

Report
Project 4: Gene Sequencing

1. Time and complexity of methods

```
#####
# def unrestricted_prep_array
# Preps array by adding first row
#
# Time Complexity:  $O(n)$  Creates  $n$  cells
# Space Complexity:  $O(n)$  Stores  $n$  cells
#####

#####
# def unrestricted_algorithm
# Sequences 2 strings
#
# Time Complexity:  $O(n * m)$  For each letter in string 1, traverse string 2
# Space Complexity:  $O(n * m)$  Makes and stores things in array of size  $n * m$ 
#####

#####
# def banded_prep_array
# Preps array by adding first row
#
# Time Complexity:  $O(k)$  Creates  $k$  cells,  $k = 4$ 
# Space Complexity:  $O(k)$  Stores  $k$  cells in list,  $k = 4$ 
#####

#####
# def banded_algorithm
# Sequences 2 strings
#
# Time Complexity:  $O(n * k)$  For each letter in string 1, visit 4 times,  $k = 4$ 
# Space Complexity:  $O(n * k)$  Makes and stores things in array of size  $n$  with 4 cells,  $k = 4$ 
#####

#####
# def get_smallest_combination
# returns smallest combination
#
# Time Complexity:  $O(n)$  visit  $n$  nodes and make a string
# Space Complexity:  $O(n)$  saves a string at about  $n$  long
#####

#####
# def get_score
# returns the score
#
# Time Complexity:  $O(1)$  Just returns the score
# Space Complexity: Not really applicable
#####
```

2. Paragraph

```
#####  
# KEY Array [i][j]  
#  
# String 1  
#  
# S j j j j j j j j  
# t i  
# r i  
# i i  
# n i  
# g i  
# 2 i  
# i  
  
# diagonal_index = [i - 1] [j - 1]  
# up_index       = [i - 1] [j]  
# left_index     = [i] [j - 1]
```

My algorithm start at index [0][0] and makes the first row of insertions, deletions to the right. Then for each subsequent row I reference the left child, upper diagonal child and upper child. I calculate the costs of each parent, determine the transition with the least amount of cost, then store that node and the new cells parent.

Because I am only storing the upper half diagonal of the data to avoid duplication, I did `self.string1_len + 1 - (i - 1)` loops on each subsequent row. So basically `n - 1` loops for each additional row.

Because I was only storing half the data and in a list format, tracking the parents was a little difficult. To access a diagonal parent, I accessed the cell right above. A upper parent was the cell right above and one the left. The left parent was the last cell in the list I was adding to.

3. Results

Gene Sequence Alignment

| | sequence1 | sequence2 | sequence3 | sequence4 | sequence5 | sequence6 | sequence7 | sequence8 | sequence9 | sequence10 |
|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|
| sequence1 | -30 | -1 | 4956 | 4956 | 4956 | 4956 | 4956 | 4956 | 4956 | 4956 |
| sequence2 | | -33 | 4948 | 4948 | 4948 | 4948 | 4948 | 4948 | 4948 | 4948 |
| sequence3 | | | -3000 | -2996 | -2956 | -2944 | -749 | -768 | -536 | -764 |
| sequence4 | | | | -3000 | -2960 | -2948 | -752 | -771 | -535 | -767 |
| sequence5 | | | | | -3000 | -2988 | -756 | -771 | -532 | -767 |
| sequence6 | | | | | | -3000 | -760 | -775 | -528 | -771 |
| sequence7 | | | | | | | -3000 | -2677 | -552 | -2673 |
| sequence8 | | | | | | | | -3000 | -592 | -2996 |
| sequence9 | | | | | | | | | -3000 | -2727 |
| sequence10 | | | | | | | | | | -3000 |

Label 3:

77808|gb|AF391541.1|Bovine coronavirus isolate BCoV-ENT, complete genome.

Sequence 3:

--gattgcgagcgattgctgctgcgtgcacccgcttcact-gatctcttgtagatctttcataatctaaactttataaa---aacatccactccctgt

Sequence 10:

ataagagtgattggcgtccgtacgtac-cctttctactctcaaaactctgttagtttaaactc-taactaaactttataaacggcacttcctgtgtgtcc

Label 10:

340|gb|AF208066.1|Murine hepatitis virus strain Penn 97-1, complete genome.

