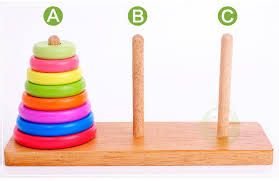
**Java program:** Prob15.java

**Input File:** Prob15.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

The Tower of Hanoi is a mathematical game or puzzle. It was invented in 1883 by French mathematician Edouard Lucas. It consists of three pegs labeled A, B, and C and a number of disks of different sizes which can slide onto any peg. The puzzle starts with the disks in a neat stack in ascending order of size on peg A, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack from peg A to peg C using peg B to help, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack; i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

With two disks, the puzzle can be solved in three moves, with three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2n - 1, where n is the number of disks.

This puzzle is well known since it touches two important topics.

* Recursive functions and stacks
* Recurrence relations

**Program Input**

The first line of the file Prob15.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* A single positive integer N denoting the number of disks in the initial stack

**Example Input:**

2

2

3

**Program Output**

Your program’s output should be as follows:

* The first line of each test case’s output should be the number of disks in the stack.
* The next 2n – 1 lines should be the disk movements in the form FromPeg->ToPeg. Since only the top disk can be moved, it is not necessary to print out the disk number.

**Example Output:**

2

A->B

A->C

B->C

3

A->C

A->B

C->B

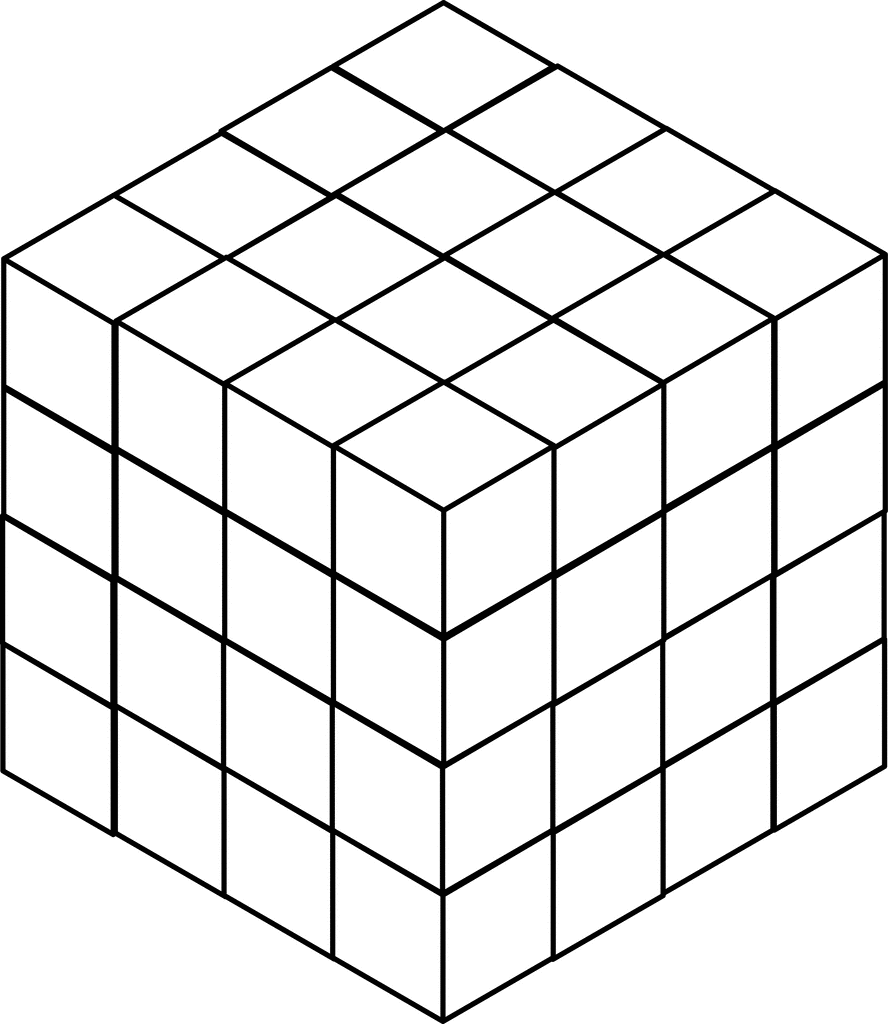
A->C

B->A

B->C

A->C

**Java program:** Prob16.java

**Input File:** Prob16.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

An exciting new planet in another planetary system has been discovered by the Kepler telescope. Scientists want to launch a satellite to this new planet and have asked you to write a program to plot a course that will require the least amount of time to get there. They believe the satellite can escape our solar system leaving with the speed of over about 17Km/s and thus taking around 20,000 years to travel a light year. The satellite has a solar chargeable battery that can hold a maximum 20 units of energy to power its instruments and thrusters. The satellite consumes 1 unit of energy every light year it travels. The idea is to hop from star to star consuming energy until the destination is reached without draining the entire battery.

