Albert Dinkins - Xiangzhong Wei - Dong Jun Kim - Group 9

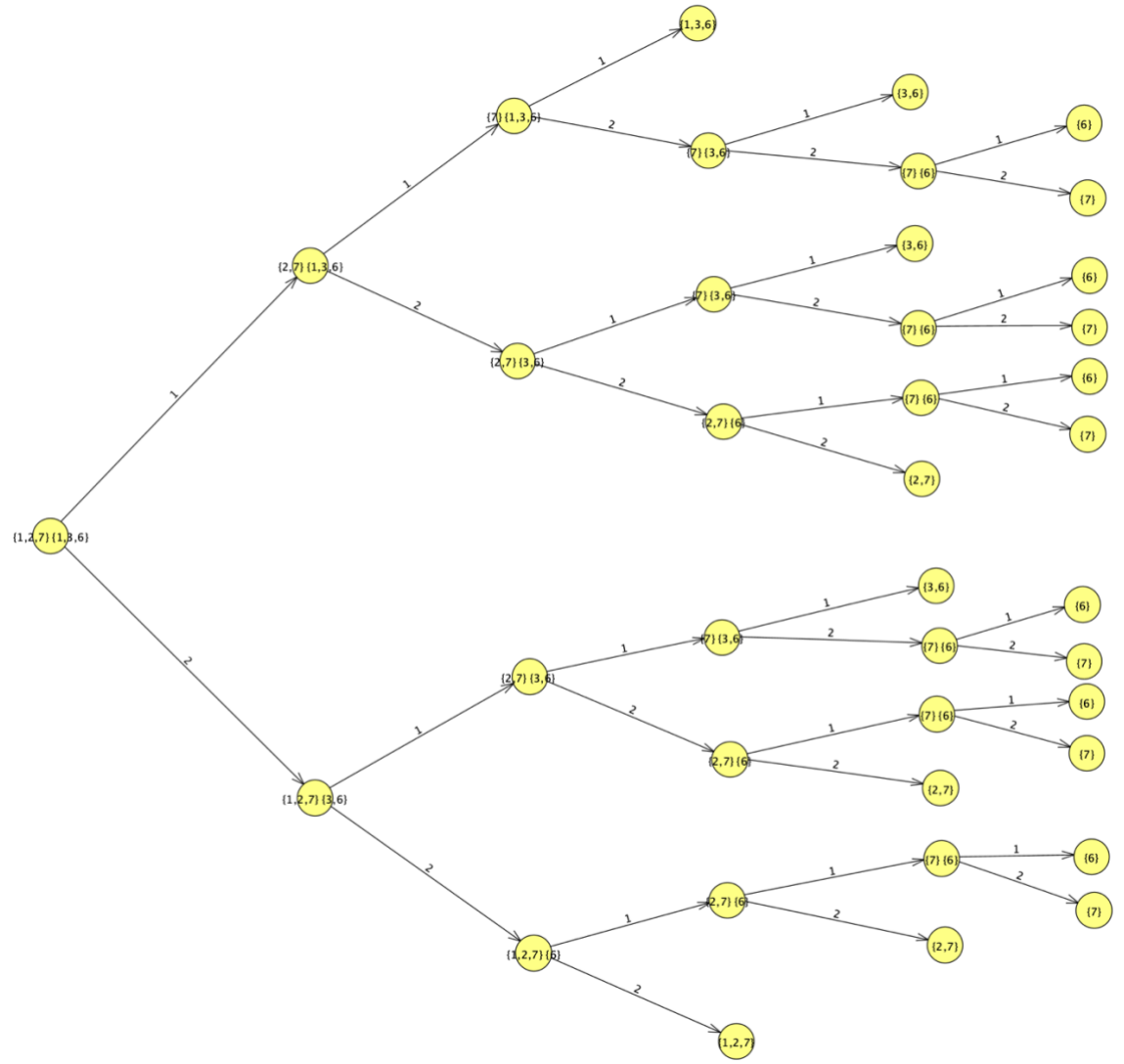
1) How would you break down the problem of calculating the maximum alignment score for a combination of two sequences seq1 and seq2 (length n and m, respectively) with the sequence target (length n + m) into one or more smaller instances of this problem? Your answer should include what subproblems you are evaluating, as well as how you are using the answers to these subproblems to solve the original problem. (8 points)

