Computer Science Competition
State 2025
Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Aspen
Problem 2	Amelia
Problem 3	Annabella
Problem 4	Guillermo
Problem 5	Heidi
Problem 6	Holden
Problem 7	Liza
Problem 8	Marshall
Problem 9	Mitchell
Problem 10	Nate
Problem 11	Wang
Problem 12	Willson

1. Aspen

Program Name: Aspen.java **Input File: none**

Aspen's team is excited about representing their region here at the championships. Help Aspen show how excited they are by displaying the phrase `2025 UIL State Championships`! in the diagonal manner shown below.

Input: None

Output: Print the phrase "`2025 UIL State Championships!`" in the pattern shown below.

Sample input: `None`

Sample output:

```
2025 UIL State Championships!
!2025 UIL State Championships
s!2025 UIL State Championship
ps!2025 UIL State Championshi
ips!2025 UIL State Championsh
hips!2025 UIL State Champions
ships!2025 UIL State Champion
nships!2025 UIL State Champio
onships!2025 UIL State Champi
ionships!2025 UIL State Champ
pionships!2025 UIL State Cham
mpionships!2025 UIL State Cha
ampionships!2025 UIL State Ch
hampionships!2025 UIL State C
Championships!2025 UIL State
 Championships!2025 UIL State
e Championships!2025 UIL Stat
te Championships!2025 UIL Sta
ate Championships!2025 UIL St
tate Championships!2025 UIL S
State Championships!2025 UIL
 State Championships!2025 UIL
L State Championships!2025 UI
IL State Championships!2025 U
UIL State Championships!2025
 UIL State Championships!2025
5 UIL State Championships!202
25 UIL State Championships!20
025 UIL State Championships!2
2025 UIL State Championships!
```

2. Amelia

Program Name: Amelia.java **Input File: amelia.dat**

Amelia is now a TA for ECS 1100 at her college program. She wants your assistance creating a sorting program to determine who the best students are, so that we can find out who deserves extra credit. The sorting algorithm is outlined as follows:

- First, sort the students based on attendance percentage (classes attended/total classes), as the students who go to all the classes deserve the best treatment (the professor's words, not ours), with the higher attendance percentage coming first.
- Next, if attendance is the same between two students, sort by improvement average between exam 1 to exam 2, and exam 2 to exam 3 (((exam2 – exam1) + (exam3 – exam2))/2), with the higher improvement coming first.
- Next, if the improvement scores are the same between two students, sort by number of daily quizzes completed, with the most quizzes coming first.
- If all other requirements are the same, sort alphabetically by last name then first name.

Input: The input will begin with an integer, `num` (`0 < num <= 1000`), denoting the number of test cases to follow. Each test case will begin with one integer, `n` (`0 < n < 1000`), denoting the number of students in the class to be sorted. Each line will begin with two strings, separated by spaces, denoting the first and last names of the student in question, respectively. These strings will be followed by 6 integers, ca, tc, e1, e2, e3, and dq, denoting the classes attended, total classes, exam 1 score, exam 2 score, exam 3 score, and daily quizzes completed, respectively. Each of the student records will appear on its own line, and all values (string and integer both) will be separated by spaces.

Output: Output the names of the students, first name then last name separated by spaces, followed by a space, followed by the average of the 3 exam grades rounded to 2 decimal places, each on its own line, in sorted order. Every test case should be followed by a line of 10 asterisks.

Sample input:

```
2
3
Benjamin Armstrong 10 11 82 89 95 9
Derrick Martin 8 11 70 80 90 11
Matthew Sheldon 10 11 92 93 94 11
2
William Armstrong 8 9 80 90 100 7
AppleBottom Jeans 6 9 70 85 100 8
```

Sample output:

```
Benjamin Armstrong 88.67
Matthew Sheldon 93.00
Derrick Martin 80.00
**********
William Armstrong 90.00
AppleBottom Jeans 85.00
**********
```

3. Annabella

Program Name: Annabella.java **Input File: annabella.dat**

Annabella is having issues with her work in the biology lab she's been working on. She needs a program that will determine how many days until a given set of cultures will reach a certain amount of biomass. Each culture will have 3 different growth rates depending on what it is being exposed to (`UV`, `LED`, `No Light`), and a starting mass. The biomass in each individual culture will all grow at the same rate each day, including mass grown in the immediately previous day. Keep in mind you may start with enough mass to satisfy the growth requirement.

