Colombian Collegiate Programming League
CCPL 2013

Contest 6 -- June 8

Dynamic Programming

Problems

This set contains 10 problems; pages 1 to 18.

(Borrowed from several sources online.)

Page

A - Allergy Test 1

B - Bus Tour 3

C - Unicycle Counting 5

D - Private Space 7

E - Machine Works 8

F - Fibonacci Words 10

G - Room Service 12

H - Honeycomb Walk 14

I - Shares 15

J - Pyramids 17

Official site `http://programmingleague.org`
Official Twitter account @CCPL2003

A - Allergy Test

Source file name: `allergy.c`, `allergy.cpp` *or* `allergy.java`

A test for allergy is conducted over the course of several days, and consists of exposing you to different substances (so called allergens). The goal is to decide exactly which of the allergens you are allergic to. Each allergen has a live duration D measured in whole days, indicating exactly how many days you will suffer from an allergic reaction *if* you are allergic to that particular substance. An allergic reaction starts to show almost immediately after you have been exposed to an allergen which you are allergic to.

The test scheme has two action points per day:

I. At 8 o'clock each morning, at most one of the allergens is applied to your body.

II. At 8 o'clock each evening, you are examined for allergic reactions.

Thus an allergen with live duration D will affect exactly D allergic reaction examinations.

Of course, if you have two or more active allergens in your body at the time of an observed reaction, you cannot tell from that information only, which of the substances you are allergic to.

You want to find the shortest possible test scheme given the durations of the allergens you want to test. Furthermore, to allow simple large scale application the test scheme must be non-adaptive, i.e., the scheme should be fixed in advance. Thus you may not choose when to apply an allergen based on the outcome of previous allergic reaction examinations.

Input

The input consists of several test cases. The first line of each test case contains a single integer k ($1 \leq k \leq 20$) specifying the number of allergens being tested for. Then follow k lines each containing an integer D ($1 \leq D \leq 7$) specifying the live duration of each allergen.

The input ends with $k = 0$, which should not be processed as a test case.

The input must be read from standard input.

Output

For each test case, output the number of days of the shortest conclusive non-adaptive test scheme.

A scheme ends the morning when you no longer have active allergens in your body, thus a test scheme for a single allergen with live duration D takes D days.

The output must be written to standard output.

Sample input	**Sample output**
3 2 2 2 5 1 4 2 5 2 0	5 10

B - Bus Tour

Source file name: `bustour.c`, `bustour.cpp` *or* `bustour.java`

Imagine you are a tourist in Warsaw and have booked a bus tour to see some amazing attraction just outside of town. The bus first drives around town for a while (a *long* while, since Warsaw is a big city) picking up people at their respective hotels. It then proceeds to the amazing attraction, and after a few hours goes back into the city, again driving to each hotel, this time to drop people off.

For some reason, whenever you do this, your hotel is always the first to be visited for pickup, and the last to be visited for dropoff, meaning that you have to suffer through two not-so-amazing sightseeing tours of all the local hotels. This is clearly not what you want to do (unless for some reason you are *really* into hotels), so let's fix it. We will develop some software to enable the sightseeing company to route its bus tours more fairly—though it may sometimes mean longer total distance for everyone, but fair is fair, right?

For this problem, there is a starting location (the sightseeing company headquarters), h hotels that need to be visited for pickups and dropoffs, and a destination location (the amazing attraction). We need to find a route that goes from the headquarters, through all the hotels, to the attraction, then back through all the hotels again (possibly in a different order), and finally back to the headquarters. In order to guarantee that none of the tourists (and, in particular, *you*) are forced to suffer through two full tours of the hotels, we require that every hotel that is visited among the first $\lfloor h/2 \rfloor$ hotels on the way to the attraction is also visited among the first $\lfloor h/2 \rfloor$ hotels on the way back. Subject to these restrictions, we would like to make the complete bus tour as short as possible. Note that these restrictions may force the bus to drive past a hotel without stopping there (this is not considered visiting) and then visit it later, as illustrated in the first sample input.

Input

The input consists of several test cases. The first line of each test case consists of two integers n and m satisfying $3 \leq n \leq 20$ and $2 \leq m$, where n is the number of locations (hotels, headquarters, attraction) and m is the number of pairs of locations between which the bus can travel.

