

1 Problem: Infer Energy Values of Ingredients

Problem Introduction

In this problem, you will apply Gaussian Elimination to infer the energy values of ingredients given a restaurant menu with calorie counts and ingredient lists provided for each item.

Platter	Calories
Hot Cake	450
Hot Cake	600
Big Breakfast	580
Deluxe Breakfast	660
Breakfast	560

Problem Description

Task. You're looking into a restaurant menu which shows for each dish the list of ingredients with amounts and the estimated total energy value in calories. You would like to find out the energy values of individual ingredients (then you will be able to estimate the total energy values of your favorite dishes).

Input Format. The first line of the input contains an integer n — the number of dishes in the menu, and it happens so that the number of different ingredients is the same. Each of the next n lines contains description a_1, a_2, \dots, a_n, E of a single menu item. a_i is the amount of i -th ingredient in the dish, and E is the estimated total energy value of the dish. If the ingredient is not used in the dish, the amount will be specified as $a_i = 0$; **beware that although the amount of any ingredient in any real menu would be positive, we will test that your algorithm works even for negative amounts $a_i < 0$.**

Constraints. $0 \leq n \leq 20$; $-1000 \leq a_i \leq 1000$.

Output Format. Output n real numbers — for each ingredient, what is its energy value. These numbers can be non-integer, so output them with at least 3 digits after the decimal point.

Your output for a particular test input will be accepted if all the numbers in the output are considered correct. The amounts and energy values are of course approximate, and the computations in real numbers on a computer are not always precise, so each of the numbers in your output will be considered correct if either **absolute or relative error** is less than 10^{-2} . That is, if the correct number is 5.245000, and you output 5.235001, your number will be considered correct, but 5.225500 will not be accepted. Also, if the correct number is 1001, and you output 1000, your answer will be considered correct, because the relative error will be less than 10^{-2} , but if the correct answer is 0.1, and you output 0.05, your answer will not be accepted, because in this case both the absolute error (0.05) and the relative error (0.5) are more than 10^{-2} . **Note that we ask you to output at least 3 digits after the decimal point, although we only require precision of 10^{-2} , intentionally: if you output only 2 digits after the decimal point, your answer can be rejected while being correct because of the rounding issues. The easiest way to avoid this mistake is to output at least 3 digits after the decimal point.**

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	1.5	5	1.5	2	5	5	3

Memory Limit. 512MB.

Sample 1.

Input:

```
0
```

Output:

There are no dishes in the menu — you don't need to output anything in this case.

Sample 2.

Input:

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

Output:

```
1.000000 5.000000 4.000000 3.000000
```

Explanation:

This is an easy test. Each dish contains just one component, and the amount used is exactly 1, so the energy value of each ingredient is just equal to the energy value of the whole dish in which it is used.

Sample 3.

Input:

```
2
1 1 3
2 3 7
```

Output:

```
2.000000 1.000000
```

Explanation:

You can see that the numbers match: $1 \cdot 2.0 + 1 \cdot 1.0 = 3$ and $2 \cdot 2.0 + 3 \cdot 1.0 = 7$. If you output 1.994000 and 1.009000 instead of 2.000000 and 1.000000 respectively, your answer will still be accepted, but don't forget to output at least 3 digits after the decimal point!

Sample 4.

Input:

```
2
5 -5 -1
-1 -2 -1
```

Output:

```
0.200000 0.400000
```

Explanation:

Beware that there will be tests with negative amounts and negative total energy values, although this is impossible in reality! Also note that the answers can be non-integer! You can check that the numbers match: $5 \cdot 0.2 + (-5) \cdot 0.4 = -1$ and $(-1) \cdot 0.2 + (-2) \cdot 0.4 = -1$.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a blank procedure and output the result. They also contain some convenience functions and data structures. You need to change the main procedure to implement Gaussian Elimination if you are using C++, Java, or Python3. For other programming languages, you need to implement a solution from scratch. Filename: `energy_values`

What To Do

Implement the Gaussian Elimination algorithm from the lectures.

Need Help?

Ask a question or see the questions asked by other learners at [this forum thread](#).

2 Problem: Optimal Diet Problem

Problem Introduction

In this problem, you will implement an algorithm for solving linear programming with only a few inequalities and apply it to determine the optimal diet.

