CSC 495 – Problem Set 11 – Due Tuesday, April 10

Problem 11.R1: Speedy Random

Required Problem
Points: 50 points
40 Extra Ninja Points distributed based on the speed of the solutions

Background

The last problem in Column 13 of *Programming Pearls* is interesting, with a lot of room for creativity. Simply stated: Write the fastest program you can to generate a sorted array of uniformly distributed random integers without duplicates.

The Problem to Solve

Given integers k and n, generate a random set of k integers drawn uniformly from the range $0, \ldots, n-1$ (inclusive). You are guaranteed that $k \leq 25,000,000$ and $n \leq 10^{14}$.

Hints and Techniques

Note that it's not really possible to test correctness in an automated way. The server will check to see that the correct number of values are output, that the list is sorted, and that there are no duplicates. However, it has no way of testing that the numbers generated are uniformly distributed. This will be checked when I grade the programs and look at your source code — it's very possible that you could submit something for which the output looks good, but a mistake in your code makes the output non-uniform. In that case, it will be graded as incorrect, and not be eligible for the speed tests, so be careful! If you have a function that returns a randomly distributed, positive 63-bit integer (call this integer `r`) you may assume that `r%n` is uniformly distributed between 0 and $n-1$ (inclusive) — note that this isn't exactly correct, but for the range of values given for n it is close enough for this problem.

Speed Test

I'll allocate ninja points for the speed test a little differently than I have in the past. I will time all correct submissions on two large inputs with different properties (the relation between k and n). Then I will take the reciprocal of each time so that the fastest program has the highest "score." Finally, for each test input I will distribute 20 points proportionate to these scores. If all running times are similar, then these points will be roughly evenly split. If one program is much faster than the others, then it will get the lion's share of the points.

Input and Output

The input will consist of a single line containing integers k and n (in that order), and you are to output the sorted list with one integer per line.

Sample Input

```
4 10
```

Sample Output

```
0
5
6
8
```

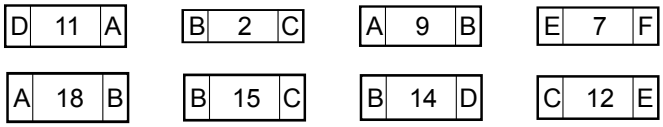
Problem 11.R2: Carry a Big Stick

Required Problem
Points: 50 points

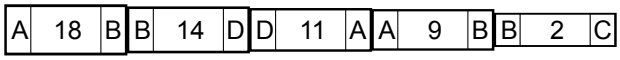
Background

Consider constructing a long pole from pieces that have the same length but different weights. The pieces have fittings (for connecting with adjacent pieces) on the ends, and can be put together if the fittings are compatible. You would like to put as many of these together as possible, making the longest possible pole under the following conditions: to attach two pieces together, the end where they attach must have compatible fittings, and the weights of the pieces must be monotonically decreasing from beginning to end. To model this, we label different fitting types with letters A–Z, so, for example, a piece might have a 'C' fitting on the front end and an 'A' fitting on the back end. Pieces can only be oriented one way (you can't flip them around), and are manufactured so that an 'A' fitting on the back end of one piece always fits an 'A' fitting on the front end of another piece — mis-matched fittings are never compatible.

For example, consider the following pieces, where fitting types are shown on the ends and the weight of the piece is given in the middle (the height of the piece in the drawing also reflects the weight):



Taking these pieces, it is possible to make a pole 5 segments long, satisfying the requirements, as follows:



This is the longest possible such sequence (there are no 6-segment poles that can be constructed).

The Problem to Solve

You are to write a program that reads in descriptions of segments (weight and end fittings) and compute the longest possible pole that can be constructed satisfying the rules described above (decreasing weights and compatible end fittings). All weights will be at most 10,000, and there will be at most 10,000 pieces. To make the problem easier, you are guaranteed that all weights will be distinct. Your program should be able to solve any such problem in under 15 seconds.

Hints and Techniques

Remember how helpful it was in Problem Set 5 to pre-sort the input before processing?

Input and Output

The input begins with a line containing an integer n, the number of segments that we can choose from. This is followed by n lines that each describe a segment, with the weight first (an integer) and the two fitting types following that. Output should be the segments (described in the same way) that make up an optimal solution, in order. If there are multiple segments with the same maximum length, any solution of that length will suffice. The sample input below reflects the problem described in the "Background" section.

