Zhoutao Zhang
Noel Kubach

UNIVERSITY OF TUBINGEN
SUMMER 2023

SSBI
ASSIGNMENT 01

Task 1 - Nussinov RNA folding algorithm

The Nussinov algorithm has been implemented in the python script kubach_nussinov_predictor.py. For documentation please refere to the provided readme file.

Task 2 - Multiple optimal solutions

Ignored Solutions:

The dynamic programming matrix from the lecture, as well as from our own implementation contains all possible (possibly multiple) solutions for the RNA folding. However, during the traceback we ignore some of the solutions. By using an if-elif-else construct, we consider one case after the other (from 1 to 4), but stop searching for a "parent cell" as soon as we found a case, that explains the current value in a cell of the DP matrix. Additionally for case 4, we are stopping the search as soon as we encounter one possible solution (break statement in line 14), thus we do not necessarily consider all possible values of k. Pseudocode (lecture 02D, slide 41):

```
1  traceback(i,j)
2  if i<j:
3      if $\gamma$(i,j) = $\gamma$(i+1,j):
4          traceback(i+1,j)
5      elif $\gamma$(i,j) = $\gamma$(i,j-1):
6          traceback(i,j-1)
7      elif $\gamma$(i,j) = $\gamma$(i+1,j-1) + $\delta$(i,j):
8          print(i,j)
9          traceback(i+1,j-1)
10     else for k = i+1 to j-1:
11         if $\gamma$(i,j) = $\gamma$(i,k) + $\gamma$(k+1,j):
12             traceback(i,k)
13             traceback(k+1,j)
14             break
```

Random optimal solution:

Instead of always comparing the scores in the same order and jumping to the next recursion immediately, we can temporary save all indexes that lead to the same max score at (i, j), and randomly choose one for our next recursive call.

Computational complexity:

To find all optimal solutions, instead of interrupting the trace back when a match was found, we must continue to find all the matches. Doing this during trace back is extremely time consuming since we have to calculate the scores all over again. Instead of calculating all scores again, we could save all indexes leading to the score of a cell already in the "forward run" in addition to the score itself. By doing this, we can get the optimal solution once we finish the matrix without a trace back step. To fill each cell of the dynamic programming matrix, the worst time complexity is $O(n)$, because in the worst case the step $max(max(\gamma(i,k) + \gamma(k+1,j))\ for\ i < k < (j-1))$ (case 4) takes n steps to complete. All other cases take only $O(1)$. Since we have to fill about half of the matrix or $\frac{n^2}{2}$ cells, The worst case time complexity is $O(n) \cdot \frac{n^2}{2} = O(n^3)$ for the entire algorithm.

Task 3 - k-loop decomposition

Base pair	Value of k	Size	Bases being accessible	Name of secondary structure
1, 2, 52	0	3	0	null-loop
(3,51)	2	0	0	stacking-loop
(4,50)	2	0	0	stacking-loop
(5,49)	3	5	6, 46, 47, 48	multi-loop
(7,21)	2	0	0	stacking-loop
(8,20)	2	3	9, 10, 19	interior-loop
(11,18)	1	6	12, 13, 14, 15, 16, 17	hairpin-loop
(23,45)	2	1	24	bulge-loop
(25,44)	2	3	26, 27, 43	interior-loop
(28,42)	2	1	41	bulge-loop
(29,40)	3	0	0	multi-loop
(30,34)	1	3	31, 32, 33	hairpin-loop
(35,39)	1	3	36, 37, 38	hairpin-loop