Process

Clear

☐ Banded Align Length: 1000

Done. Time taken: 1 mins and 2.379 seconds.

Gene Sequence Alignment

| | sequence1 | sequence2 | sequence3 | sequence4 | sequence5 | sequence6 | sequence7 | sequence8 | sequence9 | sequence10 |
|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|
| sequence1 | -30 | -1 | inf | inf | inf | inf | inf | inf | inf | inf |
| sequence2 | | -33 | inf | inf | inf | inf | inf | inf | inf | inf |
| sequence3 | | | -9000 | -8984 | -8888 | -8848 | -1857 | -1799 | -1204 | -1787 |
| sequence4 | | | | -9000 | -8888 | -8848 | -1857 | -1807 | -1207 | -1795 |
| sequence5 | | | | | -9000 | -8960 | -1864 | -1809 | -1186 | -1797 |
| sequence6 | | | | | | -9000 | -1864 | -1814 | -1185 | -1802 |
| sequence7 | | | | | | | -9000 | -8009 | -1110 | -8005 |
| sequence8 | | | | | | | | -9000 | -1120 | -8980 |
| sequence9 | | | | | | | | | -9000 | -1066 |
| sequence10 | | | | | | | | | | -9000 |

Label 3:

77808|gb|AF391541.1|Bovine coronavirus isolate BCoV-ENT, complete genome.

Sequence 3:

--gattgcgagcgattgctgctgcgtgcacccgcttcact-gatctcttgtagatctttcataatctaaactttataaa-aacatccactccctgtgtag

Sequence 10:

ataagagtgattggcgtccgtacgtac-cctttctactctcaaaactctgttagtttaaactc-taactaaactttataaacggcacttcctgtgtgtc-c

Label 10:

340|gb|AF208066.1|Murine hepatitis virus strain Penn 97-1, complete genome.

Process

Clear

☒ Banded Align Length: 3000

Done. Time taken: 1.760 seconds.

4. Code

```
#####  
# KEY Array [i][j]  
#  
# String 1  
#  
# S j j j j j j j j  
# t i  
# r i  
# i i  
# n i  
# g i  
# 2 i  
# i
```

```
from enum import Enum
```

```
#####  
# Cell  
# Stores data for each cell in array  
#####  
class Cell:  
  
    def __init__(self, parent, transition_type, cost):  
        self.parent = parent  
        self.transition_type = transition_type  
        self.cost = cost
```

```
#####  
# Transition  
# Stores different transitions  
#####  
class Transition(Enum):  
    UP = 0  
    LEFT = 1  
    DIAGONAL = 2
```

```
#####  
# Sequence  
# Sequences 2 strings  
#  
# Time Complexity: Depends on user input  
# if banded see banded_algorithm complexity  
# if not see unrestricted_algorithm complexity  
# Space Complexity: Depends on user input  
# if banded see banded_algorithm complexity  
# if not see unrestricted_algorithm complexity  
#####  
class Sequence:
```

```
    def __init__(self, string1, string2, banded):  
        self.string1 = string1  
        self.string1_len = len(string1)  
        self.string2 = string2  
        self.string2_len = len(string2)  
        self.array = []
```

```

if not banded:
    self.unrestricted_prep_array()
    self.unrestricted_algorithm()
    self.score = self.get_score()
else:
    # We do not allow more than 3 insertions/ deletion
    if self.string1_len - self.string2_len <= 4:
        self.banded_prep_array()
        self.banded_algorithm()
        self.score = self.get_score()
    else:
        self.score = float('inf')

#####
# def unrestricted_prep_array
# Preps array by adding first row
#
# Time Complexity: O(n) Creates n cells
# Space Complexity: O(n) Stores n cells
#####
def unrestricted_prep_array(self):
    # add first row to array
    self.array.append([])

    # temp cell
    tmp_cell = Cell(None, None, 0)

    # add first cell to array
    self.array[0].append(tmp_cell)

    # add remaining cells to row
    for j in range(1, self.string1_len + 1):
        tmp_cell = Cell(tmp_cell, Transition.LEFT, j * 5)
        self.array[0].append(tmp_cell)

#####
# def unrestricted_algorithm
# Sequences 2 strings
#
# Time Complexity: O(n * m) For each letter in string 1, traverse string 2
# Space Complexity: O(n * m) Makes and stores things in array of size n * m
#####
def unrestricted_algorithm(self):

    # for each row (string 2)
    for i in range(1, self.string2_len + 1):

        # add another row to array
        self.array.append([])

        for j in range(1, self.string1_len + 1 - (i - 1)):

            # diagonal_index = [i - 1] [j - 1]
            # up_index       = [i - 1] [j]
            # left_index      = [i]    [j - 1]

            # get up cost
            up_cost = self.array[i - 1][j].cost + 5

            # get left cost

