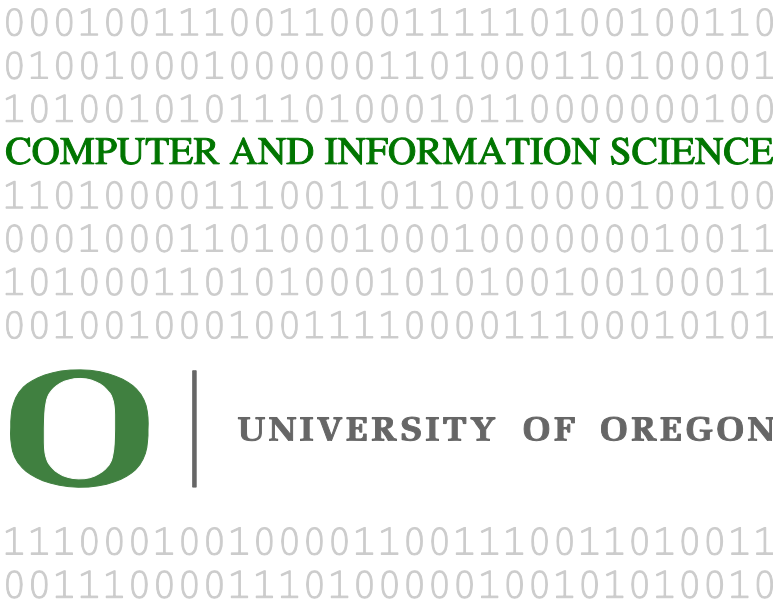
Fourth Annual Juilfs Contest

June 2, 2012



A. **Phoning it in** 7 points

You and another inhabitant of your cubicle are plotting to take a small slice off every transaction of your employer's software processes. To hatch your scheme, you need to communicate via email, but you don't want the IT guys to find out what you're up to. Your partner in crime has developed a special system to encode your messages in normal business chatter called Telephone Steganography.

The messages are hidden in the spaces between words in otherwise normal emails. Unlike normal steganography, which can have really large groups of spaces, you represent a character with two groups of spaces. The length of the first space indicates which key on a telephone keyboard contains the letter you want. The length of the second space indicates which letter from that key you want to write.

1	2	3	4	5	6	7	8	9
[space]	ABC	DEF	GHI	JKL	MNO	PQRS	TUV	WXYZ

For example, the email (spaces have been replaced with ␣ for clarity)

send␣␣␣␣␣me␣a␣␣␣␣new␣␣␣stapler␣␣␣␣␣␣soon␣please"

would translate to the message "JIM"

The input will consist of-a number $0 < n < 1000$ representing the number of test cases, followed by n lines containing strings of text with messages encoded using telephone steganography. The message is guaranteed to begin and end with a word (not a space), so you can safely strip whitespace from the beginning and end of the line.

Your output should consist of n lines containing the encoded message in all capitals.

Sample Data

Input
3 F␣␣␣␣Y␣I:␣␣␣␣␣␣all␣␣␣pets␣are␣␣␣␣␣␣to␣␣␣be␣␣␣␣␣␣␣kept␣␣␣on␣␣␣␣␣␣␣␣␣␣a␣leash! You␣␣␣␣␣␣␣␣can't␣␣␣␣eat␣␣␣␣␣␣␣␣␣in␣the␣␣␣␣␣␣␣2nd␣␣␣␣floor␣␣␣␣␣␣␣␣men's␣room. Please␣␣␣␣␣␣␣have␣␣your␣␣␣␣␣␣␣code␣␣␣submitted␣␣␣␣␣␣␣by␣␣␣5pm␣␣␣␣␣␣␣or␣␣else!!!
Output
GO NOW STOP NOON

B. **Are you smarter than a 5th grader?** 7 points

Mrs. Anderson was teaching her fifth grade class about binary numbers, when disaster struck: she lost the solution manual. So, she wants you to write a program that will output correctly formatted binary multiplication. Binary multiplication consists of only ones and zeroes and it's easy to multiply by a one or a zero. In the algorithm that she taught the students, the first number is the **multiplicand** and the second number is the **multiplier**. Starting on the right, for each 1 in the multiplier, the student writes the multiplicand on the next available line, right-justified to that multiplier digit. For each 0 in the multiplier, the student writes 0 on the next available line, right-justified to that multiplier digit. After all digits in the multiplier are processed, the student sums the products from right to left. The steps are shown in the sequence from the textbook shown below.

