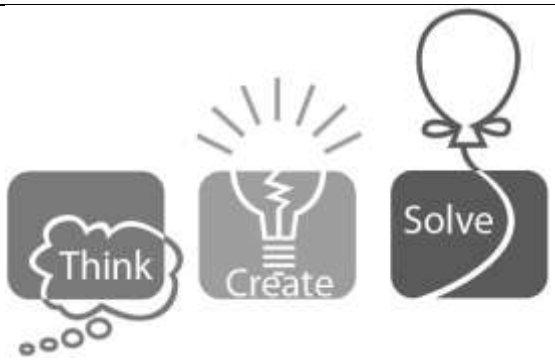
University of Central Florida



2022 Local Programming Contest (Final Round)

Problems			
Problem#	Difficulty Level	Filename	Problem Name
1	Easy	maxi	Number Maximization
2	Easy-Medium	calendar	Simplified Calendar System
3	Easy-Medium	freq	Letter Frequency
4	Medium	pseudo	Pseudo Pseudo Random Numbers
5	Medium-Hard	wordtree	Word Tree
6	Medium-Hard	house	House Prices Going Up
7	Medium-Hard	whichnum	Which Number
8	Medium-Hard	maxsat	Maximum Satisfaction
9	Medium-Hard	auction	Share Auction
10	Hard	desert	Desert Travel
11	Hard	fence	Security Fence

Call your program file: filename.c, filename.cpp, filename.java, or filename.py

For example, if you are solving Number Maximization:

Call your program file: maxi.c, maxi.cpp, maxi.java, or maxi.py

UCF Local Contest (Final Round) — September 10, 2022

Number Maximization

filename: `maxi`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

Everybody likes larger numbers since they come across as more powerful, more money, etc.

The Problem:

Given a number, determine the largest number that can be formed using the exact same digits as the given number.

The Input:

There is only one input line; it contains an integer between 0 and 999,999 (inclusive). Assume that the input number will not have leading 0's. Note, however, that the input can be just the value 0.

The Output:

Print the largest number that can be formed using the exact same digits as the input number.

Sample Input	Sample Output
2897	9872
15010	51100
99	99

UCF Local Contest (Final Round) — September 10, 2022

Simplified Calendar System

filename: `calendar`
Difficulty Level: `Easy-Medium`
Time Limit: `5 seconds`

Consider a simplified calendar system where each year is 12 months and each month is 30 days, i.e., each year is 360 days. This sure makes the math easy, or does it?

The Problem:

Given a date and what day of the week it is, determine what day of the week a second date is. The second date will be after (timewise) the first date.

The Input:

There are two input lines. The first input line contains four integers:

- d ($1 \le d \le 30$), providing what day of the month the first date is,
- m ($1 \le m \le 12$), providing what month of the year the first date is,
- y ($1000 \le y \le 2999$), providing what year the first date is, and
- n ($1 \le n \le 7$), providing what day of the week the given date is (1 represents Sunday, 2 represents Monday, …, 7 represents Saturday).

The second input line provides the second date; it contains three integers (similar to the first three integers of the first input line). Again, the second date will be later than the first date (timewise).

The Output:

Print an integer (between 1 and 7, inclusive) indicating what day of the week the second date is.

Sample Input	**Sample Output**
20 11 2021 7 29 11 2021	2
20 10 1999 5 15 4 2002	4

UCF Local Contest (Final Round) — September 10, 2022

Letter Frequency

filename: `freq`
Difficulty Level: `Easy-Medium`
Time Limit: `5 seconds`

The most common consonants in English are R, T, N, S, and L. The most common vowels are E, A, and I. It is definitely fun to find the most common letters.

The Problem:

Given a set of words, find the letter that occurs the most in each position. That is, the most frequent letter at Position 1 when considering all the words, the most frequent letter at Position 2 when considering all the words, etc. If more than one letter is the most frequent for a particular position (i.e., ties for max at that position), print those letters in alphabetical order.

The Input:

The first input line contains an integer, n ($1 \leq n \leq 20$), indicating the number of words. Each of the next n input lines contains a word. Assume that each word contains 1-30 lowercase letters and it starts in column one.

The Output:

Print each position and the letter(s) that occur the most in that position, following the format illustrated in Sample Output. Note that each output line starts with a number, immediately followed by a colon (':'), followed by a space, followed by a letter. If there are multiple letters for an output line, they are separated by exactly one space.

