Stanford ProCo

May 14, 2011

Problem Packet
Advanced Division

Sponsored by:

Advanced 2.1 **Topic nine: anagrams**

Overview: Given two strings, check whether they are anagrams of each other.

Description: One way of planting an idea into someone else's mind is to leave indirect hints of what you want the person to think. For instance, if you show the word "INCEPTION" to your debate opponents enough times, maybe they'll start thinking, "NICE POINT," which is an anagram of "INCEPTION." If you see the names of the characters Arthur, Dom, Robert, Eames, Ariadne, and Mal enough times, maybe you'll start thinking it's all "A DREAM," which is an anagram of the first initials of their names.

Help Cobb and his team complete their inception mission by checking whether they are planting the correct anagrams in their target's mind. Given a series of two strings, check whether they are anagrams of each other.

Filename: adv21.{java, c, cpp, cc, py}

Input: The input will consist of two lines. Each line will be a string. You must check whether these two strings are anagrams of each other.

Output: Output either `YES` if they are anagrams or `NO` if they are not.

Assumptions: All strings contain only uppercase letters and spaces.
Each string will have length between 0 and 20000, inclusive.
All input is valid.

Sample Input #1:
```
INTERNET ANAGRAM SERVER
I REARRANGEMENT SERVANT
```

Sample Output #1:
```
YES
```

Sample Input #2
```
ABCDEFGHIJKLMNOPQR
ZZZZZZZZZZZZZZZZZZ
```

Sample Output #2
```
NO
```

Advanced 2.2 **diff dreams reality**

Overview: Take repeated differences of a sequence to find the value at which it first becomes a constant sequence.

Description: Dom and company are multiple levels down in a dream, but they want to jump back to reality. Each dream is represented by a finite sequence of numbers. They know they are in a reality if the sequence of numbers is a constant. To jump back up a level, Dom and company need to recreate the sequence of numbers representing that level.

Given a finite sequence of numbers representing a level, take the difference of consecutive integers to create a new sequence representing one level closer to reality. More precisely, the sequence representing the level above the level (x_1, x_2, …, x_n) will be (x_2 - x_1, x_3 - x_2, …, x_n - x_{n-1}). For example, jumping up one level from “5 2 8 4 9 1” will get to "-3 5 -4 5 -8". Repeating this process a finite number of times (perhaps zero) will always eventually get to a constant sequence in which all the values of the sequence are equal (perhaps of length 1).

For example, starting with the sequence “6 3 4 9”, the next sequence is “-3 1 5”, followed by “4 4”. This is the first constant sequence, and thus you should output the number 4.

Help Dom and his team find reality by computing the value constant in the first constant sequence reached by successively taking differences. This constant represents their reality.

Filename: adv22.{java, c, cpp, cc, py}

Input: The first line contains a single integer n, the length of the input sequence.
The second line contains n integers, separated by a single space, which the values of the input sequence $x_1, x_2, \ldots, x_n$.

Output: Output a single line containing a single integer, the value of the first constant sequence.

Assumptions: $1 \le n \le 10{,}000$
There will be no integer overflow in taking consecutive differences in any sequences representing any of the dream levels.
All input is valid.

Sample Input #1:
```
4
6 3 4 9
```

Sample Output #1:
```
4
```

Sample Input #2:
```
6
5 2 8 4 9 1
```

Sample Output #2:
```
-79
```

Advanced 2.3 **Merging mergesort** (page 1 of 1)

Overview: Given two sorted arrays, merge them.

Description: Dom and Ariadne are in disagreement over the details of a particular dreamscape. As a compromise, they decide to each create only a part of the details of the dreamscape and then merge the two together into one. Each of them come up with a list of integers representing their respective ideas of the dream world. Help them merge them into one list representing the cohesive dreamscape.

