NOI 2011 TASKS OVERVIEW

Tasks
Task 1: CHANGE
Task 2: PAINT
Task 3: TOUR
Task 4: TUTOR
Task 5: SEQUENCE

Notes:

1. Each task will be tested on several data sets.
2. The maximum execution time for every task is 10 seconds.
3. Each data set is worth 20 points.
4. For each question, the data sets vary in size, starting from small, and ending with sizes around the limits given in each question.
5. Either zero mark or full mark (20 points) is awarded to your answer to each data set.
6. The task statements are contained on 11 pages following this overview page.

HAPPY PROGRAMMING!

Task 1: CHANGE

Jack Sagert is a man on a mission. He carries a pocket full of change consisting of 5, 10, 20 and 50 cent coins, and whenever he has to pay any amount, he will choose his change so that he uses the minimum number of coins possible. This is because he feels that it is his life's destiny to carry a lot of change everywhere and would like to spend as few of them as possible.

It is not known how he came to be like this. Legend has it that his mother's gynaecologist dropped him on his head when he was born.

In any case, your task is to write a program such that, given the number of 5, 10, 20 and 50 cent coins in his pocket, you work out the number of coins of each denomination and the total number of coins used, such that the total number is the minimum that he can use.

Note that Jack won't be able to pay any arbitrary amount of money. For example, if something costs $2.35 and he does not have enough change, he won't be able to pay this amount.

Input

The input (from standard input) consists of a single line of numbers, where the first four are the number of 5, 10, 20 and 50 cent coins respectively, and the fifth number is an integer representing the amount that Jack has to pay, in cents. (The number of each type of coins is smaller than $1,000,000$ while the amount is smaller than $100,000,000$ cents.)

Example 1: If he has 2 of each type of coin and he needs to pay 35 cents, the input would be:

```
2 2 2 2 35
```

Example 2: If he has three 5-cent coins, two 10-cent coins and four 50-cent coins, and has to pay $5.35 (or 535 cents), then the input would be:

```
3 2 0 4 535
```

Output

Your program should output (to standard output) the number of each type of coin that Jack should spend, as well as the total number of coins. Again this total should be a minimum so that Jack can retain as many coins as possible.

In Example 1 above, the minimum number of coins is 3 (one 5 cent, one 10 cent and one 20 cent coin, giving 35 cents). So our output is:

```
1 1 1 0 3
```

In Example 2 above, Jack does not have enough change to pay $5.35, and your program should output "−1" to indicate that Jack cannot make up this amount. So your program should output:

```
-1
```

Task 2: PAINT

A junior college in Singapore just acquired a sailing boat, and the school's student sailing team undertakes to paint the boat. There are several painting jobs to be done on the boat, such as:

- primer painting on the keel,
- painting antifouling below the waterline,
- painting the deck,
- etc

Each job can be started and finished on the same day, but in order for the paint to dry, at most one job can be done every day.

The students face the following problem. The supplier of paint charges a certain amount for the paint for each job, but is increasing the price every day by an amount that depends on the job. For example, he may charge \$100 for the primer on the first day, and increase the price by \$2 each day, resulting in a price of \$102 on the second day, \$104 on the third day, etc. For the antifouling, he may charge \$200 on the first day, but increase the price by \$4 each day, resulting in a price of \$204 on the second day, \$208 on the third day, etc. The students have no facilities for storing the paint, and therefore need to buy the paint on the day on which they use it. The students want to complete all jobs in the cheapest possible way.

Input

Your program must read from the standard input the following data. The first line of the input contains one integer, n, ranging from 2 to 20000, which specifies the number of painting jobs. We identify the jobs by numbers, ranging from job 1 to job n, regardless of when they are executed. The following line contains n numbers, each ranging from 1 to 1000. The i^{th} number in this line specifies the price for the paint for job i. The last line contains n numbers, each ranging from 0 to 10. The j^{th} number in this line specifies the amount of money by which the price for the paint for job j increases every day. An example is given below.

```
7
100 200 500 300 400 200 100
5 2 0 5 7 1 3
```

In this example, there are 7 painting jobs. On the first day, the price of the paint for job 1 is \$100, for job 2 \$200, etc. On the second day, the price of the paint for job 1 increases by \$5 and is therefore \$100+\$5 = \$105, the price of the paint for job 2 is \$200+\$2 = \$202, etc. On the third day, the price of the paint for job 1 is \$100+2*\$5=\$110, the price of the paint for job 2 is \$200+2*\$2=\$204, etc.

Output

The required output, to be written to standard output, consists of a single number, specifying the smallest amount of money sufficient to complete all jobs. For the above example, the output is:

```
1837
```

Task 3: TOUR

A tourist is holding an $R \times C$ grid map of Singapore ($1 \leq R, C \leq 20$) where '~' (tilde character) is water, '.' is land, '2', '3', ..., '9' are attractive spots, and 'C' is Changi. The figure below is an example of a 3×5 grid map with one 'C' and two attractive spots labeled with '5' and '6'.

```
~.~.~
6.~5.
~..C~
```

There can be up to N ($0 \leq N \leq 14$) attractive spots in Singapore. The label of each attractive spot i is its satisfaction point S_i ($2 \leq S_i \leq 9$), for example, an attractive spot labeled with '5' has $S_i = 5$. If the tourist visits spot i, he will gain S_i points.

