TJ IOI 2017
Programming Round Solutions

Thomas Jefferson High School for Science and Technology

Saturday, May 13, 2017

Contents

Preface

These are the solutions to the TJ IOI 2017 contest. Each solution outlines the thought process used to solve the problem, but does not include the solution code. Solution code in Java, Python, and C++ can be found in our Github repository at `https://github.com/tjsct/tjioi-2017`.

A Larry's Race

TJ IOI Inc. has chosen Larry as their corporate representative at the local track and field competition! However, the competition has a very peculiar set of rules: if Larry would like to advertise TJ IOI Inc., he must compete in the race! To get Larry in shape, Devon has built a robot to chase Larry, traveling 100 meters in T seconds ($1 \leq T \leq 100,000$).

There are N inputs to this problem ($1 \leq N \leq 100,000$). Each consists of a distance A_i that Larry runs, where A_i is a multiple of 100 ($100 \leq A_i \leq 100,000$), and the time B_i it took for him to run that distance ($1 \leq B_i \leq 100,000$), determine whether Larry could outrun Devon's robot.

Note: if Devon's robot catches Larry exactly at the finish line, Larry did not outrun it.

SHORT NAME: `race`

INPUT FORMAT:
The first line consists of two integers, N and T. The next N lines each contain an integer A_i representing a distance in meters (A_i is a multiple of 100), and a time B_i representing the time it took Larry to run that distance in seconds.

OUTPUT FORMAT:
For each input, if Larry outran Devon's robot, output "SPEEDRACER" (without quotes). Otherwise, output "POTATO" (without quotes).

SAMPLE INPUT:

```
3 22
1600 352
800 150
3200 840
```

SAMPLE OUTPUT:

```
POTATO
SPEEDRACER
POTATO
```

SOLUTION:

In this problem, we know that in order for the robot to outrun Larry, the robot's average speed must be faster than Larry's average speed. Speed is given by

$$S = \frac{D}{T}$$

where S is the speed, D is the distance travelled, and T is the time taken to travel the distance. We are provided with the distance and time for both the robot as well as each of Larry's attempts, so we can calculate the speed for the robot and compare it to the speed for each of Larry's attempts.

B Lunchbox Hunt

Devon has prepared an extravagant lunch to celebrate the one year anniversary of TJ IOI Inc.! However, to make things more interesting, he has hidden the lunchbox containing his lunch somewhere inside the huge, single-level parking garage.

Alex has decided to go on a treasure hunt to find Devon's hidden lunchbox. Fortunately for Alex, Devon has left behind a set of instructions specifying the location of his lunchbox. The instructions consist of a starting location (x_0, y_0) and N $(1 \leq N \leq 1,000,000)$ queries. Each query consists of a direction specified by the characters 'N', 'S', 'E', and 'W', and a non-negative distance. Help Alex find the coordinates of the location of Devon's lunchbox.

Note: Alex's position (x, y) at any time is guaranteed to remain within $-1,000,000,000 \leq x, y \leq 1,000,000,000$.

SHORT NAME: `lunchbox`

INPUT FORMAT:
The first line will contain three integers N, x_0, and y_0. The following N lines will describe a query consisting of a character ('N', 'S', 'E', 'W') and a non-negative integer distance.

Note: North corresponds to up, south corresponds to down, east corresponds to right, and west corresponds to left.

OUTPUT FORMAT:
The output should consist of two integers separated by a space. The first integer is the final x coordinate and the second integer is the final y coordinate.

SAMPLE INPUT:

```
4 6 -2
N 3
S 5
W 2
W 1
```

SAMPLE OUTPUT:

```
3 -4
```

SOLUTION:

In this question, we are given a series of instructions, as well as a starting location, and must determine the ending location. Following each instruction takes $O(n)$ time, and because the number of queries is less than $1,000,000$, we can simply follow each instruction and trace out Alex's path, in order to determine the location of the lunchbox. This can be accomplished with a variable storing Alex's location, and a loop: at every iteration of the loop, we adjust the location of Alex depending on the instruction, and repeat through all the instructions.

C Singing Low

After the work day is over at TJ IOI Inc., many of the employees attend office karaoke! One of these employees is Devon, who wants to see how low he can sing. Devon begins at note N ($1 \leq N \leq 1,000,000$), and would like to sing note 0. In one step, Devon may sing between 1 and K notes lower than his current note. (For example, if K is 3 and he is on note 5, he can sing either 3, 2, or 1 notes lower than his current note, taking him to notes 2, 3, or 4, respectively.)

