Challenge24

12. International 24-hour Programming Contest

12th International 24-hour Programming Contest

http://ch24.org

XII. INTERNATIONAL 24-HOUR PROGRAMMING CONTEST 2012 - BUDAPEST, HUNGARY

MAIN SPONSOR

DIAMOND GRADE SPONSORS

GOLDEN GRADE SPONSOR

3D MODELING SPONSOR

INNOVATIVE PARTNER

TALENT PARTNER

SILVER GRADE SPONSORS

SOCIAL MEDIA PARTNER

SECURITY PARTNER

PROFESSIONAL PARTNERS

ORGANIZER

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

SPECIAL THANKS TO

M Ű E G Y E T E M 1 7 8 2

Contest

Welcome to the 12th International 24-hour Programming Contest!

Rules

The contest starts at 2012-04-28 09:00 CEST and ends at 2012-04-29 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.org/.

Submission site

The same submission system will be used as during the Electronic Contest. It will be available at http://server.ch24.org/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	Score decreases with time	Penalty for wrong answer	Time delay after fail/pass	Scaling	Scheduled	Max score
A	Yes	No	Yes	-5	0/0	No	No	1000
B	Yes	No	Yes	-5	0/0	No	No	1000
C	Yes	No	No	-5	5/1	Yes	No	1000
D	Yes	No	Yes	-1	2/0	No	No	1000
E	Yes	No	Yes	-5	0/0	No	No	1000
F	Yes	No	Yes	-5	0/0	No	No	500
G	Yes	No	Yes	-5	5/0	No	No	500
H	No	Yes	No	0	0/0	Yes	Yes	2500
I	No	Yes	No	0	0/0	No	No	1000
J	No	Yes	No	0	0/0	No	Yes	2500
K	No	Yes	No	0	0/0	No	Yes	3500
L	Yes	No	No	0	0/0	Yes	No	1000

- 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.
- Score decreases with time: Submitting at the end of the contest worth 70% of the score at the begining.
- Penalty for wrong answer: Wrong answer gets -5 points (different value may be specified explicitely 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.)
- Scheduled: Must be submitted or solved at a scheduled time (cannot be postponed later).

Ports

Port	Task	Service description	Connection direction server <-> team
`80`	-	web	<-
`6667`	-	irc	<-
`u4242`	-	time server, UDP broadcast destination port	->
`u123`	-	ntp server	<-
`u53`	-	dns server	<-
`u67,u68`	-	dhcp server	<-
`u30002`	H (TV - Stream)	TV stream UDP broadcast destination port	->
`31000`	H (TV - Stream)	Aircraft ID report connection	<-
`22000`	I (Air hockey)	Practice client control connection	<-
`u22200`	I (Air hockey)	Practice UDP broadcast destination port	->
`20000`	J (Air hockey)	Tournament client control connection	<-
`u21002`	J (Air hockey)	Tournament UDP broadcast destination port	->
`10001`	K (WW3)	Practice server control	<-
`10002`	K (WW3)	Practice server game connection	<-
`10003`	K (WW3)	Practice server telnet view connection	<-
`10012`	K (WW3)	Tournament server game connection	<-
`10013`	K (WW3)	Tournament server telnet view connection	<-

Ports starting with `u` are UDP ports.

The UDP broadcast destination ports should be opened by the teams (if they are interested in the broadcast).

All other services are hosted on `server.ch24.org`.

Schedule of interactive tasks

time slot starts at		Air hockey		WW3		TV stream
relative	**CET**	**scored**	**practice**	**scored**	**practice**	**scored**
00:00	09:00	-	24 hours	-	24 hours	24 hours
00:30	09:30					
01:00	10:00					
01:30	10:30					
02:00	11:00					
02:30	11:30					
03:00	12:00					
03:30	12:30					
04:00	13:00	tournament #1 1 hour 30 minutes				
04:30	13:30					
05:00	14:00					
05:30	14:30	-				
06:00	15:00			tournament #1 max. 4 hours		
06:30	15:30					
07:00	16:00	tournament #2 5 hours				
07:30	16:30					
08:00	17:00					
08:30	17:30					
09:00	18:00					
09:30	18:30					
10:00	19:00			-		
10:30	19:30					
11:00	20:00					
11:30	20:30					
12:00	21:00	-				
12:30	21:30					
13:00	22:00					
13:30	22:30					
14:00	23:00			tournament #2 max. 4 hours		
14:30	23:30					

time slot starts at		Air hockey		WW3		TV stream
relative	**CET**	**scored**	**practice**	**scored**	**practice**	**scored**
15:00	00:00	tournament #3 9 hours	24 hours (continued)	tournament #2 max. 4 hours (continued)	24 hours (continued)	24 hours (continued)
15:30	00:30					
16:00	01:00					
16:30	01:30					
17:00	02:00					
17:30	02:30					
18:00	03:00			-		
18:30	03:30					
19:00	04:00					
19:30	04:30					
20:00	05:00			tournament #3 max. 4 hours		
20:30	05:30					
21:00	06:00					
21:30	06:30					
22:00	07:00					
22:30	07:30					
23:00	08:00					
23:30	08:30					

Contact

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

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`, `#FGH`, `#IJ`, `#K`, `#L`.

Motorcycles and Gang Life

A couple of weeks ago one of the regular problem set team meetings was crashed by a real life motorcycle gang. One of them - who looked to be the leader, since he sported the most facial hair and body modifications - started explaining something to us with extreme vehemence.

It took us some time to start to understand him, since as software developers, mathematicians and engineers, we were very intimidated by his powerful persona and leadership skills. Having finally attained a rapport, he clarified that he was called Bertie, and his gang would request to employ the considerable talents of Challenge24 finalists to solve a bunch of problems for him and his gang.

His tasks looked fun enough, and it was conspicuously easy to adapt them to our existing problem ideas; we accepted his conditions very quickly. We can only hope you will all find the tasks agreeable.

Enjoy & have fun,
 - *The Organizers*

A. Alarm system (1000 points)

source:
http://www.fugly.com/media/IMAGES/Whoops/stolen_motorcycle.jpg

There are 3 large motorcycle gangs sharing the same parking lot. They have a lot of trouble with stolen bikes lately. You sense a great business opportunity here: you will try to sell them an alarm system to guard their bikes.

The parking lot is a rectangular grid, each cell is either empty or occupied (a bike is parking there). The alarm device you are selling will emit laser beams in the 4 cardinal directions (north/south/east/west) to measure distance to the nearest object. Whenever the measured distance changes, the system triggers loud sirens, blinking lamps and a hurd of gnus.

Due to safety regulations there cannot be anything immediately near a device so only one device can be installed in a cell. A device can be placed in any empty cell, however two devices using the same wave length cannot be placed with direct visibility between them (for example in the same row without an occupied cell between them), the system gets fooled and stops working.

Fortunately you have three different models of the device, each of them operating on different wave lengths, so you plan to recommend different models to the three gangs. The different models don’t interfere in any way and they don’t block each other’s beam either.

Your goal is to sell as many devices as possible. To achieve this, you need to make a map of the most optimal placement where

- there is no interference between devices
- and the setup requires as many devices as possible

Note that it is **not** a requirement that all bikes should be protected.

Input

The first line of the input is

```
W H
```

the size of the rectangular grid.

Then H lines follows, each contains W characters followed by a newline character.

The *i*th character of the *j*th line describes the *i*th cell of row *j*: ’#’ means occupied, ’ ’ means empty.