1
```
  1010 ← multiplicand
× 101 ← multiplier
```
2
```
  1010
× 101 ← one present in multiplier
  1010← multiplicand right-justified under 1 in multiplier
```
3
```
  1010
× 101 ← zero present in multiplier
  1010
    0 ← 0 under 0 in multiplier
```
4
```
  1010
× 101 ← one present in multiplier
  1010
    0
1010  ← multiplicand right-justified under 1 in multiplier
```
5
```
  1010
× 101
  1010
    0
1010
110010 ← sum of partial products, final answer
```

The input will begin with a line with a single number $1 < n < 1000$, followed by n lines representing n multiplication problems. Each problem line will consist of a positive binary number less than 65 bits, the letter "x" and a second binary number less than 65 bits. The first binary number is the multiplicand and the second binary number is the multiplier. Your output must be correctly lined up and have all the required symbols. For the multiplication symbol, use the lowercase letter x. For underlining, use dashes on a line by themselves. The output should line up exactly like the samples below. The multiplication symbol should be left-most on the line with the multiplier; then numbers should line up by digit position, and the dashes should exactly match the width of the problem. Place a blank line after each answer.

Sample Data

Input	Output
4 0x0 100x100 10x11 1001x1	0 X0 -- 0 -- 0 100 X 100 ----- 0 0 100 ----- 10000 10 X11 --- 10 10 --- 110 1001 X 1 ---- 1001 ---- 1001

C. **Why Can't We All Just Get Along?** 10 points

Ontotech has moved into a new building following the fire. Unfortunately, the building manager didn't order enough cubicle parts, so programmers will need to share. To further make life more difficult, most of the people in the company have several enemies. Since the company doesn't have a list of dislikes, the manager, Mr. Sterling, has decided to test new cubicle arrangements until nobody complains. He also wants to make sure that every cubicle has at least one programmer in it.

Several programmers have warned him that this will take a very long time to resolve if they test every distribution of the programmers into the cubicles, leaving none of them empty.

The input will consist of a line containing an integer $0 < n < 50$ that specifies the number of problems, followed by n lines with numbers $p\ k\ t\ l$ where $0 < k < 50$ is the number of cubicles, $k \leq p < 1000$ is the number of programmers, $0 < t < 60$ is the amount of time in minutes it takes to rearrange all the programmers, and $0 < l < 52416000$ is the maximum amount of time in minutes the bosses are willing to let the programmers rearrange themselves.

The output will consist of n lines each containing "Too Long" if the rearrangements will take more than the allotted time, or "Acceptable" otherwise.

Sample Data

Input	Output
3 4 2 1 1000 20 16 2 1000 20 16 2 22500000	Acceptable Too Long Too Long

D. **BART and other Calamities** 12 points

You are planning to visit San Francisco with your significant other. Since you do not want to look foolish, you do a bit of homework on how to use mass transit to move around the city. To your amazement, you find that San Francisco is served by four independent mass transit authorities: CalTrain, BART, city busses and streetcars. Not all of them go to each point, and they all demand different fares. You want to plan your itinerary using the least amount of money. So, given a list of sights, you want to figure out the least amount of money you can use to visit all sights in order. You can transfer from one mode of transportation to another at will, passing off the transfers as stopping to see some local attraction.

As input, you will receive four fare tables and a list of itineraries. Each fare table will begin with a line containing a single integer $0 < n \leq 1000$, followed by n lines beginning with the starting point and containing a space separated tuple of the destination point and the fare. All fare amounts are in cents. Each point has at most four nearby points. The fare is always the same going either way, so only one entry will be present for each set of points. After the four tables are listed, there will be a line with a single number on it $0 < m \leq 1000$. This will be followed by m lines showing sets of sites that you want to visit in order. For each itinerary, you should output the total cost for the cheapest route using any or all of the mass transit authorities.

