A	Nuts to you	nuts.in	nuts.out	5 seconds	64 MiB
B	Prefix Goodness	prefix.in	prefix.out	5 seconds	64 MiB
C	Climb Every Mountain	mountain.in	mountain.out	5 seconds	64 MiB
D	N tasks	tasks.in	tasks.out	5 seconds	64 MiB
E	Lucky Number	lucky.in	lucky.out	5 seconds	64 MiB
F	Round Trip	trip.in	trip.out	5 seconds	64 MiB

Problem A- Nuts to You!

A very angry squirrel wants half of your stash of acorns. (Yes. This is the best plot I could think of.) You come to a bargain: you'll let him have 1 acorn to start, but then you'll take 2, and then let him have 3, and you take 4, and he takes 5, and you take 6, and so on, until the stash is divided up. The remaining acorns go to whoever was next.

Input Specification:

There will be multiple test cases (at most 100), presented one per line. Each line will contain a single integer n between 1 and 2^{30}, the number of acorns in your stash. The input will be terminated by the end of file.

Output Specification:

For each test case, output the number of acorns that you get out of your stash.

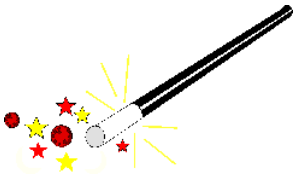
Sample Input:

```
1
2
3
4
```

Sample Output:

```
0
1
2
2
```

Problem B- Prefix Goodness



The *prefix goodness* of a set of strings is the length of the longest common prefix for the elements in the set, multiplied by the number of strings in the set. For example consider the set $\{000, 001, 0011\}$. The longest common prefix is "00", which means it has a prefix goodness of 6.

You are given a set of binary strings. Find the maximum prefix goodness among all possible subsets of these binary strings.

Input Specification:

The input begins with an integer $T \leq 20$, the number of test cases. Each test case begins with $n \leq 50000$, the number of binary strings, and is followed by n lines of binary strings, no longer than 200 characters each.

Output Specification:

For each test case output the maximum prefix goodness among all possible subsets of n binary strings.

Sample Input:

```
4
4
0000
0001
10101
010
2
01010010101010101010
11010010101010101010
3
01010101010100001000 1010
01010101010100001000 1000
01010101010100001000 1010
5
0101010101010000101001001010 0101
0101010101010000101001101010 1010
00001010101010110101
00010101010110101 01
00010101010101001
```

Sample Output:

```
6
20
66
44
```

Problem C- Climb Every Mountain

Cara has much to say about hiking. It seems like she has hiked everywhere and climbed every mountain in B.C.- and that's because she has! But now, Cara is off for a much bigger challenge: Alaska. She has obtained a book of topographical maps from the U.S. Bureau of Travel and Tourism, and she wants you to determine how much of Alaska she can hike.

Cara will hand to you as input a sequence of maps, each represented as a grid of numbers. These numbers represent the average height above sea level of that grid square. Height is important to Cara! She can only reach a higher (or lower) square if the square is adjacent (horizontally, vertically or diagonally) and if the height differential is less than or equal to one.

Input Specification:

The input will begin with a line with a single integer n. Following that will be a sequence of n test cases.

Each test case will begin with a single line containing integers Δ_x and Δ_y (satisfying $0 < \Delta_x, \Delta_y \leq 100$) which represent the dimensions of the map. Δ_x is the number of rows and Δ_y is the number of columns, and the map will display $\Delta_x \Delta_y$ acres of terrain. The next Δ_x rows contain an ASCII map, composed of the numbers 0-9, representing their heights. There will also be a single '*' on the map, denoting Cara's entry point. The entry point always has a height of 0.

Output Specification:

The output will describe the number of acres (gridsquares) that Cara can hike, including the acre containing the entry point. Leave one blank line between test cases. Otherwise, follow the format shown in the example below, paying careful attention to spacing.