Output

The first line of the output is

```
W H N
```

the size of the rectangular grid as in the input and N the maximal number of devices which can be placed in the given parking lot.

Then H lines should follow describing an optimal placement of the devices: ’#’ means occupied cell, ’ ’ means empty cell, ’1’,’2’ and ’3’ means a device placed for the three groups respectively.

Example input

```
8 5
########
#     ##
## #   #
#     ##
########
```

Example output

```
8 5 10
########
#23 1 ##
##2#321#
#312  ##
########
```

Note: there may be multiple optimal device placements, any such solution is accepted.

B. The Great Mixup (1000 points)

After a long ride out on the roads, Bertie’s gang prefers to spend some time resting (and obviously drinking) in a pub, their motorcycles are properly lined up in front of the pub. At the end of the day each rider has his or her own bike and no one likes taking another one by mistake. They usually remember their bikes by their position.

source: http://en.wikipedia.org/wiki/File:Motorcycles_parked_outside_the_Rajiv_Gandhi_Institute_of_Technology.jpg

The bikes are painted in a variety of colors, but there are still more bikes than colors. One day the bartender decides to be play a little practical joke and plans to tell Bertie that while the gang was happily getting drunk in the pub, he rushed out and reordered the bikes. The guys would surely notice if there was a bike of a different color parked in the spot where they left their own bike; so the only way to mix up the bikes is to shuffle them around in a way that the color of the bike in a given spot stays the same.

Unfortunately the bartender can’t leave the bar unnoticed at the moment so he can’t check the parking lot and count bikes. Instead, he is pondering how many different solutions exist for different number of bikes and colors.

Input is a list of *n*, *k* pairs (*n* being the number of bikes and *k* being the number of possible colors). Last line of the input is always "0 0". Your task is to calculate how many different placements of the bikes he can pick from, for each input line. However, since you do not know the exact coloring of the actual bikes, you need to sum the number of solutions for each possible color setup. The bartender dropped out of school after 6 grades, so he can count only up to 100000007. To overcome this limitation, you need to submit your results modulo 100000007.

Example input

```
1 1
1 3
6 3
2 4
10 10
100 200
35 30
64 29
32 2
1000 2
1000 3
0 0
```

Example output

```
1
3
20160
20
21262936
93294537
48151294
39886421
15453915
84702802
36100834
```

C. Functions (1000 points)

Besides roaming the country with his motorcycle gang, Bertie has a daytime job as a QA engineer at a market lead semiconductor company. In this task you are asked to help him by optimizing integrated circuit testing procedures.

Manufacturing large quantities of complicated integrated circuits is a challenging task. A large production run of ICs may only have a few faulty pieces (due to unexpected errors in materials or processes).

The cost of such failures is higher if the problem is recognized later, when the chips are already embedded in consumer products. Thus, in some situations, it is cheaper to test all chips right after manufacturing, and remove faulty ones before they're shipped. The test process must be as fast as possible, to keep costs down.

Your task is to design minimal test scripts for stateless digital circuits.

You're given descriptions of 10 different IC models to test. Each is given as a netlist, and a list of potential bugs and their frequency in percent - i.e. how likely a malfunctioning or short circuit occurs on a component.

For each product, your submission should be a list of test cases: digital inputs that are delivered to the circuit, and the expected outputs that it should generate.

We run the submitted test case list on a huge amount of circuits (of which we know whether they're faulty or not). For each model, we have a fixed set of reference circuits manufactured for the test run.

For each circuit, the digital inputs in the list are sent to the circuit, and the measured digital outputs are compared to the references specified in the submission. If every digital input specified in the test yields the digital output that matches the reference output submitted, the test considers the circuit faultless; if there is at least one digital output that's different, the test considers the circuit faulty.

If the test answers properly whether the circuits are faulty or not for at least 99.9% of all circuits, the submission passes; otherwise it's a failure.

Product files (problem input)

Header of the product file is

```
N C I O
i1 i2 ... in
o1 o2 ... on
D1 D2 D3
```

First line contains the number of networks (N), total number of components (C), number of input components (I) and number of output components (O). The next two lines are listing the input and the output component IDs.

The fourth line lists statistical probability of presence of different defects given in percents: 0.1 means 0.1%, that is every 1000th gate manufactured will have that specific problem. Note: since a single circuit consits of multiple gates, it means a given instance of the circuit may have multiple defect gates. Possible gate-defects are, in the same order as listed in the defect-line:

1. inverted output (D1)
2. output is always zero (D2)
3. output is always one (D3)

The rest of the file describes each network (network is a wire connected to multiple components) in a separate line.

```
n c1 c2 ... cn
```

First integer is the number of components connected to that network, the remaining of the line is a list of component IDs. The first component is always the electrical source for that network (a component's output), the rest of the components are all sinks (inputs).

The circuit is composed of three different component types:

- input port component is a single pin device where signals are injected into the circuit (thus they are always source of a network)
- output port component is a single pin device that transmit signals out from the circuit (always sink in a network)
- multiple-input NAND gates with a single output pin; sink in multiple networks, source in a single network; If the state of all input pins are 1, the output of the gate is 0, in all other cases the output is 1.

Sources are always strong enough to drive all connected sinks, carrying out changes so fast that the logic value of the network is always 0 or 1 and never floating.

Submission format

Teams submit plain text test patterns. Each line of the output contains I + O numbers describing the bits of the tested input-output pair. Bits are listed as 0 or 1 and are separated by a space and are listed in the same order as input and output components appear on the input and output lists on the netlist.

Example input

Product file (problem input):

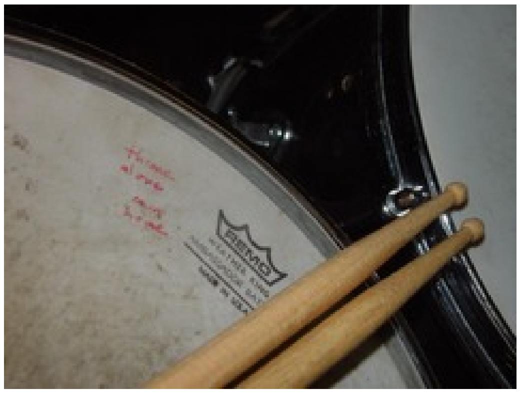
```
6 7 2 1
5 6
4
0.1 0.1 0.1
3 1 0 0
2 2 1
2 3 1
2 0 4
3 5 2 2
3 6 3 3
```

Example output

A valid solution:

```
0 0 1
1 0 0
0 1 0
```

D. Drums (1000 points)


source: http://www.public-domain-image.com/cache/objects-public-domain-images-pictures/drumsticks-laid-scross-drum_w725_h544.jpg

Bertie decides that his motorbike gang needs a theme song. He is a novice drummer and he has composed a lot of songs during the long hours he spent wandering on the highway. He always had his pocket sound recorder and a small portable drum kit with him and by now he has countless hours of recordings accumulated. He needs your help to transcribe the grooves.

Input format

Each input is a wav file containing a recording of a session. Separate wav files of each kind of drum are also provided to make it possible to associate IDs with individual drums in his kit.

drumID	sample
1	1_kick.wav
2	2_snare.wav
3	3_hihat.wav
4	4_hihat.wav
5	5_crash.wav
6	5_kettle.wav
7	5_cowbell.wav
8	8_triangle.wav

Source: http://www.drumsamples.org

NOTE: because <insert your favorite excuse here>, we accept both 3 and 4 for hihat, you don't need to distinguish between them.