ONE WEEK MEAL PLAN						
Day	Breakfast	Lunch	Dinner	Side Dish	Snack	Dessert
Mon	Banana Oat Smoothie with Honey	Beefy-Chicken Salad	Mediterranean Pasta	Steamed Potatoes	Pesto Pita with Hummus	Chocolate Zucchini
Tue	Spicy Fried Kebab and Brown Rice	Harvest Turkey Plate	Spicy Chicken	Oven Roasted Sweet Potato	Fried Salmon with Avocado	Apple Nachos
Wed	Broccoli Sprout and Lemon Detox	Lobster Lettuce	Beef in Your Mouth	Beet and Feta	Zucchini Salad with Tomato and Vinaigrette	Sausage and Cauliflower
Thu	Spicy and Tangy Broccoli Pie	Beef Lettuce Wrap	Beet Wrap	Carrot and Purple Sweet Potato	Orange Chicken	General Tso's Chicken
Fri	Breakfast and Brunch	Jambalaya	Butter Chicken with White Wine Gravy	Cauliflower Mashed Potato	Two Chees	Cheese Ravioli
Sat	Smoothies and Smoothie Bowls	Curried Chicken and Rice	Beef and Broccoli	Cold Beer and Brunch	Jerky	Guacamole and Salsa
Sun	Spiced Fish with Beets and Broccoli	Slow Cooker Pork Pot Roast	Beef Tacos with Guacamole	Oven-Roasted Vegetables	Jerky	Orange Pineapple Mousse

Problem Description

Task. You want to optimize your diet: that is, make sure that your diet satisfies all the recommendations of nutrition experts, but you also get maximum pleasure from your food and drinks. For each dish and drink you know all the nutrition facts, cost of one item, and an estimation of how much you like it. Your budget is limited, of course. The recommendations are of the form “total amount of calories consumed each day should be at least 1000” or “the amount of water you drink in liters should be at least twice the amount of food you eat in kilograms”, and so on. You optimize the total pleasure which is the sum of pleasure you get from consuming each particular dish or drink, and that is proportional to the amount $amount_i$ of that dish or drink consumed.

The budget restriction and the nutrition recommendations can be converted into a system of linear inequalities like $\sum_{i=1}^m cost_i \cdot amount_i \leq Budget$, $amount_i \geq 1000$ and $amount_i - 2 \cdot amount_j \geq 0$, where $amount_i$ is the amount of i -th dish or drink consumed, $cost_i$ is the cost of one item of i -th dish or drink, and $Budget$ is your total budget for the diet. Of course, you can only eat a non-negative amount $amount_i$ of i -th item, so $amount_i \geq 0$. The goal to maximize total pleasure is reduced to the linear objective $\sum_{i=1}^m amount_i \cdot pleasure_i \rightarrow \max$ where $pleasure_i$ is the pleasure you get after consuming one unit of i -th dish or drink (some dishes like fish oil you don't like at all, so $pleasure_i$ can be negative). Combined, all this is a linear programming problem which you need to solve now.

Input Format. The first line of the input contains integers n and m — the number of restrictions on your diet and the number of all available dishes and drinks respectively. The next $n + 1$ lines contain the coefficients of the linear inequalities in the standard form $Ax \leq b$, where $x = amount$ is the vector of length m with amounts of each ingredient, A is the $n \times m$ matrix with coefficients of inequalities and b is the vector with the right-hand side of each inequality. Specifically, i -th of the next n lines contains m integers $A_{i1}, A_{i2}, \dots, A_{im}$, and the next line after those n contains n integers b_1, b_2, \dots, b_n . These lines describe n inequalities of the form $A_{i1} \cdot amount_1 + A_{i2} \cdot amount_2 + \dots + A_{im} \cdot amount_m \leq b_i$. The last line of the input contains m integers — the pleasure for consuming one item of each dish and drink $pleasure_1, pleasure_2, \dots, pleasure_m$.

Constraints. $1 \leq n, m \leq 8$; $-100 \leq A_{ij} \leq 100$; $-1\ 000\ 000 \leq b_i \leq 1\ 000\ 000$; $-100 \leq cost_i \leq 100$.

Output Format. If there is no diet that satisfies all the restrictions, output “No solution” (without quotes).

If you can get as much pleasure as you want despite all the restrictions, output “Infinity” (without quotes). If the maximum possible total pleasure is bounded, output two lines. On the first line, output “Bounded solution” (without quotes). On the second line, output m real numbers — the optimal amounts for each dish and drink. Output all the numbers with at least 15 digits after the decimal point.

