15th International 24-hour Programming Contest

http://ch24.org

DIAMOND GRADE SPONSORS

fornax 25 years

Google

SILVER GRADE SPONSOR

BalaBit IT Security

ORGANIZER

maVe
Magyar Villamosmérnök- és Informatikus-hallgatók Egyesülete
www.eestec.hu

eestec
LC Budapest

PROFESSIONAL PARTNERS

ivsz

Contest

Welcome to the 15th International 24-hour Programming Contest!

Rules

The contest starts at 2015-06-27 09:00 CEST and ends at 2015-06-28 09:00 CEST.

No solution can be submitted after the 24 hour time is up.

Web server

General contest related information will be available on our web server at http://server.ch24.local/.

Submission site

The same submission system will be used as during the Electronic Contest. It will be available at http://server.ch24.local/sub/.

Task summary

There are various kinds of problems, with various scoring rules and submission methods. Here we provide a short summary:

Task	Web submission	Interactive	Scheduled	Score decreases with time	Penalty for wrong answer	Time delay after fail/pass	Scaling	Queue	Max score
A Pool	Yes	No	No	Yes	-5	0/0	No	No	1000
B Ping pong	No	Yes	Yes	No	0	60/60	No	Yes	1200
C PRISC	Yes	No	No	Yes	0	5/5	Yes	No	1200
D Hypercubes	Yes	No	No	Yes	-5	0/0	No	No	1000
E Matrix	Yes	No	No	Yes	-5	0/0	No	No	1000
F Carnival shooter	No	Yes	Yes	No	0	-	No	No	1500

- Web submission: A static output file must be uploaded through the submission site.
- Interactive: During the solution or the submission, either network communication or other kind of interaction is necessary.
- Scheduled: continously running task during the contest with scheduled interactions.
- Score decreases with time: Submitting at the end of the contest is worth 70% of what would be awarded at the beginning.
- Penalty for wrong answer: Wrong answer gets -5 points (different value may be specified explicitly in the task description).
- Time delay after fail/pass: Duration in minutes while no new submission is accepted after a wrong/correct answer.
- Scaling: The score for this problem may change over time depending on submissions by other teams. (Note that your last submission is considered and not your best one.)
- Queue: Only one team can work on this task at a time (hardware task) so there will be a first-come, first-served queue.

Ports

Port	Task	Service description
`80`	-	web
`6667`	-	irc
`u123`	-	ntp server
`u53`	-	dns server
`u67,u68`	-	dhcp server
`10000`	C	Pingpong video stream
`10001`	C	Pingpong control
`16600`	F	Carnival shooter

Ports starting with `u` are UDP ports. All services are hosted on `server.ch24.local`.

Contact

General contest related information and data will be published on the web at http://server.ch24.local/.

Important announcements will be made on the `#info` irc channel and will be published on the web as well.

For general discussions and questions join the `#challenge24` irc channel.

There will be separate channels for task related problems as well: `#A`, `#B`, `#C`, `#D`, `#E`, `#F`.

Prologue: Amusement Park

The Amusement Park of Budapest, or as previously called: English Park has now been closed for two years. There is a mysterious legend from this place, about some wicked toys, which have never been shown to the public. They vanished without a trace. Stephen, the janitor of the park, yesterday found an old, abandoned shed, where he found some of these toys, and he started to fiddle with them...

source

source

A. Pool (1000 points)

Samuel Fischer, the semi-professional fisherman is spending his day-off at the Amusment Park's fishing ponds. There are different types of fish, mixed in the ponds. Samuel would like to catch specific types and amounts of fish, to which he uses his unique fishnet-plates. He can catch exactly those fish that are covered perfectly by the plates. Every pond and plate has the same area and every pond has to be covered with a plate. The fisnet-plates can be mirrored or rotated.

source

Provide the sequence and orientations of the plates needed to catch the given types and amounts of fish!