The n different locations are numbered from 0 to $n-1$, where 0 is the headquarters, 1 through $n-2$ are the hotels, and $n-1$ is the attraction. Assume that there is at most one direct connection between any pair of locations and it is possible to travel from any location to any other location (but not necessarily directly).

Following the first line are m lines, each containing three integers u, v, and t such that $0 \leq u, v \leq n-1$, $u \neq v$, $1 \leq t \leq 3600$, indicating that the bus can go directly between locations u and v in t seconds (in either direction).
The last case is followed by a single line containing `0 0`.

The input must be read from standard input.

Output

For each test case, display the case number and the time in seconds of the shortest possible tour.

The output must be written to standard output.

Sample Input	**Sample Output**
5 4 0 1 10 1 2 20 2 3 30 3 4 40 4 6 0 1 1 0 2 1 0 3 1 1 2 1 1 3 1 2 3 1 0 0	Case 1: 300 Case 2: 6

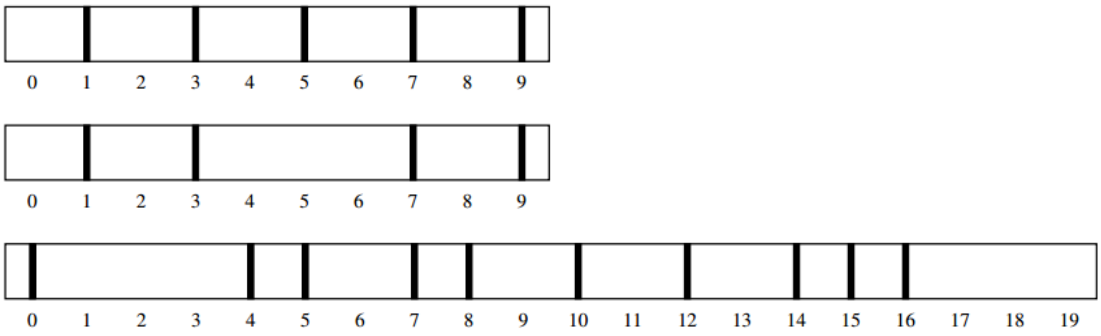
C - Unicycle Counting

Source file name: `unicycles.c`, `unicycles.cpp` *or* `unicycles.java`

Going to circus college is not as fun as you were led to believe. You are juggling so many classes. Trapeze class is sometimes up, sometimes down. There's a lot of tension in your high-wire class. And you've seen that lion taming can be cat-astrophic.

The one pleasure you find is in riding unicycles with your fellow classmates. Many people have unicycles with different sized wheels. One day you notice that all their tires leave a small mark on the ground, once per rotation. You decide to amuse yourself and avoid your classwork by trying to determine how many unicycles have passed by on a given stretch of road. In fact, you want to know the minimum number of unique unicycles that could have left the marks you observe. You make the simplifying assumption that any unicycle riding on the road will ride completely from the beginning to the end.

The figures below illustrate the sample input. Each thick black vertical stripe represents a mark left by a tire.



Input

Each line of input represents the observations on a stretch of road. A line begins with two integers $1 \leq m \leq 100$ and $1 \leq n \leq 10$, where m represents the length of the road and n represents the number of marks you observe on the road. These are followed by n unique integers $a_1, a_2, ..., a_n$, where $0 \leq a_i < m$ for all a_i. These n integers represent the positions where you observed a unicycle's tire has left a mark. There will be at most 100 lines of input. Input ends at end of file.

The input must be read from standard input.

Output

For each set of observations, print the minimum number of unicycles that could have produced the observed marks.

The output must be written to standard output.

Sample Input	**Sample Output**
10 5 1 3 5 7 9 10 4 1 3 7 9 20 10 0 4 5 7 8 10 12 14 15 16	1 2 3

D - Private Space

Source file name: `space.c, space.cpp` *or* `space.java`

People are going to the movies in groups (or alone), but normally only care to socialize within that group. Being Scandinavian, each group of people would like to sit at least one space apart from any other group of people to ensure their privacy, unless of course they sit at the end of a row.

