Содержание

1 Условие задачи

На клеточном поле расположен прямоугольный лабиринт. В некоторых стенках лабиринта могут быть двусторонние порталы. Ваша программа управляет роботом, которому неизвестна карта лабиринта. Однако после каждого движения робота программе сообщается: либо удалось сместиться на соседнюю клетку, либо робот не переместился, так как встретил стенку между клетками. Роботу по очереди выдаются задания вида «прийти в некоторую точку». При этом целевая точка задаётся не координатами, а картой окрестности точки. Требуется выполнить все задания с минимальными затратами энергии.

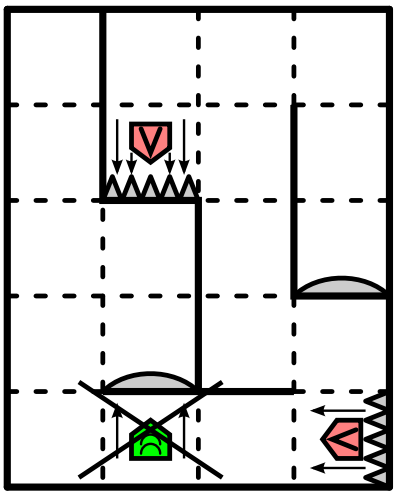
1.1 Общее описание

Лабиринт занимает прямоугольную область на клеточном поле. Все клетки поля пустые, однако между соседними клетками могут стоять стенки (вертикальные или горизонтальные перегородки). Гарантируется, что прямоугольная область лабиринта полностью окружена стенами, а снаружи лабиринта стенок нет.

У робота в любой момент есть текущее направление (одно из четырёх). Робот может поворачиваться на месте на угол, кратный 90°, а также двигаться вперёд. При движении вперёд робот успешно перемещается на соседнюю клетку в его текущем направлении, при условии, что на пути нет стенки. Если же стенка на пути есть, то робот не перемещается. В любом из этих двух случаев результат хода сообщается программе участника. В начальный момент робот направлен «вверх», а начальное положение робота вашей программе не сообщается.

Вместо некоторых стенок могут быть перегородки-порталы. У портала одна сторона «рабочая», а с другой стороны портал ведёт себя как обычная стенка. Все порталы в лабиринте разбиты на пары. При входе робота в портал с рабочей стороны он выходит в парном портале

с рабочей стороны. При этом робот не диагностирует факт прохождения через портал. Следует заметить, что рабочие стороны парных порталов могут быть ориентированы по-разному, в этом случае текущее направление робота меняется при прохождении портала. Пример прохождения портала роботом нарисован на картинке справа.



Роботу выдаётся последовательность заданий. Каждое задание — попасть в определённую клетку-цель в лабиринте. Придя в целевую клетку, робот должен сообщить об этом. Если в этот момент он действительно находится в целевой клетке, то ему выдаётся следующее задание, а иначе остаётся текущее. На каждую посылку сообщения затрачивается энергия. Гарантируется, что в любой момент робот может добраться до любой клетки лабиринта, и не может выйти за пределы лабиринта.

Роботу о каждом задании сообщается только карта некоторой окрестности клетки-цели. Окрестность имеет форму квадрата, сторона имеет длину 5. В центре окрестности расположена цель. Для каждой пары соседних клеток окрестности сообщается, есть ли между ними преграда. Преградами считаются как обычные стенки, так и перегородки-порталы. Карта окрестности ориентирована точно так же, как сам лабиринт. Заметим, что в лабиринте может быть несколько клеток с такой же окрестностью, как у цели.

У робота также имеется дальномер. При использовании дальномера программе сообщается, сколько успешных шагов вперёд может совершить робот (без поворотов) до столкновения со стенкой. Это число может оказаться бесконечным, что означает, что можно бесконечно долго идти вперёд по циклу благодаря порталам. На каждое использование дальномера затрачивается энергия.

Кроме того, робот может оставлять метки в посещённых клетках. Всего у робота есть только 3 метки: `A`, `B`, и `C`. После каждого перемещения робота ему сообщаются все метки, расположенные в клетке, в которой он оказался. Робот может выставлять и убирать метки в его текущей клетке, затрачивая при этом энергию. Ни в какой момент не может быть двух одинаковых меток в лабиринте.

