Project 2 (Global Sequence Alignment) Checklist

Prologue

Project goal: write a program to compute the optimal sequence alignment of two DNA strings

Relevant lecture material

- ⇝ Control Flow
- ⇝ Collections
- ⇝ Input and Output

Files

- ⇝ `project2.pdf` (project description)
- ⇝ `project2.zip` (starter files for the exercises/problems, `report.txt` file for the project report, `run_tests.py` file to test your solutions, and test data files)

Exercises

Exercise 1. (*Three Equal Numbers*) Write a program `equality.py` that takes three integers as command-line arguments and writes "equal" if all three are equal, and "not equal" otherwise.

```
$ python3 equality.py 5 5 5
equal
```

```
$ python3 equality.py 5 1 5
not equal
```

Exercises

equality.py

```
import stdio
import sys

# Get x, y, and z from command line, as ints.
x = ...
y = ...
z = ...

# If x, y, and z are all equal, ie, x equals y and y equals z, then
# write "equal". Otherwise, write "not equal".
if ...:
    ...
else:
    ...
```

Exercises

Exercise 2. (*k Per Row*) Write a program `k_per_row.py` that takes three integers a, b, and k as command-line arguments writes the integers between a and b (both inclusive), but only k integers per row.

```
$ python3 k_per_row.py 101 200 5
101 102 103 104 105
106 107 108 109 110
111 112 113 114 115
116 117 118 119 120
121 122 123 124 125
126 127 128 129 130
131 132 133 134 135
136 137 138 139 140
141 142 143 144 145
146 147 148 149 150
151 152 153 154 155
156 157 158 159 160
161 162 163 164 165
166 167 168 169 170
171 172 173 174 175
176 177 178 179 180
181 182 183 184 185
186 187 188 189 190
191 192 193 194 195
196 197 198 199 200
```

Exercises

k_per_row.py

```
import stdio
import sys

# Get a, b, and k from command line, as ints.
a = ...
b = ...
k = ...

# Iterate over the integers a to b (both inclusive), and
# write k of them per row.
for i in range(..., ...):
    # If i is a multiple of k, write i followed by a
    # new line. Otherwise, write i followed by a space.
    if ...:
        ...
    else:
        ...
```

Exercises

Exercise 3. (*Counting Primes*) Write a program `prime_counter.py` that takes an integer N as command-line argument and writes the number of primes less than or equal to N. Recall that a number i is prime if it is not divisible by any number $j \in [2, \sqrt{i}]$.

```
$ python3 prime_counter.py 1000
168
```

Exercises

prime_counter.py

```
import stdio
import sys

# Get N from command line, as an int.
N = ...

# Define primes to store the result (number of primes <= N).
primes = ...

# Iterate over integers 2 to N (inclusive).
for i in range(..., ...):
    # Define a variable j to store the potential divisors of i,
    # and initialize it to 2.
    ...

    # Repeat as long as j is less than or equal to i / j.
    while ...:
        # If i is divisible by j, it is not a prime so exit
        # (break) this inner loop.
        if ...:
            ...

        # Increment j by 1.
        ...

    # If j is greater than i / j, then i is a prime. So
    # increment primes by one.
    if ...:

# Write primes.
...
```

Exercises

Exercise 4. (*Birthday Problem*) Suppose that people enter an empty room until a pair of people share a birthday. On average, how many people will have to enter before there is a match? Write a program `birthday.py` that takes an integer *trials* from the command line and runs *trials* experiments to estimate this quantity, where each experiment involves sampling individuals until a pair of them share a birthday.

```
$ python3 birthday.py 1000
24
```

Exercises

birthday.py

```
import random
import stdarray
import stdio
import sys

DAYS_PER_YEAR = 365

# Get trials from command line, as an int.
trials = ...

# Define a variable count denoting the total number of
# individuals sampled across the trials number of experiments,
# and initialize it to 0.
count = ...

# Peform trials number of experiments, where each experiment
# involves sampling individuals until a pair of them share
# a birthday.
for t in range(...):
    # Setup a 1D list birthdays_seen of DAYS_PER_YEAR booleans,
    # all set to False by default. This list will keep track
    # of the birthdays encountered in this experiment.
    birthdays_seen = ...

    # Sample individuals until match.
    while True:
        # Increment count by 1.
        ...

        # Define a variable birthday with a random integer
        # from the interval [0, DAYS_PER_YEAR).
        birthday = ...

        # If birthday has been encountered, abort this experiment.
        if ...:
            ...

        # Record the fact that we are seeing this birthday for
```