The number of seats per row in the cinema starts at X and decreases with one seat per row (down to a number of 1 seat per row). The number of groups of varying sizes is given as a vector $(N_1, \ldots, N_n)$, where N_1 is the number of people going alone, N_2 is the number of people going as a pair etc.

Calculate the seat-width, X, of the widest row, which will create a solution that seats all (groups of) visitors using as few rows of seats as possible. The cinema also has a limited capacity, so the widest row may not exceed 12 seats.

Input

The input consists of several test cases. The first line of each test case contains a single integer n ($1 \leq n \leq 12$), giving the size of the largest group in the test case. Then follows a line with n integers, the i-th integer (1-indexed) denoting the number of groups of i persons who need to be seated.

The end of the input is given with a line containing `0`.

The input must be read from standard input.

Output

A single number; the size of the smallest widest row that will accommodate all the guests. If this number is greater than 12, output `impossible` instead.

The output must be written to standard output.

Sample input	**Sample output**
3 0 1 1 3 2 1 1 0	3 4

E - Machine Works

Source file name: `works.c`, `works.cpp` *or* `works.java`

You are the director of Arbitrarily Complex Machines (ACM for short), a company producing advanced machinery using even more advanced machinery. The old production machinery has broken down, so you need to buy new production machines for the company. Your goal is to make as much money as possible during the restructuring period. During this period you will be able to buy and sell machines and operate them for profit while ACM owns them. Due to space restrictions, ACM can own at most one machine at a time.

During the restructuring period, there will be several machines for sale. Being an expert in the advanced machines market, you already know the price P_i and the availability day D_i for each machines M_i. Note that if you do not buy machine M_i on day D_i, then somebody else will buy it and it will not be available later. Needless to say, you cannot buy a machine if ACM has less money than the price of the machine.

If you buy a machine M_i on day D_i, then ACM can operate it starting on day $D_i + 1$. Each day that the machine operates, it produces a profit of G_i dollars for the company.

You may decide to sell a machine to reclaim a part of its purchase price any day after you've bought it. Each machine has a resale price R_i for which it may be resold to the market. You cannot operate a machine on the day that you sell it, but you may sell a machine and use the proceeds to buy a new machine on the same day.

Once the restructuring period ends, ACM will sell any machine that it still owns. Your task is to maximize the amount of money that ACM makes during the restructuring.

Input

The input consists of several test cases. Each test case starts with a line containing three positive integers N, C, and D. N is the number of machines for sale ($N \leq 10^5$), C is the number of dollars with which the company begins the restructuring ($C \leq 10^9$), and D is the number of days that the restructuring lasts ($D \leq 10^9$).

Each of the next N lines describes a single machine for sale. Each line contains four integers D_i, P_i, R_i and G_i, denoting (respectively) the day on which the machine is for sale, the dollar price for which it may be bought, the dollar price for which it may be resold and the daily profit generated by operating the machine. These numbers satisfy $1 \leq D_i \leq D$, $1 \leq R_i < P_i \leq 10^9$ and $1 \leq G_i \leq 10^9$.

The last test case is followed by a line containing three zeros.

The input must be read from standard input.

Output

For each test case, display its case number followed by the largest number of dollars that ACM can have at the end of day $D + 1$.

Follow the format of the sample output.

The output must be written to standard output.

Sample Input	**Sample output**
6 10 20 6 12 1 3 1 9 1 2 3 2 1 2 8 20 5 4 4 11 7 4 2 10 9 1 0 0 0	Case 1: 44

F - Fibonacci Words

Source file name: `fibonacci.c`, `fibonacci.cpp` *or* `fibonacci.java`

The Fibonacci word sequence of bit strings is defined as:

$$F(x) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{if } n > 2 \end{cases}$$

Here + denotes concatenation of strings. The first few elements are

$F(0) = 0$
$F(1) = 1$
$F(2) = 10$
$F(3) = 101$
$F(4) = 10110$
$F(5) = 10110101$
$F(6) = 1011010110110$
$F(7) = 101101011011010110101$
$F(8) = 10110101101101011010110110101101$
$F(9) = 1011010110110101101011011010110110101101011011010110101$

Given a bit pattern p and a number n, how often does p occur in $F(n)$?