You are given two sorted arrays of integers. Write a program that will merge both arrays into a single, sorted, output array. The merge algorithm is as follows: compare the first element of both arrays and determine the smaller element (ties may be broken arbitrarily). Place this element in the next open spot in the output array. If this element came from the first array, then the next comparison should compare the second element of the first array with the first element of the second array (and vice versa if the element came from the second array). Continue in this manner until one of the arrays runs out of numbers, in which case simply place all the numbers remaining in the other array at the end of the output array.

Filename: adv23.{java, c, cpp, cc, py}

Input: The first line contains two integers, *a* and *b*, separated by a single space, representing the number of elements in the first and second arrays. The next line contains *a* integers separated by spaces, representing the *a* elements of the first array. The next line contains *b* integers separated by spaces, representing the *b* elements of the second array.

Output: The output should a sorted list of *a*+*b* integers, printed on a single line with a single space between each integer.

The output is to be formatted exactly like the sample output given below.

Assumptions: $1 \le a, b \le 1000$
All input is valid.

Sample Input #1:
```
2 3
2 5
1 3 4
```

Sample Output #1:
```
1 2 3 4 5
```

Sample Input #2:
```
5 3
2 7 7 11 17
5 7 17
```

Sample Output #2:
```
2 5 7 7 7 11 17 17
```

Advanced 2.4 **Shifting dreams**

Overview: Rotate a permutation by one.

Description: Cobb has obtained a password to access a Swiss Bank account from his last extraction target. However, in a moment of weakness, he manages to accidentally scramble some of the digits that make up the password. He needs your help to unscramble the digits he was given in order to gain access to the account.

Cobb will give you a permutation p of the integers 0 through n - 1, inclusive. Let $p(i)$ be the i-th digit in p. So for p = 30214, then $p(0) = 3$, $p(1) = 0$, etc. Your goal is to rotate the elements in the permutation so that each index i is replaced by the element at $p(i)$. For example, after rotation, the number at index 0 will be 1, the element at index 1 will be 3, and so on.

Filename: adv24.{java, c, cpp, cc, py}

Input: The first line of input will be an integer n, representing the length of the permutation. The second line of the string will be the permutation p, where each character will be an integer between 0 and n - 1, inclusive.

Output: The output should contain the permutation after the rotation operation has been applied.

Assumptions: $1 \le n \le 10$
All input is valid.

Sample Input #1:
```
5
30214
```

Sample Output #1:
```
13204
```

Sample Input #2:
```
4
1230
```

Sample Output #2:
```
2301
```

Advanced 5.1 **Black and white and gray** (page 1 of 1)

Overview: Convert from Gray code to binary.

Description: Yusuf, left behind in the first level dream, counts time using an old electromechanical clock that increments once per second. However, the clock is very unstable and will produce spurious readings sometimes, when the individual bit switches on the clock get out of synchronization. Yusuf modifies the clock to display the time in terms of the Gray code, but now he has trouble reading the numbers on the digit. Help him read the clock by converting Gray Code integers to binary.

In normal binary representation, an increment from 3 (011) to 4 (100) would involve the flipping of all 3 bit switches at once. Due to the unreliable nature of old electromechanical switches, these 3 bit switches might not change at exactly the same moment, leading to spurious results when the value is read in a transition state instead of a final state. Gray codes encode binary values such that only one bit is changed whenever a binary value is incremented or decremented.

To construct an *n*-bit Gray code, we first would need to generate a list of *(n-1)* bit Gray codes. We then create a new list of twice the length which consists of the original Gray Code list followed by the original list reversed. To generate the *n*-bit code, we prefix elements in the original list with zeros and the reversed list with ones.

For example, the 1-bit Gray code is merely the list {0, 1}, corresponding to {0, 1} in normal binary. The 2-bit Gray code can be generated by taking {0, 1}, reversing the list to be {1, 0}, concatenating the original with the reversed {0, 1, 1, 0} and prefixing the original with zeros and the new list with ones {00, 01, 11, 10}. This corresponds to the sequence {00, 01, 10, 11} in binary.

Filename: adv51.{java, c, cpp, cc, py}

Input: The first line contains the integer *n*. The second line will contain *n* bits (`0` or `1`) in an *n*-bit Gray code to be translated to normal binary.