```

```

if len(self.array[i]) > 0:
    left_cost = self.array[i][-1].cost + 5
else:
    left_cost = float("inf")

if self.string1[j - 1 + (i - 1)] == self.string2[i - 1]:
    strings_match = True
else:
    strings_match = False

if strings_match:
    # strings are the same, cost: -3
    diagonal_cost = self.array[i - 1][j - 1].cost - 3
else:
    # strings are not the same, cost: +1
    diagonal_cost = self.array[i - 1][j - 1].cost + 1

# stores smallest parent
if diagonal_cost <= up_cost and diagonal_cost <= left_cost:
    if strings_match:
        self.array[i].append(Cell(self.array[i - 1][j - 1],
                                   Transition.DIAGONAL,
                                   diagonal_cost))
    else:
        self.array[i].append(Cell(self.array[i - 1][j - 1],
                                   Transition.DIAGONAL,
                                   diagonal_cost))
elif up_cost <= left_cost:
    self.array[i].append(Cell(self.array[i - 1][j],
                              Transition.UP,
                              up_cost))
else:
    self.array[i].append(Cell(self.array[i][-1],
                              Transition.LEFT,
                              left_cost))

#####
# def banded_prep_array
# Preps array by adding first row
#
# Time Complexity: O(k) Creates k cells, k = 4
# Space Complexity: O(k) Stores k cells in list, k = 4
#####
def banded_prep_array(self):
    # add first row to array
    self.array.append([])

    # temp cell
    tmp_cell = Cell(None, None, 0)

    # add first cell to array
    self.array[0].append(tmp_cell)

    # add remaining cells to row
    for j in range(1, 4):
        tmp_cell = Cell(tmp_cell, Transition.LEFT, j * 5)
        self.array[0].append(tmp_cell)

#####
# def banded_algorithm

```

[illegible]

```

        else:
            self.array[i].append(Cell(self.array[i - 1][j - 1],
                                     Transition.DIAGONAL,
                                     diagonal_cost))
    elif up_cost <= left_cost:
        self.array[i].append(Cell(self.array[i - 1][j],
                                   Transition.UP,
                                   up_cost))
    else:
        self.array[i].append(Cell(self.array[i][j - 1],
                                   Transition.LEFT,
                                   left_cost))

#####
# def get_smallest_combination
# returns smallest combination
#
# Time Complexity: O(n) visit n nodes and make a string
# Space Complexity: O(n) saves a string at about n long
#####
def get_combo_string(self):

    if self.score == float('inf'):
        return ["No Alignment Possible", "No Alignment Possible"]

    combo_string_a = ""
    combo_string_b = ""

    tmp_cell = self.array[self.string2_len][-1]
    string1_index = self.string1_len - 1
    string2_index = self.string2_len - 1

    while tmp_cell is not None:

        if tmp_cell.transition_type == Transition.UP:
            combo_string_a = "-" + combo_string_a
            combo_string_b = self.string2[string2_index] + combo_string_b
            string2_index -= 1

        elif tmp_cell.transition_type == Transition.LEFT:
            combo_string_a = self.string1[string1_index] + combo_string_a
            combo_string_b = "-" + combo_string_b
            string1_index -= 1

        elif tmp_cell.transition_type == Transition.DIAGONAL:
            combo_string_a = self.string1[string1_index] + combo_string_a
            combo_string_b = self.string2[string2_index] + combo_string_b
            string1_index -= 1
            string2_index -= 1

    tmp_cell = tmp_cell.parent

    return [combo_string_b[:100], combo_string_a[:100]]

#####
# def get_score
# returns the score
#
# Time Complexity: O(1) Just returns the score
# Space Complexity: Not really applicable

```



```
#####  
def get_score(self):  
    return self.array[self.string2_len][-1].cost
```