Rules

1. There are **six problems** to be completed in **four hours**.
2. All questions require you to read the test data from standard input[1] and write the results to standard output[2].
3. No whitespace should appear in the output except between printed fields.
4. All whitespace, either in the input or output, will consist of **exactly one** consecutive character.
5. The allowed programming languages are C, C++, Python 3, and Java.
6. All programs will be re-compiled prior to testing with the judges' data.
7. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages deemed dangerous by contest officials (e.g., that might generate a security violation).
8. The input to all problems will consist of multiple test cases.
9. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
10. All communication with the judges will be handled by the PC^2 environment.
11. Judges' decisions are final. No cheating will be tolerated.

[1] For example, in C++ you can use `cin` or `scanf`, and in Python you can use `input()`.
[2] For example, in C++ you can use `cout` or `printf`, and in Python you can use `print()`.

A 312-Avoiding Permutations

Consider a permutation of the numbers 1 through n represented by

$$\pi_1, \pi_2, \ldots, \pi_n$$

where each π_i is one number in the permutation. Such a permutation is called "312-avoiding" if it contains no 312 configurations. A 312 configuration is a subsequence of three indices i, j, and k such that $i < j < k$ and $\pi_j < \pi_k < \pi_i$. That is, it is a subsequence of three numbers in the permutation where the first number is the largest, the second number is the smallest, and the third number is in between the two in size.

For example, consider the permutation

$$3, 2, 7, 4, 5, 6, 1$$

This is *not* 312-avoiding because the subsequence 7-4-5 is an instance of a 312 configuration, as are the subsequences 7-4-6 and 7-5-6. But the permuation of

$$4, 3, 2, 1$$

is 312-avoiding, as there is no subsequence of three numbers such that the largest number comes first, the smallest number second, and the middle number third.

Input

Each problem instance is given by two lines. The first line consists of an integer n indicating the length of the sequence. The second line consists of a permutation of the numbers 1 through n separated by spaces. The length of each permuation will be no more than 30. A value of $n = 0$ indicates the end of the problem instances.

Output

For the i^{th} problem instance (starting from 1), your program should print a single line of text consisting of "Permutation i:" followed by either "yes" or "no". It should print "yes" if the the i^{th} problem's permutation is 312-avoiding, and "no" otherwise.

Sample Input

```
4
1 2 3 4
4
4 3 2 1
4
4 2 1 3
7
3 2 7 4 5 6 1
0
```

Sample Output

```
Permutation 1: yes
Permutation 2: yes
Permutation 3: no
Permutation 4: no
```

B BOGO50%

Jalani is part of the special rewards program at his local grocery store and has been for a long time. He goes shopping there so consistently and spends so much that the store has now offered him a special one-time promotion: for each item he buys, he can get another item of equal or lesser cost for half the price. In the case that half the price would not have a whole number of cents, the price is rounded up to the nearest cent. Jalani can tell that this promotion has a lot of potential, but he wants to make sure that he is using it optimally.

For example, suppose Jalani just wanted to buy one pack of gum for \$0.50. The promotion would not save Jalani any money in this case, as there are no other items on his shopping list to get for half price.

However, suppose Jalani's shopping list consists of 3 peppers for \$2.00 each, 4 onions for \$1.00 each, and a pack of tofu for \$5.00. Jalani can buy the tofu (\$5.00) to get one pepper at half price (\$1.00), then buy a pepper at full price (\$2.00) to get another pepper at half price (\$1.00), and finally buy two onions at full price (\$1.00 $\times$ 2) to get two onions at half price (\$0.50 $\times$ 2). This way Jalani pays a total of \$12.00 instead of the full price of \$15.00, saving \$3.00 with the promotion.

Finally, suppose Jalani's shopping list consists of just 10 cans of beans for \$1.95 each. Jalani can buy 5 cans at full price (\$1.95 $\times$ 5) to get the other 5 cans at half price (\$0.975 rounded up to \$0.98 $\times$ 5). In this case, the promotion lets Jalani pays a total of \$14.65 instead of the full price of \$19.50, saving \$4.85.

Jalani has begun to plan for using this promotion optimally by making lists of the items he wants to buy each time he goes to the grocery store for the next few weeks. He just needs to calculate how much money the promotion could potentially save him each time he shops to see which trips will benefit most from the promotion.