1.2 Протокол

Ваша программа управления роботом общается с программой проверки через каналы (pipes). Вам нужно читать информацию от программы проверки из стандартного потока ввода, и писать команды в стандартный поток вывода. В конце каждой команды должен быть перевод строки (в конце каждого ответа также стоит перевод строки). После вывода каждой команды нужно выполнить flush потока вывода.

В начале работы вашей программы на вход подаётся общая информация о тесте, а также информация о первом задании. Далее ваша программа должна выдать первую команду. После этого на вход поступает результат выполнения этой команды. Затем ваша программа выдаёт вторую команду, получает ответ, выдаёт третью команду, и т.д. Если ваш робот обошёл все цели (об этом можно узнать по результатам команд), вашей программе управления следует завершиться с нулевым кодом возврата.

Описание каждой команды начинается с угла поворота, кратного 90°, и меньшего 10^6 по модулю. Перед выполнением команды текущее направление робота поворачивается на этот угол по часовой стрелке без затрат энергии. После поворота выполняется сама команда. Например, при выполнении команды "`-90 M`" робот сначала повернётся против часовой на 90°, а затем постарается переместиться вперёд на одну клетку. Поворот всегда сохраняется, даже если выполнение команды (например, движения) завершается неуспешно.

Результат выполнения команды всегда начинается с флага успешности выполнения команды. В начале описания каждой команды указано, сколько на неё затрачивается энергии. Энергия на выполнение команды тратится всегда, даже если команда завершается неуспешно.

Ниже подробно описываются все возможные команды робота и ответные сообщения для них.

- `[угол] M` — команда движения вперёд (1 Дж).

  Результат выполнения зависит от того, что находится между текущей клеткой робота и соседней вдоль его текущего направления клеткой .

  1. Если ничего нет, то робот перемещается в эту соседнюю клетку, направление робота не меняется, и команда завершается успешно.
  2. Если стоит обычная стенка, то робот остаётся на месте, направление робота не меняется, и команда завершается неуспешно.
  3. Если стоит портал, причём робот попадает в него с рабочей стороны, то робот проходит через портал. При этом его направление изменяется на направление выхода из парного портала, и он перемещается в клетку, смежную парному порталу с рабочей стороны. Команда завершается успешно.
  4. Если стоит портал, но робот заходит в него не с рабочей стороны, то происходит всё то же, что и в случае 2.

  Не забывайте, что перед собственно движением вперёд всегда выполняется поворот на `[угол]`, который сохраняется в любом случае.

  `[успех] [метки]` — ответное сообщение о результате.

  Вместо `[успех]` стоит `Y`, если команда выполнилась успешно, и `N` в противном случае. Вместо `[метки]` записываются все метки, находящиеся в клетке робота после выполнения команды. Метки записываются в алфавитном порядке без пробелов. Если меток нет, то пишется `0`. Даже если команда выполнена неуспешно, метки в клетке робота всё равно сообщаются.

- `[угол] G` — команда отправки сообщения о достижении цели (100 Дж).

  Эту команду нужно посылать, когда ваша программа управления полагает, что робот находится в клетке-цели текущего задания. Если это действительно так, то команда выполняется успешно, и роботу даётся следующее задание. Если нет, то команда завершается неуспешно, и текущее задание робота не меняется.

  `[успех]` — ответное сообщение о результате.

  Вместо `[успех]` стоит одно из:

– `Y` — робот успешно выполнил задание. Ниже приводится окрестность следующей клетки-цели.
– `N` — робот находится не в клетке-цели, задание ещё не выполнено.
– `FINISHED` — робот выполнил последнее задание. При получении такого ответа следует завершить программу.

Если роботу выдаётся следующее задание, то выводится окрестность цели в следующих 9 строках, по 9 символов в каждой. Символ в нечётных строке и столбце обозначает клетку лабиринта, символ в позиции с разной чётностью строки и столбца обозначает место для перегородки, а символ в чётных строке и столбце ничего не обозначает. Для вертикальных преград используется символ ASCII 124, а для горизонтальных — ASCII 45. В центральной клетке (пятая строка и столбец) записана буква `G`, обозначающая цель. Любой символ в чётных строке и столбце является плюсом (ASCII 43). Все остальные символы заполняются точками (ASCII 46).