Exercises

birthday.py

```
        # the first time.
        else:
            ...

# Write the average number of people that must be sampled before
# a match, as an int.
stdio.writeln(...)
```

Exercises

Exercise 5. (*Pascal's Triangle*) Pascal's triangle $\mathcal{P}_n$ is a triangular array with $n+1$ rows, each listing the coefficients of the binomial expansion $(x+y)^i$, where $0 \leq i \leq n$. For example, $\mathcal{P}_4$ is the triangular array:

$$
\begin{array}{ccccc}
1 & & & & \\
1 & 1 & & & \\
1 & 2 & 1 & & \\
1 & 3 & 3 & 1 & \\
1 & 4 & 6 & 4 & 1
\end{array}
$$

The term $\mathcal{P}_n(i,j)$ is calculated as $\mathcal{P}_n(i-1,j-1) + \mathcal{P}_n(i-1,j)$, where $0 \leq i \leq n$ and $1 \leq j < i$, with $\mathcal{P}_n(i,0) = \mathcal{P}_n(i,i) = 1$ for all i. Write a program `pascal.py` that takes an integer n as command-line argument and writes $\mathcal{P}_n$.

```
$ python3 pascal.py 10
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

Exercises

pascal.py

```
import stdarray
import stdio
import sys

# Get n from command line, as an int.
n = ...

# Construct a 2D ragged list a of integers. The list must
# have n + 1 rows, with the ith (0 <= i <= n) row a[i] having
# i + 1 elements, each initialized to 1. For example, if n = 3,
# a should be initialized to [[1], [1, 1], [1, 1, 1], [1, 1, 1, 1]].
a = []
for i in range(...):
    a += ...

# Fill the ragged list a using the formula for Pascal's triangle
#     a[i][j] = a[i - 1][j - 1] + a[i - 1][j]
# where 0 <= i <= n and 1 <= j < i.
for i in range(..., ...):
    for j in range(..., ...):
        ...

# Write out the elements of the ragged list a.
for i in range(..., ...):
    for j in range(..., ...):
        # If j is not the last column, write the element with a
        # space after.
        if ...:
            ...
        # Otherwise, write the element with a newline after.
        else:
            ...
```

Problems

Student

The guidelines for the project problems that follow will be of help only if you have read the description of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Instructor

Please summarize the project description for the students before you walk them through the rest of this checklist document.

Problems

Problem 1. (*Calculating Edit Distance Using Dynamic Programming*) Write a program `edit_distance.py` that reads strings `x` and `y` from standard input and computes the edit-distance matrix `opt`. The program should output `x`, `y`, the dimensions (number of rows and columns) of `opt`, and `opt` itself.

Hints

- ⇝ Read the sequences `x` and `y` from standard input, as strings
- ⇝ Create an `(M + 1) x (N + 1)` edit-distance matrix `opt` with all elements initialized to `0`, where `M` and `N` are the lengths of `x` and `y` respectively
- ⇝ Note that the value `opt[i][j]` respresents the optimal edit distance between the strings `x[i:]` and `y[j:]`
- ⇝ Initialize the bottom row of `opt` to `2 * (N - j)` and its right column to `2 * (M - i)`, where `0 <= j <= N` and `0 <= i <= M`
- ⇝ For example, if `x = 'HAM'` (`M = 3`) and `y = 'SPAM'` (`N = 4`), then the corresponding `opt` matrix after the above step is:

```
       |  0  1  2  3  4
   x\y |  S  P  A  M  -
----------------------
 0  H  |  0  0  0  0  6
 1  A  |  0  0  0  0  4
 2  M  |  0  0  0  0  2
 3  -  |  8  6  4  2  0
```

- ⇝ Exercise: Work out the initial `opt` matrix by hand for the strings `x = 'CUTE'` (`M = 4`) and `y = 'CHUTE'` (`N = 5`)

Problems

- Fill in the rest of the `opt` matrix, starting at `opt[M - 1][N - 1]` and ending at `opt[0][0]`, as follows: if `x[i]` and `y[j]` are the same, where `0 <= i <= M - 1` and `0 <= j <= N - 1`, then

```
opt[i][j] = min(opt[i + 1][j + 1], opt[i + 1][j] + 2, opt[i][j + 1] + 2)
```

  and