Output: The output should contain the *n*-bit (`0` or `1`) binary representation of the input.

Assumptions: $1 \le n \le 20$
All input is valid.

Sample Input #1:
```
3
100
```

Sample Output #1:
```
111
```

Sample Input #2:
```
6
010101
```

Sample Output #2:
```
011001
```

Advanced 5.2 **Flasks in flux** (page 1 of 1)

Overview: Given flasks of certain sizes, determine if you can measure a certain volume in one of the flasks.

Description: Oh no! Ariadne has taken most of Yusuf's flask set from him (something about building a dream model with glass, she says). Now Yusuf only has 2 flasks to work with, of volume *a* and *b*. Fortunately, Ariadne hasn't made off with any of Yusuf's chemical supply, so he still has an unlimited supply of his top secret dream-inducing chemical mix.

Yusuf would like to measure a certain volume of liquid, *t*, into one of the flasks, using the two flasks that he has left. During each step, he may perform any of these actions: fill up any flask to the brim with liquid, pour out all the liquid in a flask, or pour liquid from one flask to another until one of the flasks is either completely empty or completely full. You cannot pour liquid into a full flask.

For instance, given two flasks of volume 5 and 3, Yusuf can measure out 4 units of liquid by the following method.

- Fill the volume 5 flask with liquid. (5, 0)
- Pour liquid from the volume 5 flask into the volume 3 flask. (2, 3)
- Empty the volume 3 flask. (2, 0)
- Pour liquid from the volume 5 flask into the volume 3 flask. (0, 2)
- Fill the volume 5 flask. (5, 2)
- Pour liquid from the volume 5 flask into the volume 3 flask. (4, 3)

Given the desired volume and the volumes of the flasks Yusuf has left, help him determine if it is possible to obtain the desired volume in one of the flasks.

Filename: adv52.{java, c, cpp, cc, py}

Input: The input will consist of three lines with integers *t*, *a* and *b*, with one integer on each line. *t* represents the desired volume, while *a* and *b* represent the volumes of the first and second flasks, respectively.

Output: The output should consist of a single line containing `YES` if it is possible to obtain the desired volume in one of the flasks, and `NO` otherwise.

Assumptions: $1 \le a, b \le 100$
$0 \le t \le \max(a, b)$
All input is valid.

Sample Input #1:
```
4
5
3
```

Sample Output #1: `YES`

Sample Input #2:
```
5
6
4
```

Sample Output #2: `NO`

Advanced 5.3 **Roundabout passage** (page 1 of 1)

Overview: Find the shortest path in a toroidal maze.

Description: Before letting Ariadne join his team as an architect, Cobb tests her by challenging her to construct a maze that he cannot solve in one minute. Ariadne cleverly designs a maze filled with twisty passages and dead ends and gives it to Cobb. Cobb is confused. To make things worse, normal rules of physics do not apply and the maze bends space. Upon reaching one side of the maze, Cobb also discovers that the building wraps around to the opposite side! Can you help him find his way out?

The maze will be represented by a rectangular grid of cells, where each cell can either be the starting position, the exit, an empty space, or an impassable wall. Each step, you can move a single space horizontally or vertically into an adjacent empty space. The edges of the maze wrap around to the opposite edges. In other words, if you are on a empty space on an edge, and the corresponding space on the opposite edge is empty, you may move to that space. Write a program to determine the minimum number of steps you need to reach the exit.

Filename: adv53.{java, c, cpp, cc, py}

Input: The first line will consist of two integers, *r* and *c*, where *r* represents the number of rows and *c* represents the number of columns in the maze.
The following *r* lines will each contain *c* characters, each representing a single cell of the maze.

- `.` - (period) An empty space
- `*` - (asterisk) Impassable wall
- `S` - (uppercase “s”) Your starting position
- `E` - (uppercase “e”) The exit

Output: The output will consist of a single integer, representing the length of the shortest path to the exit.

Assumptions: $3 \le r, c \le 50$
The maze has at least one path from the starting position to the exit.
All input is valid.