However, Devon's note cannot decrease by any given amount more than once. (For example, if he went from note 5 to note 3 in the previous example by going down 2 notes, he would not be able to go to 1 as this be another decrease by 2 notes.) Please help Devon calculate the smallest value of K that will allow him to get to note 0.

SHORT NAME: `singing`

INPUT FORMAT:
The first line will contain the integer N, the note that Devon begins on.

OUTPUT FORMAT:
The output should consist of one integer, K, the lowest value which will allow Devon to reach note 0.

SAMPLE INPUT:

```
8
```

SAMPLE OUTPUT:

```
4
```

SOLUTION:

The key observation for this problem is to realize that for any given K, the maximum number of notes that Devon can go down is fixed. In other words, the order in which Devon's notes go down does not matter. For a given K, we can go down 1, 2, 3, etc. notes, all the way until K notes. The total is given by $1+2+3+\cdots+(K-1)+K$. Therefore, if N is less than this sum, then we are guaranteed[1] that Devon can reach note zero. Otherwise, Devon cannot reach note zero. Therefore, we can try increasingly large values of K until this sum exceeds N. However, we cannot afford to recalculate the sum every time, as that would require $O(K^2)$ time, as for each value of K, we must loop over K elements.

We can use the formula

$$\sum_{k=1}^{M} k^2 = \frac{M(M+1)}{2}$$

to reduce this operation to $O(K)$ time, as we can determine the sum of the first K integers in constant time, which is fast enough to solve all test cases. This formula can be proven using induction, or by pairing the terms from the "front" and the "back".

[1]The proof of this is slightly more involved, but the idea is as follows: Suppose that N is less than K, but not more than K less. Then, use all note differences aside from $(N-K)$. If N is more than K, then omit note difference K, and repeat the process on the first $(K-1)$ notes.

D Pencils

TJ IOI Inc. has reached a net worth of a million dollars! Kevin, the CEO, is very happy of this achievement, and asks Alex to write a report on the financial standing of the company, first thing tomorrow. However, Kevin decides that instead of typing the report, he will make Alex hand write it, in order to build character.

That night, Alex has gathered N ($1 \leq N \leq 100,000$) pencils in front of him. However, his pencil bag only has room for K pencils ($1 \leq K \leq N$). Each of his pencils has an integer length in the range of 1 to 1,000, inclusive. Because Kevin expects an extravagant and exhaustively detailed report, Alex will have to write a lot if he wants to please Kevin, so Alex wants to choose the longest K pencils to place into his pencil bag. Please help Alex determine the sum of the lengths of those pencils.

SHORT NAME: `pencils`

INPUT FORMAT:
The first line of input contains two integers N and K. The next N lines describe the length of the pencils.

OUTPUT FORMAT:
Output a single integer, the sum of the K longest pencils.

SAMPLE INPUT:

```
7 3
1
4
5
3
8
14
2
```

SAMPLE OUTPUT:

```
27
```

SOLUTION:

Whenever we are tasked with finding the N greatest values of a set, we are immediately motivated to use a sort. This is because a sort will arrange a data set "in order[2]", which is very useful for our purposes. This is because if the pencils are sorted in order of height, we can simply sum the first N pencil lengths (or last N, depending on which way you sort the elements).

One may use any of the standard library sorts (`sorted` in Python, `Arrays.sort()` in Java, and `sort` in C++) to accomplish this task in $O(N \log N)$ time, fast enough to solve all cases. Alternatively, one may implement their own $O(N \log N)$ sort, though this is not the intended solution.

[2]The word "order" is loosely defined, as we can define what the order is. For example, consider a group of people. We could define the order as height from greatest to least, or age from least to greatest. Regardless, there must be some metric by which we sort.

E Puck Puck Moose

TJ IOI Inc. is having its annual corporate retreat to build rapport and develop synergy amongst its employees. Larry invites Alex and some other employees to form a circle of N people ($1 \leq N \leq 10$) in order to play Alex's favorite game, Puck Puck Moose. Some of the people in the circle do not get along, however, and Alex and Larry must accommodate their friends. Given the pairs of people who do not want to sit next to each other, determine the number of possible ways that the N people can sit down in the circle.

Note: A circle is rotationally symmetric, so any configuration that can be rotated into another is considered the same configuration.

SHORT NAME: `puck`

INPUT FORMAT:
The first line of input contains two integers, N and K ($1 \leq K \leq \binom{N}{2}$), where N is the number of people in the circle, numbered from 0 to $N-1$, and K is the number of pairs to follow. The next K lines consist of two integers, denoting the indices of the two people that do not want to sit next to each other.