   To begin, both sequences can be stored as queues to more easily cooperate with the instruction that no element in its respective sequence can come before the other. Taking these queues, we can break down the problem into several recursively produced subproblems in which each call of the recursive function operates on a queue that is one element smaller – resulting in base cases where one of the two queues is empty and all that's left to do is test the remaining elements of the non-empty queue. Instead of producing and testing all possible "in-order permutations" of the two original sequences against the target, we can recursively choose the best option starting from the back of the two queues and work our way forward. Once we have a "best" alignment score from the sub-queues we can introduce more elements from the original queue and test them against our previous answers and the target.

   In terms of non-memoized code, this is done by recursively popping an element from the first queue (and the first queue alone), multiplying it against the front of the target (store this in a variable called productA for example), and adding productA to the return value of the recursive call that will operate on the newly reduced queue 1 and unchanged queue 2. In parallel, we can also recursively pop an element of the second queue (and the second queue alone), multiply it against the front of the target, store it in some variable (call it productB), and add productB to the return value of the recursive call that will operate on the newly reduced queue 2 and unchanged queue 1.

   The max of productA and productB will hold the maximum alignment score as each of the respective recursive calls (choosing the maximum results derived from either popping queue 1 or 2) will have swept out all possible in order permutations of the combined sequences.

   This is essentially the same recursive process that occurs in the LCS algorithm. Pictured below is the sample recursive process for the sample input provided as part of this project:



2) What are the base cases of this recurrence? (4 points)

There are 4 base cases:

When queue 1 is empty and queue 2 has 1 element
When queue 1 is empty and queue 2 has 2 or more elements
When queue 2 is empty and queue 1 has 1 element
When queue 2 is empty and queue 1 has 2 or more elements

3) What data structure would you use to recognize repeated problems? You should describe both the abstract data structure as well as its implementation. (4 points)

I would use a 2D vector that holds doubles to recognize repeated problems where row 0 represents queue 1 with 0 elements, row 1 represents queue 1 with 1 element, row 2 represents queue 1 with 2 elements and so on. Likewise, column 0 would represent queue 2 with 0 elements, column 1 would represent queue 2 with 1 element, column 2 would represent queue 2 with 2 elements and so on. This 2D vector will be initialized to any value greater than 1 since subproblems are normalized and between 1 and -1. This data structure and its implementation would represent the repeated comparisons of similar sized queues against each other as, if a queue is a certain size, we know exactly what is in that queue.

4) Give pseudocode for a memoized dynamic programming algorithm to find the maximum alignment score when combining seq1 and seq2 to align with target. (8 points)

Algorithm: memoMASwrapper
Input: queue1[0…n], queue2[0…n], target[0…n+m]

memoMASwrapper:

initialize 2d array of size (n+1)x(m+1) to 9
return memoMAS(queue1, queue2, target, 2dArray)

//end wrapper

```
Algorithm: memoMAS
Input: queue1[0…n], queue2[0…m],  target[0…n+m], 2dArray[(n+1)x(m+1)]

memoMAS:

if both queues are not empty and 2dArray[n][m] == 9:
        prodA = multiply front of q1 times the first element in target
        prodB = multiply front of q2 times the first element in target

        a = prodA + memoMAS(queue1[1..n], queue2[0…m], target[1…n+m])
        b = prodB + memoMAS(queue1[0..n], queue2[1…m], target[1…n+m])

        2dArray[n][m] = max(a,b)

else:
        return 2dArray[n][m]


if queue1 is empty and queue2 is not and 2dArray[n][m] == 9:
        for the rest of the i elements in target
                memo[0][m] = queue2.front * target(i)
                queue2.pop
        return 2dArray[0][m]

else:
        return 2dArray[n][m]


if queue2 is empty and queue1 is not and 2dArray[n][m] == 9:
        for the rest of the i elements in target
                memo[n][0] = queue1.front * target(i)
                queue1.pop
        return 2dArray[n][0]

else:
        return 2dArray[n][m]

return 2dArray[n][m]
//end memoMAS
```