Input: Input will begin with an integer, `num` (`0 < num <= 1000`), denoting the number of test cases to follow. Each test case will begin with 2 space separated integers, `n` (`0 < n <= 100`) and `m` (`0 < m <= 1000000`), denoting the number of cultures in the set, and the target amount of total biomass, followed by a space, followed by a string denoting the type of light the cultures are being exposed to. Each of the following `n` lines will each contain 4 floating point values, the amount of mass the culture starts with, and the growth rate under `UV`, `LED`, and `No Light`, respectively.

Output: For each test case, output the integer day in which the total biomass level combined across all the cultures reaches the target value. This will always be possible.

Sample input:

```
2
2 10 UV
1.0 1.2 1.3 1.4
1.0 1.5 1.4 1.3
3 50 LED
2.0 2.0 2.1 1.9
3.0 1.6 1.7 1.9
4.0 1.2 1.3 1.9
```

Sample output:

```
5
4
```

4. Guillermo

Program Name: Guillermo.java **Input File: guillermo.dat**

You just started your new internship, and your boss Guillermo has asked you to schedule several 1 hour meetings with the team on different days. The problem is that this is a remote team and so your coworkers work all over the globe. And, being an intern and all, you feel as though if you can't schedule these meetings your return offer is as good as gone and you'll have to work at McDonalds. The time zones of each of your coworkers vary from
UTC −12:00 to UTC +14:00. A person's availability may span multiple time windows throughout the day and may even wrap past midnight.

Input: The first line contains an integer N ($1 \leq N \leq 50$) denoting the number of days your boss wants a meeting scheduled. Each day starts with a single integer P ($1 \leq P \leq 50$) - the number of people. Each person's data will be given as a JSON object (see example input) describing their timezone offset in the format ±HH:MM and their availability throughout the day.

Output: For each day, output the lexicographically smallest 1 hour meeting time in the format HH:MM − HH:MM in UTC time zone. If no such slot exists, output "`The fries are in the bag.`" to prepare for your new job since you won't be getting a return offer.

Sample input:

```
1
3
{
      "timezone": "+02:00",
      "availability": [("09:00", "12:00"), ("14:00", "18:00")]
},
{
      "timezone": "-04:00",
      "availability": [("08:00", "11:00"), ("13:00", "16:00")]
},
{
      "timezone": "+00:00",
      "availability": [("13:00", "17:00")]
}
```

Sample output:

```
13:00-14:00
```

Explanation:

```
Coworker1 (+02:00)
   ● 09:00-12:00 local -> 07:00-10:00 UTC
   ● 14:00-18:00 local -> 12:00-16:00 UTC
Coworker2 (-04:00)
   ● 08:00-11:00 local → 12:00-15:00 UTC
   ● 13:00-16:00 local → 17:00-20:00 UTC
Coworker3 (+00:00, already UTC)
   ● 13:00-17:00
```

The earliest 1 hour overlap of these 5 time windows is 13:00-14:00.

5. Heidi

Program Name: Heidi.java **Input File: heidi.dat**

Heidi has begun thieving. She needs your help determining what she can fit in her bag when she robs a place. Her bag will have a maximum capacity, and each object will have a weight and a value on the black market. You need to determine what the maximum value she can obtain is, and which objects can be used to get that value.

Input: Input will begin with an integer, `num` (`0 < num <= 100`), denoting the number of test cases to follow. Each test case will begin with two space separated integers, `n` (`0 < n <= 100`) and `c` (`0 < c <= 1000`), denoting the number of items, and maximum weight Heidi can carry in her bag. The following line will contain `n` integers, denoting the values of each item Heidi can steal. The line after will contain `n` integers denoting the weights of each item. It can be assumed that the `i`th value corresponds with the `i`th weight. All weights and values will be strictly positive.

Output: Output the maximum value Heidi can obtain, followed by a colon and a space. Followed by a list of items Heidi chose in numerical order (they will be 1-indexed, so the first item in the list is 1, and the last is `n`). If there are no items small enough to fit in the pack, output "`0: None`". If there are multiple combinations with the same weight and value, select the option that contains individual objects with the largest values, even if there are less of them.