Sample Input	Sample Output
5 this is an example ink	1: i 2: n 3: a i k 4: m s 5: p 6: l 7: e
3 this is longlonglong	1: i l t 2: h o s 3: i n 4: g s 5: l 6: o 7: n 8: g 9: l 10: o 11: n 12: g

UCF Local Contest (Final Round) — September 10, 2022

Pseudo Pseudo Random Numbers

filename: `pseudo`
Difficulty Level: `Medium`
Time Limit: `5 seconds`

It turns out that if numbers were truly random, then each possible bit string (string of 0's and 1's) of length ***n*** would be equally likely. For example, `111111` would be just as likely as `010110` to occur. Unfortunately, most people believe that any time they see the same bit over and over again, that the process can't be truly random.

You are in charge of generating random bit strings of length ***n*** for use in a video game. However, the producer of the game has asked you to remove all possibilities where there are **more than *k*** 0's or 1's in a row. For example, if ***n*** = 4 and ***k*** = 2, then the 10 valid bit strings would be `0010`, `0011`, `0100`, `0101`, `0110`, `1001`, `1010`, `1011`, `1100`, and `1101` (the other 6 strings of 4 bits either have more than two 0's in a row or more than two 1's in a row, so they are not valid).

The Problem:

Given the values of ***n*** and ***k***, determine the number of ***n*** bit strings that do not contain any runs of 0's or 1's of length greater than ***k***.

The Input:

There is only one input line; it contains two integers: ***n*** ($2 \le n \le 20$), indicating the length of the bit string for the problem, and ***k*** ($1 \le k \le n$), indicating the maximal length of a run of 0's or 1's for the bit strings to be created.

The Output:

Print the number of valid bit strings of length ***n*** that do not contain any runs of the same bit of length greater than ***k***.

Sample Input	Sample Output
4 2	10
5 1	2
20 20	1048576

UCF Local Contest (Final Round) — September 10, 2022

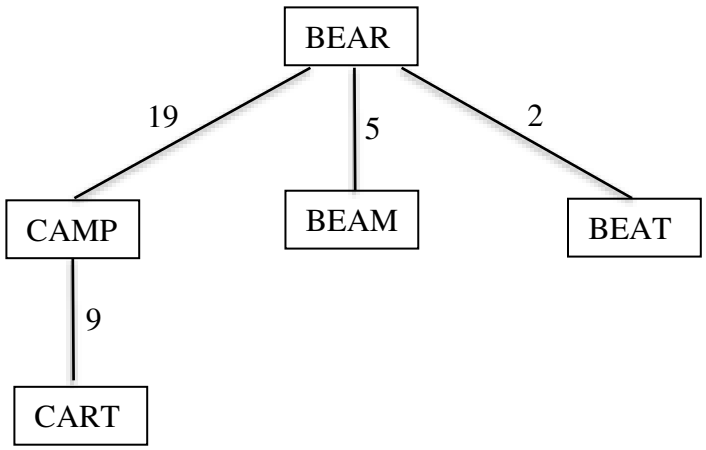
Word Tree

filename: `wordtree`
Difficulty Level: `Medium-Hard`
Time Limit: `5 seconds`

Given a set of ***n*** words, each of the same length, we can form a tree structure between the words by connecting pairs of words with an edge. In particular, we must add exactly ***n***−1 edges and make sure that there is a single simple path from every word to every other word.

We define the cost of an edge between two words as the sum of the costs between each pair of corresponding letters of the two words. We define the cost between two letters as being the absolute value of the difference between their Ascii values. For example, the cost of an edge between "CAMP" and "BEAR" would be 1 + 4 + 12 + 2 = 19 because |'C' − 'B'| = 1, |'A' − 'E'| = 4, |'M' − 'A'| = 12, and |'P' − 'R'| = 2.

Here is an illustration of a word tree with the words BEAM, BEAR, BEAT, CAMP and CART:



Finally, because we like only connecting words that are "similar", our goal is to minimize the value of the maximum edge cost of the tree. In the example above, it is impossible to connect the words in a tree structure where each edge has a cost of less than 19.

Note that there are no constraints on the tree structure, i.e., any word can be listed as the child of any other node. For example, it does not have to be a binary search tree. The structure must, of course, be a tree and the cost is as defined above.

The Problem:

Given a set of ***n*** words, each consisting of ***k*** uppercase letters, of all possible word trees that could be formed with those words, determine the minimum possible value of the maximum edge cost.