The ponds and plates are square-shaped, the lenght of the edges is `3 <= n <= 6`. The number of ponds and plates is `1 <= k <= 2000`. The maximum types of fish in a pond is `1 <= t <= 30` (Not every type of fish is neccesary given in every pond)

Input

The first line of the input file gives the number of test cases included in the file (`testcases`).
After that `testcases` number of test cases will be provided in the following format:
1. line: size of the ponds, number of ponds, number of types of fish: `n k t`
2. line: number of to-be-caught fishes: `t` integers: the `i.` element denotes how many fish have to be caught of the `i.` type (divided by space characters)
Next `k` rows: ponds' details: lines of `n×n` numbers separated by spaces: given type of fishes on each segment (0 denotes a segment with no fish in it)
Next `k` lines: the plates: lines of `n×n` numbers `(0 or 1)` separated by spaces. Each plate can only catch fishes on the positions of 1st.
After this, without any separations, comes the next test case (if there is any more left)

Output

`k` lines, every line contains 3 numbers. The i. line's 1. number shows which plate to be used (indexes go 1...k),
The 2. number denotes how many times the plate has to be mirrored to it's vertical axis (0...1)
The 3. number shows how many times the plate has to be rotated around it's center clockwise (0...3)
IMPORTANT: FIRST MIRROR, THEN ROTATE!
The test cases don't have to be separated from each other.

Example input

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

It means, that Samuel has the following pond and plate:

```
123   001
041   000
100   110
```

He would like to catch one type 1 and one type 2 fish

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

Example output

```
1 1 2
```

It means, that the first plate is used in the first pond, mirrored vertically then rotated by 2×90 = 180 degrees.

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

B. Pingpong (1200 points)

While Little Timmy was wandering arount the rides in the theme park, he noticed the Railway of Horror, and could not resist to go for a round. Just as the carriages started rolling and he found himself head on with the first scary puppets, he knew it was a very bad decision to go, and he will panic if he canât get to the exit really soon.

You can help to get Little Timmy out in time using a model of the theme park ride. The carriage is substitutsed with a table tennis ball, while control method is a fan that can be turned around the perimeters of the play area to control the direction of the air flow and its strength can be adjusted as well.

The rule is simple: get the ball from the starting point to the hole in the opposite corner of the area. You have 2 minutes to score, reaching the timeout results a fail in that round.

- You get 100% of the points if you can score in the first 20 seconds,
- while you get 70% if you score in the last second.
- In between, score is linearly distributed between 70% and 100% according to the remaining time.

The game is divided into rounds. One round lasts for 1.5 hours, in which time slot each of the 30 teams get 2 minutes to play plus some technical and in-between time. During the 24 hours, 16 rounds will be played. The first 4 are out of scoring, meant to try out the controls, make experiments, etc. All the other rounds are scored as described. The second 4 are done on a plain area with no obstacles. The third 4 are done on an area with groove obastacles, and the last 4 are done with groove and cone obstacles.

Protocol

You can access the video stream from the camera overlooking the area on server:10000 with CH24/CH24 authenication. For controlling the game, you can connect to port 10001. Timing starts when you receive the message `Start`. You can control the speed and position of the fan with sending a packet containing `!SSS PPP\n` where SSS stands for speed, ranging between 000 and 100 percent. PPP stands for the position, ranging between `000` and `400`. The default state is `200`, and `200` units mean a full circle, so you can turn fully in both directions with the fan. The number formats are delimited, you have to write e.g. 005 for 5 percent.

Submission

For submission, fill out one of your tickets (given by the organizers), and hand it to the booth. Don’t forget, that each ticket is valid for only one season.

Dont forget, when you want to play, we have to set up the connection.

