Dynamic Programming

Kamryn Suh
Keylin Sanchez
Ricardo Frumento

University of South Florida
Tampa, FL
Fall 2021

1 Breaking down the problem

One way to break down in small problems the calculation to find the alignment score between the combination of two sequences and a target sequence is first understand how the combinations can be formed

$$a = [a_1, a_2, \cdots, a_n] \quad \& \quad b = [b_1, b_2, \cdots, b_m] \quad \& \quad t = [t_1, t_2, \cdots, t_{n+m}]$$

$$seq_1 = [a_1, a_2, \cdots, a_n, b_1, b_2, \cdots, b_m]$$
$$seq_2 = [a_1, a_2, \cdots, b_1, a_n, b_2, \cdots, b_m]$$
$$\vdots$$
$$seq_m = [b_1, b_2, \cdots, b_m, a_1, a_2, \cdots, a_n]$$

Then it is a matter of finding the alignment score and figuring out what are the repeated structures

$$score_1 = a_1 \cdot t_1 + a_2 \cdot t_2 + \cdots + a_n \cdot t_n + b_1 \cdot t_{n+1} + b_2 \cdot t_{n+2} + \cdots + b_m \cdot t_{n+m}$$

$$score_2 = a_1 \cdot t_1 + a_2 \cdot t_2 + \cdots + b_1 \cdot t_n + a_n \cdot t_{n+1} + b_2 \cdot t_{n+2} + \cdots + b_m \cdot t_{n+m}$$

It is important to note that between the previous two score calculations $a_1 \cdot t_1 + a_2 \cdot t_2 + \cdots + a_{n-1} \cdot t_{n-1}$ is repeated. The same idea can be used with any part of this sum and therefore it can be used for the first element for example, which repeats for half the alignment score calculations (half of the combinations start with the first element of a). That being said the function call should look like this until we reach the base cases discussed in the previous items.

$$Function(a[i..n], b[j..m], t[k..n+m])$$

Should result in the max value between

$$a_i \cdot t_k + Function(a[i+1..n], b[j..m], t[k+1..n+m])$$

and

$$b_j \cdot t_k + Function(a[i..n], b[j+1..m], t[k+1..n+m])$$

2 Base cases

Observing the base cases, there are two states that require attention. When one of the sequences is empty, and when both the sequences are empty. The first has a clear meaning, the algorithm has run through one of the sequences and there are only elements on the other to be computed, so there is no need to make two function calls only one incrementing the sequence that is not empty. For the second case there are no more elements in the sequences so the function should return 0.

3 Data structure used

Observing how the results repeat themselves a matrix implemented as a 2D-array would be sufficient to store the results.

4 Memoized pseudocode for max alignment score

```
   Data: seq2, seq2
   Data: i, j, k
   Data: array target
 1 MemoAlignment(seq1[i..n], i, seq2[j..m], j, target, k, arr):
 2 |  if arr[i,j] ≠ ∞ then
 3 |  |  return arr[i,j]
 4 |  else if seq1 = ϵ and seq1 = ϵ then
 5 |  |  return 0
 6 |  else if seq1 = ϵ and seq2 ≠ ϵ then
 7 |  |  arr[i][j] = seq2[j] · target[i+j] + MemoAlignment(seq1[i..n]), i,
   |  |   seq2[j+1..m], j+1, target);
 8 |  else if seq1 ≠ ϵ and seq2 = ϵ then
 9 |  |  arr[i][j] = seq1[i] · target[i+j] + MemoAlignment(seq[i+1..n]),
   |  |   i+1, seq2[j..m], j, target);
10 |  else
11 |  |  score1 = seq1[i] · target[i+j] + MemoAlignment(seq[i+1..n]),
   |  |   i+1, seq2[j..m], j, target);
12 |  |  score2 = seq2[j] · target[i+j] + MemoAlignment(seq1[i..n]), i,
   |  |   seq2[j+1..m], j+1, target);
13 |  |  score = max(score1, score2);
14 |  |  arr[i][j] = score;
15 |  end
16 return arr[i][j];
```

```
   Data: array seq2
   Data: array seq2
   Data: array target
 1 Wrapper(seq1[1..n], seq2[1..m], target[1..n+m]):
 2 |  arr = Array(n, m);
 3 |  Initialize arr to ∞;
 4 return MemoAlignment(seq1, 1, seq2, , target, 1, arr);
```