Sample input:

```
3
3 50
60 100 120
10 20 30
4 60
40 100 50 60
20 10 40 30
5 100
10 25 30 40 50
30 40 50 60 70
```

Sample output:

```
220: 2 3
200: 1 2 4
65: 2 4
```

6. Holden

Program Name: Holden.java **Input File: holden.dat**

Holden is the main character in a video game you've been playing recently. He is the captain of a spaceship, and has to travel the universe helping people who are in trouble. You have decided that you want to determine the most optimal route to get between different areas in the universe (least possible distance travelled). You will only move directly between places in a straight line, but some of these direct routes must be avoided, due to piracy, black holes, strange creatures etc. You are also limited by the amount of fuel you have, so you can only travel a certain distance. You may not always be in the same ship, so your gas capacity may change. You travel 1 unit per gallon of fuel.

Input: The input will begin with three integers, `n` (`0 < n <= 1000`), denoting the number of test cases to follow, s (`0 < s <= 1000`), denoting the number of stations you know of, and `b` (`0 < n <= 1000`), denoting the number of paths that are blocked by spatial phenomena. Each of the following `s` lines will each contain a string, the name of the station, followed by 3 space separated floating point numbers, `x`, `y`, and `z`, denoting the spatial coordinates of the station. The next `b` lines will each contain a pair of space separated names of space stations, denoting a path that is blocked. The following `n` lines will each contain a pair of space separated names of space stations for you to find the shortest path between, followed by a space separated integer, `g`, denoting how much fuel your ship currently has. You will start from the station whose name is given first.

Output: For each test case, output the shortest possible distance needed to travel between the two given stations, rounded to two decimal places and with commas in the appropriate places. If it is not possible to get between the two stations, output "`Stuck in the Slow Zone.`". If you skate into a station exactly as you run out of fuel, you make it.

Sample input:

```
3 4 2
Ceres -1334.29 -1239.23 -12.43
Europa 2489.29 1948.01 39.23
Medina 2000.02 -1212.12 -1.34
Ganymede -843.23 999.99 123.43
Medina Europa
Ganymede Europa
Ceres Medina 3500
Europa Medina 1000
Ganymede Europa 7500
```

Sample output:

```
3,334.44
Stuck in the Slow Zone.
7,274.50
```

7. Liza

Program Name: Liza.java **Input File: liza.dat**

Liza has started working at the local frozen yogurt shop, and they have given her a special task. She has in turn asked you to write a program to handle it. She needs a program that takes a given amount of money that a customer has and determines the most expensive thing(s) they can afford (multiple items may cost the same amount of money). The menu is as follows:

Item Name	**Item Price**	**Item Name**	**Item Price**
Small Yogurt Cup	3.00	Medium Yogurt Cup	5.00
Large Yogurt Cup	7.00	Shake	5.50
Small Concrete	4.00	Large Concrete	6.00
Flavor	**Extra Cost**	**Flavor**	**Extra Cost**
Vanilla	0.00	Chocolate	0.25
Strawberry	0.35	Banana	0.70
Cake Batter	1.00	Brownie Batter	1.25
Topping	**Extra Cost**	**Topping**	**Extra Cost**
Cheesecake Bites	2.50	Gummy Bears	1.00
Chocolate Chips	0.50	Caramel	0.60
Sprinkles	0.10	Boba	2.00
M&Ms	1.20	Hot Fudge	1.40

Yogurt Cups can have up to three added toppings each, but are not required to have any, and can be any flavor. Shakes cannot have toppings, only flavors. Concretes will have one topping mixed in, so it will be half price. To calculate the price of one item, take the base cost, add the flavor cost, and the topping(s) cost (cut in half if it is a concrete). You can only use each topping once per order, and only one flavor.

Input: The input will begin with an integer, `n` (`0 < n <= 1000`), denoting the number of test cases to follow. Each test case will consist of one floating point number denoting the amount of money that the customer has.

Output: Output the order (or list of orders, separated by commas and spaces as seen in the sample output) that costs the maximum amount that the customer can afford. An order should be formatted with the flavor coming first, followed by a space, followed by the base order. If there are toppings they will come after the base order, followed by " `with` ", followed by the first topping, followed by the string " `and` ", followed by the second topping if there is one, followed by the string " `and` ", followed by the third topping if there is one. If the customer cannot afford anything, output "`Nothing`". Multiple orders of the same price should be output in lexicographic order.

