2. Longest common subsequence.

Given two strings A and B,

Find the length of the longest common subsequence.

A=“ACHEFMGLP”

B=“XYCEPQMLG”

output = 4 , “CEMG” is the longest common subsequence

Given two strings A and B, common subsequence is the sequence of characters not necessarily contiguous which is present in both the string.

A="ACHEFMGLP"

B="XYCEPQMLG"

Another example of subsequences

A="ACHEFMGLP"

B="XYCEPQMLG"

A="ACHEFMGLP"

B="XYCEPQMLG"

Longest common subsequence is CEMG

Find the longest common subsequence.

Hint : Start by comparing last characters of two strings and enumerate all the candidates for next comparisions

1. State

1. State

Parameters

i - Last index of the substring of A

j - Last index of the subtring of B

Cost function

lcs(i,j,A,B) - length of longest common subsequence in the string A ending at index i and string B ending at index j.

We check one character at a time.

2. Transitions

2. Transitions

Base case

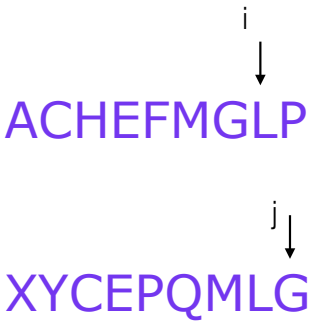
i = -1 or j = -1 , return 0 , Either one of the string is empty

Choices

Longest common subsequence

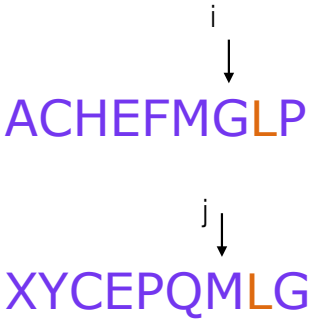
Longest common subsequence

Longest common subsequence



Longest common subsequence

Longest common subsequence



2. Transitions

Base case

i = -1 or j = -1 , return 0 , Either one of the string is empty

Choices

lcs(i,j,A,B)

Case 1 : If A[i] != B[j]

lcs(i-1,j,A,B)

lcs(i,j-1,A,B)

Optimal choice

lcs(i,j,A,B) = MAX(lcs(i-1,j,A,B),lcs(i,j-1,A,B))

Case 2: If A[i] == B[j]

lcs(i,j,A,B) = lcs(i-1,j-1,A,B)+1

Recurrence relation

lcs(i,j,A,B) = -1, if i=-1 or j=-1

lcs(i,j,A,B) = lcs(i-1,j-1,A,B)+1 , if A[i] == B[j]

lcs(i,j,A,B) = MAX(lcs(i-1,j,A,B),lcs(i,j-1,A,B)) , if A[i] != B[j]

3. Recursive solution

3. Recursive solution

Pseudo code

```
lcs(i,j,A,B)
   if i==-1 or j==-1
      return 0

  if A[i] == B[j]
     return lcs(i-1,j-1,A,B) +1
  else
    return MAX(lcs(i-1,j,A,B),lcs(i,j-1,A,B))
```

Java

```java
public static int lcs(int i,int j,String A,String B){
    if(i == -1 || j == -1){
        return 0;
    }
    if(A.charAt(i) == B.charAt(j)){
        return lcs(i-1,j-1,A,B)+1;
    }else {
        return Math.max(lcs(i-1,j,A,B),lcs(i,j-1,A,B));
    }
}
```