Sample Data

Input

```
4
Mission (GlenPark 125)(CivicCenter 125) (Castro 100)(ChinaTown 200)
CivicCenter (Embarcadero 130)
Embarcadero (WestOakland 150)
GlenPark (BalboaPark 75)
3
Castro (GlenPark 40)(GoldenGatePark 120)
ChinaTown (CivicCenter 30)(Mission 110)
GoldenGatePark(Embarcadero 120)
1
BalboaPark (Mission 30)
3
Castro (GoldenGatePark 260)
ChinaTown (Castro 300)
GoldenGatePark (ChinaTown 270)
3
GlenPark WestOakland Embarcadero
BalboaPark Castro ChinaTown GoldenGatePark
CivicCenter Mission Embarcadero
```

Output

```
580
595
380
```

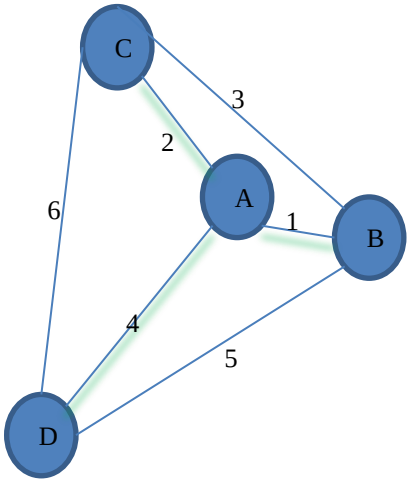
Recall that the algorithm for computing total weight from starting vertex to destination vertex, given the weight matrix is as follows:

input(*starting vertex*, *destination vertex*, *weight*)
$\forall$ *x*, *tw*(*x*) $\leftarrow \infty$
tw(*starting vertex*) $\leftarrow$ 0
finished vertices $\leftarrow \emptyset$
current vertex $\leftarrow$ *starting vertex*
while *destination vertex* $\notin$ *finished vertices*
    for each *neighbor* of *current vertex*
        if *tw*(*current vertex*) + *weight*(*current vertex*, *neighbor*) < *tw*(*neighbor*)
        then *tw*(*neighbor*) $\leftarrow$ *tw*(*current vertex*) + *weight*(*current vertex*, *neighbor*)
    finished vertices $\leftarrow$ *finished vertices* $\cup$ {*current vertex*}
    current vertex $\leftarrow$ vertex with shortest *tw* value that is not in finished vertices
output *tw*(*destination vertex*)

E. Same Old Same Old

12 points

At the Juilfs contest planning meeting, Jim said, "I have put the minimum spanning tree problem in the contest every year, and I will keep doing that until somebody solves it!" So, your job is to make sure this problem doesn't show up next year.



As input, you will receive a graph with weighted edges. You are supposed to find a minimum spanning tree for the graph. Recall that a spanning tree is a subgraph with no cycles that allows you to get from any point to any other point. A minimum spanning tree is a spanning tree with total edge weights less than or equal to any other spanning tree for that graph.

The general algorithm is to consider the edges one by one. If the edge belongs in the MST, then keep it. If the edge does not belong in the MST, then throw it out. The order that you should consider the edges in, and the way you determine if it is needed varies according to the three methods you learned in Discrete Math.

For a complete graph (a graph with edges between every pair of vertices), it is only necessary that you have the lower triangular distance matrix. First, we give the number of vertices, followed by the lower triangular weight matrix. For example, the input on the left represents the weight matrix on the right:

```
4
1
2 3
4 5 6
```

	A	B	C	D
A	0	1	2	4
B	1	0	3	5
C	2	3	0	6
D	4	5	6	0

In this case, the MST connects vertices B, C and D through vertex A, so the total distance is 7.

The input will consist of a single number $0 < n < 100$ indicating the number of problems. Each problem begins with a line containing a single number $0 < m < 1000$, indicating the number of vertices. This is followed by m–1 lines of data. For each problem, you should output a single number on a line by itself indicating the weight of the MST.

Sample Data

Input	Output
3 4 2 1 3 3 1 2 5 2 2 2 2 1 2 2 2 1 1 6 10 1 5 5 1 4 2 5 1 4 5 2 4 1 3	4 5 7