The tourist can only travel in the four directions (N/E/S/W) but only over the land (denoted with '.'). As Singapore is a modern city, it is always possible to reach all other land ('.') cells from any land ('.') cell. Traveling is tiring, so every step contributes to dissatisfaction (or negative satisfaction). The tourist says that each step contributes -2 satisfaction points. In the example above, if he travels as directly as possible from Changi 'C' on row 2 column 3 to spot '6' on row 1 column 0 (0-based indexing), he must use 4 steps with $4 * -2 = -8$ satisfaction points.

This tourist lands on Changi 'C', wants to visit these attractive spots, and then finally goes back to Changi 'C' to fly home. Each point on land (including attractive spots) can be visited multiple times, but the satisfaction points for each attractive spots can be collected only once. What is the maximal number of satisfaction points achievable by the tourist, using a given map of Singapore?

If his tour is 'C' → '6' → '5' → 'C', he will get: $4*(-2)+6+5*(-2)+5+1*(-2) = -8+6+-10+5-2 = -9$ satisfaction points. If his tour is 'C' → '5' → 'C', he will get: $1*(-2)+5+1*(-2) = -2+5-2 = 1$ satisfaction point (this is the maximum over all possibilities), i.e. this tourist skips '6' as it is not worth his time.

Input

The input (from standard input) starts with two integers R, C in one line. Then, R lines with exactly C characters per line will appear next. There is always exactly one 'C' and between zero and fourteen attractive spots in the input. For the above example, the input is:

```
3 5
~.~.~
6.~5.
~..C~
```

Output

Your program should output a single integer (to standard output), which represents the maximum number of satisfaction points achievable, using the given map of Singapore. For the above example, the output is:

```
1
```

Task 4: TUTOR

You need to compute a *sequence of actions* in order to maximize `cash` earned in a tutor simulation game. The actions involve earning `cash` from tuition (**TEACH**), studying in college to increase your `knowledge` (**TRAIN**), and buying books (**BUY**). The **TRAIN** action allows you to increase tuition income. The **BUY** action allows you to reduce time units consumed for each **TRAIN** action. Each of these actions consumes time units and there are only `maxTimeUnits` available for work.

Like all games, the puzzle difficulty depends on the game variables and certain rules. For this task, there are 4 game variables and 5 game rules. Value ranges are provided where appropriate.

Game Variables

1. `maxTimeUnits` (10 - 1000): The maximum number of time units in the simulation game.

2. `learningRate` (1, 2, 4, or 8): Scales down the time units consumed by a **TRAIN** action, based on the number of `book` in your possession.

3. `paybackRate` (5, 10, or 20): Scales up the tuition `income` earned from a **TEACH** action, based on the `knowledge` level.

4. `bookCost`: An array of 4 integers in non decreasing order where the i-th integer is the cost of the i-th book. (The cost of each book is in the range of \$5 - \$500).

Game Rules

1. Before the start of the simulation, you have `maxTimeUnits` left; your `cash` is 0; your `knowledge` is 0; and you have 0 `book`.

2. The simulation game lasts for `maxTimeUnits`. Your aim is to obtain as much `cash` as possible.

3. Every **TEACH** action spends 2 time units. The formula below gives the tuition `income` per **TEACH** action. That is, the more `knowledge` that you have, the higher your `income` will be.
   ```
   income = 10 + min(20, knowledge) * paybackRate
   ```

4. Every **TRAIN** action costs 20 dollars and it will increase the `knowledge` level by 1 unit. The formula below gives the `trainingTime` per **TRAIN**

action. That is, the more `book` or the higher `learningRate` that you have, the lower your `trainingTime` will be.
`trainingTime = max(1, (int)(8 / max(1, book * learningRate)))`

5. There are 4 books in this game. The i-th BUY action will buy the i-th book. It takes i time units to buy the i-th book (0-based indexing, i.e. the first book can be bought with 0 time unit) and the cost of the i-th book is stated in `bookCost[i]`. As there are at most 4 books, you can perform at most 4 **BUY** actions.

Your Task

Write a program that reads in the game variable values (see sample input), and determines the best possible sequence of actions. You need to implement a planner so that your `cash` is maximum at a certain time unit `t` where `t` $\in$ `[0 .. maxTimeUnits]`. However, you should not violate the following important constraints:

1. You cannot overshoot the `maxTimeUnits`, when choosing an action.
2. You cannot incur negative `cash` at any point; i.e. you must be able to pay for any **TRAIN** or **BUY** action.

For example, suppose `maxTimeUnits`, `learningRate`, and `paybackRate` are 13, 8, and 20, respectively. Assume the cost of the 4 books are \$5, \$50, \$100, and \$200.

