Olympic pizza

Pizza is a widely-known traditional Italian food, consisting of oven-baked, flat bread covered with tomato sauce, melted mozzarella cheese and many other optional toppings. There are so many kinds of pizzas according to their shape (circular, rectangular or square), their thickness (from few millimeters to a couple of centimeters) and, most importantly, their toppings (cheese, ham, salami, sausage, fries, mushrooms, olives, etc., and any combination of them). That's why several pizza places (*pizzerias*) can serve such a large variety of pizzas!

Olympic pizza

Our crowded pizzeria at IOI needs your help to manage its orders and make its famous pizza. Its secret is to use very fresh ingredients and it needs frequent deliveries of new ingredients to achieve this. Because of frequent traffic jams in Italy, many deliveries arrive late, causing some ingredients to be unavailable at some times. When an ingredient is not available, the pizzeria cannot any make pizza that requires that ingredient.

Your task is to write a program that will receive notifications of new pizza orders and new ingredient deliveries. Then, it has to tell the pizza maker when she can bake a pizza: this has to be done as soon as all the needed ingredients are available.

Statement

Your task is to write a program that, receiving orders and deliveries, tells the pizza maker when she can bake pizzas as described above. Specifically, there are exactly 8 ingredients for topping a pizza and they are identified by the integers ranging from 0 to 7. The pizza orders are also numbered consecutively starting from 0 (which represents the first pizza order). Your program has to implement the following routines.

- `Init`() — it is called exactly once at the beginning of the execution (before any other routines) to indicate that the pizzeria can start receiving orders or deliveries.
- `Order`(N, A) — given a positive integer $N \leq 8$ and an array A of N integers (ranging from 0 to 7, with no two equal elements), it notifies your program that a pizza has been ordered and that it requires the N ingredients described by the elements of A (in arbitrary order).
- `Delivery`(I) — given an integer I between 0 and 7, it notifies your program that a delivery of *one portion* of ingredient I has been received, so it can be used for *one* pizza.

Your program has to call the following routine provided by the system to tell the pizza maker to bake a pizza:

- `Bake`(K) — given an integer $K \geq 0$, it indicates that the pizza for order K has to be baked.

You must call `Bake` as soon as its ingredients are ready: if there are more pizzas that can be baked, the one ordered first must be chosen. Some of the ordered pizzas might never be made because of the lack of some ingredients.

Example

In the following example, the left column reports the calls to your routines, while the right column reports the calls to the system routine that your program should make.

Your routines	System routine
`Init`()	
`Delivery`(1)	
`Delivery`(1)	
`Delivery`(1)	
`Delivery`(2)	
`Delivery`(2)	
`Order`(3, [1, 2, 3])	
`Delivery`(4)	
`Delivery`(4)	
`Order`(3, [1, 2, 4])	`Bake`(1)
`Delivery`(3)	`Bake`(0)
`Order`(4, [1, 2, 3, 4])	
`Delivery`(2)	

Subtask 1 [25 points]

The routines `Order` and `Delivery` will be called at most 100 times in total.

Subtask 2 [25 points]

The routines `Order` and `Delivery` will be called at most 5 000 times in total.

Subtask 3 [20 points]

The routines `Order` and `Delivery` will be called at most 100 000 times in total. Also, all delivery calls will happen before any order call.

Subtask 4 [30 points]

The routines `Order` and `Delivery` will be called at most 100 000 times in total.

Implementation details

You have to submit exactly one file, named `pizza.c`, `pizza.cpp` or `pizza.pas`, which implements the routines described above using the following signature.

C/C++ programs

```
void Init();
void Order(int N, int *A);
void Delivery(int I);
```

The `Bake` function will have the following signature:

```
void Bake(int K);
```

Pascal programs

```
procedure Init();
procedure Order(N : LongInt; var A : array of LongInt);
procedure Delivery(I : LongInt);
```

The `Bake` procedure will have the following signature:

```
procedure Bake(K : LongInt);
```

These routines must behave as described above. Of course you are free to implement other routines for their internal use. Your submissions must not interact in any way with standard input or standard output, nor with any other file.

Run-time limits

- Time limit: 1 second.
- Memory limit: 256 MiB.

Italian queue

Sometimes (fortunately less often than many people think) in Italy the usual concept of monodimensional waiting line is replaced by a far less rigorous idea, to which we are going to refer as *Italian Queue*.

The Big Mess Theory

While it is a common thought that the Universe undergoes some supreme order, a long time ago we Italians realized, as Galileo's descendents, that the Universe is more complicated than expected. We empirically found that the Universe is better modeled after the *Big Mess Theory* and the Italian Queue is a good illustrative example. What for a foreigner seems to be a chaotic queue, for us it is just experimental validation of a theory…

Italian Queue (IQ)

Suppose that there is a large square room containing many people at the same time, and a special place in the room that is very interesting for some reason: we call this place the Very Interesting Point (VIP). At some time, more people decide to enter the already crowded room, one by one, to get close to the VIP: instead of queuing in the waiting line in arrival order, as normally expected, they form an IQ so that each newcomer reaches the currently free place that is closest to the VIP. (Ties are broken arbitrarily when there are many free places at the same distance from the VIP.) As expected, the resulting IQ is not a simple waiting line and this seems to confirm our theory.

