# CSCI532 Homework 2

Lin Shi, Gideon Popoola

February 24, 2022

1. Exercise 1; pg 312

   (a) part a

   *Proof.* Given the heaviest-first greedy algorithm, if the nodes are $3, 8, 6$, based on the algorithm, we will get 8. However, the actual maximum weight is 9 with the node 3 and 6. □

   (b) part b

   *Proof.* Given the even-odd algorithm, if the nodes are $8, 6, 3, 6$. Based on the algorithm, the even is 11, the odd is 12 (index count from 1). However, the true maximum weight is 14 with node 8 and 6. □

   (c) part c

   For this problem, we will use the dynamic programming approach. Let S = $v_1, v_2, ...v_n$, where v represents each node. Then we will store the local maximum weight in a list, X, where $X_0 = 0$. Next we will start iterating through, the local maximum weight will be either maximum of $X_{i-1}$ or $v_i + X_{i-2}$.

2. Exercise 3; pg 314

   (a) part a

   *Proof.* Given the example (1, 2), (1,3), (2, 5), (3,4), (4, 5). Based on this algorithm, we will take (1, 2) and (2, 5). The algorithm identifies this path as the longest path. However, the longest path is (1,3), (3,4), (4,5). □

   (b) part b

   For this problem, we will use the dynamic programming approach. Let S = $v_1, v_2, ...v_n$, where v represents each node. Additionally, we will store the local longest path in X, where $X_1 = 0$. Next, we will start iterating through, the local longest path for any $i$ location will be 1 + maximum($X_j$), where j is the starting node for all edges connected to i. As we start to build up X, we will reach our longest path count.

3. Exercise 7; pg 191

   In this problem, based on the statement, we know the length of time that it takes for the supercomputer to finish all tasks are the same regardless of its ordering. Therefore, since there are the same amount of PC as jobs, we would like to start with the longest tasks with that takes the longest time, so we could minimize the total completion time. We will use the idea of a mergesort to get our schedule. We will merge and sort in decreasing time order. If the total amount of time is the same for supercomputer and PC, then we will take the one with complete the one with longest PC time first. After the sort is completed, we will start with the ones with the longest time and end with the job that requires the least amount of time. Thus, that will be the algorithm used to find the schedule. The algorithm uses merge sort, which will be $O(nlogn)$.

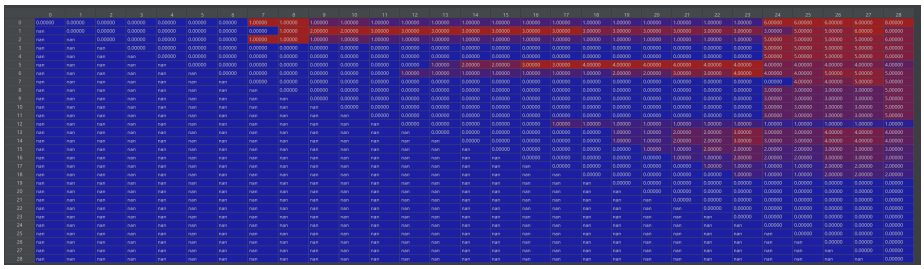
4. Programming Project For this programming project, I have decided to implement the RNA secondary structure.

   Below is the random example:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 1.0 | 6.0 | 6.0 | 6.0 | 6.0 | 8.0 |
| 1 | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 3.0 | 3.0 | 3.0 | 5.0 | 5.0 | 6.0 | 6.0 |
| 2 | NaN | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 1.0 | 5.0 | 5.0 | 5.0 | 5.0 | 6.0 |
| 3 | NaN | NaN | NaN | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 5.0 | 5.0 | 5.0 | 5.0 | 6.0 |
| 4 | NaN | NaN | NaN | NaN | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 5.0 | 5.0 | 5.0 | 5.0 | 6.0 |
| 5 | NaN | NaN | NaN | NaN | NaN | 0.0 | 0.0 | ... | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 |
| 6 | NaN | NaN | NaN | NaN | NaN | NaN | 0.0 | ... | 3.0 | 4.0 | 4.0 | 4.0 | 5.0 | 5.0 | 5.0 |
| 7 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 0.0 | 4.0 | 4.0 | 5.0 | 5.0 |
| 8 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 3.0 | 3.0 | 3.0 | 3.0 | 5.0 |
| 9 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 3.0 | 3.0 | 3.0 | 3.0 | 5.0 |
| 10 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 3.0 | 3.0 | 3.0 | 3.0 | 5.0 |
| 11 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 3.0 | 3.0 | 3.0 | 3.0 | 5.0 |
| 12 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 13 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 2.0 | 3.0 | 3.0 | 3.0 | 4.0 | 4.0 | 4.0 |
| 14 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 2.0 | 3.0 | 3.0 | 3.0 | 4.0 | 4.0 | 4.0 |
| 15 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 2.0 | 2.0 | 2.0 | 2.0 | 3.0 | 3.0 | 3.0 |
| 16 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 1.0 | 2.0 | 2.0 | 2.0 | 3.0 | 3.0 | 3.0 |
| 17 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 1.0 | 1.0 | 1.0 | 1.0 | 2.0 | 2.0 | 2.0 |
| 18 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 1.0 | 1.0 | 1.0 | 2.0 | 2.0 | 2.0 |
| 19 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 20 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 21 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 22 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 23 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 24 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 25 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | 0.0 | 0.0 | 0.0 | 0.0 |
| 26 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | 0.0 | 0.0 | 0.0 |
| 27 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | 0.0 | 0.0 |
| 28 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | 0.0 |

