Problem A: Pascal's Travels

Source file: `pascal.{c, cpp, java}`
Input file: `pascal.in`

An *n* x *n* game board is populated with integers, one nonnegative integer per square. The goal is to travel along any legitimate path from the upper left corner to the lower right corner of the board. The integer in any one square dictates how large a step away from that location must be. If the step size would advance travel off the game board, then a step in that particular direction is forbidden. All steps must be either to the right or toward the bottom. Note that a 0 is a dead end which prevents any further progress.

Consider the 4 x 4 board shown in Figure 1, where the solid circle identifies the start position and the dashed circle identifies the target. Figure 2 shows the three paths from the start to the target, with the irrelevant numbers in each removed.

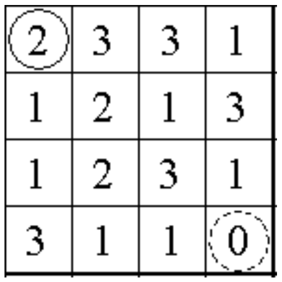

Figure 1

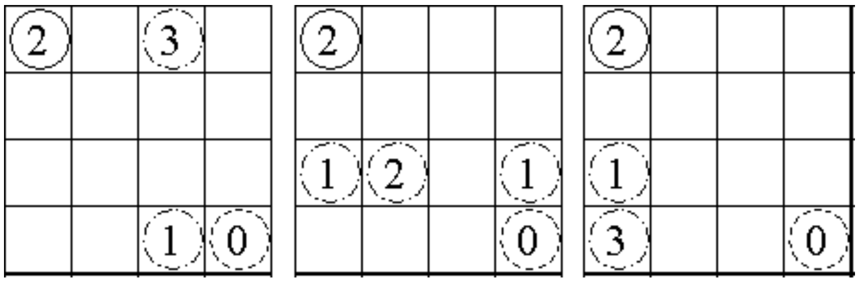

Figure 2

Input: The input contains data for one to thirty boards, followed by a final line containing only the integer -1. The data for a board starts with a line containing a single positive integer *n*, 4 ≤ *n* ≤ 34, which is the number of rows in this board. This is followed by *n* rows of data. Each row contains *n* single digits, 0-9, with no spaces between them.

Output: The output consists of one line for each board, containing a single integer, which is the number of paths from the upper left corner to the lower right corner. There will be fewer than 2^{63} paths for any board.

Warning: Brute force methods examining every path will likely exceed the allotted time limit. 64-bit integer values are available as *long* values in Java or *long long* values using the contest's C/C++ compilers.

Example input:	**Example output:**
`4`	`3`
`2331`	`0`
`1213`	`7`
`1231`	
`3110`	
`4`	
`3332`	
`1213`	
`1232`	
`2120`	
`5`	
`11101`	
`01111`	
`11111`	
`11101`	
`11101`	
`-1`	

Last modified on October 26, 2005 at 6:45 PM.

H: Robot Challenge

You have entered a robot in a Robot Challenge. A course is set up in a 100m by 100m space. Certain points are identified within the space as targets. They are ordered – there is a target 1, a target 2, etc. Your robot must start at (0,0). From there, it should go to target 1, stop for 1 second, go to target 2, stop for 1 second, and so on. It must finally end up at, and stop for a second on, (100,100).

Each target except (0,0) and (100,100) has a time penalty for missing it. So, if your robot went straight from target 1 to target 3, skipping target 2, it would incur target 2's penalty. Note that once it hits target 3, it cannot go back to target 2. It must hit the targets in order. Since your robot must stop for 1 second on each target point, it is not in danger of hitting a target accidentally too soon. For example, if target point 3 lies directly between target points 1 and 2, your robot can go straight from 1 to 2, right over 3, without stopping. Since it didn't stop, the judges will not mistakenly think that it hit target 3 too soon, so they won't assess target 2's penalty. Your final score is the amount of time (in seconds) your robot takes to reach (100,100), completing the course, plus all penalties. Smaller scores are better.

Your robot is very maneuverable, but a bit slow. It moves at 1 m/s, but can turn very quickly. During the 1 second it stops on a target point, it can easily turn to face the next target point. Thus, it can always move in a straight line between target points.

Because your robot is a bit slow, it might be advantageous to skip some targets, and incur their penalty, rather than actually maneuvering to them. Given a description of a course, determine your robot's best (lowest) possible score.

The Input

There will be several test cases. Each test case will begin with a line with one integer, **`N`** (**`1`** ≤ **`N`** ≤ **`1000`**) which is the number of targets on the course. Each of the next **`N`** lines will describe a target with three integers, **`X`**, **`Y`** and **`P`**, where (**`X,Y`**) is a location on the course (**`1`** ≤ **`X,Y`** ≤ **`99`**, **`X`** and **`Y`** in meters) and **`P`** is the penalty incurred if the robot misses that target (**`1`** ≤ **`P`** ≤ **`100`**). The targets will be given in order – the first line after **`N`** is target 1, the next is target 2, and so on. All the targets on a given course will be unique – there will be at most one target point at any location on the course. End of input will be marked by a line with a single `0`.

The Output

For each test case, output a single decimal number, indicating the smallest possible score for that course. Output this number rounded (NOT truncated) to three decimal places. Print each answer on its own line, and do not print any blank lines between answers.