Sample input:

```
3
3.84
3.50
3.94
```

Sample output: (*indented lines are continuation of previous line*)

```
Banana Small Yogurt Cup with Sprinkles
Vanilla Small Yogurt Cup with Chocolate Chips
Chocolate Small Yogurt Cup with Caramel, Chocolate Small Yogurt Cup with Chocolate
      Chips and Sprinkles, Strawberry Small Yogurt Cup with Chocolate Chips)
```

8. Marshall

Program Name: Marshall.java **Input File: marshall.dat**

Marshall is working on an audio editing tool that processes digital waveforms. A waveform is represented as a sequence of volume levels over time. To ensure audio clarity, Marshall wants to find the longest segment of the waveform where the volume is consistently controlled – that is, the volume at every point in the segment does not exceed a certain threshold.

Marshall wants to identify the maximum-length contiguous segment of the waveform where the volume never exceeds a given threshold V, and the segment is at least L time units long.

Input: The first line of input contains a single integer N ($1 \le N \le 100$), the number of test cases. Each test case begins with a line containing three integers:

S ($1 \le S \le 10^5$): the number of time units in the waveform

V ($0 \le V \le 10^6$): the maximum acceptable volume

L ($1 \le L \le S$): the minimum required segment length

The next line contains S integers: the volume levels at each time unit.

Output: For each test case, output a single integer – the length of the longest contiguous segment of at least length L where all volumes are ≤ V. If no such segment exists, output −1.

Sample input:

```
2
7 3 2
2 3 5 1 6 2 1
5 10 3
4 7 5 6 9
```

Sample output:

```
2
5
```

9. Mitchell

Program Name: Mitchell.java **Input File: mitchell.dat**

As you've been writing all of these programming problems, your laptop is filling up with space. Now you're looking through your computer to try and free up space but it's difficult to tell how much space you would free up because your computer isn't telling you the size of the directories for some reason. You decide to write a program to see how much space you could save by deleting a specific file or directory.

Input: The first input line will contain a single integer N (1≤ N≤100) denoting the number of test cases that follow. Each test case will start with two space separated integers F (1≤ F≤10^5) and Q (1≤ Q≤10^4) denoting the total number of entries (files + directories) and the number of queries respectively. The following F lines will contain either a file entry or a directory entry. The input formats of these can be seen below. The next Q lines will contain a single value V, being the Id of the directory/file you want to query.

File format – "Id": ID, "type": TYPE, "name": NAME, "size": SIZE, "parentId": ID
Directory format – "Id": ID, "type": TYPE, "name": NAME, "parent": ID

ID will be an integer value, TYPE will be either directory or file, NAME will be a string, SIZE will be an integer representing the file size in KB, parent will be an integer representing the ID of the parent directory.

Note that parent Id will be −1 if it is a top-level entry, and every parent Id will be greater than its children's Ids.

Output: Output the amount of space that would be saved by deleting the provided file/directory id for each query.

Sample input:

```
1
9 3
"Id": 0, "type": directory, "name": root, "parentId": -1
"Id": 1, "type": directory, "name": programs, "parentId": 0
"Id": 2, "type": directory, "name": stuff, "parentId": 0
"Id": 3, "type": file, "name": image.jpg, "size": 40, "parentId": 2
"Id": 4, "type": file, "name": file.txt, "size": 212, "parentId": 2
"Id": 5, "type": directory, "name": morestuff, "parentId": 2
"Id": 6, "type": file, "name": somedoc.docx, "size": 50, "parentId": 5
"Id": 7, "type": file, "name": anotherfile.dat, "size": 30, "parentId": 5
"Id": 8, "type": file, "name": pic.png, "size": 13, "parentId": 0
0
5
2
```

Sample output:

```
345
80
332
```

10. Nate

Program Name: Nate.java **Input File: nate.dat**

Your neighbor Nate works for the city creating roads. He's recently been tasked with constructing roads to connect a bunch of towns. However, laying down roads is intense and expensive labor so Nate wants to minimize the amount of road he needs to pour. Nate has asked you to help him determine the minimum amount of road he needs to pour to connect all the towns, given the coordinates of each town.

Input: The first input line will contain a single value N ($1 \leq N \leq 1000$) denoting the number of towns that follow. The following N lines will contain two floating point values in the form X,Y ($1 \leq X, Y \leq 1000$) representing the coordinates for a town.

The distance in miles between two towns is the Euclidian distance between their coordinates.

Output: Output "`X miles of road are required.`" where `X` is the minimum amount of road Nate needs to pour to connect all the towns, rounded to 2 decimal places with commas.