For Drums, you will get only 5 inputs at the beginning of the contest. You will receive further inputs during the contest.

Output format

Output is a list of events, one event per line. An event consists an integer time stamp (miliseconds elapsed since the beginning of the recording), a space and an integer drumID. Events should be generated for each and every stroke and are accepted only if the drumID matches and time stamp is within +-20 ms. Output must be sorted by time stamp in increasing order. In case there are multiple events for the same time stamp, they should be ordered by drumID (lower ID comes first). Since Bertie has 4 limbs to play with, he can strike at most 4 instruments in the same moment. However, some cymbals may emit a long enough sound to overlap more than 4 other events.

Example

A valid solution for in0.wav (provided with the real inputs):

```
0 1
0 3
444 2
444 4
666 2
888 1
888 4
1110 2
1332 2
1332 4
1776 1
1776 5
2220 4
2220 6
2442 6
2664 1
2664 4
2886 7
3108 4
3108 7
3552 1
3552 3
3996 2
3996 8
4218 2
4440 1
4440 8
4662 2
4662 8
4884 2
4884 8
```

E. Interval (1000 points)

Bertie recently bought a multimedia smartphone with a touch screen, to edit his drum recordings while on the road. He bought an expensive mixer program that was obviously optimized for much larger screens, resulting in all the little knobs and switches being too small. Whenever he tries to operate the GUI with his finger, he toggles a group of switches instead of a single switch. The more switches are turned on, the higher the power consumption of the expensive amplifiers is.

source:
http://howtosmile.org/record/492

In this task, you will need to simulate Bertie's actions and estimate total power consumption by summing how many of the important switches were on during the simulation.

Given n switches numbered from 0..n-1. Each switch is either turned on or off. Initially all of them are off.

There are two operations Bertie can perform on the switches and each of them has two parameters, a and b:

operation	description
toggle(a,b)	toggle all switches in the closed interval [a,b]
query(a,b)	query how many switches are turned on in the closed interval [a,b] (these are the important switches)

Your task is to simulate a sequence of k operations and calculate the sum of all query results.

Simulation

The simulation details are specified below using pseudo code.

```
function seed(s) {
        x := s
}

function rand() {
        x := (6364136223846793005*x + 1442695040888963407) mod 2^64
        return x div 2^32
}

function simulation(n, k, p, s) {
        seed(s)
        sum := 0
        repeat k times
                len := rand() mod n
                a := rand() mod (n-len)
                b := a+len
                r := rand()
                if r < p then
                        sum := sum + query(a, b)
                else
                        toggle(a, b)
                end if
        end repeat
        print(sum)
}
```

Input

only one line: n k p s

Output

only one number: the sum of all of the answers for the queries

Example input

```
10 10 2147483648 12345678987654321
```

Example output

```
8
```

Note: in this example the steps were

```
toggle(1, 6)
toggle(2, 7)
toggle(6, 6)
toggle(1, 2)
query(0, 6)
toggle(0, 8)
query(1, 7)
query(1, 3)
query(6, 7)
toggle(1, 8)
```

FGH. TV

source: http://www.nowpublic.com/vintage_televison_set

Bertie is visiting his remote relatives in a far away country. Initially he has a lot of trouble getting along with people in this country, but he figures capturing the local TV broadcast will help him overcome some of the social difficulties. Unfortunately the country developed it's own standards that are incompatible with the built-in TV set of Bertie's motorcycle.

Your task is to decode the stream and use the resulting video as input for solving different subtasks for Bertie.

Signal

There is a continuous stream of 8 bit, signed integers delivering frames. Frames are broken up into approx. `NROWS` rows, each row having about `NCOLS` pixels. Each pixel is described by a single intensity value found in the next 8 bit integer - intensities are positive, between 0 (black) and +127 (white).

Because of the inaccuracy of transmission, `NROWS` and `NCOLS` pixels can not be counted in number of samples; instead sync signals separate rows and frames. A sync signal is a continuous series of samples with value less than `STHRS`. Values between 0 and `STHRS` should be considered black pixels.

Each new row starts after the last sample of a horizontal-sync (hsync) signal not shorter than `HSYNC_MIN` and no longer than `HSYNC_MAX`. A new frame starts at the end of a vertical sync (vsync) signal which is longer than `VSYNC_MIN` and shorter than `VSYNC_MAX`. A vsync signal also functions as a hsync for the first row.

Constants:

name	value	description
`NROWS`	100	nominal number of rows
`NCOLS`	160	nominal number of columns
`STHRS`	-32	sync signal threshold
`HSYNC_MIN`	2	minimal length of a hsync signal in number of samples
`HSYNC_MAX`	8	maximum length of a hsync signal in number of samples
`VSYNC_MIN`	9	minimal length of a vsync signal in number of samples
`VSYNC_MAX`	16	maximum length of a vsync signal in number of samples

F. TV - Maze (500 points)

The city Bertie needs to go through is a maze - quite literally. GPS is made impossible to use by local authorities. The only alternative is a special video stream service that (after registration) shows the map of the city with your current position marked. Your task is to find the way out, the path between the marked location and the border of the map and direct Bertie through the maze of narrow streets.

Input

A tv stream showing the map. White pixels are streets, black pixels are walls and a little grey area marks Bertie’s current position.

Output

A series of single-character directions in one long line, each causing Bertie to move to the adjacent cell.

direction character	**meaning**
n	go north (up)
e	go east (right)
s	go south
w	go west

Example

Input 0 can be decoded into

```
###########
#         #
### ##### #
# # #S  # #
# # ### # #
#       # G
###########
```

and one of the solutions is

```
eesswwwwnnnneeeeeessssse
```

G. TV - Count (500 points)

One of the TV channels broadcasted a strange program after the evening news. It showed the faces of all the politicians recently elected, being happy or sad. Bertie is interested in the general mood in the local parliament, because it is rumored that some new laws regarding motorbike traffic regulations are forthcoming - and he thinks happy politicians result in lenient laws.

Your task is count smiling and sad faces in recordings of the program.

Input

A tv stream showing faces.

Output

The output should contain two numbers separated by whitespace: first the number of happy faces then the number of sad faces.

H. TV - air defense stream (3000 points)

Bertie only got the visa necessary for this far away country by agreeing to a number of ridiculous demands. Some of these pertained to personal hygiene and other inconsequential trifles, but one has given him a bit of pause before he signed the necessary forms. It is apparently the custom of this country that for the duration of one day, he would be asked to participate in the defense efforts against the hated and villainous neighboring country.

As it turns out, the neighboring country has mounted a ferocious attack by air. Friendly airspace is swarming with enemy aircraft. The local ground-to-air defense arsenal may be adequate, but friend-or-foe identification is a problem.

The country has a number of TV cameras pointed towards the skies. An army of volunteers watches the screens, keeping their eyes peeled to sort friend from adversary. Their input data controls flak cannons and other assorted surprises.

Enemy aircraft look pretty similar to friendly ones - they even have the same sort of identification codes - but luckily, they prefer sleek, angled wings.

- The front edge of friendly wings is fairly close to perpendicular to the body of the aircraft
- The front edge of enemy wings is angled at least 45° backwards

All aircraft are identified by a sort of barcode painted on their bodies. The barcode is read from nose to tail. A wide bar encodes a `1` bit, a narrow bar a `0` bit. The last bar (nearest to the tail) is a parity bit. If the count of `1` bits before it is even, it is narrow (`0`); if it's odd, it is wide (`1`) - thus, the number of wide bars in such a barcode is always even.