The amounts you output will be inserted into the inequalities, and all the inequalities will be checked. An inequality $L \leq R$ will be considered satisfied if actually $L \leq R + 10^{-3}$. The total pleasure of your solution will be calculated and compared with the optimal value. Your output will be accepted if all

the inequalities are satisfied and the total pleasure of your solution differs from the optimal value by at most 10^{-3} . We ask you to output at least 15 digits after the decimal point, although we will check the answer with precision of only 10^{-3} . This is because in the process of checking the inequalities we will multiply your answers with coefficients from the matrix A and with the coefficients of the vector pleasure , and those coefficients can be pretty large, and computations with real numbers on a computer are not always precise. This way, the more digits after the decimal point you output for each amount — the less likely it is that your answer will be rejected because of precision issues.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	2	30	1.5	2	30	30	4

Memory Limit. 512MB.

Sample 1.

Input:

```
3 2
-1 -1
1 0
0 1
-1 2 2
-1 2
```

Output:

```
Bounded solution
0.000000000000000 2.000000000000000
```

Explanation:

Here we have only two items, and we know that $(-1) \cdot \text{amount}_1 + (-1) \cdot \text{amount}_2 \leq -1 \Rightarrow \text{amount}_1 + \text{amount}_2 \geq 1$ from the first inequality, and also that $\text{amount}_1 \leq 2$ and $\text{amount}_2 \leq 2$ from the second and the third inequalities. We also know that all amounts are non-negative. We want to maximize $(-1) \cdot \text{amount}_1 + 2 \cdot \text{amount}_2$ under those restrictions — that is, we don't like dish or drink number 1, and we twice as much like dish or drink number 2. It is optimal then to consume as few as possible of the first item and as much as possible of the second item. It turns out that we can avoid consuming the first item at all and take the maximum possible amount 2 of the second item, and all the restrictions will be satisfied! Clearly, this is a diet with the maximum possible total pleasure! Note that integers 0 and 2 in the output are printed with 15 digits after the decimal point. Don't forget to print at least 15 digits after the decimal point, as the answers to some tests will be non-integer, and you don't want to get your answer rejected only because of some rounding problems.

Sample 2.

Input:

```
2 2
1 1
-1 -1
1 -2
1 1
```

Output:

```
No solution
```

Explanation:

The first inequality gives $amount_1 + amount_2 \leq 1$ and the second inequality gives $(-1) \cdot amount_1 + (-1) \cdot amount_2 \leq -2 \Rightarrow amount_1 + amount_2 \geq 2$. But $amount_1 + amount_2$ cannot be less than 1 and more than 2 simultaneously, so there is no solution in this case.

Sample 3.

Input:

```
1 3
0 0 1
3
1 1 1
```

Output:

```
Infinity
```

Explanation:

The restrictions in this case are only that all amounts are non-negative (these restrictions are always there, because you cannot consume negative amount of a dish or a drink) and that $amount_3 \leq 3$. There is no restriction on how much to consume of items 1 and 2, and each of them has positive *pleasure* value, so you can take as much of items 1 and 2 as you want and receive as much total pleasure as you want. In this case, you should output “Infinite” (without quotes).

Starter Files

The starter solutions for this problem read the data from the input, pass it to a blank procedure and output the result. You need to implement this procedure if you are using C++, Java, or Python3. For other programming languages, you need to implement a solution from scratch. Filename: diet

What To Do

There are at most 8 inequalities (16 if you count in the inequalities $amount_i \geq 0$) with at most 8 variables. You can use this fact and the fact that the optimal solution is always in a vertex of the polyhedron corresponding to the linear programming problem. At least m of the inequalities become equalities in each vertex of the polyhedron. If there are n regular inequalities, m variables and m inequalities of the form $amount_i \geq 0$, you need to take each possible subset of size m out of all the $n + m$ inequalities, solve the system of linear equations where each equation is one of the selected inequalities changed to equality, check whether this solution satisfies all the other inequalities, and in the end select the solution with the largest value of the total pleasure out of those which satisfy all inequalities. The running time of this algorithm is $O(2^{n+m}(m^3 + mn))$, which is good enough to pass. 2^{n+m} is to go through all the subsets of the inequalities (although you will need only subsets of size m), m^3 is for Gaussian Elimination and mn is to check a solution of a system of linear equations against all the inequalities. Various ways to traverse all the subsets of some set are described [here](#) and [here](#).

The only case that you would miss this way is the case when the correct answer is “Infinity”. It is guaranteed that in all test cases in this problem, if a solution is bounded, then

$$amount_1 + amount_2 + \dots + amount_m \leq 10^9.$$

Thus, to distinguish between bounded and unbounded cases, just add this inequality to the initial problem. If the resulting program has the last inequality among those m that define the vertex, output “Infinity”, otherwise output “Bounded solution” and the solution of the augmented problem on the second line.

Need Help?

Ask a question or see the questions asked by other learners at [this forum thread](#).