Input

Each problem instances is given by multiple lines representing Jalani's shopping list for a single trip to the grocery store. The first line of a problem instance consists of an integer n between 0 and 100, inclusive, indicating the number of kinds of items on Jalani's shopping list. The next n lines each consist of information on the different kinds items on Jalani's shopping list. Each of these lines begins with an integer representing the number of units of that item that Jalani wants to buy, followed by a space, then a (unique) description of the item (which may contain spaces), followed by another space, and ending with a decimal number representing the price per unit. The number of units is always between 1 and 10, inclusive. The price always has exactly 2 decimal places, at least one digit before the decimal point, and a value between 0.01 and 100.00, inclusive.

A value of $n = 0$ indicates the end of the problem instances.

Output

For the i^{th} problem instance (starting from 1), your program should print a single line of text consisting of "Savings i:" followed by a single number indicating the total amount of money Jalani saves by using the promotion optimally on that trip. This number should be a decimal value with exactly two decimal places, even if the savings is a whole number of dollars. This number should also have at least one digit before the decimal point, in particular a 0 if the savings is less than one dollar. Otherwise, this number should not contain any leading zeros.

Sample Input

```
1
1 Pack of Gum 0.50
3
3 Peppers 2.00
4 Onions 1.00
1 Pack of Tofu 5.00
1
10 Cans of Beans 1.95
4
1 Baguette 3.00
1 Bottle of Mayonnaise 4.00
2 Boxes of Cereal 5.00
1 Gallon of Milk 3.50
0
```

Sample Output

```
Savings 1: 0.00
Savings 2: 3.00
Savings 3: 4.85
Savings 4: 4.25
```

C Bonus Pay

Inyoung works as a contractor, and her main client, the company Comp & Knee, pays out a daily commission for the work of each day. As a contractor, she has the freedom to choose which days she works, which means that she can choose to not work for Comp & Knee on days that wouldn't be profitable enough for her.

To incentivize consistent work, Comp & Knee's contracts pay out a bonus for working on consecutive business days. Specifically, on the second consecutive day of work in a row, Inyoung would recieve a bonus of $300, and on the third consecutive day of work, she would recieve a bonus of $100. These incentives reset every time Inyoung takes a day off.

What Comp & Knee don't realize is that this incentive system actually can incentivize Inyoung to skip work on certain days to re-earn the bonuses. For example, consider the following schedule of commissions for five business days, starting at $250 on the first day and ending at $300 on the last. Days where Inyoung might work for Comp & Knee are marked with a checkmark and days where she might not are marked with an X. Days earning bonus payouts are marked with superscripts. If Inyoung works all 5 days for Comp & Knee, she earns the bonuses for days 2 and 3 for a total of $1500. However, if she skips day 3, she can earn an additional second-day bonus from Comp & Knee on day 5 for a total of $1550. Thus, she makes more money by taking a day off!

$250	$200	$150	$200	$300	total
✓	✓$^{+300}$	✓$^{+100}$	✓	✓	$1500
✓	✓$^{+300}$	✗	✓	✓$^{+300}$	$1550

Inyoung has made a schedule of the upcoming commissions she expects are possible from Comp & Knee, and she now wants to plan her work schedule so as to maximize her earnings.

Input

Each problem instance is given by two lines. The first line consists of an integer n between 0 and 100, inclusive, indicating the number of business days in the schedule. The second line consists of n non-negative integers separated by spaces, indicating the expected commission for each day from Comp & Knee in order. Each commission will be no more than 1000 dollars, and no cents will be considered, so every commission is an integer. A value of $n = 0$ indicates the end of the problem instances.

Output

For the i^{th} problem instance (starting from 1), your program should print a single line of text consisting of "Schedule i:" followed by a single integer indicating the maximum amount of money Inyoung can earn by optimally choosing which days to work for Comp & Knee.

Sample Input

```
5
250 200 150 200 300
7
150 250 350 450 550 650 750
9
250 350 200 300 300 250 400 200 350
0
```

Sample Output

```
Schedule 1: 1550
Schedule 2: 3550
Schedule 3: 3150
```

D Piece of Cake

Juhi runs a bakery specializing in fancy tiered cakes like wedding cakes. She offers small, medium, and large sizes for each tier of her cakes and can stack them as tall as the customer wants—she is very good at balancing cakes!

She recently set up a website where customers can customize their cakes by choosing the number of tiers and the size of each tier. The orders have been rolling in every day, and Juhi needs to ensure she has enough ingredients to fulfill them all. While she had been working out her ingredients by hand, Juhi realizes that with so many orders, it might be faster to write a program to calculate the ingredients she needs. Then her website could just tell her automatically!

Juhi has decided to start by calculating how many eggs she needs. A small tier requires 2 eggs, a medium tier requires 5 eggs, and a large tier requires 10 eggs. Thus, for example, a 3-tier cake with one of each size tier would require $2 + 5 + 10 = 17$ eggs.