Python

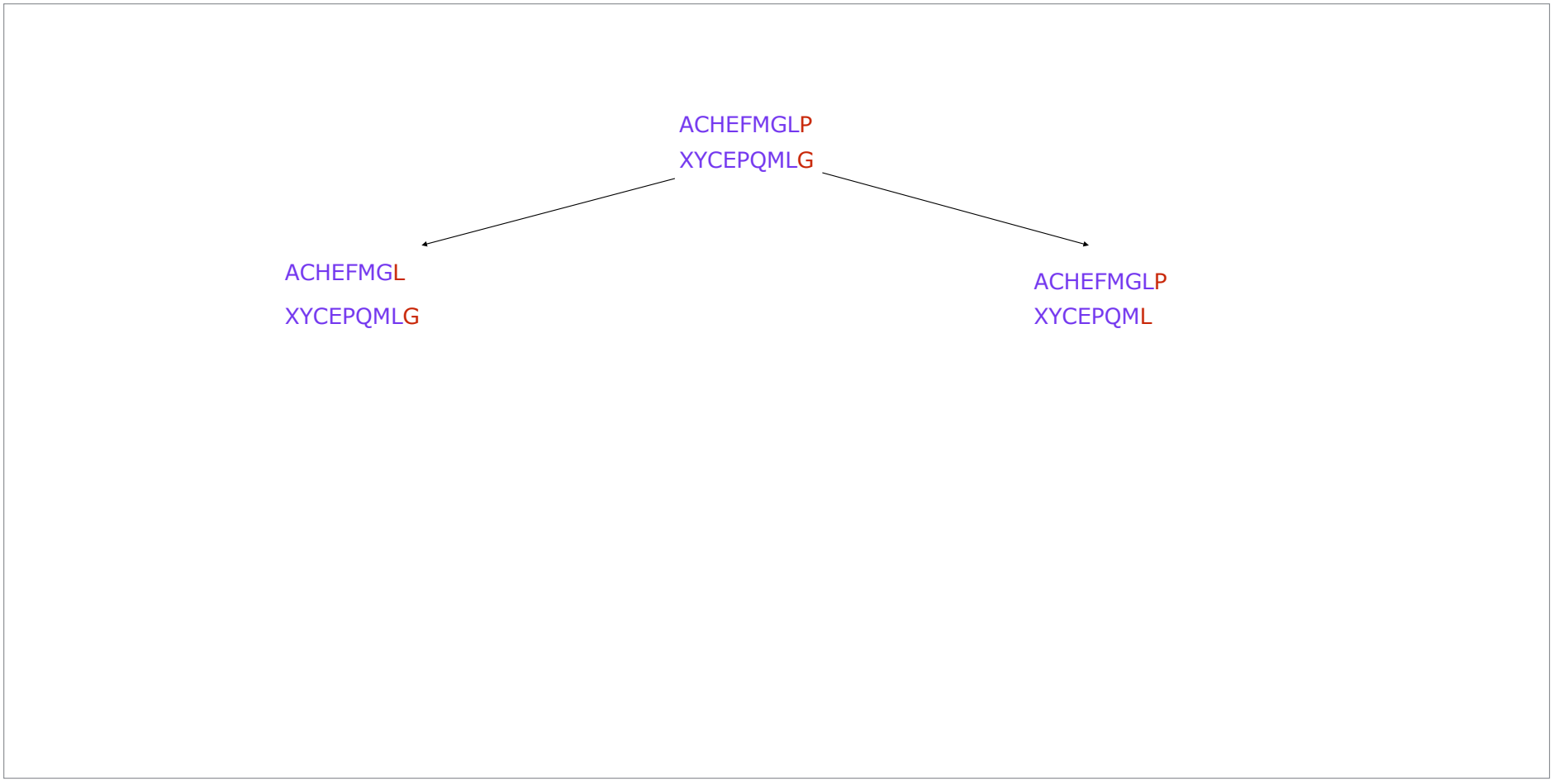
```python
def lcs(i, j, A, B):
    if i == -1 or j == -1:
        return 0
    if A[i] == B[j]:
        return lcs(i - 1, j - 1, A, B) + 1
    else:
        return max(lcs(i - 1, j, A, B), lcs(i, j - 1, A, B))
```