Sample Input

```
1
50 50 20
3
30 30 90
60 60 80
10 90 100
3
30 30 90
60 60 80
10 90 10
0
```

Sample Output

```
143.421
237.716
154.421
```

Problem I: RIPOFF

Source file: `ripoff.{c, cpp, java}`
Input file: `ripoff.in`

Business has been slow at Gleamin' Lemon Used Auto Sales. In an effort to bring in new customers, management has created the Rebate Incentive Program Of Fabulous Fun (or RIPOFF). This is a simple game which allows customers to try and win a rebate on an automobile purchase. The RIPOFF game is a board game where each square is labeled with a rebate amount. The customer advances through the board by spinning a spinner. Each square he lands on adds to his total rebate amount. When he reaches the end of the board he is rewarded with the total rebate amount.

Of course, given the company involved, it should come as no surprise that there are a couple of catches written in the fine print. The first is that there is a limit to the number of turns the customer has to finish the game; if he doesn't reach the end within the allotted number of turns then he loses his rebate. The second is that some of the squares actually have a negative amount which subtract from the rebate instead of adding to it. A particularly unlucky customer might even come out of the game with a negative rebate.

Even with these catches, the management of Gleamin' Lemon is concerned that someone might win a particularly large rebate—something they would like to avoid at all costs. Your job is to take a particular configuration for the RIPOFF game and decide the maximum rebate a customer could possibly obtain.

Consider, for example, the game board below. Assume we have 5 turns to finish the game, and each turn we can move between 1 and 4 spaces depending on what we spin. Notice that we must start just before the board begins, so spinning a 1 causes us to land on the first square. Also notice we must end by landing past the end of the last square. It does not have to be exact; any number that gets us off of the board will work.

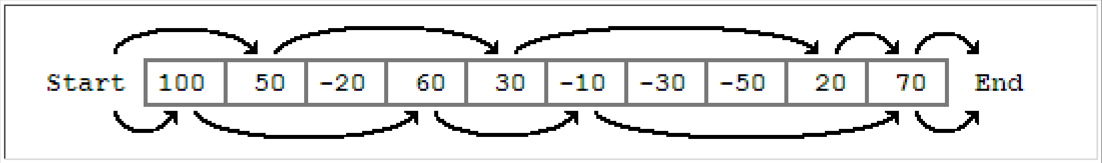

Figure 1

The illustration shows two different possible ways the game might go. Following the arrows on the top, if we spin a 2, 3, 4, 1, and 1 respectively, we will win a total rebate of 50 + 30 + 20 + 70 = $170. However, the best possible rebate we could win would be $220. We would win this amount if we spun a 1, 3, 2, 4, and 1 respectively, as shown by the lower path. Notice that we did not land on every square with a positive number; if we had we wouldn't have been able to make it to the end of the board before the 5 turns was up.

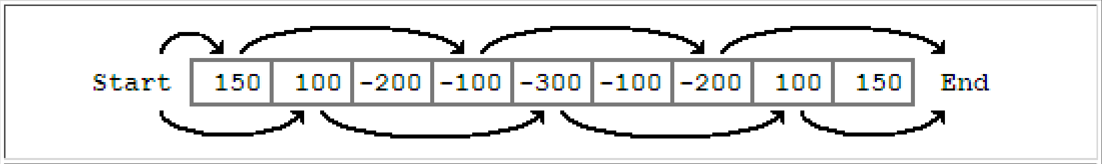

Figure 2

The illustration in Figure 2 shows a game where we have 4 turns to finish the game, and can move up to 3 spaces each turn. Again, two different paths are shown, the one on top earning a rebate of -\$150, and the one on bottom earning a rebate of -\$100. In fact, -\$100 is the highest possible rebate we could earn for this game (a fact that would no doubt please the management of Gleamin' Lemon). Of course, there also might be a sequence of moves in which we do not reach the end before the turn limit—e.g. spinning a 1 every time. Although not finishing would actually be preferable to finishing with a negative rebate, in this problem we are only going to consider sequences of moves which allow us to reach the end before the turn limit.

Input: The input consists of one to twenty data sets, followed by a line containing only 0.

The first line of a data set contains three space separated integers N S T, where

> N is the total number of squares on the board, $2 \le N \le 200$.
> S is the maximum number of spaces you may advance in each turn, $2 \le S \le 10$.
> T is the maximum number of turns allowed, where $N + 1 \le ST$ and $T \le N + 1$.

The data set ends with one or more lines containing a total of N integers, the numbers on the board. Each number has magnitude less than 10000.

Output: The output for each data set is one line containing only the maximum possible rebate that can be earned by completing the game.

To complete the game you must advance a total of $N + 1$ spaces in at most T turns, each turn advancing from 1 to S spaces inclusive. It will always be possible to complete a game. However, there may be a very large number of different turn sequences that will finish, so you will need to be careful in choosing your algorithm.

The sample input data corresponds to the games in the Figures.

Example input:	**Example output:**

`10 4 5`
`100 50 -20 60 30`
`-10 -30 -50 20 70`
`9 3 4`
`150 100 -200`
`-100 -300 -100`
`-200 100 150`
`0`
	`220`
	`-100`

Last modified on October 18, 2009 at 9:58 AM.