Input

The input contains several test cases. The first line of each test case contains the integer n ($0 \leq n \leq 100$). The second line contains the bit pattern p. The pattern p is nonempty and has a length of at most 100000 characters.

The input ends with $n = -1$.

The input must be read from standard input.

Output

For each test case, display its case number followed by the number of occurrences of the bit pattern p in $F(n)$. Occurrences may overlap. The number of occurrences will be less than 2^{63}.

The output must be written to standard output.

Sample input	**Sample output**
6 10 7 10 6 01 6 101 96 10110101101101 -1	Case 1: 5 Case 2: 8 Case 3: 4 Case 4: 4 Case 5: 7540113804746346428

G - Room Service

Source file name: `room.c`, `room.cpp` *or* `room.java`

You are working for a company designing cute, funny robot vacuum cleaners. At a high level, the robots' behavior is divided into three modes:

1. Exploration
2. Vacuuming
3. Rampant Killing

Unfortunately, while consumer testing shows that the last two modes are working perfectly, the exploration mode still has bugs. You've been put in charge of debugging.

At the beginning of the exploration mode, the robot is placed into a convex polygonal room. It has sensors that should tell it where all the walls are. Your job is to write a program that verifies that these readings are correct. To do this, the robot needs to physically touch every wall in the room.

Your problem is this: given the shape of a convex polygonal room with N walls and a starting point P inside it, determine the shortest route that touches each wall and then returns to P. Touching a corner counts as touching both incident walls.

Input

The input contains several test cases. Each test case starts with a line containing the number of vertices N of the polygon ($3 \leq N \leq 100$) and the integer coordinates P_x and P_y of the robot's starting point ($-10000 \leq P_x, P_y \leq 10000$). This is followed by N lines, each containing two integers x, y ($-10000 \leq x, y \leq 10000$) defining a vertex of the polygon. Vertices are given in counterclockwise order, all interior angles are less than 180 degrees, the polygon does not self-intersect, and the robot's starting point is strictly inside the polygon.

The last case is followed by a line containing 0.

The input must be read from standard input.

Output

For each test case, display the case number and the length of the desired route, accurate to two decimal places.

The output must be written to standard output.

Sample input	**Sample output**
4 0 0 -1 -1 1 -1 1 1 -1 1 3 10 1 0 0 30 0 0 20 0	Case 1: 5.66 Case 2: 36.73

H - Honeycomb Walk

Source file name: `honey.c, honey.cpp` *or* `honey.java`

A bee larva living in a hexagonal cell of a large honeycomb decides to creep for a walk. In each ''step'' the larva may move into any of the six adjacent cells and after n steps, it is to end up in its original cell.

Your program has to compute, for a given n, the number of different such larva walks.

Input

The first line contains an integer giving the number of test cases to follow. Each case consists of one line containing an integer n, where $1 \leq n \leq 14$.

The input must be read from standard input.

Output

For each test case, output one line containing the number of walks. Under the assumption $1 \leq n \leq 14$, the answer will be less than 2^{31}.

The output must be written to standard output.

Sample input	**Sample output**
2 2 4	6 90

I - Shares

Source file name: `shares.c`, `shares.cpp` *or* `shares.java`

You are a successful business man who uses to invest some money in the shares market. As a successful man you manage a network of well prepared spies assistants that can assure you the values of the shares for the next day. Each day you have a capital that you can spend in the market according to your assistants suggestions. In addition, you can only buy packs of shares from several salesmen.

Your goal is to select which packs should be bought in order to maximize the profits without exceeding the amount of capital you have.

Input

The input consists of several test cases. The first line of each test case contains the maximum capital C that you can invest ($0 < C \leq 2^{30}$). The next line has two integers, the number of total shares N ($0 < N \leq 500$) and the number of packs P ($0 < P \leq 50000$). Each one of the following N lines describe the N shares. Each line contains two integers a_i and t_i representing the current price and the expected price for the next day of the ith share ($1 \leq i \leq N$), respectively. Finally, the following P lines contain the information of the packs, one per line. For each line, the first integer R represents the number of different shares that contains this pack. Then for each share type you have two integers s_j and q_j ($1 \leq j \leq R$), where s_j is the id of the jth share and q_j is the quantity of the jth share in this pack.