Sample Input #1:
```
3 5
*****
.S*E.
*****
```

Sample Output #1:
```
3
```

Sample Input #2:
```
7 7
*****.*
*S....*
****.**
*....**
***.*E*
*.....*
*****.*
```

Sample Output #2:
```
8
```

Advanced 5.4 **Mastermind** (page 1 of 2)

Overview: Play a game of Mastermind against the machine interactively.

Description: Before anyone can successfully enter another person's subconscious mind, tough mental training must be undertaken. Ariadne, new to the Inception team, is undergoing the training, supervised by Cobb. Part of the training involves repeatedly playing the game of Mastermind one thousand times in her dream world. However, Ariadne is fed up with Mastermind by the 10th game, so she enlists you to construct a computer program to solve the next 990 Mastermind games for her.

You will play the game of Mastermind against Cobb. Cobb will select four pegs, each of which will be one of 6 possible colors (note that Cobb may select multiple pegs of the same color). You must guess the correct color and arrangement of those pegs he selected with at most twelve guesses. This is an interactive problem, where your program will iteratively make guesses, and our program (playing the role of Cobb, the code maker) will respond to each guess.

The possible colors in the game are Red, Green, Blue, Cyan, Magenta, Yellow and are represented in game as R, G, B, C, M, Y respectively. After you make each guess, the judge will inform you of *p,* the number of pegs that are both the correct color and in the correct position, and *c*, the number of pegs that of the correct color but in the wrong position.

Note that if a peg in the guess is already matched to a peg in the correct solution, those pegs cannot be matched again in counting *c*. For example, if the correct solution is RRGG and the guess is RGRR, the first position is the only one correct, so *p* is 1, and *c* is 2 because G is correct but in the wrong position, and one of the Rs is correct but in the wrong position. Note that the third R in the guess does not count towards *c*, because there are only two Rs in the correct solution. The second G in the correct solution also doesn't count towards *c*, because there is only one G in the guess. In other words, *p*+*c* is the maximum number of positions in which the correct solution and some permutation of the guess match. See the sample games for more examples.

File name: adv54.{java, c, cpp, cc, py}

Input/Output: This is an interactive problem. This means that your program will receive input from the grading environment based on the output your program produces. All input and output will be done through the console.

Output: Rules of interaction:

1. Your program should output a 4-character string consisting of only the characters (`R`, `G`, `B`, `C`, `M`, `Y`).
2. You MUST output a new line character and flush the output stream after each output! (See the sample contest problem)

Advanced 5.4 **Mastermind** (page 2 of 2)

3. Each query will result in two integers, *p* and *c*, with *p* representing the number of correct positions in the most recent guess, and *c* representing the number of other pegs that are the correct color but are not in the correct position.
4. Your program must terminate upon receiving a response of `4 0`, which represents a correct solution.
5. You must solve the game in at most 12 guesses and you must use at most 1 second of CPU time to solve each game. Any violation of this will result in your program being judged incorrect.

Assumptions: $0 \le p, c$
$p+c \le 4$

Sample Game #1:

```
[contestant] YYCC
[judge]      0 1
[contestant] MRRY
[judge]      1 1
[contestant] GMGY
[judge]      1 1
[contestant] RRMY
[judge]      0 2
[contestant] CRGM
[judge]      1 3
[contestant] GCRM
[judge]      4 0
```

Sample Game #2:

```
[contestant] RRRR
[judge]      2 0
[contestant] RYRY
[judge]      1 1
[contestant] GGYB
[judge]      1 1
[contestant] GGGB
[judge]      2 0
[contestant] RRGB
[judge]      4 0
```

Advanced 9.1 **Dreams in a tile** (page 1 of 1)

Overview: Determine whether it is possible to divide a domino board into distinct dominoes.