Note: the order of the two indices does not matter. For example if 0 and 1 are a disallowed pair, 0 may not be neither to the left nor the right of 1.

OUTPUT FORMAT:
Output a single integer, the number of ways that the people can sit down in a circle, such that no pair that does not want to sit next to each other is together. If no configurations are possible, print 0.

SAMPLE INPUT:

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

SAMPLE OUTPUT:

```
2
```

SOLUTION:

Because the bounds for this problem are low enough, we can actually generate all possible configurations and check if they work! This is because the number of ways that we can arrange a set of N people in a circle is given by $(N-1)!$[3], and the bound on N is sufficiently low enough to run in time. Therefore, we can use a brute force solution.

We can make a further optimization by building each configuration recursively. At each step, we add another person to the configuration. However, if at any point, adding a specific person creates a conflict, we can cut off that branch of the recursion search tree immediately, saving a lot of time.

[3]This is due to the fact that there are $N!$ ways to arrange N people in a line, but the circular permutation allows for rotations, which removes a degree of freedom. You can think of this as fixing one person, and arranging the rest in a line after him/her.

F Candy Fest

TJ IOI Inc. is hosting its annual Candy Fest! Larry, who is organizing the event, has found a store that offers a massive sale on candy. Unfortunately, after purchasing a large amount of N different kinds of candy ($1 \leq N \leq 100,000$), he realized that the receipt of length M ($1 \leq M \leq 1,000$) has no spaces in it! Frustrated with this receipt format, Larry instead ate all of the candy himself! Help Larry figure out how much sugar Larry will consume after he eats all of the candy.

Note: It is guaranteed that no candy name is a prefix of another candy name and that a solution exists.

SHORT NAME: `candy`

INPUT FORMAT:
The first line of input contains the integer N, the number of different kinds of candy. The next N lines contain the name of the candy, a string of length L ($1 \leq L \leq 100$) given in all capital letters, followed by the amount of sugar in that candy a_i ($1 \leq a_i \leq 100,000$), separated by a space. The final line will contain the receipt, a string of capital letters.

OUTPUT FORMAT:
Output a single integer, the total amount of sugar that Larry will consume.

SAMPLE INPUT:

```
4
KITKAT 20
TWIX 28
REESES 8
SNICKERS 9
TWIXTWIXKITKATREESESTWIXSNICKERSTWIX
```

SAMPLE OUTPUT:

```
149
```

SOLUTION:

We would like to figure out how the N strings make up the receipt (with length M). It is guaranteed that there is only one way to do this, since no string is a prefix of another.

To do this, we first store each of the strings in a hash map (a dictionary in Python, a `HashMap` in Java, or an `unordered_map` in C++); the keys and values of the entries are the candy name and the amount of sugar, respectively.

Next, we define a variable that stores the total sugar in the candy, T, and set this initially to zero. Now we iterate through the receipt with an index i initially set to zero. While i is less than M, we have a nested loop which iterates a variable j from i to M. When the substring of the receipt from i to j is found in the hash map, this means that we have found the next candy name that that composes the receipt. We can add its sugar amount to the T, add j to i, and break from the inner loop. This algorithm would eventually go through each candy in the receipt.

We can bound the runtime my establishing the worst-cases for each of the loops. The outer loop could at most iterate up to M; the inner loop could at most iterate up to L, the maximum length of any given candy name. Thus, the runtime is $O(ML)$.

G Larryopoly

Tired of digging out change at the TJ IOI Inc. vending machines, Larry, being rich, decides to invent his own currency to be used by all TJ IOI employees. He creates N ($1 \leq N \leq 100$) different types of bills, each with a unique dollar value d_i ($1 \leq d_i \leq 1,000$).

Although many employees are initially skeptical, Niki decides to use this currency. Niki wants to feel rich, so he wants to maximize the number of bills he can hold. However, Niki refuses to take more than one bill of each kind. Niki will ask M times ($1 \leq M \leq 100,000$) if he can hold X amount of value in Larry's currency ($1 \leq X \leq 1,000,000$) and if so, how many bills that will take.

SHORT NAME: `larryopoly`

INPUT FORMAT:
The first line of input contains two integers N and M, where N is the number of bills and M is the number of queries. The next N lines each contain one integer, denoting the value of that bill, followed by M lines, each consisting of a query in the form of a value X.

OUTPUT FORMAT:
Output M lines, where each line contains the maximum number of bills that could make the value in the corresponding query, or -1 if this is impossible.