Note that two towns are considered connected if there is one or more roads you can use to travel between them.

Sample input:

```
10
958.31,324.64
612.11,659.05
688.23,438.73
487.35,16.83
16.69,88.52
539.12,812.09
123.62,856.53
989.12,362.49
470.64,385.00
689.53,291.41
```

Sample output:

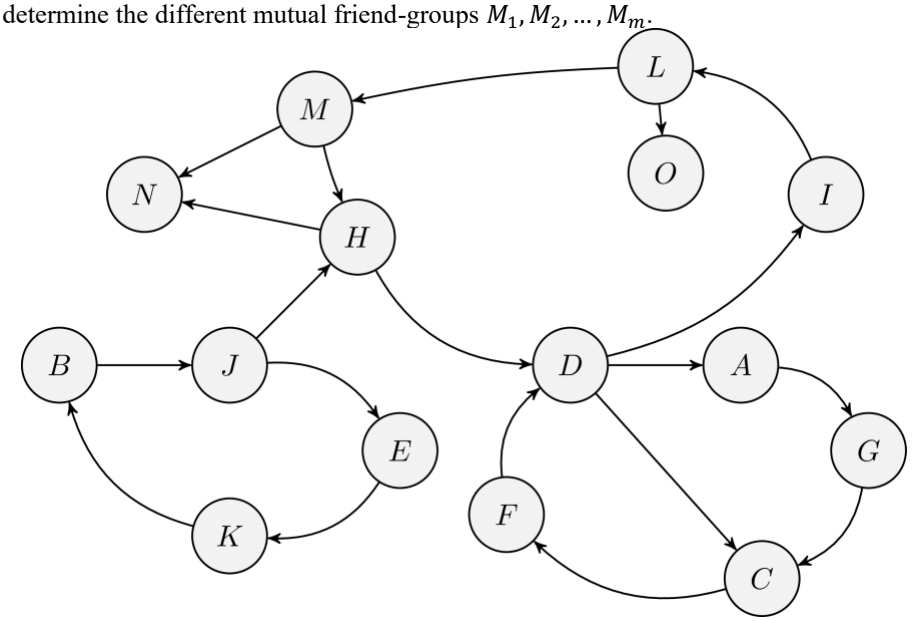
```
2,328.68 miles of road are required.
```

11. Wang

Program Name: Wang.java **Input File: wang.dat**

Wang is quite the social butterfly and loves getting to know new people on the regular. However, due to the breadth of her friendships with so many people, Wang often gets confused with which of her friends are mutually friends with one another. This makes it hard to know what friends to invite to different functions, since she does not want to force others to socialize with people they are not comfortable with.

As far as Wang is concerned, she considers friendship to be a directional relationship, rather than a unidirectional (or bidirectional) relationship. That is, person A may feel as though person B is their friend, but person B may not inherently be too fond of person A. Moreover, Wang will consider the definition of a "mutual friend-group" to be the following: let P be the set of all people. The set M is a mutual friend-group ($M \subseteq P$) so long as for every person p in P, they are either directly or indirectly friends with every other person in M (denoted $\forall q \in M,\ p \rightsquigarrow q$). More formally, for every pair of individuals p and q in M, there exists some path from p to q in the directed graph (we will assume that every person inherently thinks of themselves as a friend). Note that Wang will only consider a set of mutual friends to be a mutual friend-group so long as there are no other people in P that could possibly belong to the friend group. Given a set of people P, and the set of their perceived friendship relations, help Wang determine the different mutual friend-groups $M_1, M_2, \ldots, M_m$.



Input: First line of input will consist of a single integer, T ($1 \leq T \leq 500$), denoting the number of test cases to follow. Each test case will begin with two single space-separated integers n_i ($2 \leq n_i \leq 10^4$) and e_i ($1 \leq e_i \leq 2 \cdot 10^4$), denoting the number of people in P and the number of perceived friendship relations for the i^{th} test case. The next line will consist of n_i single space-separated strings denoting the names of each of the n_i people in P. You may assume that all names are unique and only comprise of alphanumeric characters. The following e_i lines will consist of two single space-separated strings, the j^{th} of which denotes the names of the two people involved in the j^{th} friendship relation. You may assume that the first name that appears believes that they are friends with the second name that appears on each of the e_i lines. Lastly, you may assume that the only names that will appear on the e_i lines will be names that appear among the n_i names of P.