[29 rows x 29 columns]
The total score of fold: 8.0

Below is the dataframe of the example above:



Below is an randomized example of a rna with length 100:

```
     0    1    2    3    4    5    6   ...    93    94    95    96    97    98    99
0   0.0  0.0  0.0  0.0  0.0  1.0  1.0  ...  28.0  28.0  28.0  28.0  28.0  28.0  28.0
1   NaN  0.0  0.0  0.0  0.0  0.0  0.0  ...  28.0  28.0  28.0  28.0  28.0  28.0  28.0
2   NaN  NaN  0.0  0.0  0.0  0.0  0.0  ...  27.0  28.0  28.0  29.0  29.0  29.0  30.0
3   NaN  NaN  NaN  0.0  0.0  0.0  0.0  ...  27.0  27.0  27.0  27.0  27.0  27.0  27.0
4   NaN  NaN  NaN  NaN  0.0  0.0  0.0  ...  23.0  23.0  27.0  27.0  27.0  27.0  27.0
..  ...  ...  ...  ...  ...  ...  ...  ...   ...   ...   ...   ...   ...   ...   ...
95  NaN  NaN  NaN  NaN  NaN  NaN  NaN  ...   NaN   NaN   0.0   0.0   0.0   0.0   0.0
96  NaN  NaN  NaN  NaN  NaN  NaN  NaN  ...   NaN   NaN   NaN   0.0   0.0   0.0   0.0
97  NaN  NaN  NaN  NaN  NaN  NaN  NaN  ...   NaN   NaN   NaN   NaN   0.0   0.0   0.0
98  NaN  NaN  NaN  NaN  NaN  NaN  NaN  ...   NaN   NaN   NaN   NaN   NaN   0.0   0.0
99  NaN  NaN  NaN  NaN  NaN  NaN  NaN  ...   NaN   NaN   NaN   NaN   NaN   NaN   0.0

[100 rows x 100 columns]
The total score of fold: 28.0
```

Below is the exact code used:

```
import numpy as np
import pandas as pd


class rna:
    def __init__(self, rna):
        length = len(rna)
        self.table = pd.DataFrame(np.nan, index=range(length), columns=range(length))
        self.rna = rna
        for i in range(self.table.shape[0]):
            for j in range(5):
                if (i + j < self.table.shape[0]):
                    self.table.loc[i, i + j] = 0
                else:
                    break

    def fold(self):
        self.calculateFold(0, len(self.rna))
        print(self.table)
        print("The total score of fold:", str(self.table.loc[0, len(self.rna)-1]))


    def findMax(self, array):
        temp = 0
        for i in array:
            if i > temp:
                temp = i
        return temp
```

```
    def calculateFold(self, x, y):
        if (x >= y - 4):
            return 0
        elif (pd.notna(self.table.loc[x, y-1])):
            return self.table.loc[x, y-1]
        else:
            for i in range(y):
                for j in range(i-4):
                    tempMax = self.table.iloc[j, i - 1]
                    splitMax = 0
                    splitTempMax = [0]
                    if (self.rna[j] == "A" and self.rna[i] == "U")
                    or (self.rna[j] == "U" and self.rna[i] == "A")
                    or (self.rna[j] == "C" and self.rna[i] == "G")
                    or (self.rna[j] == "G" and self.rna[i] == "C"):
                        for k in range(j, i+1):
                            one = self.calculateFold(j+1, k - 1)
                            two = self.calculateFold(k, i-1)
                            splitTempMax.append(1 + one + two)
                        splitMax = self.findMax(splitTempMax)
                    self.table.loc[j, i] = max(tempMax, splitMax)
        return splitMax


## Driver class

from rnaFold import rna

if __name__ == '__main__':
    # rnaSturcure = rna("AUAAACGUAAAACGGGGGGCCCCCUACAU")
    # rnaSturcure.fold()
    rna2 = rna("UUGUCAGGCGCCUCGGGUAAGUUGUCGUCUGAUAAAGUUGAGGAGCAUUCGC
    AUCUCAGUCCAAGGGACCUUGUAAGAGCAGGUCACUCACUAACGCUUC")
    rna2.fold()
```