Data Structures and Algorithms in Java

Assignment 2 (Global Sequence Alignment) Discussion

Reverse.java

Standard input	a sequence of strings
Standard output	the strings in reverse order

~/workspace/global_sequence_alignment

```
$ javac -d out src/Reverse.java
$ java Reverse
b o l t o n
<ctrl-d>
n o t l o b
$ java Reverse
m a d a m
<ctrl-d>
m a d a m
```

Read all strings from standard input into an array a (use `StdIn.readAllStrings()`)

Set n to the size of a

For each int $i \in [0, n/2)$
- Exchange $a[i]$ with $a[n - i - 1]$

For each int $i \in [0, n)$
- If $i < n - 1$, write $a[i]$ with a space after; otherwise, write $a[i]$ with a newline after

Distance.java

Command-line input	n (int)
Standard input	two size-n arrays x and y of doubles
Standard output	the Euclidean distance between the two vectors represented by x and y

>_ ~/workspace/global_sequence_alignment

```
$ javac -d out src/Distance.java
$ java Distance 2
1 0 0 1 <enter>
1.4142135623730951
$ java Distance 5
-9 1 10 -1 1 -5 9 6 7 4 <enter>
13.0
```

Accept n (int) as command-line argument

Create a size-n array x of doubles

For each int $i \in [0, n)$
- Set $a[i]$ to a double read from standard input (use `StdIn.readDouble()`)

Create and initialize a size-n array y of doubles similar to x

Set *sum* (double) to 0

For each int $i \in [0, n)$
- Increment *sum* by $(x[i] - y[i])^2$

Write $\sqrt{sum}$

Transpose.java

Command-line input	m (int) and n (int)
Standard input	$m \times n$ doubles representing the elements of an $m \times n$ matrix a
Standard output	the transpose of a

~/workspace/global_sequence_alignment

```
$ javac -d out src/Transpose.java
$ java Transpose 2 2
1 2 3 4 <enter>
1.0 3.0
2.0 4.0
$ java Transpose 2 3
1 2 3 4 5 6 <enter>
1.0 4.0
2.0 5.0
3.0 6.0
```

Accept m (int) and n (int) as command-line arguments

Create an $m \times n$ array a of doubles

For each int $i \in [0, m)$
- For each int $j \in [0, n)$
  - Set $a[i][j]$ to a double read from standard input (use `StdIn.readDouble()`)

Create an $n \times m$ array c of doubles

For each int $i \in [0, n)$
- For each int $j \in [0, m)$
  - Set $c[i][j]$ to $a[j][i]$

For each int $i \in [0, n)$
- For each int $j \in [0, m)$
  - If $j < m - 1$, write $c[i][j]$ with a space after; otherwise, write $c[i][j]$ with a newline after

StrangeMatrix.java

Command-line input	m (int) and n (int)
Standard output	an $m \times n$ (strange) matrix

~/workspace/global_sequence_alignment

```
$ javac -d out src/StrangeMatrix.java
$ java StrangeMatrix 4 5
  147     66     27     10      3
   54     27     12      5      2
   17     10      5      2      1
    4      3      2      1      0
```

Accept m (int) and n (int) as command-line arguments

Create an $m \times n$ array a of ints

For each int $i \in [0, m)$
- Set $a[i][n-1]$ to $m - i - 1$

For each int $j \in [0, n)$
- Set $a[m-1][j]$ to $n - j - 1$

For each int $i \in [m-2, 0]$
- For each int $j \in [n-2, 0]$
    - Set $a[i][j]$ to $a[i][j+1] + a[i+1][j+1] + a[i+1][j]$

For each int $i \in [0, m)$
- For each int $j \in [0, n)$
    - If $j < n-1$, write $a[i][j]$ with the format string `"%5d "`; otherwise, write $a[i][j]$ with the format string `"%5d\n"`

Pascal.java

Command-line input	n (int)
Standard output	Pascal's triangle $\mathcal{P}_n$

>_ ~/workspace/global_sequence_alignment

```
$ javac -d out src/Pascal.java
$ java Pascal 5
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Accept n (int) as command-line argument

Create a 2D array a of ints with $n + 1$ rows (leave the column capacity empty)

For each int $i \in [0, n]$
- Set $a[i]$ to an array of $i + 1$ ints
- For each int $j \in [0, i]$
    - Set $a[i][j]$ to 1

For each int $i \in [0, n]$
- For each int $j \in [1, i)$
    - Set $a[i][j]$ to $a[i-1][j-1] + a[i-1][j]$

For each int $i \in [0, n]$
- For each int $j \in [0, i]$
    - If $j < i$, write $a[i][j]$ with a space after; otherwise, write $a[i][j]$ with a newline after

Goal: find an optimal alignment for two DNA sequences `x` and `y`

We are permitted to insert gaps in either sequence to make them have the same length

We pay a penalty for each gap that we insert and also for each pair of characters that mismatch

Operation	Cost
Insert a gap	2
Align two characters that do not match	1
Align two characters that do match	0

Edit distance is the cost of the best possible alignment between the two genetic sequences over all possible alignments

Two possible alignments of the sequences `x = "AACAGTTACC"` and `y = "TAAGGTCA"`

x	y	cost
A	T	1
A	A	0
C	A	1
A	G	1
G	G	0
T	T	0
T	C	1
A	A	0
C	-	2
C	-	2
		8

x	y	cost
A	T	1
A	A	0
C	-	2
A	A	0
G	G	0
T	G	1
T	T	0
A	-	2
C	C	0
C	A	1
		7

Edit distance for the two sequences is 7

`m` and `n` denote the lengths of `x` and `y`, respectively

`x[i]` denotes the `i`th character of the sequence `x`

`x[i..m]` denotes the suffix of `x` consisting of the characters `x[i]`, `x[i + 1]`, ..., `x[m - 1]`

`opt` is the `(m + 1) x (n + 1)` edit-distance matrix

`opt[i][j]` denotes the edit distance of `x[i..m]` and `y[j..n]`

Example: if `x = "AACAGTTACC"` and `y = "TAAGGTCA"`, then

- `m = 10` and `n = 8`
- `x[2]` is `'C'`
- `x[5..m]` is `"CAGTTACC"` and `y[8..n]` is `""`
- `opt` is a `11 x 9` matrix
- `opt[0][0]` is the edit distance of `x` and `y`

Case 1 (`x[i]` is matched with `y[j]`): `opt[i][j] = opt[i + 1][j + 1] + 0 or 1` depending on whether `x[i]` equals `y[j]`

Case 2 (`x[i]` is matched with a gap): `opt[i][j] = opt[i + 1][j] + 2`

Case 3 (`y[j]` is matched with a gap): `opt[i][j] = opt[i][j + 1] + 2`

We compute `opt[i][j]` by taking the minimum of the three quantities