Here are the guidelines for your journey:

* In order to simplify your distance calculations, you break space up into a series of cubes, each with a volume of one cubic lightyear. This means that it will take one unit of energy to travel from the center of one cube to the center of any adjacent cube.
* Diagonal movements are not allowed. You have carefully managed the orientation of the space cubes such that movements along the x, y, and z axes will avoid collisions with stars and black holes.
* You start your journey in the origin cube (0, 0, 0). Your destination is in cube (N-1, N-1, N-1).
* The battery can only hold a maximum of 20 units of energy, and you start with a full battery.
* The battery can be briefly charged by flying through a cube that contains a sun. The amount of energy units gained is based on the star type in the table below. Remember, 20 units is a fully charged battery. Any excess solar energy is lost.

|  |  |
| --- | --- |
| Star Type | Solar Energy |
| M | 3 |
| K | 4 |
| G | 5 |
| F | 6 |
| A | 7 |
| B | 8 |
| O | 9 |

* If you approach a cube without a sun with the battery drained, then the mission fails. For example, if you have 1 energy unit left and then move to the cube below you and no star isthere, then the mission fails; however, if you move above you with a G-type star, then you’ll end up with 5 units of energy in that cube with the star.
* You can’t return to a star you already flew by as this will affect your momentum.

**Program Input**

The first line of the file Prob16.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will contain a positive integer L denoting the width, length, and depth of the star map you created in lightyears. For example, if L is 2, then space is broken into 8 cubes (2x2x2).
* The second line of each test case will contain a positive integer N denoting the number of stars available for you to harness on your journey.
* The next N lines will each contain the star-type and the x,y,z coordinates of the space cube containing that star all separated by commas.

**Example Input:**

2

10

4

M,1,2,1

G,3,4,1

M,1,8,3

F,2,2,3

12

12

M,2,10,7

B,8,1,2

B,1,7,5

A,9,2,9

B,3,9,2

A,7,2,5

A,3,7,5

O,2,6,5

O,6,8,4

M,1,7,2

B,10,3,3

B,7,9,3

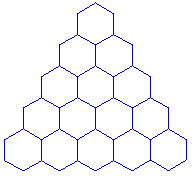
**Program Output**

Your program should display the shortest distance the satellite must travel to reach the destination in lightyears. From the example above, in one possible solution you travel to M(1,2,1) and lose 4 energy units, gain 3 energy units because it’s a M-type star, travel to F(2,2,3) and lose 3 energy units, gain 6 (only 4 are usable to fill your tank though), and then lose 20 to travel to the destination at (9,9,9). The total travel distance is 4+3+20=27 lightyears.

**Example Output:**

27

33

**Java program:** Prob17.java

**Input File:** Prob17.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

A honeycomb is an efficient way to visualize a set of interconnected spaces. In a honeycomb, each space is connected to at least two other spaces and at most six other spaces. Four of these possible connections are diagonal: up left, up right, down left, and down right. The other two are simply left and right.

In this problem, you are a honey bee. You are currently located at the top of the honeycomb triangle. You are supposed to leave soon to go gather pollen, but you left your pollen brush in the bottom right corner of the triangle. And you just remembered that your pollen bag is in the bottom left corner! You need to collect your things, and you need to do it quickly so you don’t get left behind. Since the door to the bee hive is just above the top of the honeycomb, you need to find the fastest route to go from the top of the honeycomb to the bottom right, then to the bottom left, and back to the top again.

Oh, and since the bees have been hard at work, there is honey in your way. Each section of the honeycomb has a different amount of honey in it, so each space will take a different amount of time to get through.

**Program Input**

The first line of the file Prob17.in.txt will contain a positive integer T denoting the number of test cases that follow. Each test case will have the following input:

* The first line of each test case will contain a positive integer N denoting the number of rows in the honeycomb. Each honeycomb will have at least 3 rows.
* The next N lines of input will contain a comma separated list of positive integers denoting the time it takes to get through each space in the honeycomb. The first line will have a single integer, the second line will have 2, and so on.

**Example Input:**

2

5

5

1,3

1,2,4

2,4,3,1

6,8,3,7,9

3

1

1,10

1,1,1

**Program Output**

For each test case, your program should print out the minimum time required to gather your supplies in order and get back to the top of the honeycomb.

**Example Output:**

40

7

In the first test case above, you start at the top of the honeycomb. You move down left (1), down right (2), down right (3), right (1), and down right(9). You are now in the bottom right and have spent 16 time units getting there. Now you move up left (1), left (3), up left(2), left(1), down left(2), and down left(6). Getting to the bottom left cost you 15 more time units. Finally, you move up right (2), up right (1), up right (1), and up right (5). Getting to the top cost you 9 more time units for a grand total of 40 time units spent on the trip.