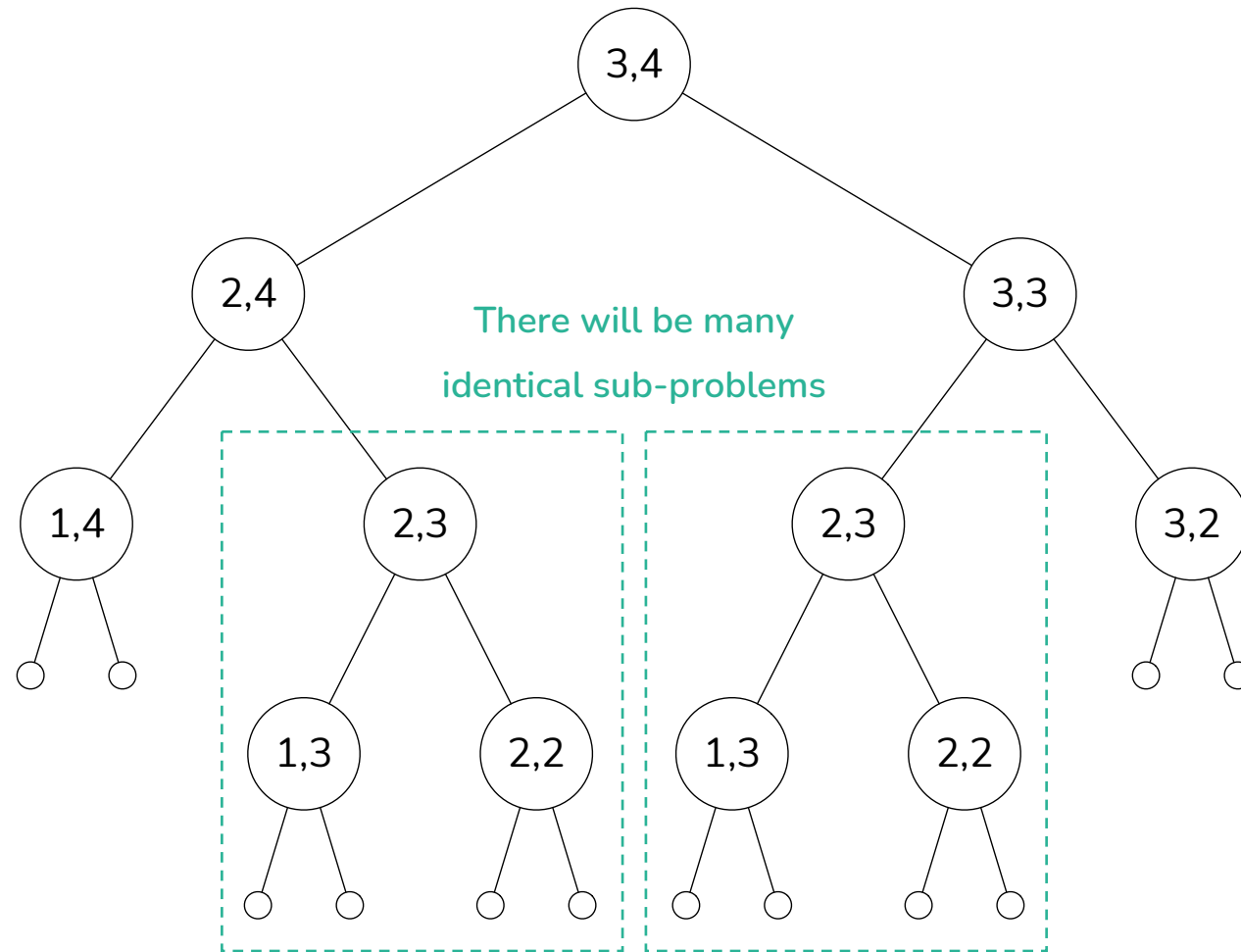


...continued



- Note that the height is $n+m$ not $\max(n,m)$. Observe the repetition (rather than decrementing) of “2” down the red line.
- This gives us an exponential number of nodes $2^{(n+m)}$ to compute – not good... but...

...continued



Memoization

- We will keep a **memo**. The idea is like using a lookup table.

Optimise by avoiding repeatedly calculating the results for previously processed inputs.

Example: Computing factorial without memoization

```
Factorial(n) {  
    if (n==1)  
        return 1  
    else  
        return n*Factorial(n-1)  
}
```

Example: Computing factorial with memoization

```
Factorial(n) {  
    if (Lookup(n) exists)   
        return Lookup(n)  
    else if (n==1)  
        return 1  
    else  
        result      = n*Factorial(n-1)  
        Lookup(n)    = result // put in lookup.  
        return result  
}
```

} So, computing 5! requires 5 recursive calls, but subsequently computing 7! only requires 2.

Recursion tree analysis

- Back to the recursion tree, further analysis shows that there although there are $2^{(n+m)}$ nodes to evaluate, there are $n \times m$ distinct subtrees (sub problems) – one for every combination of n and m .
- We use a memoization algorithm to compute. This gives:
 - Time $O(nm)$.
 - Space $O(nm)$ ← required to store the lookup table.

Working LCS as a DP

	ϵ	A	B	C	B	D	A	B
ϵ	0	0	0	0	0	0	0	0
B	0							
D	0							
C	0							
A	0							
B	0							
A	0							

The zeroes are our base case.

The rest will be filled in using the results of our theorem.

The length of the LCS will be here.

Completed example

	ϵ	A	B	C	B	D	A	B
ϵ	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

```
if x[i]=y[i] c[i-1,j-1] + 1  
else max(c[i, j-1), c[i-1, j])
```

There are 4 characters in the LCS.

Annotating for traceback

The arrows indicate when we chose to pick the diagonal +1 because the two characters at the two indices were the same.

	ϵ	A	B	C	B	D	A	B
ϵ	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Finding the LCS by tracing backwards

We can move left or up when we have the same value and diagonally if there is an 'arrow'.

The dot shows a decrease in length. When the length decreases, we have a common character.

	ϵ	A	B	C	B	D	A	B
ϵ	0	0	0	0	0	0	0	0
B	0	0	1 ○	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2 ○	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3 ○	3	3	4
A	0	1	2	2	3	3	4 ○	4

Finding the LCS (a different trace backwards)

We can move left or up when we have the same value and diagonally if there is an 'arrow'.

The dot shows a decrease in length. When the length decreases, we have a common character.

	ϵ	A	B	C	B	D	A	B
ϵ	0	0	0	0	0	0	0	0
B	0	0	1	1	1 _○	1	1	1
D	0	0	1	1	1	2 _○	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3 _○	3
B	0	1	2	2	3	3	3	4 _○
A	0	1	2	2	3	3	4	4

Conclusion

- Time = $O(nm)$.
- Space = $O(nm)$.
 - Note that space can be decreased substantially. One way is to keep only row above the current one or the column to the left of the current one (whichever one is smaller): $O(\min(n,m))$.
 - This makes reconstruction of the LCS string a bit more complex but still possible.

Further reading

- These notes should be supplemented by:
 - Introduction to Algorithms (Clifford Stein, Thomas H Cormen, Ronald L Rivest, Charles E Leiserson – MIT Press)
 - Also see Eric Demaine: http://videolectures.net/mit6046jf05_leiserson_lec15/