Seasons

9:00	-	10:30	Without scoring	0-0 points
10:30	-	12:00	Without scoring	0-0 points
12:00	-	13:30	Without scoring	0-0 points
13:30	-	15:00	Without scoring	0-0 points
15:00	-	16:30	Plain area	49-70 points
16:30	-	18:00	Plain area	49-70 points
18:00	-	19:30	Plain area	49-70 points
19:30	-	21:00	Plain area	49-70 points
21:00	-	22:30	Groove obstacles	70-100 points
22:30	-	0:00	Groove obstacles	70-100 points
0:00	-	1:30	Groove obstacles	70-100 points
1:30	-	3:00	Groove obstacles	70-100 points
3:00	-	4:30	Groove and cone obstacles	105-150 points
4:30	-	6:00	Groove and cone obstacles	105-150 points
6:00	-	7:30	Groove and cone obstacles	105-150 points
7:30	-	9:00	Groove and cone obstacles	105-150 points

Good to know

The arm takes 12 seconds to make a full circle, and when calculating steps, add 2 extra after every stopping and restarting the arm to compensate for the lost steps at start.

C. PRISC (1200 points)

During a huge storm a lighting struck the long-forgotten warehouse of the Park. Due to some dark arcane miracle, the computers stored there gained conciousness and now are on the loose! The only way to stop them is to overwrite their murderous code! Bob, the Parkâs sysadmin is the only one capable of such feat! He learned from old handbooks, that the program of the machines was written in PRISC (Pretty Reduced Instruction Set Computing)!

Your task will be to help him create binaries in this code, which when run, outputs a specific file. The goal is to minimize the size of the binary (which consist of the code and any static data needed to output the file)

Opcodes

```
 OPCODE |    MNEMONIC      | EFFECT
-------------------------------------------------------------------------
  0x00  |  MOV  op1, op2   | op1=op2
  0x01  |  RMOV op1, op2   | op2=op1
  0x02  |  ADD  op1, op2   | op1=op1+op2
  0x03  |  SUB  op1, op2   | op1=op1-op2
  0x04  |  MUL  op1, op2   | op1=op1*op2
  0x05  |  DIV  op1, op2   | op1=op1/op2
  0x06  |  MOD  op1, op2   | op1=op1%op2
  0x07  |  AND  op1, op2   | op1=op1&op2
  0x08  |  OR   op1, op2   | op1=op1|op2
  0x09  |  XOR  op1, op2   | op1=op1^op2
  0x0a  |  JZ   op1, addr  | if (op1==0) jump to addr
  0x0b  |  JNZ  op1, addr  | if (op1!=0) jump to addr
  0x0c  |  JNEG op1, addr  | if (op1&0x80000000!=0) jump to addr
  0x0d  |  JMP  addr       | jump to addr
  0x0e  |  OUT  addr       | output 1 or 4 bytes from addr
  0x0f  |  IN   addr       | input 1 or 4 bytes into addr
```

Operands

Every operand is a memory address. These addresses don’t need to be aligned.

Example:

```
Memory, before ADD 4,8:
|05|00|00|00| |10|20|30|00| |12|AB|45|60| |01|00|00|00|
Memory, after:
|05|00|00|00| |22|CB|75|60| |12|AB|45|60| |01|00|00|00|
```

The second operand (or the only operand in case of JMP and OUT) will be interpreted as a pointer if prefixed with an asterisk:

```
Memory, before ADD 4,*12:
|05|00|00|00| |01|02|00|00| |AB|00|00|00| |08|00|00|00|
Memory, after:
|05|00|00|00| |AC|02|00|00| |AB|00|00|00| |08|00|00|00|
```

All operations can be postfixed with "B", so they operate on single bytes:

```
Memory, before ADDB 4,8:
|05|00|00|00| |10|20|30|00| |12|AB|45|60| |01|00|00|00|
Memory, after:
|05|00|00|00| |22|20|30|00| |12|AB|45|60| |01|00|00|00|

    Also, if memory is:
    |00|00|00|00| |00|01|00|00| |00|00|00|00| |00|00|00|00|
    "JZ 4, label" will take the jump, "JZB 4, label" won't
```

Non-B operations work on signed, little-endian 32 bit integers.
"B" instructions work on unsigned bytes.