Statement

This is an output-only task. You are given 10 text files named `input0.txt`, `input1.txt`, …, `input9.txt` containing the initial arrangement of the room and you are to submit files named `output0.txt`, `output1.txt`, …, `output9.txt` containing a possible final arrangement of the room (in other words, a simulation of where people might go).

The first line of each input file contains two positive integers N and K, where N is the size of the room (recall that the room is square), and K is the number of people that are going to enter the room (that is, the people whose behaviour you are going to simulate).

A description of the initial arrangement of the room follows. The VIP is represented by capital letter '`O`', a person is represented by capital letter '`X`', a free place is represented by the dash symbol '`-`'.

Each output file must contain the final description of one of the possible final states of the room, using the same format above. Since K people will have entered the room (settling in as many free places), we have that K '`-`' will be replaced by so many '`X`'.

Remarks

Each character of the room's description represents what we have so far called a *place*, whose coordinates are non-negative integers where (0, 0) is the coordinate of the topmost and leftmost places. The K new people can reach *every* free place, included the ones surrounded by other people. The distance between any two places is defined as the *Euclidean distance* between their integer coordinates.

Example

Suppose you are given the following input file.

```
5 3
--XX-
-XO-X
-XXX-
X--
-X-XX
```

The first person entering the room has only one choice (marked in red), at distance 1 from the VIP:

```
--XX-
-XOXX
-XXX-
X--
-X-XX
```

The second one is going to settle in the only position at distance √2 from the VIP:

```
-XXX-
-XOXX
-XXX-
X--
-X-XX
```

Finally, the third person has two possible positions (both at distance 2 from the VIP):

```
-XXX-
XXOXX
-XXX-
X-X--
-X-XX
```

Since only one solution is required, you can choose between the two possibilities. For example, a valid output file would be the following:

```
-XXX-
-XOXX
-XXX-
X-X--
-X-XX
```

Subtasks [10 points each]

Each of the 10 test cases is worth 10 points.

Touristic plan

An eccentric tourist has decided to visit some of the most beautiful Italian cities. In his luggage, together with an Italian/wherever-he-comes-from phrasebook, he has a number of large coins made of pure gold.

The tour

The tourist has already chosen a list of N cities to travel across, and intends to follow his plan until he runs out of money (so he might not reach the last cities in his list). He starts in the first city, and travels from one city to the next one, following the order he has established and paying one golden coin per route.

When he is in a city (included the first one), he can spend the night in a hotel in order to visit that city. If he does so, he has to pay the hotel one golden coin, and his happiness increases by a value that depends on the city.

Clearly, the tourist is interested in maximizing the happiness that he gains by suitably using the coins during this holiday.

Statement

Your task is to write a program that finds the maximum total happiness that the tourist can gain. Specifically, you have to implement a routine called `GreatestHappiness`(N, K, H) that, given the initial number K of golden coins, the number N of cities and the array H of N positive integers representing the amount of happiness for each of the cities, returns the required total happiness. Cities are numbered from 0 to N - 1: when the tourist visits city i, he gains happiness H[i], where $1 \leq H[i] \leq 1\,000$.

Note that the tourist must visit the cities in increasing order: for each city i = 0, 1, 2, …, N - 1, either he visits i (and spends one coin) or skips it (and does not spend a coin); in both cases, he must spend one coin anyway for travelling from city i - 1 to city i, when i ≥ 1.

Example

Suppose N = 4, K = 5 and H = [4, 5, 2, 7].

The tourist might visit the first city, travel, visit the second city, travel and visit the third city. He would spend 2 coins for travelling and 3 coins for visiting. By doing so, he would gain a total of happiness of 4 + 5 + 2 = 11.

But a better idea is to travel, visit the second city, travel, travel again and finally visit the fourth city. He would spend 3 coins for travelling and 2 for visiting, and he would gain a happiness of 5 + 7 = 12. This is the best achievable result.

Subtask 1 [18 points]

It holds N ≤ 10.

Subtask 2 [1 point]

It holds N = 42.

Subtask 3 [22 points]

It holds N ≤ 200.

Subtask 4 [26 points]

It holds N ≤ 1 000.

Subtask 5 [33 points]

It holds N ≤ 1 000 000.

Implementation details

You have to submit exactly one file, named `tourist.c`, `tourist.cpp` or `tourist.pas`, which implements the routine described above using the following signatures.

C/C++ programs

```
int GreatestHappiness(int N, int K, int *H);
```

Pascal programs

```
function GreatestHappiness(N, K: LongInt; var H : array of LongInt) : LongInt;
```

These routines must behave as described above. Of course you are free to implement other routines for their internal use. Your submissions must not interact in any way with standard input or standard output, nor with any other file.

Run-time limits

- Time limit: 1 second.
- Memory limit: 256 MiB.