Description: You want to join in on an inception mission, but before you can, Dom says you must have your own totem. Because you like dominoes, you want to make yours a specially designed domino board populated with numbers. Because 2 is the best number ever, you decide to make it a 2 x *n* board. If it is possible to select 2 x 1 or 1 x 2 pairs such that no pairs are "repeated" -- that is, such that no pairs consist of the same integers in either order (so [3,4] is the same as [4,3]) -- then you are in a dream, but if it is not possible, then you know you are in reality.

For example, on the following board, it is possible to divide the board into dominoes [4,2], [3,1] and [1,2]:

4	2	1
3	1	2

Given an 2 x *n* array of integers representing your totem, output `DREAM` if you it is possible to divide the board into pairs such that no pairs are repeated or `REALITY` if it is impossible.

Filename: adv91.{java, c, cpp, cc, py}

Input: The first line contains one integer *n*, representing the length of the 2 x *n* domino board. The next line contains *n* integers, separated by spaces, representing the values in the top row of the 2 x *n* board. The next line also contains *n* integers, separated by spaces, representing the values of the bottom row of the 2 x *n* board.

Output: The output should be either `DREAM` or `REALITY`.

Assumptions: $1 \le n \le 25$
Each integer in the board is between 0 and 9, inclusive.
All input is valid.

Sample Input #1:
```
4
4245
1362
```

Sample Output #1:
```
DREAM
```

Sample Input #2:
```
4
4241
1824
```

Sample Output #2:
```
REALITY
```

Advanced 9.2 **Airborne escape** (page 1 of 2)

Overview: Compute the weight of a maximum spanning tree of a graph.

Description: Oh no! Cobb has many enemies that are after him, and he must hide from them as quickly as possible! Saito has graciously provided him with a private jet to fly from city to city, but alas, funds are limited so in the graph that will include cities that Cobb will fly to (nodes) and flight paths from city to city (edges), Saito can afford no cycles.

The graph that denotes Cobb's possible flight plans is a spanning tree. Cobb is safest when in the air, so he would like to find the spanning tree across different cities such that the sum of the flying times for all possible flight connections he can make is the greatest.

A spanning tree is a collection of edges such that all the nodes are connected and there are no cycles (that is, there is a unique path between each pair of nodes).

A maximum spanning tree is a spanning tree such that its total weight (the sum of the weights of the edges) is greater than the weights of all other possible spanning trees.

Saito has hired you to help Cobb determine the maximum spanning tree for the given set of cities. Can you help him? Cobb's mission, after all, depends on you. No pressure.

Filename: adv92.{java, c, cpp, cc, py}

Input: The first line will contain two numbers (in this order): the number of nodes n in the graph and the number of edges m in the graph.

The next m lines contain 3 integers separated by spaces, representing the m edges. The first two integers on each line are between 0 and n-1 inclusive and represent the two nodes that are the endpoints of the edge. The third integer represents the flying time for that edge.

Output: The output should be the weight of the maximum spanning tree of the graph.

Assumptions: $0 \le n \le 500$
$0 \le m \le 1000$
The given graph will be connected.
All weights (flying times) will be at most 15.
All input is valid.
There are no duplicated edges.

Advanced 9.2 **Airborne escape**

Sample Input #1:

```
4 5
0 1 1
2 3 2
3 0 13
2 1 6
0 2 8
```

Sample Output #1:

```
27
```

Sample Input #2:

```
4 6
0 1 9
2 0 6
3 1 11
3 2 12
2 1 8
0 3 7
```

Sample Output #2:

```
32
```

Advanced 9.3 Informed extraction

Overview: Given a message encoded as an (8,4)-Hamming code (possibly corrupted), determine the original message.

Description: Cobb and his team of Extractors are trying to extract information from Saito. Due to the nature of dreams, the information they extract from Saito is unreliable, with some bits of information corrupted. Fortunately, the team discovers that the information being extracted is encoded in a (8,4)-Hamming code format, and the team can extract the information even though some parts of the message are corrupted. Your job is to assist the team in writing a converter to decode the potentially corrupted data and output its pristine, correct form, if possible.