The Input:

The first input line contains two integers: n ($2 \leq n \leq 1000$), indicating the number of words, and k ($1 \leq k \leq 20$), indicating the length of each of the words. Each of the following n input lines contains one of the words. These words are guaranteed to be distinct, consist of exactly k uppercase letters, and start in column one.

The Output:

Print the minimum value of the maximum edge cost over all possible word trees for the set of input words.

Sample Input	**Sample Output**
5 4 BEAM BEAR BEAT CAMP CART	19
6 3 BAN BAR BAT CAN CAP CAR	2

UCF Local Contest (Final Round) — September 10, 2022

House Prices Going Up

filename: `house`
Difficulty Level: `Medium-Hard`
Time Limit: `5 seconds`

The house prices have been going up and the Virtual County Tax Collection Agency needs help to keep track of prices.

The Problem:

We assume the houses in Virtual County (VC) are numbered 1 through n, i.e., the house numbers are 1, 2, 3, …, n. VC keeps track of the house prices as they go up and would like to know the total price for different sections of the county, e.g., the total price for houses 4 through 15.

The Input:

The first input line contains an integer, n ($1 \le n \le 5 \times 10^5$), indicating the number of houses in VC. Each of the next n input lines contains an integer (between 1 and 10^9, inclusive) indicating the initial price for a house; first integer is the price for the first house, second integer is the price for the second house, etc.

After the initial price information for all the houses, the input will have a set of transactions (operations) to be processed. This section of the input starts with an integer, t ($1 \le t \le 10^5$), indicating the number of transactions. Each of the next t input lines contains a transaction to be processed. There will be two types of transactions:

- Update Transaction: This input line starts with the letter U in the first column, followed by one space, followed by a valid house number, followed by a space, followed by an integer (between 1 and 10^9, inclusive), indicating the increase in the price of that house.

- Retrieve Transaction: This input line starts with the letter R in the first column, followed by one space, followed by a valid starting house number, followed by a space, followed by a valid ending house number. The total price for this range is being requested. Assume that the ending house number will not be less than the starting house number, i.e., the requested range is valid.

The Output:

There is no output required for Update transactions. For each Retrieve transaction, output a separate line providing the total price for the range being requested.

Sample Input **Sample Output**

5 100 200 150 300 250 8 R 2 4 U 2 20 U 3 40 R 2 4 R 1 5 U 2 10 U 4 100 R 1 5	650 710 1060 1170

UCF Local Contest (Final Round) — September 10, 2022

Which Number

filename: `whichnum`
Difficulty Level: `Medium-Hard`
Time Limit: `2 seconds`

Natasha likes counting, but she has been told that a certain set of prime numbers are bad luck. Thus, she skips those numbers and all of their multiples entirely. For example, if `3`, `5` and `11` are the primes she is avoiding, then when she starts counting, she'll list the following integers:

`1, 2, 4, 7, 8, 13, 14, 16, 17, 19, 23, …`

You are curious, what is the n^{th} number Natasha will say?

The Problem:

Given the prime numbers whose multiples Natasha avoids, determine the n^{th} number she will say when she starts counting from 1.

The Input:

The first input line contains two integers: n ($1 \leq n \leq 10^{17}$), indicating the number for the query, and k ($1 \leq k \leq 14$), indicating the number of distinct prime numbers that Natasha avoids when counting (again, the multiples of these primes are avoided as well when counting). The second input line has k distinct prime numbers on it, representing the numbers (and multiples) which Natasha avoids. Assume that the product of all these primes will not exceed 10^{17}, e.g., the list of prime numbers can be {2, 3, 5, 11} since their product (330) does not exceed 10^{17} but the list of prime numbers will not be {1000000007, 1000000009} since their product exceeds 10^{17}. Note that, as illustrated in the Sample Input, the primes can be listed in any order (i.e., they are not necessarily listed in increasing order).

The Output:

Print the n^{th} number Natasha will say.

Sample Input	Sample Output
11 3 3 5 11	23
9 4 2 7 3 5	37

UCF Local Contest (Final Round) — September 10, 2022

Maximum Satisfaction

filename: `maxsat`
Difficulty Level: `Medium-Hard`
Time Limit: `2 seconds`

UCF is trying to make its COP 2500 class as accessible as possible, by creating s different sections of the course. Of course, it costs money to add sections of a course and UCF requires that each section have at least k students in it.