Opcode encoding:

The first byte is the opcode and the options as bits:

```
bits 0-3: opcode
bit 4: Is operand 1 wide
bit 5: Is operand 2 wide
bit 6: Does the opcode work on bytes (postfixed with B)
bit 7: Is the second operand indirect (asterisked)
```

The next few bytes are the two operands. Their width depends on bit 4-5 of opcode byte. If the bit is 0, the operand is represented on one byte, if it is 1, the operand is the next four bytes. One byte operands are zero- extended to 4 bytes.

Other assembler features:

- `comments`: Everything after a pound sign (#) is a comment.
- `labels`: A line is a label, if it ends with ":". Labels don't need to be forward declared, you can use a label before declaring it.
- `data`: The DB keyword will emit one byte, the DW keyword will emit a 4 byte integer, and the ascii keyword will emit the string until EOF

Start and end of the code:

The binary code is loaded to address 0x100 of the memory, and the Program Counter is set to 0x100. The execution finishes, if an exception occurs (invalid memory access, division by zero), or the PC is set to 0. (e.g. with JMP 0)

Assembling:

Using the provided assembler.py python2 script, you can assemble PRISC binaries. Usage:

```
$ python assembler.py --infile file_to_assemble.asm --outfile out.bin
```

Running:

```
$ python runner.py out.bin
```

(Of course you may write it in any other prefered language, this is just an example.)

The task:

You need to optimize 3 given binaries, so they do their job in the least amount of clock ticks. The number of clock ticks an instruction takes is equal to its length in bytes. Keep in mind that loading the program takes 4 clock ticks per byte in the beginning, so optimize for size too!

Submission:

You get the 3 binaries, plus one example binary with source, encoded in base64. You need to decode them to get the actual runnable binaries, optimize them, and reencode to base64 to be able to submit them.

Tips:

- Some instruction operands can be represented in 3 bytes, some can be as much as 9, depending on the operands. Use them wisely
- The first 0x100 bytes of the memory can be used as a register file. It’s useful to have a "register" containing 0, and one containing 1, since there are’t any arithmetic operations with immediate second operands. Instructions that only access this lower part of memory also tend to be shorter, and as such, faster.

D. Hypercubes (1000 points)

The newest attraction of the Amusement Park is the *Tetrix-9000 K-Dim Ultra*: a k dimensional game of fun, logic and splendid visuals. Jack and Jill, the two siblings, are the best players far and wide. They would like to understand the deeper connections of the game better, so they start to analyze the different possible positions of the various in-game levels. Help them with their calculations!

Task

Find the position of `n` pieces of `k` dimensional hypercubes in the `k` dimensional space, where hypecubes placed next to each other cannot overlap, but interconnect with their `k-1` dimensional sides. What is the maximum number of these joint `k-1` dimensional sides?

Input

First line: `t`, the number of test cases. Next `t` line: 2 numbers each line: the dimension number `2 <= k < 100` and the number of hypercubes `0 <= n < 2`^{50}

Output

`t` lines: first number each line: the maximal number of `k-1` dimensional sides which are bounding two different hypercubes

Example input	Example output
6 2 9 2 8 3 27 3 26 4 81 4 80	12 10 54 51 216 212

E. Matrix (1000 points)

Little Timmy and his father roams around the Amusement Park, searching for yet another source of fun and excitement. As they pass a small booth, a 100 years old looking woman with crooked teeth and warted nose shrieks after them: *Oh, oh little boy! I see great things happening to you in your future! You will have a fair share of fame and love, luck and prosperity! Come closer little boy! Read out this paper, answer the question in it, and I will tell you more!*

Little Timmy approaches her, takes the paper from her wrinkly hands, opens it, and starts to read...