~ **Wang** ***continues next page*** ~

~ Wang *continued* ~

Output: For each of Wang's T queries, on its own line, print out the set of mutual friend-groups. Each mutual friend group will take the format of "{p_1, p_2, …, p_ℓ}", where p_1 is the lexicographically smallest name among the ℓ members of the group, p_2 is the second smallest, and so on. Mutual friend groups should be separated by a single comma followed by a single space, and should be listed first by decreasing set size, and then by lexicographical order in the case of ties.

Sample input:

```
1
15 19
A B C D E F G H I J K L M N O
B J
J E
E K
K B
D I
I L
L M
M H
H D
A G
G C
C F
F D
D A
J H
M N
L O
D C
H N
```

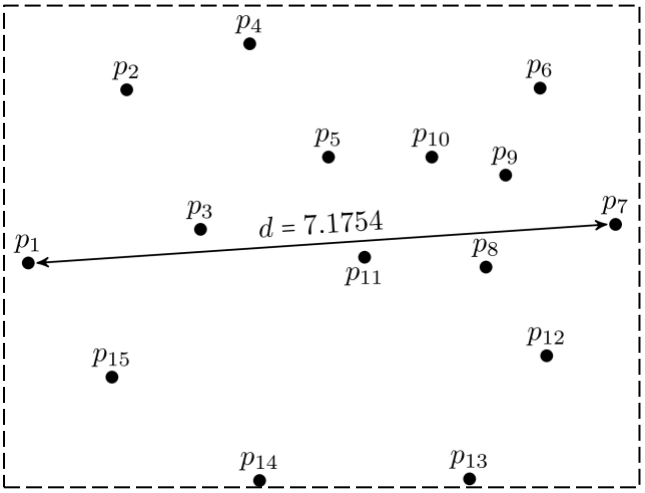
Sample output:

```
{A, C, D, F, G, H, I, L, M}, {B, E, J, K}, {N}, {O}
```

12. Willson

Program Name: Willson.java **Input File: willson.dat**

Willson isn't close to many people, but you and Willson are good friends. In fact, Willson recently came to you and admitted that he is antisocial and just doesn't like being around too many people. Moreover, he is nervous about the upcoming (required) school dance because he fears that he will be forced to socially interact with many people whom he does not wish to. And to Willson's dismay, the school has unfortunately assigned dance partners to each student! Thankfully, for Willson, two things are unique about his school dance. First, rather than having a mosh-pit style school dance, there are assigned spots for each individual student. Second, whoever books the tickets to the school dance gets to pick not only their own assigned spot, but also the assigned spot of their dance partner. Due to Willson's aversion to his assigned dance partner, he wishes to select the two available spots for himself and his dance partner that are the furthest away from one another, as to minimize his required social (and dancing) interactions with his assigned partner. Willson has asked you to help him determine, among the available spots, what the farthest distance he can be from his assigned partner is, disregarding how close he may be to other people.



Input: The first line of input will consist of a single integer, T ($1 \leq T \leq 3$), denoting the number of test cases to follow. Each test case will begin with a single space-separated integers n_i ($2 \leq n_i \leq 10^6$), denoting the number of available dance spots for the i^{th} test case. The following n_i lines will each contain two space-separated floating points x_j, y_j, the j^{th} of which denotes the (x_j, y_j) center of the j^{th} available dance spot. You may assume that $-(2^{17}) \leq x_j, y_j \leq 2^{17}$, holds for all j. You may also assume that all n_i points are in general position.

Output: For each of Willson's T simulations, on its own line, print out a single floating-point value, d_i, denoting the maximum distance Willson can be from his assigned dance partner for the current test case. Express this floating-point value to four decimal places of precision.

Sample input:

```
2
15
1.44 2.65
2.64 4.76
3.54 3.06
4.14 5.32
5.10 3.94
7.68 4.78
8.60 3.12
7.02 2.60
7.26 3.72
6.36 3.94
```

~ ***Input continued*** ~

```
5.54 2.72
7.76 1.52
6.82 0.02
4.26 0.00
2.46 1.26
14
2.64 4.76
3.54 3.06
4.14 5.32
5.10 3.94
7.68 4.78
8.60 3.12
```

~ ***Input continued*** ~

```
7.02 2.60
7.26 3.72
6.36 3.94
5.54 2.72
7.76 1.52
6.82 0.02
4.26 0.00
2.46 1.26
```

Sample output:

```
7.1754
6.4155
```