Longest common subsequence

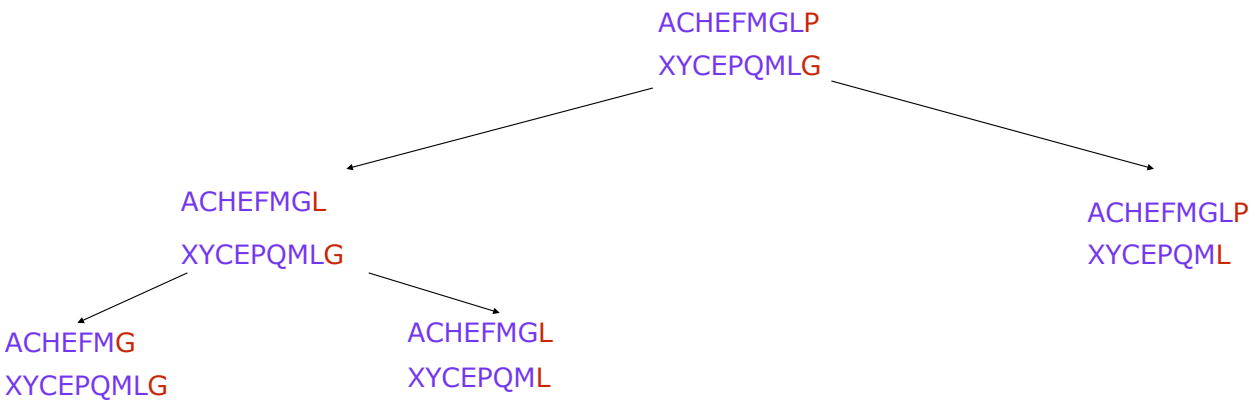
ACHEFMGLP

XYCEPQMLG

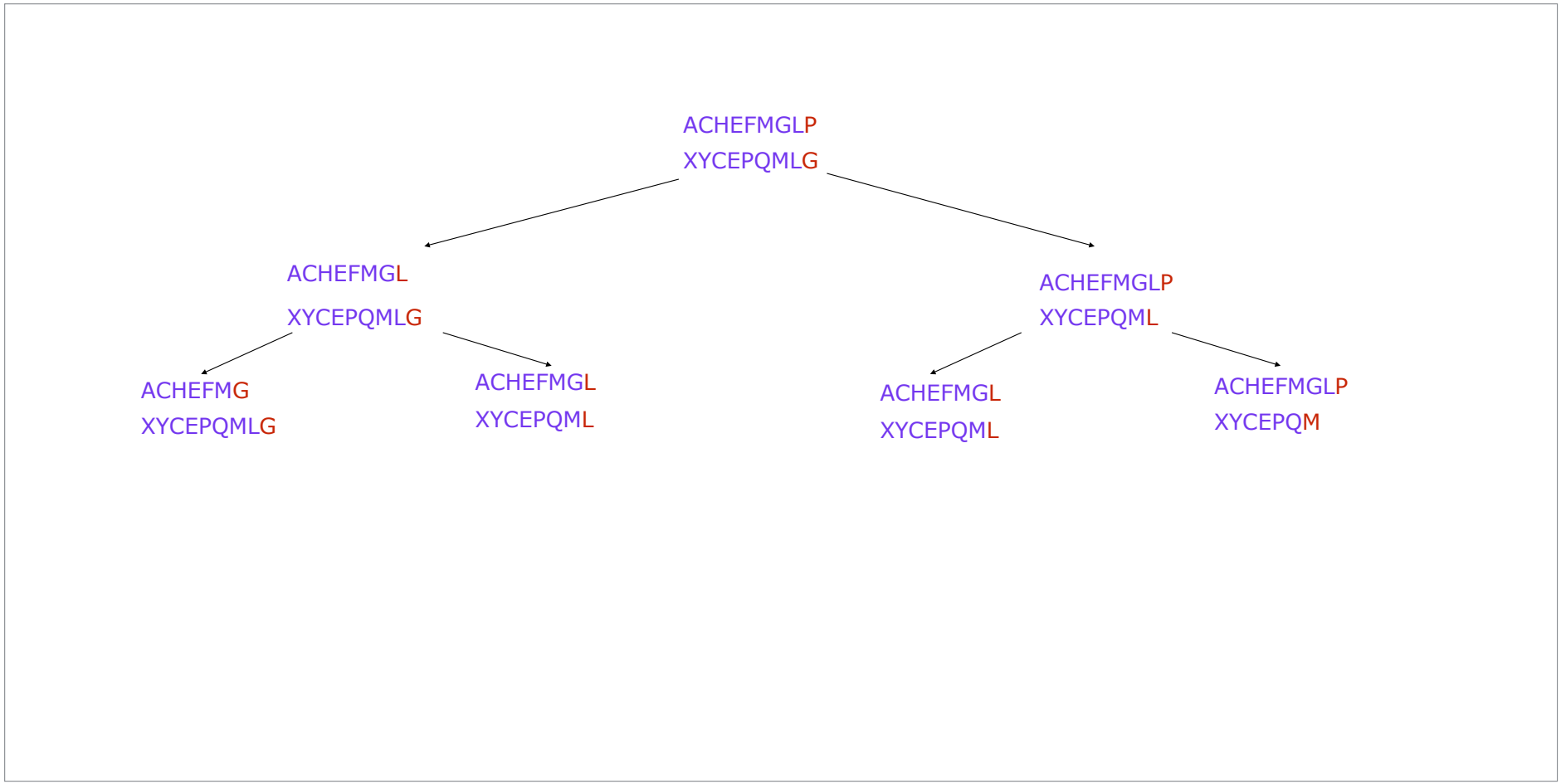
Longest common subsequence



Longest common subsequence



Longest common subsequence



4. Memoize

4. Memoize

We can cache the results in a 2D array and reuse it whenever needed again

Key -> (i,j)

Value -> LCS in the substring of A ending at i and substring of B ending at inde j