Bertie receives his live TV stream via **UDP broadcast** on UDP port **30002**. He is requested to report his findings by **TCP connection** to **server.ch24.org** port **31000**.

For every aircraft he sees and identifies, he should send via TCP (on a single line terminated by LF):

- **enemy 01010101** - for enemy aircraft; or
- **friend 01010101** - for friendly aircraft (always inserting the correct barcode).

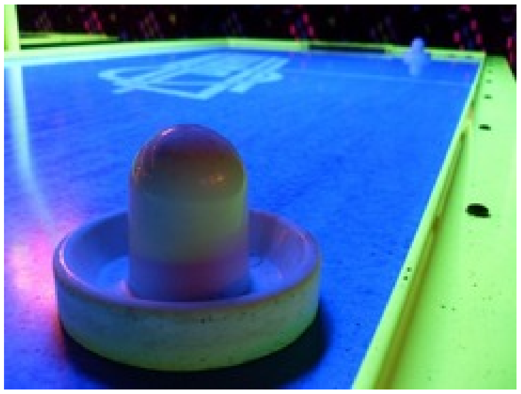
These commands will only work when the given aircraft is still visible **-** otherwise it’s too late for air defense to do anything!

For every correctly identified plane, Bertie is rewarded **10 Cents of Courage -** he is however penalized **1 Cent of Courage** for every misidentified plane (which is a very good deal; people before have received harsher punishment for needlessly sending civilians to their deaths). One plane may be identified only once (correctly or incorrectly).

This task uses scaled scoring. Your team will receive a score given by comparing the number of Cents of Courage collected against other teams, the score scaled accordingly. The team who collects the most receives maximum score, which is **XYZ**. All teams receive the same TV stream, and see the same planes; but more than one team may identify a single plane (and all teams will get the proper score accordingly).

Note that this task may contain secrets, which may lead its finders to more Coins or other rewards.

IJ. Air hockey



Bertie’s gang often meets other motorcycle gangs. Although the members are all mostly peaceful human beings, from time to time there are confrontations. To spare motorcycles and human lives, gangs have worked out a less costly method for settling their disputes: they all visit a local pub that runs a special air hockey game built for 30 players enabling at most 30 gangs to play the same game. Bertie quickly figures a simple rule - who rules air hockey rules the land. After several years of training at the game, he realizes this is something better done by computers (the Gang Code doesn’t prohibit that).

Your task is to develop the best air hockey AI (to be used by Bertie in the future) that beats any other player - allowing Bertie to rule all other gangs with an iron fist. All other teams at Challenge24 got the same task. Bertie wants to pick the best AI by the end of the day. To compare AIs at Challenge24, there will be tournaments where all 30 teams shall participate. Tournaments are scheduled at regular, preset times.

To be able to test your AI, you’ll get your own 30-player table (this is what we call the "practice server"). You can play this game alone any time during the contest. The practice table comes with a few automated puck-guns and there are some for-score practice setups to prove your AI can handle some simple situations.

Game rules and physics

To avoid corner cases (literally) the table is mostly a round shape. Each player controls a small circular **mallet** by accelerating it within a given **zone**. There is a handful of **pucks**, which are also circular, that can freely move on the desk (or rather, slightly above the desk as they are hovering on high pressure air, providing essentially friction-free movement).

Players also have a short goal line in the middle of the backside of their zone. Any time a puck crosses the goal line, the team gets negative score and the puck is returned on the table (after a certain delay).

A player of this game should control their mallet to kick incoming pucks away from their goal line and shoot them in other teams’ goal lines.

There is a way teams can receive positive scores. Ideally the team who "scored" the goal should receive a positive score - but deciding which team scored the goal is slightly complicated (because there are many pucks and many teams in the game at the same time). Every puck carries a sorted, timestamped list of *which* team’s mallet touched it and *when*. When pucks collide, the lists of the two pucks are merged, and both pucks leave with the same merged list. If a team touched both pucks, only the more recent touch is recorded in the merged list. If there’s a goal, the puck’s carried list is checked. The team with the most recent touch will receive a positive score, except if it’s the same team who receives the goal; in that case, the team with the second most recent touch will receive the positive score.

Specific sizes, velocity and acceleration limits and scores are transmitted as part of the communication protocol.

The physics should mostly work as expected, but you can always make measurements to determine the precise dynamics.

Networking, protocol

The standard game table features an Ethernet connector for control and monitoring. The firmware detects the position of all moving objects precisely and reports them in a series of UDP broadcast **status update** packages periodically. Furthermore each player (at most 30) shall connect to a specific TCP port where the firmware sends all game constants and then accepts mallet control commands from the player. Both protocols are simple, line based text protocols. Fields written in *italic* are numbers.

Status updates

UDP status messages	
format	**description**
t *K P*	status at time tick *K*; there are multiple UDP packets per tick and this is *P*th one (starting from 0)
g *A B*	team *A* scored a goal against team *B* (*A* can be -1 to indicate "no team"); both are integers
m *T X Y XV YV*	team *T*'s mallet is at (*X*, *Y*) and has velocity (*XV*, *YV*)
p *ID T1 T2 X Y XV YV ANG ANGV*	puck *ID* is at *(X, Y)* with velocity *(XV, YV)*, turning angle *ANG*, angular velocity *ANGV*. *T1* and *T2* are the teams who last touched the puck (most recent and second most recent, respectively).
e	marks the last packet of a status update

Each message is terminated by a newline character.

Each UDP packet starts with a **t** message and contains zero or more **g**, **m** and **p** messages. The last packet of a tick ends with an **e** message.

Team id, tick and packet numbers are integers; velocities and other physical properties are floating point numbers. Velocities are measured in unit/seconds.

Angles are radians, positive is counterclockwise.

Constants and mallet control

TCP initial constants (server -> player)	
format	**description**
playernum *N*	number of players (int)
scorestart *S*	starting score for each team (int)
scoremin *S*	minimum score a team can achieve (int)
scoremax *S*	maximum score a team can achieve (int)
scoregoalsend *S*	score awarded for shooting a goal (int)
scoregoalrecv *S*	score awarded for receiving a goal (negative int)
puckr *R*	radius of puck (float)
malletr *R*	radius of mallet (float)
maxaccel *A*	max acceleration of mallet (float)
maxvel *V*	max velocity of mallet (float)
ticktime *N*	time between two status updates in milliseconds (int)
wall *T X0 Y0 X1 Y1*	line segment (*X0,Y0*)-(*X1,Y1*) is a wall or a goal line of team *T* if *T* >= 0
zone *T X Y R*	team *T*'s zone is a circle centered at (*X*,*Y*) with radius *R*
score *T S*	team *T* has score *S*
done	sent after sending the above commands, to terminate the list of constants
These constants are sent by the server immediately after the connection is established. Team ID and scores are always int, coordinates, sizes and other physical properties are floats.	

TCP mallet control (player -> server)	
format	**description**
XA YA	acceleration vector for player's mallet (float)
Can be sent any time. Last sent value is used in each tick (it doesn't make sense to send more than one control message in a single tick).	

Disclaimer

We are using a packet switched network that may cause at least the following errors (listed in order of expected frequency, most common ones first):

- variable delay in delivering UDP packets
- variable delay in delivering TCP packets (not synchronized to UDP packets in any way)
- UDP packets lost
- order of UDP packets is mixed up

Servers

- Tournament: TCP **server.ch24.org:20000**
- Tournament: UDP broadcast **21002**
- Practice: TCP **server.ch24.org:22000**
- Practice: UDP broadcast **22200**

I. Air hockey practice (1000 points)

Each team gets their own practice game table. A simple protocol is provided to control the behavior of the automated part of the table.