```
opt[i][j] = min(opt[i + 1][j + 1] + 0 or 1, opt[i + 1][j] + 2, opt[i][j + 1] + 2)
```

Direct computation of this recursive scheme is spectacularly inefficient

We use dynamic programming

Key idea: break up a large problem into smaller subproblems, store the answers to those smaller subproblems, and use the stored answers to solve the original problem

Write a program called `EditDistance.java` that reads strings `x` and `y` from standard input; computes the edit-distance matrix `opt`; and outputs `x`, `y`, the dimensions of `opt`, and `opt`

```
>_ ~/workspace/global_sequence_alignment

$ javac -d out src/EditDistance.java
$ java EditDistance < data/example10.txt
AACAGTTACC
TAAGGTCA
11 9
  7   8  10  12  13  15  16  18  20
  6   6   8  10  11  13  14  16  18
  6   5   6   8   9  11  12  14  16
  7   5   4   6   7   9  11  12  14
  9   7   5   4   5   7   9  10  12
  8   8   6   4   4   5   7   8  10
  9   8   7   5   3   3   5   6   8
 11   9   7   6   4   2   3   4   6
 13  11   9   7   5   3   1   3   4
 14  12  10   8   6   4   2   1   2
 16  14  12  10   8   6   4   2   0
```

Read sequences `x` (String) and `y` (String) from standard input

Set `m` (int) and `n` (int) to the lenghts of `x` and `y`, respectively (use `GSA.length()`)

Create an `(m + 1) x (n + 1)` array `opt` of ints

Initialize the rightmost column of opt to `2(m - i)`, where `0 <= i <= m`

Initialize the bottommost row of opt to `2(n - j)`, where `0 <= j <= n`

Fill in the rest of `opt`, starting at `opt[m - 1][n - 1]` and ending at `opt[0][0]`, as follows (use `GSA.charAt()` and `GSA.min()` where needed)

- If `x[i] = y[j]` then `opt[i][j] = min(opt[i + 1][j + 1], opt[i + 1][j] + 2, opt[i][j + 1] + 2)`
- Otherwise, `opt[i][j] = min(opt[i + 1][j + 1] + 1, opt[i + 1][j] + 2, opt[i][j + 1] + 2)`

Write the following output, each starting on a new line

- `x`
- `y`
- `m` and `n` separated by a space
- `opt` using the format string `"%3d "` for elements not in the last column, and `"%3d\n"` for the last-column elements

Write a program `Alignment.java` that reads from standard input the output produced by `EditDistance.java`; recovers an optimal alignment between `x` and `y`; and writes the edit distance and the alignment

>_ ~/workspace/global_sequence_alignment

```
$ javac -d out src/Alignment.java
$ java EditDistance < data/example10.txt | java Alignment
7
A  T  1
A  A  0
C  -  2
A  A  0
G  G  0
T  G  1
T  T  0
A  -  2
C  C  0
C  A  1
```

Read sequences `x` (String) and `y` (String) from standard input

Set `m` (int) and `n` (int) to the lenghts of `x` and `y`, respectively

Read the edit-distance matrix `opt` from standard input (use `StdArrayIO.readInt2D()`)

Write the edit distance between `x` and `y`, ie, the value of `opt[0][0]`

Set ints `i` and `j` both to `0`

Recover and output the optimal alginment, starting at `opt[0][0]` and ending at `opt[m - 1][n - 1]`, as follows

- If `opt[i][j] = opt[i + 1][j] + 2`, then align `x[i]` with a gap and penalty of 2, and increment `i`
- Otherwise, if `opt[i][j] = opt[i][j + 1] + 2`, then align `y[j]` with a gap and penalty of 2, and increment `j`
- Otherwise, align `x[i]` with `y[j]` with a penalty of 0 or 1 depending on whether `x[i]` equals `y[j]`, and increment both `i` and `j`

If `y` is exhausted before `x` (ie, `i < m`), align the remaining `x` with gaps and penalty of 2

If `x` is exhausted before `y` (ie, `j < n`), align the remaining `y` with gaps and penalty of 2