5) Give pseudocode for an iterative algorithm to find the maximum alignment score when combining seq1 and seq2 to align with target. This algorithm does not need to have a reduced space complexity relative to the memoized solution. (8) points)

```
Algorithm: memoMASITwrapper
Input: seq1[0…n], seq2[0…n],  target[0…n+m]

memoMASITwrapper:

initialize 2d array of size (n+1)x(m+1) to 0
return memoMAS(seq1, seq2, target, 2dArray)

//end wrapper



Algorithm: memoMASIT
Input: seq1 [0…n], seq2[0…m],  target[0…n+m], 2dArray[(n+1)x(m+1)]

memoMAS:

from 1:n as i
        2dArray[i][0] = 2dArray[i-1][0] + target(n+m – i + 1)*seq1(n - i)

from 1:m as j
        2dArray[0][j] = 2dArray[0][j-1] + target target(n+m – i + 1)*seq2(m - i)

from 1:n+1 as k
        from 1:m+1 as h
                2dArray[k][h] = max(2dArray[k][h-1] + seq1(k-h) * target(n+m-h-1),
                                    2dArray[k-1][h] + seq2(k-h)*target(n+m-k-1))


Return 2dArray[n][m]
```

6) Can the space complexity of the iterative algorithm be improved relative to the memorized algorithm? Justify your answer. (4 points)

   The space complexity of the iterative algorithm can be improved because the iterative process only relies on rows/columns 1 away from the cell being filled, so we can eliminate rows/columns 2dArray[k][h-2] and 2dArray[k-2][h] once we're finished with an iteration. In terms of the math, the maximum alignment score from 2 rows/columns away has already been solidified and will be properly chosen each time we iterate.

7) Give pseudocode for an algorithm that identifies the sequence which will achieve the maximum alignment score. If there is more than one sequence that yields the maximum score, you may return any such sequence. Your algorithm may be iterative or recursive. (4 points)

   RECURSIVE SOLUTION

   Algorithm: memoMASwrapper
   Input: queue1[0…n], queue2[0…n], target[0…n+m], seqList
   seqList is passed by reference and is mutable

   memoMASwrapper:

   initialize 2d array of size nxm to 9
   return memoMAS(queue1, queue2, target, seqList, 2dArray)

   //end wrapper

Algorithm: memoMAS
Input: queue1[0…n], queue2[0…m], target[0…n+m], seqList, 2dArray[(n+1)x(m+1)]
seqList is passed by reference and its mutable

memoMAS:

if both queues are not empty and 2dArray[n][m] == 9:

prodA = multiply front of q1 times the first element in target
prodB = multiply front of q2 times the first element in target

a = prodA + memoMAS(queue1[1..n], queue2[0…m], target[1…n+m], seq1)
b = prodB + memoMAS(queue1[0..n], queue2[1…m], target[1…n+m], seq2)

if(a >= b):
add q1.front to seqList
add rest of seq 1 to seqList
else:
add q2.front to seqlist
add rest of seq 2 to seqList

2dArray[n][m] = max(a,b)

else:
return 2dArray[n][m]

if queue1 is empty and queue2 is not and 2dArray[n][m] == 9:

add all remaining elements of queue2 to sequence

for the rest of the i elements in target
memo[0][m] = queue2.front * target(i)
queue2.pop

return 2dArray[0][m]

else:
return 2dArray[n][m]

if queue2 is empty and queue1 is not and 2dArray[n][m] == 9:

add all remaining elements of queue1 to sequence

```
        for the rest of the i elements in target
                memo[n][0] = queue1.front * target(i)
                queue1.pop

        return 2dArray[n][0]

else:
        return 2dArray[n][m]



return 2dArray[n][m]
//end memoMAS
```