Два примера описания окрестностей представлены ниже. На левом примере показаны все возможные стенки, поэтому он невозможен на практике (нет связности).

```
.|.|.|.|.
-+-+-+-+-
.|.|.|.|.
-+-+-+-+-
.|.|G|.|.
-+-+-+-+-
.|.|.|.|.
-+-+-+-+-
.|.|.|.|.
```

```
.|.|.....
-+.+-+.+-
.....|.|.
-+.+-+.+-
....G....
-+-+-+-+.
.........
-+.+.+-+-
.|.|...|.
```

- `[угол] D` — команда для использования дальномера (3 Дж).

  Дальномер определяет, сколько шагов вперёд может успешно выполнить робот из текущего положения до первого столкновения со стенкой (положение робота при этом не изменяется).

  `[успех] [расстояние]` — ответное сообщение о результате.

  `[успех]` равен `N`, если можно бесконечно долго идти вперёд из текущего положения робота. В этом случае `[расстояние]` равно 10^9. В противном случае `[успех]` равен `Y`, а в `[расстояние]` записано целое неотрицательное число — количество клеток, которое можно двигаться вперёд до столкновения.

- `[угол] P [метка]` — команда для выставления метки (5 Дж).

  В `[метка]` должна быть записана одна из букв `A`, `B` или `C` — название метки, которую вы хотите поставить. Если такая метка ещё не установлена в лабиринте, то метка выставляется в текущей клетке робота, и команда завершается успешно. В противном случае команда завершается неуспешно, и ничего не изменяется.

  `[успех]` — ответное сообщение о результате.

  `[успех]` равно `Y` в случае успеха (метка поставлена) и `N` в случае неудачи (метка занята).

- `[угол] T [метка]` — команда для убирания метки (5 Дж).

  В `[метка]` должна быть записана одна из букв `A`, `B` или `C` — название метки, которую вы хотите убрать. Если такая метка находится в текущей клетке робота, то она убирается, и команда завершается успешно. В противном случае команда завершается неуспешно, и ничего не изменяется.

  `[успех]` — ответное сообщение о результате.

  `[успех]` равно `Y` в случае успеха (метка убрана) и `N` в случае неудачи (такой метки здесь нет).

При запуске вашей программы ей на вход подаются следующие данные. В первой строке записаны параметры, переданные на вход генератору при создании теста. Всего записано семь целых чисел (часть из них описана ниже): T — номер теста, W — размер лабиринта по горизонтали в клетках, H — размер лабиринта по вертикали в клетках, γ — количество дополнительно убранных стенок (см. далее), P — количество порталов в лабиринте, T — могут ли быть разворачивающие порталы (1 = да, 0 = нет), G — общее количество целей в тесте. В следующих строках задана окрестность цели для первого задания в таком же формате, в котором окрестности задаются в ответе для команды отправки сообщения о достижении цели.

1.3 Система оценки

После завершения работы вашей программы очки за тест определяются как

$$10^6 k - E$$

где k — количество выполненных заданий, а E — энергия, потраченная роботом к моменту выполнения последнего выполненного им задания (в Дж). Например, если робот выполнил 789 первых заданий, причём к моменту выполнения последнего из них он потратил 55 555 энергии, то решение получает 788 944 445 очков за тест, даже если после этого робот тратит всю оставшуюся батарею, бесцельно бродя по лабиринту. Если решение не выполняет ни одного задания, то оно набирает 0 баллов за тест. Ёмкость батареи робота в начале работы всегда равна 10^6 Дж.

Если в какой-то момент истекает время работы программы, или превышается ограничение по памяти, или нарушен формат вывода команды, то программа завершается досрочно и ей начисляются очки за тест по состоянию дел на тот момент. Очки за тест начисляются аналогично и в случае некорректного завершения. Ограничение по процессорному времени работы вашей программмы составляет 5 секунд в сумме на каждый тест. Ограничение по памяти — 256 мегабайт.

1.4 Тесты жюри

Тесты жюри делятся на четыре группы в порядке увеличения сложности:

1. Известные тесты: тесты, для которых можно посмотреть лабиринт и начальное положение робота.

2. Тесты без порталов.
3. Тесты, в которых порталы не меняют направление робота при прохождении.
4. Тесты с произвольными порталами.