SAMPLE INPUT:

```
3 3
3
6
5
6
11
4
```

SAMPLE OUTPUT:

```
1
2
-1
```

SOLUTION:

Initially, this problem appears to resemble the knapsack problem, which is discussed in depth in the Study Guide. However, there is an additional restraint: Niki will take at most one of each type of bill. Thus, we take a similar approach as the knapsack problem, but which a slightly different recurrence relation.

Let $dp[i]$ be the maximum number of bills that could form i dollars and $A[j]$ be the dollar value of the jth dollar. Then we have

$$dp[i + A[j]] = \max(dp[i] + 1, dp[i + A[j]])$$

for all i and j. In order to ensure that each dollar is used only once, we must loop i backgrounds, however.

Once we construct the dp array, we can answer each of the queries in constant time. The final time complexity of this solution is $O(ND + M)$, where D is the maximum dollar amount (1000).

Note that the maximum size of each of Niki's queries is $1,000,000$, so if we initialize the dynamic programming array with size $100,000$ (the maximum amount that Niki could possibly hold), we have to check for whether Niki's query lies within the bounds of our dynamic programming array. Failing to do so would result in an `ArrayIndexOutOfBoundsException` in Java, or undefined behavior in C++.

H Cookie Baking

Devon, a dedicated member of the culinary staff at TJ IOI Inc., has made cookies for Kevin's birthday! However, smelling the aroma of chocolate chips coming from the kitchen, Alex devises a plot to bring cookies to him and his fellow employees.

Devon has N large piles of cookies ($1 \leq N \leq 100,000$) on the kitchen counter, where the i^{th} pile ($1 \leq i \leq N$) contains a_i cookies ($1 \leq a_i \leq 1,000,000$). Alex, on the other hand, wants to steal his cookies, but he brings along a different number of employees each time.

When he steals cookies, he wants to make sure he is able to split the cookies evenly among him and his $X-1$ employees (X people total). Alex always chooses X to be prime, because he likes prime numbers. Since the kitchen door is located next to cookie pile 1, Alex wants to take cookies from the first possible pile (i.e. the minimum value of i), such that he can split the pile's cookies amongst X people evenly.

Since Devon is a very efficient chef, whenever Alex takes cookies from a pile, he is able to restock the pile with exactly the same number of cookies. This means that the sizes of the cookie piles effectively do not change. Whenever Alex arrives, help him determine the best pile to take cookies from.

INPUT FORMAT:
The first line contains N and Q ($1 \leq Q \leq 100,000$). The second line contains N integers representing a_i. The next Q lines contains queries. Each of these lines consists of one integer X ($1 \leq X \leq 1,000,000$), where X is prime.

OUTPUT FORMAT:
For each query, output the minimum i such that X divides a_i, or output -1 if no such a_i exists.

SHORT NAME: `cookie`

SAMPLE INPUT:

```
5 6
2 15 49 11 17
3
7
2
3
13
5
```

SAMPLE OUTPUT:

```
2
3
1
2
-1
2
```

SOLUTION:

This problem gives us a list of integers A_i, and for each query value X, asks us to find the minimum index i such that A_i divides X. The primary insight for this problem is that because each query value X is prime, and we're querying the factors of A_i, we only need to consider the prime factorization of A_i.

Once we've accomplished this, we can simply create an array of sets, where each index of the array k represents the value of a prime factor, and each value in the set represents the index i corresponding to a value A_i such that k divides A_i. We should perform this as a pre-processing step before we process any

queries. This requires that we implement an algorithm for factoring integers, which can be done in $O(\sqrt{N})$ by simply checking whether all integers less than or equal to $\sqrt{N}$ are factors.

Then, for each query value X, we simply find the minimum value in the set corresponding to $k = X$. If there is no such set, then k is not a prime factor of any value in A_i and we output -1. Otherwise, we output the minimum value in the set to get the first possible index of i. This solution runs in time.

A simple optimization we can make is because we will only ever need the first index of i, we can simply maintain the minimum value in the set.

I Hungry Hungry Larrys

At the end of a long day at work, Devon stands in his cubicle at the northwest corner of an $N \times N$ grid ($1 \leq N \leq 800$) of cubicles on the 100^{th} floor of TJ IOI Inc. Each minute, Devon can move either one cubicle to the south or one cubicle to the east, and would like to reach the elevator, located in the cubicle at the southeast corner of the floor.

However, a number of Larrys $L_{i,j}$ reside in each cubicle ($0 \leq L_{i,j} \leq 9$), pretending to do work. Every time Devon moves into a new cubicle, each Larry in the cubicle that Devon moves into will reach into Devon's wallet and take one dollar. It is guaranteed that there are no Larrys in Devon's own cubicle (i.e., the one he begins on).