Default value -> -1

Java

```java
public static int lcsMemo(int i,int j,String A,String B,int[][] cache){
    if(i == -1 || j == -1){
        return 0;
    }
    if(cache[i][j] != -1){
        return cache[i][j];
    }
    if(A.charAt(i) == B.charAt(j)){
        cache[i][j] = lcsMemo(i-1,j-1,A,B,cache)+1;
        return cache[i][j];
    }else {
        cache[i][j] = Math.max(lcs(i-1,j,A,B),lcsMemo(i,j-1,A,B,cache));
        return cache[i][j];
    }
}
```

Java

```java
public static int lcsMemo(String A,String B){
    int[][] cache = new int[A.length()][B.length()];
    for(int[] row : cache){
        Arrays.fill(row,-1);
    }
    return lcsMemo(A.length()-1,B.length()-1,A,B,cache);
}
```

Python

```
def lcs_memo(i, j, A, B,cache):
    if i == -1 or j == -1:
        return 0
    if cache[i][j] != -1:
        return cache[i][j]
    if A[i] == B[j]:
        cache[i][j] = lcs_memo(i - 1, j - 1, A, B,cache) + 1
        return cache[i][j]
    else:
        cache[i][j] = max(lcs_memo(i - 1, j, A, B,cache),
lcs_memo(i, j - 1, A, B,cache))
        return cache[i][j]
```

5. Bottom up approach

Recurrence relation

lcs(i,j,A,B) = 0 , if i=-1 or j=-1

lcs(i,j,A,B) = lcs(i-1,j-1,A,B)+1 , if A[i] == B[j]

lcs(i,j,A,B) = MAX(lcs(i-1,j,A,B),lcs(i,j-1,A,B)) , if A[i] != B[j]

Bottom up equation

dp[i][j] = 0, if i=0 or j=0

dp[i][j] = dp[i-1][j-1]+1 , if A[i-1] == B[j-1]

dp[i][j] = MAX(dp[i-1][j],dp[i][j-1]) , if A[i] != B[j]

		0	1	2	3	4
			Y	X	A	E
0						
1	X					
2	P					
3	A					
4	E					

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X					
2	P					
3	A					
4	E					

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0				
2	P	0				
3	A	0				
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0				
2	P	0				
3	A	0				
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0			
2	P	0				
3	A	0				
4	E	0				


0 1 2 3 4
Y X A E
0 0 0 0 0 0
1 X 0 0 1
2 P 0
3 A 0
4 E 0


0 1 2 3 4
Y X A E
0 0 0 0 0 0
1 X 0 0 1 1
2 P 0
3 A 0
4 E 0

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0				
3	A	0				
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0			
3	A	0				
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1		
3	A	0				
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0				
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0			
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0	1		
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0	1	2	
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0	1	2	2
4	E	0				

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0	1	2	2
4	E	0	0			

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0	1	2	2
4	E	0	0	1		

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0	1	2	2
4	E	0	0	1	2	

		0	1	2	3	4
			Y	X	A	E
0		0	0	0	0	0
1	X	0	0	1	1	1
2	P	0	0	1	1	1
3	A	0	0	1	2	2
4	E	0	0	1	2	3

Java

```java
public static int lcsDP(String A,String B){
    int M = A.length();
    int N = B.length();
    int[][] dp = new int[M+1][N+1];
    for(int i=1;i<=M;i++){
        for(int j=1;j<=N;j++){
            if(A.charAt(i-1) == B.charAt(j-1)){
                dp[i][j] = dp[i-1][j-1]+1;
            }else{
                dp[i][j] = Math.max(dp[i-1][j],dp[i][j-1]);
            }
        }
    }
    return dp[M][N];
}
```

Python

```python
def lcs_dp(A,B):
    M = len(A)
    N = len(B)
    dp = [[0 for _ in range(0,N+1)] for _ in range(0,M+1)]
    for i in range(1,M+1):
        for j in range(1,N+1):
            if A[i-1] == B[j-1]:
                dp[i][j] = dp[i-1][j-1]+1
            else:
                dp[i][j] = max(dp[i-1][j],dp[i][j-1])
    return dp[M][N]
```

Time and space complexity analysis

Recursive implementation

Recursion tree

Number of children at each node , 2 , Binary tree

Height of the tree , MAX(M,N)

Number of nodes in the tree , 2^N

Time complexity , $O(2^N)$, Exponential

Space complexity, O(1)

Dynamic programming

Two for loops.

Outer for loop, i=0,1,…M

Inner for loop, j=0,1,2…N

Total time complexity O(MN)

Space complexity O(MN), because we use a 2D array of size MN