TCP mallet control (player -> server)	
format	**description**
start	starts game with current settings (resets scores first)
stop	stops game
length *L*	set length of game in seconds (int)
autopucknum *N*	how many pucks the generator will try to generate (int)
autopuckavgtime *T*	expected delay of respawning a single puck after a goal (or after start); float
autopuckangvar *V*	angle variance of speed of new pucks in [0,1] (0 means shoot at the center of your zone, 1 means shoot at your whole zone); float
autopuckfrom *F1 F2 F3 ...*	list of generators, possible values are 0..*N*-1 (a value *Fn* means there is a generator next to team *Fn*, to their right); *Fn* is always integer
genpuck *X Y VX VY ANG ANGV*	create new puck manually; floats
level *L*	load level *L*; issue a "start" command to start playing it
Can be sent any time.	

The server answers a command with a single line prefixed with *ack*, except for the **level** command, for which the server answers multiple lines prefixed with *task*. Task lines specify the task to be done for the given level. There are 10 levels numbered from 0 to 9.

Scoring

You may try to solve a level any time without penalty; completing it is worth 100 points.

J. Air hockey tournament (2500 points)

There will be three tournaments where the AIs of all 30 teams compete against each other.

A tournament consists of several rounds. Rounds starts periodically every 15 minutes during a tournament. In each round you should connect to the tcp control server to get initial constants for the round and to be able to control your mallet. At the end of the round the server will close the tcp connection.

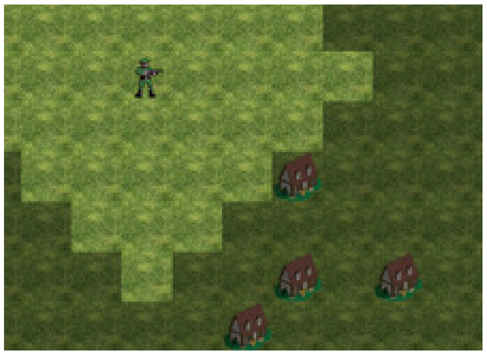
Scoring

Each playing team has an in-game score based on the number of received and scored goals. At the end of a round the team gets real scores based on its in-game ranking compared to other teams in that round.

The maximum score a team can get for a round is different in each tournament.

Tournament #	**Rounds**	**Max score for one round**
1	6	20
2	20	30
3	36	50

K. WW3 (3500 points)



Although air hockey is the common method for resolving problems among the gangs, Bertie prefers to have a plan B, just in case. The alternative to a friendly game is to purchase heavy weapons and tanks and start a large scale war. Before things escalate to this undesired level, Bertie wants to make sure he has the winning strategy. Especially that he does not have any secret weapons, he can only buy the same stuff anyone else can.

Your task is to develop an AI that can come up with the winning strategy in a war. You will need to test your AI against other teams' AIs in tournaments before it qualifies for use by Bertie.

Rules

Terminology

The map is a grid of X*Y rectangular tiles. Each tile has one of the following types:

TYPE	SYMBOL	DEFENSE	SLOW	REMARK
grass	g	+2	1	
forest	f	+4	2	
town	t	+3	4	reinforcements, new units, income
water	w	n/a	n/a	impassable

Players have a budget, may own towns (a town can be owned by only one player at a time) and arbitrary amount of the following units:

TYPE	SYMBOL	BASE_HP	MOVES	ATTACK_RANGE	REMARK
scout	s	10	10	1	
tank	t	50	4	1	
artillery	a	20	6	4	
infantry	i	10	4	1	can take over towns

Turns

There is a constant number of 2 players participating in a battle. A battle is split into N turns (N % 2 == 0). Each player may move units, buy new units, buy reinforcements in his/her own turn and attack enemy units. The turn begins by increasing the budget by (W*TW) where W is the "town income constant" and TW is the number of towns owned by the player. After the player announced end of turn, the turn timed out (see constant T) or no more moves are possible, the next turn of the next player starts.

Moving units

Units can be moved only to adjacent tiles and can not move diagonally. Each unit has a move-counter that is reset to MOVES+M at the begining of each turn, where MOVES depends on the unit type (see the above table) and M is the "move bonus constant". By moving to an adjacent tile, move-counter is decreased by the SLOW factor of the target tile. The move-counter is evaluated before the move is executed: if move-counter would be decreased below zero as a result of the move, the move fails. If move-counter is zero, the unit can not move or attack or reinforce any more in that turn. If a unit attacks, or it is reinforced, move-counter is automatically decreased to zero. Units can not move on water.

Other units affect movements the following ways:

1. a unit cannot move onto a tile occupied by another unit
2. after moving from one of the neighboring 8 tiles of an enemy unit to another of those 8 tiles, the move-counter is set to 0 (the enemy unit exerted "a zone of control"). To put it another way: if the before and after position both has the same enemy unit in the neighboring 8 tiles, the move-counter is set to 0.

Attacks

An unit can fire any tile not further than ATTACK_RANGE tiles away, counted in moves on grass. Any unit staying on the tile under attack will lose truncate((A+B)/D)+1 of its HP, where B is the BASE_HP of the attacking unit, D is the DEFENSE of the attacked tile, A is the "attack bonus constant". If the HP of a unit decreases to or below zero, the unit is destroyed (removed from the game).

Fog of war

Town ownership and enemy units are reported only for areas visible to any of the units controlled by the player. Units see as far as they could move on grass. In the beginning of each turn the server reports all visible enemy unit positions and their HP. If an enemy unit previously seen is not reported again, the location of that unit is unknown.

Towns

If an infantry unit is moved into a town, the town becomes owned by the player who controls the infantry. If any other unit type moves into an enemy town, the enemy player loses the town and the new status of the town is "not owned". An unit with non-zero move-counter staying in a town owned by the same player can be reinforced; this means the HP of the unit will be increased by R, the budget of the user decreased by R. If the resulting HP is larger than the BASE_HP of the given unit type, it is truncated to BASE_HP.

The user may buy a new unit at any empty town owned. Cost of a unit is truncate(ATTACK_RANGE*(BASE_HP+MOVES)/C), where C is the unit cost modifier constant. A new unit has 0 move-counter in the turn it was bought.

Goal of a battle and Scoreboard

After N turns of the battle, the battle ends. The current budget of the two contestants are compared. The player with the more money gets +1 points on the scoreboard, the player with the less money gets 0 points. In case of tie, both players get 0.

A series of battles are played in a tournament governed by a swiss system. At the end of each tournament teams get actual contest scores depending on their position in the scoreboard. When a tournament starts, points of all teams on the scoreboard are reset to zero, but initial pairing of teams (by the swiss system) will depend on their ranking in the previous tournament.

Game dependent constants

id	meaning
A	attack bonus (integer; A >= 0)
C	unit cost modifier (integer; C >= 1)
M	move bonuns constant (integer; M >= 0)
N	number of turns for this battle (integer; N % 2 == 0; N > 30)
P	number of players participating in the battle (P = 2)
Q	the actual map (string of X*Y characters)
R	reinforcement value (integer; R >= 0)
T	turn timeout in seconds (integer; T >= 2; tolerance: 0.02 + network delays)
W	town income (integer; W >= 2)
X	width of the map (integer; 256 > X >= 16)
Y	height of the map (integer; 256 > Y >= 16)

Communication protocol

A command is a three characters long keyword, a space, a keyword-specific arguments and a \n. An argument may contain any character except \n and space. \n is the newline character (ASCII 10). The server may send commands any time, the client may send commands only when the server gives permission.