Since you have 13 time units, a simple solution is to perform **TEACH** 6 times. You can already get $6*(10+\min(20,0)*20) = 6*10 = \$60$. However, this is not the best answer (you will not get any marks by giving this answer). The optimal answer for this test case has `cash` = \$95 and shown below:

`t`	`cash`	`knowledge`	`book`	remarks
0	0	0	0	start of simulation
2	10	0	0	**TEACH** (`income = 10`)
2	5	0	1	**BUY** (the 0-th book, 5\$, no change in `t`)
4	15	0	1	**TEACH** (`income = 10`)
6	25	0	1	**TEACH** (`income = 10`)
7	5	1	1	**TRAIN** (we have 1 book, `trainingTime = 1`)
9	35	1	1	**TEACH** (`income = 30`)
11	65	1	1	**TEACH** (`income = 30`)
13	95	1	1	**TEACH** (`income = 30`)

Input

The input (from standard input) consists of two lines. The first line contains 3 integers, which are the `maxTimeUnits`, `learningRate`, and `paybackRate`. The second input line contains 4 integers where the i-th integer is the cost of the i-th book. For the example above, the input is:

```
13 8 20
5 50 100 200
```

Output

The required output (to standard output), consists of a single number, specifying the maximum `cash` that you can gain. For the example above, the output is:

```
95
```

Task 5: SEQUENCE

The task is to predict sequences according to the easiest underlying law which applies. For this task, the following three laws are considered. Every integer in the sequences is in the range $[0, 1, \ldots, m-1]$ where m is a parameter given at the input; m is never larger than 10000.

Eventually constant sequences: An eventually constant sequence is a sequence $a_0\ a_1\ \ldots\ a_n\ \ldots$ such that there is a number d with $a_n = a_d$ for all $n \geq d$; the least such number d is called the degree of the sequence. For example, 0 8 6 6 6 ... is an eventually constant sequence of degree 2.

Periodic sequences: A periodic sequence is a sequence given by a seed $a_0\ a_1\ \ldots\ a_d$ such that for all $n > d$ it holds that $a_n = a_{n-d-1}$. The degree d of a periodic sequence is the least number d such that $a_0\ a_1\ \ldots\ a_d$ is a seed of the sequence. For example, 1 2 3 4 1 2 3 4 1 2 ... is a degree-3 periodic sequence with seed 1 2 3 4.

Polynomial sequences: A polynomial sequence modulo m of degree d is a sequence $a_0\ a_1\ \ldots\ a_n\ \ldots$ generated from an integer-valued polynomial of degree d. For example, 0 0 1 3 6 10 15 1 ... is a degree-2 polynomial modulo 20 generated by $a_n = (\frac{1}{2}n^2 - \frac{1}{2}n) \mod 20$. An alternative approach to define a polynomial sequence is as follows. A sequence $a_0 a_1 \ldots a_n \ldots$ is a polynomial sequence modulo m of degree d if it is a constant sequence (i.e., $a_n = a_0$ for all n) with degree $d = 0$ or it is a sequence derived from another nonzero sequence $b_0\ b_1\ \ldots$ of degree $d-1$ such that $a_{n+1} = (a_n + b_n) \mod m$ for all n. The second approach enables us to build a prediction method without using multiplications and divisions. (Note: $x \mod y$ equals the reminder r when x is divided by y. We assume $0 \leq r < y$.)

Note that all three concepts are related as the constant sequences are just the sequences of degree 0 of any of these three concepts.

Our task is to predict the concept of least degree which matches the data. For example, suppose we want to predict a_3 for a periodic sequence with $a_0 = 0$, $a_1 = 1$, and $a_2 = 0$. Then, we predict $a_3 = 1$ since the data follows the periodic sequence of degree 1 with seed 0 1. Note that we should not predict $a_3 = 0$ which follows the periodic sequence of degree 2 with seed 0 1 0.

Furthermore, some prediction task asks to identify, among the concepts of eventually constant, periodic and polynomial sequences, the concept that

gives the lowest degree and then to predict the next value along that concept. All prediction tasks given admit a unique solution.

Input

Your program must read from the standard input the following data. Each task is given in one of the following formats: "`Ec` m a_0 a_1 ... a_{n-1}" (for eventually constant sequences), "`Pe` m a_0 a_1 ... a_{n-1}" (for periodic sequences), "`Po` m a_0 a_1 ... a_{n-1}" (for polynomial sequences), "`Se` m a_0 a_1 ... a_{n-1}" (for general prediction tasks where the program has to chose the adequate concept with the lowest degree). There can be several prediction tasks which are separated by semicolons and a full stop follows the last prediction task. The number n is at most 90 and the number m is at most 10000. Also, for every a_i, $1 \leq a_i \leq m$.

Below shows an example input.

```
Pe 5 0 1 2 0 1;
Ec 2 0 1 1 1;
Po 100 0 1 4 9 16;
Se 50 1 1 0 1.
```

Output

For each input sequence, exactly one number should be output to standard output; this number has to be the next element in the sequence which matches the description.

For the above example, the output is:

```
2
1
25
1
```