Input

Each problem instance is given by multiple lines representing the orders Juhi has received for the day. The first line consists of an integer n between 0 and 100, inclusive, representing the number of cake orders. Each of the next n lines consists of a single cake order given by three integers between 0 and 10, inclusive, separated by spaces. The first integer is the number of small tiers in the order, the second is the number of medium tiers, and the third is the number of large tiers.

A value of $n = 0$ indicates the end of the problem instances.

Output

For the i^{th} problem instance (starting from 1), your program should print a single line of text consisting of "Day i:" followed by a single integer indicating the total number of eggs Juhi needs to fulfill all of the cake orders for that day.

Sample Input

```
3
1 1 1
0 2 0
2 0 3
4
0 0 3
1 2 1
1 0 0
0 1 2
5
1 0 10
4 1 0
0 3 3
3 3 3
0 4 0
0
```

Sample Output

```
Day 1: 61
Day 2: 79
Day 3: 231
```

E Plow Away

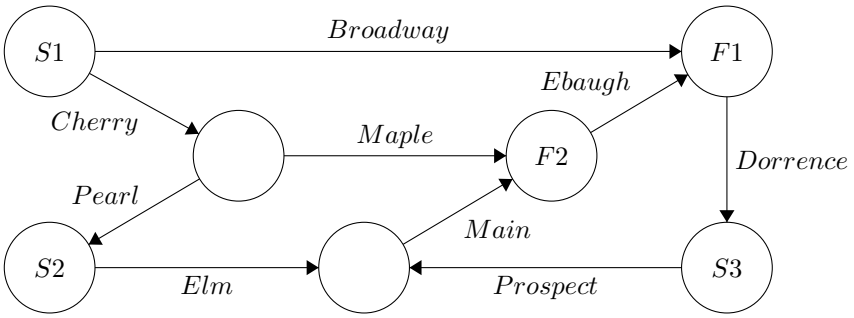
In January 2026, Winter Storm Fern brought in record-breaking amounts of snow to various regions across the United States, which highlighted that many of these affected regions were not adequately prepared to handle that much snow on the ground.

With the storm past us, Franklin County (where Columbus, OH is located) is being proactive and is looking at better ways of plowing its roads. They find that the current process is disorganized, with plows often working on the same road multiple times, and so they believe the process could use a refresh. However, the county has a limited budget for drivers and has a limited number of plows. They are thus open to finding algorithmic means to get the most use out of what they have. In particular, they would like to ensure that all a given region's plows can be used efficiently, without any plow having to re-plow any road. If this is not possible, it suggests that the region might be able to hire fewer drivers or reallocate plows to other regions.

Each region of the county consists of intersections joined together by one-way roads. To focus their plowing efforts, the county has identified important intersections for the starting and finishing points of traffic flow in each region, such as a starting intersection by a residential neighborhood and an ending intersection by a grocery store. The efficacy of plowing efforts will be measured only by how efficiently the plows can use their time to plow the roads connecting these starting and finishing intersections. It is not critical that all these roads get plowed, but rather that every plow spends all its time plowing these roads.

Ideally, the plows would attain maximum efficiency with their time by following the *No-Double-Plowing* rule: never send more than one plow down a given road. This rule allows plows to share starting intersections, finishing intersections, and any intersections in between, as long as they do not share any roads. By following this rule, every plow continuously plows snow all the way from its starting intersection to its finishing intersection without traversing an already-plowed road. A region with k plows is said to be *snow-good* if all k plows can be positioned at designated starting intersections for that region and make their way to designated finishing intersections for that same region without violating the No-Double-Plowing rule. Regions that are not snow-good are called *snow-bad*. Snow-good regions are achieving maximum efficiency with their plows, while snow-bad regions are not.

For example consider the map of the following region where the circles represent intersections and the arrows represent one-way roads. If the designated starting intersections are $S1$, $S2$, and $S3$, and the designated finishing intersections are $F1$ and $F2$, then this region is snow-good with 3 plows: one plow can go from $S1$ to $F1$ via Broadway, another can go from $S1$ to $F2$ via Cherry and Maple, and the last plow can go from $S2$ to $F2$ via Elm and Main. However, this region would be snow-bad with 4 plows, since the fourth plow would have to share a road with one of the existing plows no matter how the plows were arranged.



Input

Each problem instance is given by multiple lines representing a region to plow. The first line consists of five integers n, m, s, f, and k, in that order, each separated by a single space and falling between 0 and 100, inclusive. The number of plows in the region is k. The intersections in the region are uniquely labeled with the integers in $[n]$.[3]

The next m lines each contain two integers (separated by a single space) in $[n]$ indicating a one-way road from the first intersection given by the first integer to the intersection given by the second integer. No more

[3] $[n]$ is a shorthand for the set $\{1, \ldots, n\}$.

than one road connects any two intersections in the same direction, and no road connects an intersection to itself.