Solve `Ay=c` linear equation system, where `A` is an `N×N` dimensional matrix, `y` and `c` are `N` dimensional vectors, `y` is unknown, `A` and `c` are inputs.

The vectors and the numbers in the matrices have the following structure: `a+b×i`, where `a` and `b` are elements of $Z_P = \{0,1,...,P-1\}$, ie. are elements of the modulo `P` residue class (`0<=a<P`, `0<=b<P` are integers), `i` is the square root of the greatest number smaller than `P`, that has no square root mod P.

Find components `a` and `b` of vector `y`!

Input:

The first row contains 2 numbers, `N` and `P` separated with a space character. `P` is prime and N>0. The next `N` rows each contain 2×N numbers, which are the values `a` and `b` of the current row of matrix `A` (respectively).

The next `N` rows contain 2 numbers each, which are the current components of vector `c` (respectively).

Output:

N rows, each row contains 2 components of vector `y` – `a` and `b`. You have to write these two numbers separated with a space character.

Example input	Example output
2 7 1 1 0 2 0 0 5 3 0 1 1 0	5 6 2 3

F. Carnival shooter (1500 points)

William Tell wanted to find some souvenir to his wife and when he sight the Carnival Shooter, he relieved. No one else can be better in that game, than William. Unfortunately the usage of the bow was forbidden, just airgun can be used int Carnival Park. So William realized, that he should looking for someone else who can use the gun. That is why we are asking you to try to shoot a souvenir to William's wife. Let's shoot a cute, huge Teddy Bear!

The rules of the game are similar to the Tic Tac Toe, but there are some special things, to make the game more fun. Every participant plays in the same game. Initially the field is 6x6, but if a shot is exactly next to the borderline, the field will be increased by one more row in the next round. All of the participants take shoot in a same time, but if two or more team shoot in a same place, these shots are invalid and these won't change the status of the field. One shot can change the field status in two different ways. If the point, where the shot was shot, is plain, there will be in the hole, but if that is not empty, that hole will be refilled. The game starts with the allocation of the new team numbers. When the results or the differences will be sent out, that number will represent your team. One game takes 100 rounds. In the beginning of the round, the game is waiting for 500 millisecs. Than updating of the actual field is starting and the changes of the field are broadcasting to all of the connected teams. After the rounds, evaluation part starting, which means searching the longest row of the teams. Rows are looked for in vertical, horiyontally and diagonal. Than the result is broadcasted, longest row of the teams are send out, teams are represented, the number which was sent in the beginning the game. After all of these the a new game starts with the sending of the new team numbers.

Input

The teams should just send the coordinates (x and y) of the actual shot. If you are sending more packets, between two rounds, the last sent packet will be part of the evaluation. Teams whose sending inadequate messages or more than 3 packets between two rounds, will be banned from the actual game. Packets are going through TCP/IP protocol, here is the waited pattern:

```
{ x: 3 y: 4 }
```

Output

The game will start, when the new team numbers will be sent out to every connected team. After a round, the difference from the previous status will be sent out. This packet will contains the removed and added coordinates. The final output of a game is the scoreboard. That will contain the length of the longest line of every team.

Scoring

One session of game is one hour. Teams get points after the length of the longest lines. Length of that line is the point that the team gained in the actual game. These points are summarized and at the end of one session these will define the final order of the session. The points, that can be gained from a session, are increases linearly. 1500 points can be gained by winning the 24 sessions.

Example input

One shot:

```
{ "x": 1, "y": 1}
```

```
{ "id": 4}
```

Example output

New team number:

Difference output:

```
[{ "x": 1, "y": 1, "t": 3, "f":a }, { "x": 5, "y": 2, "t": 7, "f":r }]
```

Result:

```
[{ "t": 1, "r": 5 }, { "t": 3, "r": 3 }, { "t": 2, "r": 1 }]
```