Sample Input:

```
3
1 1
*
5 5
*1234
02274
05566
05754
05854
10 10
2323234323
2222222222
000000*000
3333311333
5555511555
2323234666
5555578887
5555578987
5555578887
5555577777
```

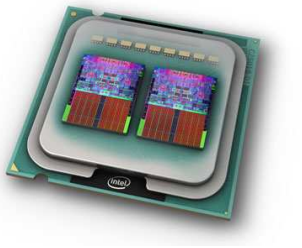
Sample Output:

```
Map #1:
     Cara can hike up to 1 acres.

Map #2:
     Cara can hike up to 12 acres.

Map #3:
     Cara can hike up to 47 acres.
```

Problem D- N Tasks



You boasted about the operating system you built just last week. Most of it was normal, perhaps even substandard, but the task scheduler was awesome. Just awesome.

A job is broken up into N concurrent tasks and split up to run on two processors: M_A and M_B. The cost to complete task i on M_A is $C_A[i]$. There is a similar cost vector, $C_B[N]$, for running the tasks on M_B, but there is also a cost *matrix*, $C_{N \times N}$, for communication between tasks. There is only a communication cost if tasks i and j are on different machines.

Your task scheduler will compute the minimum cost to complete the N tasks, which is the sum of all costs to run on each processor, plus all the communication costs.

Input Specification:

The input consists of an integer, $T \leq 100$, the number of cases, followed by T test cases. The first line of each case will contain the positive integer $N \leq 100$, the number of tasks. Then come $N + 2$ lines of N integers each, which represent (in order):

- the cost vector C_A;
- the cost vector C_B; and
- the communication matrix C.

All integers fall in the range 0-10^6. The matrix C is symmetric with a diagonal of 0.

Output Specification:

For each test case you will output one line containing "`Case #X: Y`" where `X` is the number of the test case and `Y` is the minimum cost to complete the N tasks.

Sample Input:

```
2
2
1 10
10 1
0 1
1 0
3
1 1 1
10 10 10
0 100 100
100 0 100
100 100 0
```

Sample Output:

```
Case #1: 3
Case #2: 3
```

Problem E- Lucky Number

After the Olympic opening ceremonies, scheduled on August 8, 2008, I found out that it was a lucky number because it was 8/8/08. An 8 is lucky in Chinese culture, but for the western world, July 7, 2007 was totally sold out for weddings, it being the lucky 7/7/07.

But if you think you've missed the boat, you can always plan something for January 2, 2034. You can have your dinner toast at 6 minutes and 7 seconds after 5 o'clock, to give an unlikely: 1/2/34/5:06:07. Try and top that one!

So I got to thinking about the luckiest numbers in lieu of these special dates. Maybe it's possible to have a number that's close to being lucky. An anagram of a lucky number, perhaps. Proximity is the key to determining luckiness.

Input Specification:

The first line contains an integer T ($0 < T \leq 100$) denoting the number of test cases that follow. Each test is composed of two positive integers, n and k, where n is no more than 30 digits long, and k is a single "lucky" digit in the range 5-8. n will never begin with a "0".

Output Specification:

For each test case, output the anagram of n, that is closest in value to the integer with the same length as n, but whose digits are all k. (Again, your anagram should not begin with a "0".) If it's a tie, print out the largest one.

Sample Input:

```
3
256 5
8192 8
32086 7
```

Sample Output:

```
562
8912
80236
```

Problem F- Round Trip

For this problem, you are to govern a robot through a lattice with $m \times n$ intersections, picking up any gold you find along the way. You begin at point $(0, 0)$, and, by travelling along the grid lines, you have barely enough power to get to cell $(m-1, n-1)$ where you can recharge the robot with enough juice to barely get you back to $(0, 0)$ again.

So, you greedy driver: How much gold can you pick up in one round-trip?

Input Specification:

The first line of input gives the number of cases, $N \leq 101$. N test cases follow.

On the first line of each case will be positive integers $m, n \leq 20$. On the following m lines, there will be n single-digit integers, which form a matrix $A_{m \times n}$, the amounts of gold to be found in the lattice. $(0, 0)$ is the upper-left corner of the matrix.

Output Specification:

For each test case you will output one line containing "`Case #X: Y`" where `X` is the number of the test case and `Y` is the maximum amount of gold that can be collected in the trip.

Sample Input:

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

Sample Output:

```
Case #1: 1
Case #2: 10
```