The input ends with EOF, i.e., with the end of file.

The input must be read from standard input.

Output

For each test case of the input, print an integer that indicates the maximum expected profit for the next day.

The output must be written to standard output.

Sample Input	**Sample output**
500 4 6 10 15 8 6 20 15 12 12 3 1 6 2 7 3 8 3 3 8 1 10 2 4 3 4 10 2 5 1 10 2 1 4 2 4 1 3 2 2 4 3 2 1 200000000 5 30 2800 3500 1400 4800 2900 2800 500 3800 3300 4700 2 2 13 4 15 4 4 1 1 22 3 17 5 22 1 3 2 1 3 6 4 1 11 2 5 3 7 5 15 1 5 1 4 2 26 1 21 3 8 5 26 2 3 5 2 26 4 2 30 4 12 3 7 5 14 3 3 8 2 20 5 3 1 5 30 2 1 29 3 3 5 3 3 1 20 5 26 4 9 2 25 3 1 2 2 16 3 5 2 5 5 4 26 5 2 18 5 10 4 18 1 12 3 30 3 2 5 3 27 5 4 4 3 2 4 8 1 20 2 6 3 2 14 1 1 4 22 5 2 23 3 26 1 27 5 3 4 6 1 2 16 4 1 13 4 10 2 23 5 2 1 1 14 1 2 20 1 3 14 2 3 21 1 22 1 2 27 3 5 24 1 26 3 13 5 4 15 3 3 2 21 1 5 5 16 4 2 22 5 1 4 10 1 30	52 2168800

J - Pyramids

Source file name: `pyramids.c`, `pyramids.cpp` *or* `pyramids.java`

It is not too hard to build a pyramid if you have a lot of identical cubes. On a flat foundation you lay, say, 10×10 cubes in a square. Centered on top of that square you lay a 9×9 square of cubes. Continuing this way you end up with a single cube, which is the top of the pyramid. The height of such a pyramid equals the length of its base, which in this case is 10. We call this a *high* pyramid.

If you think that a high pyramid is too steep, you can proceed as follows. On the 10×10 base square, lay an 8×8 square, then a 6×6 square, and so on, ending with a 2×2 top square (if you start with a base of odd length, you end up with a single cube on top, of course). The height of this pyramid is about half the length of its base. We call this a *low* pyramid.

Once upon a time (quite a long time ago, actually) there was a pharaoh who inherited a large number of stone cubes from his father. He ordered his architect to use all of these cubes to build a pyramid, not leaving a single one unused. The architect kindly explained that not every number of cubes can form a pyramid. With 10 cubes you can build a low pyramid with base 3. With 5 cubes you can build a high pyramid of base 2. But no pyramid can be built using exactly 7 cubes.

The pharaoh was not amused, but after some thinking he came up with new restrictions.

1. All cubes must be used.
2. You may build more than one pyramid, but you must build as few pyramids as possible.
3. All pyramids must be different.
4. Each pyramid must have a height of at least 2.
5. Satisfying the above, the largest of the pyramids must be as large as possible (i.e., containing the most cubes).
6. Satisfying the above, the next-to-largest pyramid must be as large as possible.
7. And so on...

Drawing figures and pictures in the sand, it took the architect quite some time to come up with the best solution.

Write a program that determines how to meet the restrictions of the pharaoh, given the number of cubes.

Input

The input consists of several test cases, each one on a single line. A test case is an integer c, where $1 \leq c \leq 10^6$, giving the number of cubes available.

The last test case is followed by a line containing a single zero.

The input must be read from standard input.

Output

For each test case, display its case number followed by the pyramids to be built. The pyramids should be ordered with the largest first. Pyramids are specified by the length of their base followed by an L for low pyramids or an H for high pyramids. If two differenct pyramids have the same number of cubes, list the high pyramid first. Print "`impossible`" if it is not possible to meet the requirements of the pharaoh.

Follow the format of the sample output.

The output must be written to standard output.

Sample input	**Sample output**
29 28 0	`Case 1: 3H 3L 2H` `Case 2: impossible`