On the other hand, UCF wants its students to be satisfied. Each student who is going to take COP 2500 has predicted how satisfied they will be if they are placed in each of the different s sections. We assume the students are infallible and that their predictions are perfect. Each prediction is an integer score from 0 to 1000 (0 means not satisfied, 1000 means very satisfied).

Dr. Heinrich is very busy doing some geo-caching, so he would like you to write a program that determines the maximum sum of student satisfiability possible amongst all possible class assignments where each section has at least k students.

The Problem:

Given the number of sections of COP 2500 UCF is offering, the minimum number of students each section must have, and a list for each student of how satisfied they would be in each of the sections of the course, determine the maximum sum of satisfaction of all the students.

The Input:

The first input line contains three integers: n ($1 \leq n \leq 200$), indicating the number of students, s ($1 \leq s \leq n$), indicating the number of sections of COP 2500, and k ($1 \leq sk \leq n$), indicating the minimum number of students required to be assigned to each section. Each of the following n input lines contain s non-negative integers. The j^{th} value on line i is $a_{i,j}$, the satisfaction rating student i will have in section j.

The Output:

Print the maximum possible sum of student satisfaction scores within the constraint that each section must have at least k students assigned to it.

Sample Input **Sample Output**

5 2 2 10 3 6 8 9 4 11 2 12 1	45
4 4 1 1000 0 0 0 0 1000 0 0 0 0 1000 0 0 0 0 1000	4000

Note:

For the first Sample input, the maximum possible sum of student satisfaction scores can be achieved by the following assignments:

- Student 1 to Section 1
- Student 2 to Section 2
- Student 3 to Section 2
- Student 4 to Section 1
- Student 5 to Section 1

So, the total satisfaction is: `10 + 8 + 4 + 11 + 12 = 45`

UCF Local Contest (Final Round) — September 10, 2022

Share Auction

filename: `auction`
Difficulty Level: `Medium-Hard`
Time Limit: `3 seconds`

The Bold And Diverse Investors LLC is working on a new project. They have recently acquired several auction vouchers that can be used to purchase shares of some new companies. You have been left in charge of determining which lots to bid on. When you bid on a lot you will get a percentage of the lot based on the number of vouchers you bid on the lot and the total number of vouchers bid by others on the lot. For example, let's assume you bid 22 vouchers on a lot and the total vouchers bid by others on the lot is 66, then your percentage of the lot will be 22 / (22 + 66) = 25%. If this lot has a profit of $500, then your profit from this lot will be $500 * 25% = $125.

You have paid quite handsomely for information regarding how many vouchers (not including yours) are going to be placed on each lot, and the expected profit of each lot. Your job will be to place your bids such that you maximize your expected profit.

The Problem:

Given the number of vouchers you possess, the value of each lot, and the number of bids placed by others on each lot, determine the maximum expected profit you can get by bidding optimally.

The Input:

The first input line contains two integers: N ($1 \le N \le 10^5$) representing the number of lots to bid on, and V ($1 \le V \le 10^9$) representing the number of vouchers you possess. Each of the following N input lines contains two integers, representing (respectively) the expected profit and the number of vouchers placed by others for a lot. The i^{th} input line contains p_i and v_i ($1 \le p_i \le 10^9$; $1 \le v_i \le 10^9$), providing the info for the i^{th} lot.

The Output:

Print on a single line by itself a single number: the maximum possible expected profit. Answers within 10^{-6} absolute or relative of the expected answers will be considered correct.

Sample Input **Sample Output**

1 22 500 66	125.0
2 3 1 1 1 1	1.1666666666666667

UCF Local Contest (Final Round) — September 10, 2022

Desert Travel

filename: `desert`
Difficulty Level: `Hard`
Time Limit: `8 seconds`

You are stuck in a desert. The desert can be thought of as a Cartesian grid. There are many oases in the desert of which you know the *xy*-coordinates. You are going to be making a series of trips between pairs of oases, i.e., each trip has a starting oasis and an ending oasis. For any trip, you can stop in other oases along the way (as many as you'd like).

You obviously need water to travel in the desert. It turns out that every 1 unit traveled in the desert requires 1 unit of water. You need to determine what canteen size to use for each trip. You'd like to use the smallest canteen size for each trip (no one wants to carry more than what is needed).