Sample Input

```
8
11 D A
18 A B
2 B C
15 B C
9 A B
14 B D
7 E F
12 C E
```

Sample Output

```
18 A B
14 B D
11 D A
9 A B
2 B C
```

Problem 11.C1: Permutation Detection

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points
20 Extra Ninja Points distributed based on the speed of the solutions (based on a single test, with points distributed as explained in Problem 11.R1)

Background

A "permutation matrix" is an $n \times n$ matrix in which each entry is a 0 or 1, and each row and each column contains exactly one entry with a 1 (to see why this terminology makes sense consider what happens if you multiply this matrix times a vector). Of the two matrices below, the left one is a permutation matrix, and the right one is not.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Your goal is to write a program that reads coordinates of all the 1 values in an $n \times n$ matrix, and then determine whether it is a permutation matrix.

The Problem to Solve

You are to read coordinates of the 1's in a $n \times n$ (all other entries are zero), and output "Yes" or "No" depending on whether the matrix is a permutation matrix. Coordinates are zero-based (so are $0, \ldots, n-1$), and n can be as large as 1,000,000.

Hints and Techniques

In case you're thinking about actually building the matrix, figure out how much memory a $1,000,000 \times 1,000,000$ matrix would require. Given that that is impossible, the solution is pretty obvious, given the theme of Column 13 in *Programming Pearls*.

Input and Output

The input will consist of a line containing an integer n, giving the size of the matrix. Following this is a list of lines containing valid coordinates in this matrix (so each one is in the range $0, \ldots, n-1$). You should read until the end of file — there are an unknown number of coordinates given in the file. The output of your program is a single line, containing "Yes" or "No" indicating whether or not the matrix is a permutation matrix. The two samples below correspond to the two matrices given in the "Background" section.

Sample Input 1

```
4
2 0
0 1
1 3
3 2
```

Sample Output 1

```
Yes
```

Sample Input 2

```
4
1 3
3 1
2 0
0 1
```

Sample Output 2

```
No
```

Problem 11.C2: Balance Out The Tubes

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points

Background

Imagine that you have a large number of cylindrical objects that need to be packed into tubes and shipped, such as core samples of different depths which are individually wrapped up. These samples are sequenced and need to be unpacked in the same order that they were packed, so you can't rearrange the samples. Our goal is to pack the objects into the tubes as evenly as possible.

As an example, consider samples of sizes 7, 2, 4, 7, and 1, in that order, packed into tubes of length 10. The greedy strategy is to pack as many as you can into the first tube and ship it, and repeat until finished. However, this leads to a more uneven distribution that we'd like, as shown below:

Tube Number	Greedy Strategy	Alternate Strategy
Tube 1	7,2 (1 empty)	7 (3 empty)
Tube 2	4 (6 empty)	2,4 (4 empty)
Tube 3	7,1 (2 empty)	7,1 (2 empty)

Notice how the second strategy evens out the empty space better than the greedy strategy. How do we quantify "more evenly spread out?" Here's a reasonable technique: for any assignment of samples to tubes, square the amount of empty space in each tube and add all of those up. This gives us a "penalty" score, where more uneven distributions have higher penalties. In the example above, the greedy strategy has penalty of $1^2 + 6^2 + 2^2 = 41$, whereas the second strategy has penalty $3^2 + 4^2 + 2^2 = 29$. To see why this is a sensible measure, consider what would happen if we could cut samples up and put fractional samples in each tube: this penalty is minimized when there is exactly the same amount empty space in each tube (that's a basic Calculus problem — can you prove it?). So your problem is this: how can you assign samples to the mailing tubes in order to minimize the penalty?

The Problem to Solve

You are given an integer tube capacity t and the sizes of n samples that need to be shipped. You are to calculate the shipping strategy that minimizes the penalty, as defined above. You are guaranteed that $t \leq 200$ and $n \leq 10,000$, and that every sample size is an integer value $\leq t$. Your program should be able to solve any such problem in under 10 seconds.

Input and Output

The input consists of integers t and n on the first line, where t is the capacity of the shipping tube and n is the number of samples. This is followed by n lines each containing an integer capacity. Your program should output, on the first line, the penalty of the optimal solution. Following this are lines that correspond to shipping tubes, with the sample sizes listed (in order) separated by spaces. The first sample below corresponds to the problem described in the "Background" section.

Sample Input 1

```
10 5
7
2
4
7
1
```

Sample Output 1

```
29
7
2 4
7 1
```

Sample Input 2

```
20 6
10
10
9
11
1
20
```

Sample Output 2

```
165
10
10 9
11 1
20
```