Thankfully, Devon's wallet has an infinite amount of money, but he still would like to lose as little money as possible. Determine the least amount of money that Devon must lose to the hungry hungry Larrys along the way, in order to reach the elevator and exit the building.

Note: North corresponds to up, south corresponds to down, east corresponds to right, and west corresponds to left.

SHORT NAME: `hungry`

INPUT FORMAT:
The first line of input contains the integer N, the size of the grid of cubicles. The next N lines each contain N integers, and together describe the number of Larrys within each cubicle.

OUTPUT FORMAT:
Output a single integer, the least amount of money that Devon must lose in order to get to the elevator.

SAMPLE INPUT:

```
4
0 2 5 1
2 9 3 0
4 6 1 2
8 2 2 6
```

SAMPLE OUTPUT:

```
16
```

SOLUTION:

In this problem, we would like to find essentially what is the shortest path between the top left cell and the bottom right cell of the grid, where the metric of distance is given by the number of Larry's passed over along the way. However, we can make an observation that simplifies the problem: because Devon can only travel to the right or down, in order to reach any specific cell, Devon must come from either the cell directly above or the cell directly to the left. The fact that Devon cannot move "backwards" allows us to not have to use a shortest path algorithm such as Dijkstra's algorithm, and instead use a more efficient approach.

As this problem is easily decomposable into smaller, similar problems, we are motivated to use a dynamic programming approach. Because any path through a cell must go through either the cell above or to the left, we can keep track of the shortest path to each individual cell by building it up from previous cells. We know that the shortest path to the starting cell is 0. We then find that the shortest path to any given cell $C_{i,j}$ is the minimum of the shortest paths to $C_{i-1,j}$ and $C_{i,j-1}$ (if these cells exist) plus the number of Larrys on cell $C_{i,j}$. We can then use this to build up a grid of shortest paths, using previous results to calculate the shortest path to the next cell. After calculating this grid, the answer will be found at $C_{N,N}$ (for a 1-indexed grid).

J Grocery Shopping

On the opening day of the TJHSST Third Floor Grocery Shop, N ($1 \leq N \leq 100,000$) students arrive, where the ith student would like to buy A_i ($1 \leq A_i \leq 100$) items. The grocery shop has a total of M ($1 \leq M \leq 100,000$) checkout stations, and the number of seconds it takes for a student to checkout is exactly equal to the number of items that student purchases.

The grocery shop's management team has contracted TJ IOI Inc. to help ease some of the checkout congestion. The consultants at TJ IOI Inc. have devised an unusual plan to facilitate grocery checkouts in a calm (but inefficient) manner: all N students will line up in the order of arrival, and the management team will divide the students into M contiguous segments. Each segment will check out at a different checkout station, and all of the stations may operate at the same time. Please help the management team determine the minimum amount of time necessary to checkout all students if they divide the students optimally.

SHORT NAME: `grocery`

INPUT FORMAT:
The first line contains N and M. The next N lines contain an integer representing A_i.

OUTPUT FORMAT:
Output the minimum amount of time necessary to checkout all students.

SAMPLE INPUT:

```
5 3
1
2
3
4
5
```

SAMPLE OUTPUT:

```
6
```

SOLUTION:

Because the problem is asking for us to compute sums over contiguous ranges, we are first motivated to implement the prefix-sum data structure, which is defined as follows. Let $P[i] = \sum_{k=0}^{i} A[k]$. Then, to compute the sum between i and j, which is normally a $O(N)$ operation, we can merely compute $P[i]-P[j-1]$ instead, reducing it to constant time. Note that the prefix-sum array can be computed in linear time since $P[i] = P[i-1] + A[i]$.

Next, we employ the strategy of binary searching on the answer. The central premise of this idea is to check whether a given solution (time needed to checkout all students) is attainable; if it is then we try a smaller solution and if it isn't then we try a larger solution. By choosing solution values similarly to a binary search, we can add a $\log(AN)$ factor as opposed to an AN factor.

In order to check whether any given solution, we perform a series of binary searches. Suppose that the solution is x and we want to start a contiguous range at i. We want to find the largest j such that $P[j] - P[i-1] \leq x$. To find j efficiently, we can binary search the value $x + P[i-1]$ on the prefix-sum array to find j in $O(\log M)$. We then repeat this process M times (once for each checkout station), and update i on each iteration accordingly.

If the final index that we obtain from the binary search is less than M, then x is too small and we try a larger value. Otherwise, we can try smaller values of x.

The final time complexity of this solution is $O(M \log(M) \log(AN))$.