You can fill up the canteen used in a trip at each oasis along the way to hold you until the next oasis. Thus, the maximum Euclidean distance between pairs of consecutive oases along your path for the trip determines the canteen size you need. For example, if you traveled from (0, 0) to (2, 4) to (4, 5), then your maximum distance would be approximately 4.472, the distance between (0,0) and (2,4), because this is longer than the distance between (2,4) and (4,5). This distance determines the canteen size needed for the trip.

The Problem:

Given the number of oases in the desert and several trips (each trip with a starting oasis and an ending oasis), determine the maximum Euclidean distance between pairs of consecutive oases along the path for each trip.

The Input:

The first input line contains an integer, O ($1 \le O \le 5{,}000$), representing the number of oases. Each of the following O input lines contains two integers, the i^{th} of which contains x_i and y_i ($-10^6 \le x_i \le 10^6$ and $-10^6 \le y_i \le 10^6$), representing the x and y coordinate of the i^{th} oasis. (Assume the oases are numbered 1, 2, 3, …, O.)

The next input line contains an integer, Q ($1 \le Q \le 500{,}000$), representing the number of trips. Each of the following Q input lines contains two integers, the i^{th} of which contains s_i and e_i ($1 \le s_i \le O$ and $1 \le e_i \le O$), representing the starting and ending oasis numbers for the i^{th} trip.

The Output:

For each trip, print on a single line by itself a single number: the maximum Euclidean distance between pairs of consecutive oases along the path for that trip. Answers within 10^{-6} absolute or relative of the expected answers will be considered correct.

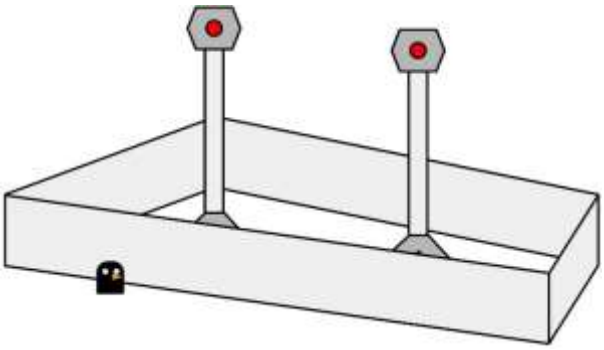
Sample Input **Sample Output**

6 0 0 3 7 1 2 2 0 4 5 2 5 3 1 2 6 5 3 1	3.1622776601683795 2.0 2.23606797749979

UCF Local Contest (Final Round) — September 10, 2022

Security Fence

filename: `fence`
Difficulty Level: `Hard`
Time Limit: `7 seconds`



You are the last vestige of hope in a world overrun by penguins. You are going to set up a sonic noise emitter array to keep the deranged beasts away from your camp. You have the perfect location to do so. There is already a fence set up to help keep out the beasts while building your weapon. The fence has been constructed by joining a series of rods embedded in the ground with straight sections of reinforced titanium sheets. Luckily the fence's interior angles between the sheets of titanium are each less than 180 degrees. You cannot adjust the location of the fence's rods or the titanium sheets, but the fence's current location should hopefully suffice for the task at hand.

To complete the sonic noise emitter array, you need to build two towers and there is a minimum distance required between these two towers. You don't want the penguins to interfere in your construction. To utilize the fence as much as possible, the towers need to be as far from the titanium sheets as possible while inside the fenced area.

The Problem:

Given the location of the rods of the fence and the minimum distance required between the two towers, determine the maximum possible distance the towers can be from their closest titanium sheets.

The Input:

The first input line contains two integers, N and D ($3 \le N \le 10^5$ and $1 \le D \le 10^6$), representing the number of rods that comprise the fence and the minimum distance required between the two towers. Each of the following N lines contains two integers, the ith of which are x_i and y_i ($-10^7 \le x_i \le 10^7$ and $-10^7 \le y_i \le 10^7$) representing the x and y coordinate of the i^{th} fence rod. The rod locations are given in a clockwise order. It is guaranteed that both towers can exist in the interior of the fenced region.

The Output:

Print on a single line by itself a single positive number: the furthest possible distance between the closest titanium sheet and towers. Answers within 10^{-6} absolute or relative of the expected answers will be considered correct.

Sample Input	Sample Output
3 1 0 0 0 3 3 0	0.6715728752538098
4 1 0 0 0 5 4 5 4 0	1.9999999999999998