Occasionally in hardware, bits are corrupted in transmission. Error correction codes seek to correct or at least detect any corrupted bits. The (8,4)-Hamming code is one such code that encodes 4 data bits for every 8 bits transmitted; the other 4 bits are parity bits that are used to verify the integrity of the message. The (8,4) code is a SECDED (single error correcting, double error detecting) code.

The following is the layout of bits in the (8,4)-Hamming code:

Bit position	0	1	2	3	4	5	6	7
Type	Parity	Parity	Parity	Data	Parity	Data	Data	Data
Coverage parity bit 0	X	X	X	X	X	X	X	X
Coverage parity bit 1		X		X		X		X
Coverage parity bit 2			X	X			X	X
Coverage parity bit 4					X	X	X	X

The bits of the original message should be written into the data bits of the encoded message sequentially. That is, the first bit of data is written at position 3 in the encoded message, the second at position 5, and so on.

Each parity bit covers certain bits of the data stream, including itself. As indicated in the table above, parity bit 0 covers all the bits, parity bit 1 covers the odd-numbered bits, and so on. The value of a parity bit should be set so that the parity of all the covered bits is 0. In other words, there should be an even number of 1 bits in the covered bit positions. So, the value of each parity bit should be 1 if the sum of the other covered bits is odd and 0 if the sum is even. Note that the parity bit at position 0 is

Advanced 9.3 Informed extraction

calculated after all of the other parity bits have been calculated; all of the other parity bits are only dependent on data bits.

For example, the data string 1011 would be encoded as 00110011, where the parity bits have been underlined. Single bit corruptions can be detected and corrected by observing which parity bits are inconsistent. There is always a unique way to correct a single bit corruption in a message.

If no single correction would yield a valid (8,4)-Hamming code, then at least two bits have been corrupted. Any message with exactly two corrupted bits can be detected in this way. However, double errors can only be detected, not corrected.

Given a Hamming-encoded message, which may have up to two corrupted bits, determine the original message or detect a double corruption.

File name: adv93.{java, c, cpp, cc, py}

Input: The input consist of 8 bits (`0` or `1`) encoded in an (8,4)-Hamming code.

Output: The output should contain the 4 data bits, corrected of any single corruptions, or "`DOUBLE CORRUPTION`" if a double corruption is detected.

The output is to be formatted exactly like the sample output given below.

Assumptions: The message will have at most two corrupted bits.
All input is valid.

Sample Input #1:
```
10011110
```

Sample Output #1:
```
1110
```

Sample Input #2:
```
11111010
```

Sample Output #2:
```
DOUBLE CORRUPTION
```

Advanced 9.4 **Tri-to find the-angle** (page 1 of 1)

Overview: Given a set of points in the plane, find the triangle with the largest minimum angle.

Description: As a last test before Cobb will let you permanently join his inception team, he gives you a puzzle. He promises you a spot in his next mission if you can solve it in the time allotted. Cobb gives you *n* points in the plane and challenges you to find the triangle with vertices among those points that satisfies the following conditions:

1. None of the other points is in the interior of the triangle.
2. Out of all triangles that satisfy condition 1, the smallest angle of this triangle is maximized.

Can you solve this last problem to be a part of Cobb's dream team?

File name: adv94.{java, c, cpp, cc, py}

Input: The first line of the input will consist of a single integer *n*, representing the number of blocks. The next *n* lines will each consist of two integers, *x* and *y*, separated by spaces, representing the location of the point.

Output: The output should consist of three integers, each separated by spaces, representing the 1-indexed indices, in ascending order, of the three points that form the desired triangle.

The output is to be formatted exactly like the sample output given below.

Assumptions: $3 \le n \le 50$
$-1000 \le x, y \le 1000$ for all pairs x, y
No three points are collinear.
There will be a unique triangle that satisfies the conditions.

Sample Input #1:
```
3
528 491
-528 -491
491 528
```

Sample Output #1:
```
1 2 3
```

Sample Input #2:
```
5
174 739
304 -4
-667 -672
-175 812
-498 -566
```

Sample Output #2:
```
2 4 5
```