```
opt[i][j] = min(opt[i + 1][j + 1] + 1, opt[i + 1][j] + 2, opt[i][j + 1] + 2)
```

  otherwise

- The `opt` matrix for the above example after the preceding step is:

```
       |  0  1  2  3  4
   x\y |  S  P  A  M  -
----------------------
 0  H  |  3  1  2  4  6
 1  A  |  4  2  0  2  4
 2  M  |  6  4  2  0  2
 3  -  |  8  6  4  2  0
```

- Exercise: Work out rest of the `opt` matrix by hand for the strings `x = 'CUTE'` and `y = 'CHUTE'`

Problems

- Write the following output, each starting on a new line:
  - String `x`
  - String `y`
  - Dimensions of the `opt` matrix separated by a space
  - Elements of `opt`, using format string `'%3d '` for elements *not* on the last column, and `'%3d\n'` for the last-column elements

Problems

Problem 2. (*Recovering the Alignment*) Write a program `alignment.py` that reads from standard input, the output produced by `edit_distance.py`, ie, input strings `x` and `y`, and the `opt` matrix. The program should then recover an optimal alignment, and write to standard output the edit distance between `x` and `y` and the alignment itself.

Hints

- ⇝ Read from standard input the sequences `x` and `y` as strings, and the matrix `opt` as a 2D array of integers
- ⇝ Write the edit distance between `x` and `y`, ie, the value of `opt[0][0]`
- ⇝ Recover and output the alignment, starting at `opt[0][0]` and ending at `opt[M - 1][N - 1]`, as follows:
  - ⇝ If `opt[i][j]` equals `opt[i + 1][j] + 2`, then align `x[i]` with a gap and penalty of 2, and increment `i` by 1
  - ⇝ Otherwise if `opt[i][j]` equals `opt[i][j + 1] + 2`, then align a gap with `y[j]` and penalty of 2, and increment `j` by 1
  - ⇝ Otherwise, align `x[i]` with `y[j]` and penalty of 0/1 based on whether `x[i]` and `y[j]` match or not, and increment both `i` and `j` by 1

  If x (or y) is exhausted before y (or x), align a character from y (or x) with a gap and penalty of 2

Problems

- ⇝ For our running example, the optimal alignment (with edit distance 3) produced by the previous step is:

```
- S 2
H P 1
A A 0
M M 0
```

- ⇝ Exercise: Work out the optimal alignment by hand for the strings `x = 'CUTE'` and `y = 'CHUTE'`

Problems

Test your programs thoroughly using the short test data files and actual genomic data files under the `data` directory

Optimal edit distances for some of the supplied files:

```
ecoli2500.txt   118
ecoli5000.txt   160
fli8.txt          6
fli9.txt          4
fli10.txt         2
ftsa1272.txt    758
gene57.txt        8
stx1230.txt     521
stx19.txt        10
stx26.txt        17
stx27.txt        19
```

Problems

Test cases with unique optimal alignments:

```
$ python3 edit_distance.py < data/endgaps7.txt | python3 alignment.py
Edit distance = 4
a - 2
t t 0
a a 0
t t 0
t t 0
a a 0
t t 0
- a 2
```

```
$ python3 edit_distance.py < data/fli10.txt | python3 alignment.py
Edit distance = 2
T T 0
G G 0
G G 0
C T 1
G G 0
G G 0
A T 1
A A 0
C C 0
T T 0
```

Epilogue

Use the template file `report.txt` to write your report for the project

Your report must include

- ⇝ Time (in hours) spent on the project
- ⇝ Difficulty level (1: very easy; 5: very difficult) of the project
- ⇝ A short description of how you approached each problem, issues you encountered, and how you resolved those issues
- ⇝ Acknowledgement of any help you received
- ⇝ Other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Epilogue

Before you submit your files

- ⇝ Make sure your programs meet the style requirements by running the following command on the terminal

```
$ pycodestyle <program>
```

  where `<program>` is the `.py` file whose style you want to check

- ⇝ Make sure your programs meet the input and output specifications by running the following command on the terminal

```
$ python3 run_tests.py -v [<items>]
```

  where the optional argument `<items>` lists the exercises/problems (`Exercise1`, `Problem2`, etc.) you want to test, separated by spaces; all the exercises/problems are tested if no argument is given

- ⇝ Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and running time
- ⇝ Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit

1. `equality.py`
2. `k_per_row.py`
3. `prime_counter.py`
4. `birthday.py`
5. `pascal.py`
6. `edit_distance.py`
7. `alignment.py`
8. `report.txt`