In command description **bold** means string literal, *italic* means name of an argument and _ means space (ASCII 32). Besides the explicitly indicated _ spaces, there are no other whitespace characters in the commands.

For each command sent by the client, either an ACK or an ERR is sent back. Optionally a series of other server commands may follow. The client is not required to wait for the ACK/ERR, and may insert multiple commands at once, however, in this case, the client is responsible for pairing requests with answers (order guaranteed).

Commands from the server

format	meaning
BTL _ *opponent*	A new battle starts against *opponent*, which is either a '?' when the opponent is unspecified or an integer between 0 and 29.
CNS _ *name* _ *value*	Set value of a constant. *value* may be an integer or a string but may never contain \n. Note: *name* is always 1 character long. Map (constant called Q) is a string representing a row-major matrix of the terrain (tile types) with their SYMBOLs. First character is for coord x=0;y=0. A series of CNS commands are sent immediately after BTL, before the first TRN. It is guaranteed that all known constants are sent once (in random order, except for X and Y, which will always precede Q).
MOV _ *whose* _ *fx* _ *fy* _ *tx* _ *ty* _ *type* _ *HP*	Unit moved. *whose* is **f** for friendly unit (owned by the player) and **e** for enemy (opponent contestant's units). *fx* and *fy* are the previous, *tx* and *ty* are the current coordinates of the unit. *type* is the SYMBOL of the unit type, *HP* is the current HP of the unit. HP is never 0. fx;fy may be the same as tx;ty to announce HP change.
NEW _ *whose* _ *x* _ *y* _ *type* _ *HP*	Position of a new unit. *x* _ *y* are the same as fx/fy in MOV. This message is generated if an unseen unit is sighted or a new unit bought (in sight). HP is never 0.
DEL _ *whose* _ *x* _ *y* _ *type* _ *HP*	Position of a unit which disappeared. *x* _ *y* are the same as fx/fy in MOV. This message is generated if a unit is going out of sight (coordinates are the last known ones) or when the unit is destroyed (HP == 0).
ATK _ *ux* _ *uy* _ *tx* _ *ty*	Unit at coordinates *ux* and *uy* attacked tile at *tx* _ *ty*. *ux*;*uy* or *tx*;*ty* may be -1;-1, in case source or target tile is not visible. ATK is not sent if neither of those are visible. NOTE: a MOV/DEL command may follow.

TWN _ *x* _ *y* _ *new owner*	Town at coordinates *x y* is owned by *new owner*, which has the same format as the "whose" field of MOV but also can be **n** which means the town is not owned.
TRN _ *whose*	A new turn starts for *whose*. *whose* is the same as for MOV, plus it may be 'o' when the battle is over. In case of the player's own turn, a BDG and an RDY commands are also sent.
BDG _ *budget*	Reports the player's budget (happens after TRN).
ACK	Last command is valid and is being executed. Zero or more status updates and a closing RDY will be sent.
ERR _ *code*	Last command was bogus and can not be executed at all. *code* is the error code explaining the problem with the request. If the player can send more commands in this turn, an RDY is sent.
RDY	No more status updates to be sent, waiting for player's commands.

Commands from the client

format	meaning
atk _ *ux* _ *uy* _ *tx* _ *ty*	Unit at coordinates *ux* and *uy* should attack tile at *tx* _ *ty*. NOTE: a DEL command may follow.
mov _ *ux* _ *uy* _ *seq*	Unit at coordinates *ux* and *uy* should move *seq*. *seq* is a string sequence of **s** (for south), **n** (for north) **e** (for east), **w** (for west). If any of the moves in the sequence would cause an error, the whole seqence is ignored and an error is reported. If ACK is sent, a series of NEW/DEL commands may follow.
rnf _ *ux* _ *uy*	Reinforce unit at coordinates *ux* and *uy*. A MOV command will follow (for updating the HP). (For costs refer to section *Towns*.)
buy _ *ux* _ *uy* _ *type*	Buy a new unit at coordinates *ux* and *uy* of type *type* One or more NEW command(s) or an ERR command will follow. (For costs check section *Towns*.)
end	Ends turn.

Error codes

code	meaning	result of
01	unit not owned	atk, mov
02	no unit at coordinates	atk (ux/uy), mov, rnf
03	no town at coordinates	rnf, buy
04	out of range	atk
05	not enough money for operation	rnf, buy
06	town is not empty	buy
07	town is not owned	buy, rnf
08	failed move or operation: move-counter is too low	mov, atk, rnf
09	failed move: tile already occupied	mov
10	syntax error: invalid keyword	n/a
11	syntax error: invalid argument	(any)
12	syntax error: coordinates out of range	(any)
13	command sent out of own turn **- no RDY follows**	(any)
14	number of arguments mismatch	(any)
15	connection is closed because of a new connection from the same team	(spontaneous)

Communication example

Please test the protocol on the practice server.

TCP ports

10001	practice server control (raw CLI, use telnet client)
10002	practice server game connection (see protocol)
10003	practice server telnet view connection (use telnet client, press ’h’ for help)
10012	tournament server game connection (see protocol)
10013	tournament server telnet view connection (use telnet client, press ’h’ for help)

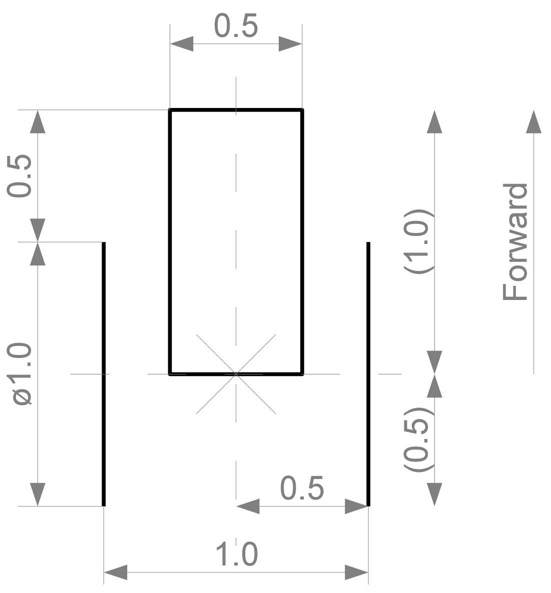
L. Wheelchair (1000 points)

Even completely normal use of a motorbike - according to all road laws and regulations - can be quite dangerous. Doing wheelies at high speed in between of heavy artillery is not the best life insurance policy. Bertie and his gang are aware of the dangers, and they're conscious of - among other things - wheelchair accessibility of public buildings.

Navigating a badly designed building in a wheelchair can be an arduous undertaking. Your task here is to find short paths through mazes using a wheelchair. Apparently the architects of these mazes haven't considered wheelchair accessibility their first concern...

The mazes are sets of walls, as line segments in two dimensions. The wheelchair has an initial position and orientation, and a given target point. To solve a maze, the wheelchair may be pushed forward or backward, and turned around either wheel, finally to reach the vicinity of the target point - without having the wheelchair intersect any walls.

The Wheelchair



The collision zone of the wheelchair is modeled using two line segments for the wheels, and one rectangle for the body (and the legs of the user).