5 Iterative pseudocode for max alignment score

```
   Data: seq1, seq2
   Data: i, j
   Data: array target
 1 IterAlignment:
 2 |  arr = Array(i,j);
 3 |  for x = 1 to n do
 4 |  |  for y = 1 to m do
 5 |  |  |  if x = 0 and y = 0 then
 6 |  |  |  |  arr[x][y] = 0;
 7 |  |  |  else if x = 0 and y ≠ 0 then
 8 |  |  |  |  arr[x][y] = arr[x][y-1] + target[x+y]  seq2[y];
 9 |  |  |  else if x ≠ 0 and y = 0 then
10 |  |  |  |  arr[x][y] = arr[x-1][y] + target[x+y] · seq1[y];
11 |  |  |  else
12 |  |  |  |  arr[x][y] = max(arr[x-1][y] + target[x+y] · seq1[y],
   |  |  |  |    arr[x][y-1] + target[x+y] · seq2[y]);
13 |  |  |  end
14 |  |  end
15 |  end
16 return arr[x][y];
```

6 Can space complexity of iterative be improved

Yes, as there is an order to the iterative algorithm there is no need to store all of the values once you are past it. The algorithm walks through the array in two possible ways, left to right or top to bottom. Which means that anything to the left of the previous column and to the right of the current column is not necessary immediately.

7 Pseudocode for sequence and max score

```
   Data: seq1, seq2
   Data: i, j
   Data: array target
 1 IterAlignmentSequence:
 2 |  arr = Array(i,j);
 3 |  counter = 1;
 4 |  answer = Array(i+j);
 5 |  for x = 1 to n do
 6 |  |  for y = 1 to m do
 7 |  |  |  if x = 0 and y = 0 then
 8 |  |  |  |  arr[x][y] = 0;
 9 |  |  |  else if x ≠ 0 and y = 0 then
10 |  |  |  |  arr[x][y] = arr[x][y-1] + target[x+y]  seq2[y];
11 |  |  |  else if x ≠ 0 and y = 0 then
12 |  |  |  |  arr[x][y] = arr[x-1][y] + target[x+y] · seq1[y];
13 |  |  |  else
14 |  |  |  |  arr[x][y] = max(arr[x-1][y] + target[x+y] · seq1[y],
   |  |  |  |    arr[x][y-1] + target[x+y] · seq2[y]);
15 |  |  |  |  if arr[x-1][y] + target[x+y] · seq1[y] > arr[x][y-1] +
   |  |  |  |    target[x+y] · seq2[y] then
16 |  |  |  |  |  arr[i][j-1]=0;
17 |  |  |  |  else
18 |  |  |  |  |  arr[i-1][j]=0;
19 |  |  |  |  end
20 |  |  |  end
21 |  |  end
22 |  end
23 |  r = i;
24 |  c = j;
25 |  counter1 = 0;
26 |  counter2 = 0;
27 |  for z = 1 to m + n do
28 |  |  if arr[r-1][c] > arr[r][c-1] then
29 |  |  |  answer[z] = seq1[i-counter1];
30 |  |  |  r = r - 1;
31 |  |  |  counter1 = counter1 + 1;
32 |  |  else
33 |  |  |  answer[z] = seq2[j-counter2];
34 |  |  |  c = c - 1;
35 |  |  |  counter2 = counter2 + 1;
36 |  |  end
37 |  end
38 return answer;
```