The next s lines each have one positive integer in $[n]$ indicating the starting intersections for the region.

The next f lines each have one positive integer in $[n]$ indicating the finishing intersections for the region. No finishing region is also a starting region.

A line with the value 0 for each of n, m, s, f, and k indicates the end of the input. No other lines contain any zeros.

Output

For the i^{th} problem instance (starting from 1), your program should print a single line of text consisting of "Region i:" followed by either "snow-good" or "snow-bad". It should print "snow-good" if the region is snow-good, and "snow-bad" otherwise.

Sample Input:

```
6 5 3 3 2
1 4
2 4
2 5
3 5
3 6
1
2
3
4
5
6
5 6 2 1 2
1 2
2 3
1 4
4 3
3 5
5 4
2
1
5
5 5 1 1 2
1 2
1 3
2 4
3 4
4 5
1
4
0 0 0 0 0
```

Sample Output:

```
Region 1: snow-good
Region 2: snow-bad
Region 3: snow-good
```

F Quick Succession

The country of Oldlostland's parliament is made up of an upper house and a lower house. The upper house is made up of $m > 0$ members, and the lower house is made up of $n > 1$ members.

Recently, the country of Oldlostland's prime minister has resigned from the position following a scandal. In fact, this was the latest in a string of short-serving prime ministers. It seems that between resignations and no-confidence votes, they can barely keep a prime minister around longer than a month!

As has been tradition, the parliament's upper house selects one of the n members of the lower house to serve as prime minister. This selection is made by consensus, which means everyone in the upper house must agree on the selected lower house member. Each time the position of prime minister is vacant, another member of the lower house must be selected, and finding consensus can take a lot of time and debate. Unfortunately, all the meetings to select the recent prime ministers are rapidly eating into the legislative schedule, preventing the upper house from getting anything else done.

Some political officials have noticed that the preferences of members of the upper house do not really have time to change between each selection, and so they have suggested a new procedure to make prime minster selection run smoothly, at least for the near future: Rather than select a single prime minister each time, instead rank all the members of the lower house once, and select the new prime minister by choosing the highest-ranked lower house member who has not yet been prime minister. That way, if the next prime minister is quickly ousted again, the preferences can be reused to quickly determine the next prime minister. However, setting this up in a way that keeps all the members of upper house happy enough for consensus is tricky and not always possible!

By some miracle, the members of the upper house have very simple preferences. Each such member has a preference represented by a pair (a, b), where both a and b are the governmental ID numbers of members of the lower house. The preference (a, b) is said to be *satisfied* if the lower house member with ID number a is ranked above the lower house member with ID number b in the final ranking.

Your goal is to compute a ranking of all lower house members while satisfying all preferences, if possible. If there are multiple solutions, you should output the lexicographically smallest one. That is, you should output the ranking that comes first in alphanumeric ordering. For example, if the upper house has only preferences $(1, 2)$ and $(1, 3)$, then the ranking 1 2 3 satisfies both preferences, as does the ranking 1 3 2. However, the lexicographically smaller of these two rankings is 1 2 3, since the rankings are the same up until the second position, where 2 is smaller than 3.

Input:

Each problem instance is given by multiple lines representing a collection of preferences. The first line consists of two integers m and n separated by a space and falling between 0 and 100, inclusive. The size of the upper house is m, the size of the lower house is $n \geq 2$. The next m lines each contain two integers a and b separated by a space, representing the preference (a, b), where a and b are distinct governmental ID numbers. Each of the n elected officials has a unique governmental ID number between 1 and n, inclusive.

A line with a value of 0 for m indicates the end of the input. No other lines contain any zeros.

Output:

For the i^{th} problem instance (starting from 1), your program should print a single line of text consisting of "Ranking i:" followed by either a sequence of space-separated governmental ID numbers or "none". If no ranking of candidates satisfies all preferences, then "none" should be printed. Otherwise the sequence of governmental ID numbers should correspond to the lexicographically smallest ranking that satisfies every member of the upper house, where the highest ranked member of the lower house is put first.

Sample Input:

```
5 5
1 2
1 4
4 3
2 4
3 5
5 4
1 2
3 1
3 2
2 4
4 2
3 4
3 2
4 3
4 3
2 3
1 2
1 3
0 2
```

Sample Output:

```
Ranking 1: 1 2 4 3 5
Ranking 2: none
Ranking 3: 1 4 3 2
Ranking 4: 1 2 3
```