The origin point of the wheelchair's coordinate system is the intersection of the wheel axis and the centerline.

The wheels are parallel; their distance is 1 unit. Their diameter (the length of the line segments) is 1.

The body rectangle is 0.5 wide and 1 long (along the centerline). It is positioned exactly between the wheels. Its longer side begins right at the wheel axis, and ends 1 unit forward - thus the resulting bounding box of the wheelchair is 1 unit wide and 1.5 units long.

Input Format

- **Line 1:** `wall_cnt chair_x chair_y chair_dir target_x target_y`
- **Lines 2 .. (wall_cnt+1):** `wall_x1 wall_y1 wall_x2 wall_y2`

Where:

- `wall_cnt` (integer) number of wall segments
- `chair_x, chair_y` (float) initial position of wheelchair (origin point)
- `chair_dir` (float) initial direction of wheelchair
- `target_x, target_y` (float) position of target point
- `wall_x1, wall_y1, wall_x2, wall_y2` (float) wall segment coordinates

Direction is given in radians. `0.0` points "east" (`X` positive), `π/2` points "north" (`Y` positive).

Output Format

The output is a series of actions for the wheelchair that gets it to the target point without running into any walls on the way. One action goes on one line. Each line should consist of an **action letter**, a single space, then a float **amount**.

Available actions:

- **Push** (**`P`** *n*): push wheelchair forward by *n* units. Negative amount pulls the wheelchair backwards.
- **Left** (**`L`** *n*): lock left wheel, turn wheelchair counterclockwise around left wheel by *n* radians.
- **Right** (**`R`** *n*): lock right wheel, turn wheelchair counterclockwise around right wheel by *n* radians.

The turning actions rotate the wheelchair around the centerpoint of the specified wheel, by the specified amount. The wheel's centerpoint stays in place, but the origin of the wheelchair will change (describing an arc with *r=0.5* around the wheel's centerpoint). Maximum allowed turning amount is **+/- 2 π**.

During pushes and turns, the collision zones of the wheelchair may not intersect a wall segment, at any point in time (i.e. not only the end position and orientation should be considered).

After the last action, the wheelchair's origin must be within **0.5 units** from the target point.

Scoring

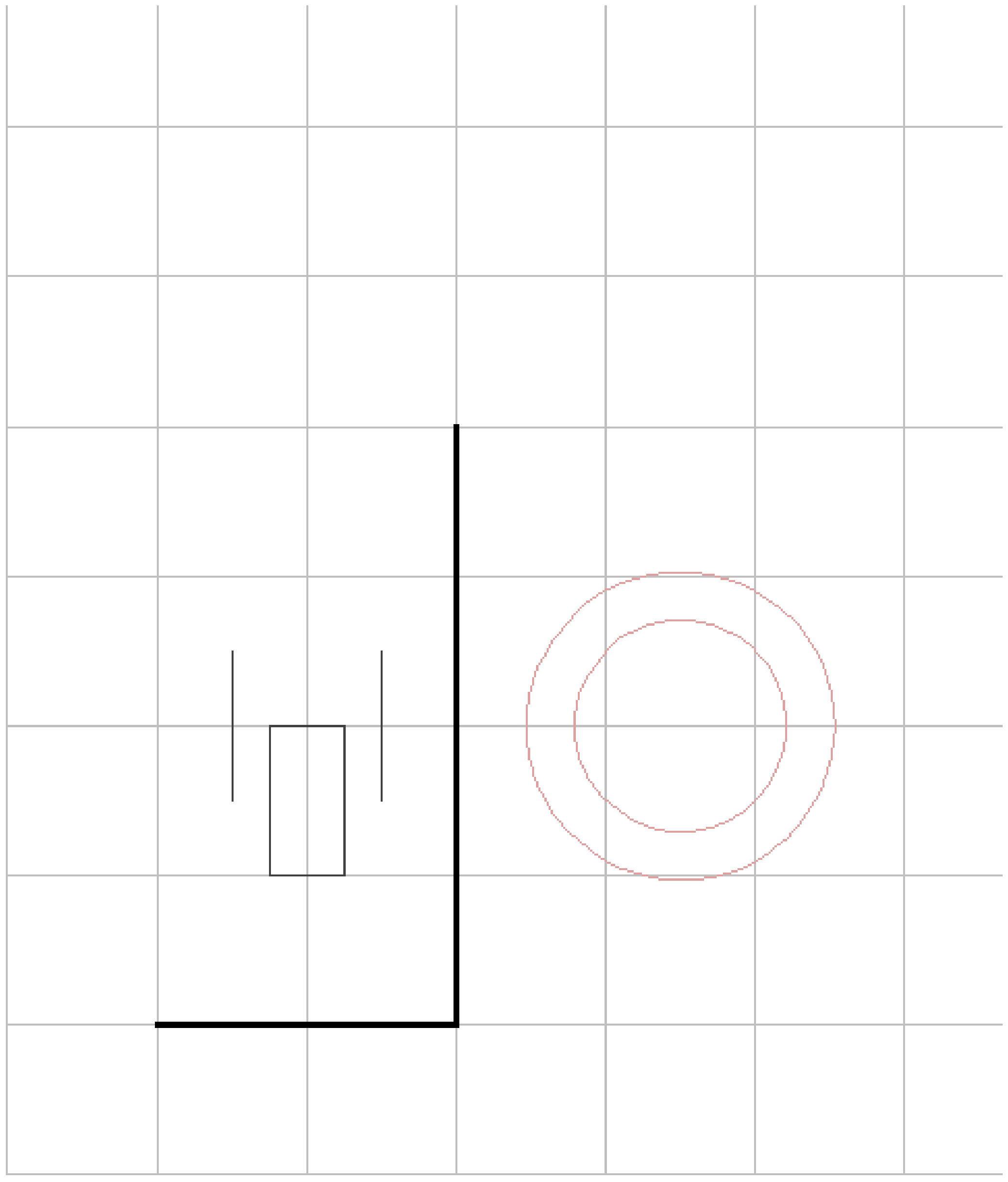
This task uses scaled scoring, based on the wheelchair's traveled distance. For each input, the solution with the shortest traveled distance gets maximum score (100 points).

The traveled distance is calculated using the wheelchair’s origin point as reference.

Sample Input

```
2 2 3 -1.57079632679 4.5 3
1 1 3 1
3 1 3 5
```

A graphical interpretation of the above input:



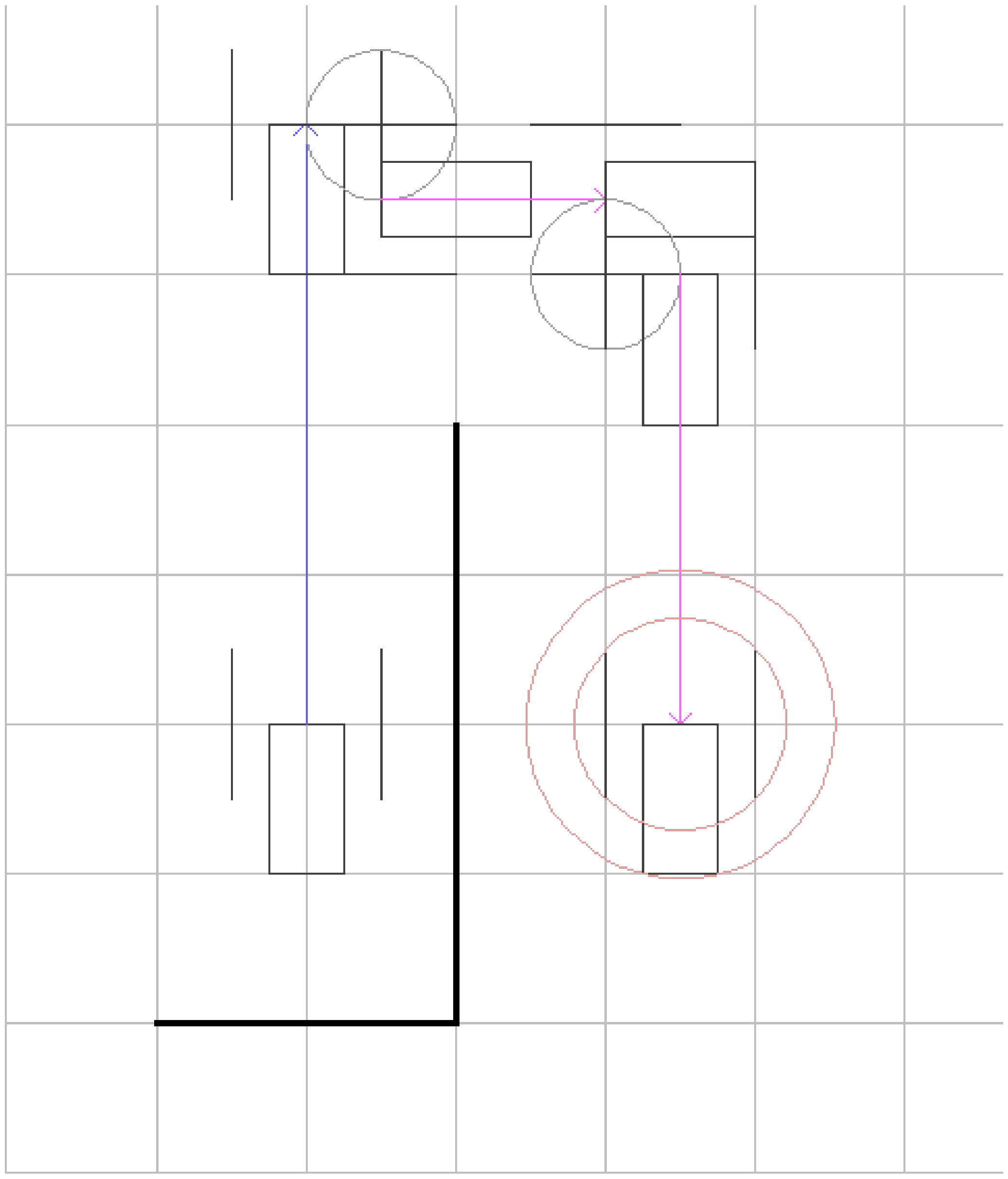
Sample Output

```
P -4
L 1.57079632679
P 1.5
R -1.57079632678
P 3
```

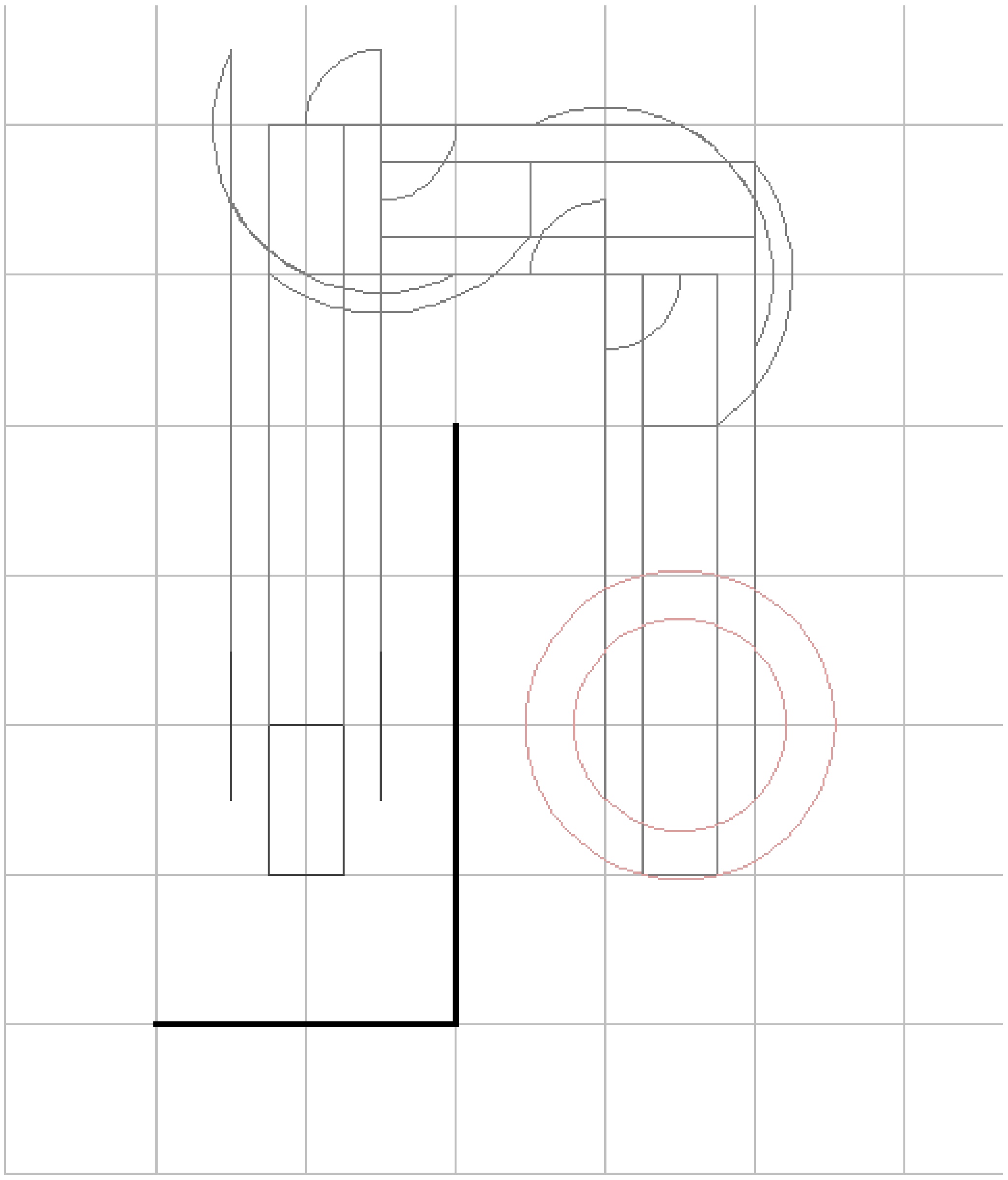
Explanation:

1. Pull wheelchair back 4 units
2. Lock left wheel, rotate around left wheel counterclockwise by 90° ($\pi/2$)
3. Push wheelchair forward 1.5 units
4. Lock right wheel, rotate around right wheel clockwise by 90° ($\pi/2$)
5. Push wheelchair forward 3 units

A graphical interpretation:



Outline of wheelchair collision zones on this path:



Floating Point Disclaimer

This task involves a lot of FP calculations, and as such, there may be inaccuracies. If your solution barely evades a wall by a nanounit in your calculations, in our evaluator it may just barely intersect it; thus it may be advisable to leave a bit of breathing room. To assist in debugging such problems, our evaluator will report the position and nature of collisions in failure messages.