Характерные размеры лабиринта в тестах каждой группы примерно одинаковые. Всего есть 50 тестов жюри. Количество тестов в каждой группе пропроционально её номеру. Все тесты из группы 1 можно найти в материалах задачи.

Все тесты жюри были сгенерированы одной программой-генератором, которая есть в ваших материалах. Программа принимает на вход набор параметров, которые подробно описаны ниже. В материалах также есть bat-скрипт, в котором указаны конкретные параметры, использовавшиеся при создании тестов жюри. За исключением тестов из группы 1, все остальные тесты жюри были сгенерированы с неизвестным вам rand-seed.

1.5 Проверка по время тура

Во время тура вы можете посылать ваши решения в систему тестирования на вкладке «Сдать». Посылать нужно исходный код программы управления роботом в виде одного файла. В системе тестирования программа будет автоматически скомпилирована и запущена на некотором наборе тестов. Результат запуска можно увидеть на вкладке «Результаты».

Тестирование в системе проводится на подмножестве из 14 тестов жюри. Используются тесты с номерами $1, 2, 3, 4, 5, 6, 15, 16, 22, 30, 31, 36, 41, 46$. Жюри может изменить набор тестов в системе во время тура: в таком случае об этом будет опубликована новость. Кроме того, в очереди в любой момент может находиться не больше одного решения от каждого участника: при отправке нового решения все непротестированные к этому моменту решения того же участника будут убраны из очереди.

Для каждой посылки на вкладке «Результаты» показывается следующая информация. В строке «Время:» показаны время посылки, используемый компилятор и ссылка на текст посланной программы. В строке «Результат:» записана строка вердиктов проверки на тестах (по одному символу на каждый тест). В строке «Баллы:» через пробел записаны очки за каждый тест. И вердикты, и очки записаны в порядке возрастания номера теста.

Ваше решение может получать следующие вердикты:

- `A: Accepted` — решение успешно прошло тест, найдя все цели.
- `W: Wrong Answer` — решение нарушило протокол взаимодействия, либо энергия закончилась до того, как были найдены все цели.
- `T: Time Limit Exceeded` — решение не уложилось в отведённое процессорное время.
- `D: Deadlock` — решение не уложилось в отведённое астрономическое время.
- `M: Memory limit exceeded` — решение не уложилось в отведённое ограничение по памяти
- `R: Run-time error` — решение вернуло код ошибки, отличный от нуля.
- `S: Security violation` — решение совершило действие, запрещённое правилами.

Во время тура вы можете задавать вопросы жюри на вкладке «Вопросы». Задавать вопросы в непонятных ситуациях рекомендуется: жюри постарается отвечать на все вопросы, на которые оно в состоянии ответить. Кроме того, рекомендуется периодически просматривать новости в системе тестирования.

1.6 Итоговые результаты

После конца тура от каждого участника берётся одно последнее посланное решение в качестве окончательного. Будьте внимательны и осторожны в конце тура: если окажется, что последнее решение не работает, это будет вашей проблемой.

Выбранное решение участника будет протестировано на всех 50 тестах жюри. В итоговом рейтинге участники ранжируются по сумме очков, набранных на этих тестах. Чем больше сумма очков, тем лучше место.

2 Материалы

В новостях в системе тестирования доступна ссылка для скачивания материалов соревнования (`materials.exe`). Пароль к архиву:

aperturescience

Прежде всего рекомендуется:

1. Распаковать архив материалов, перейти в директорию с содержимым, используя Far Manager или cmd.
2. `gen_tests.bat`: сгенерировать 50 тестов, подобных тестам жюри.
3. Скомпилировать решение `Task.cpp` или `Task.java` при помощи `cl`, `g++` или `javac`.
4. `WithJsVis.bat Task.exe 34.in`: запустить решение с интерактивным визуализатором. `WithJsVis.bat "java Task" 34.in`: то же самое в случае Java.
5. `TestSolution.bat Task.exe`: запустить локальное тестирование на 50 сгенерированных тестах.

В архиве присутствуют:

Файл или директория	Описание содержимого
`interactor.exe`	Проверяющая программа.
`runner.py`	Простая программа для совместного запуска решения и проверяющей программы.
`WithInterSimple.bat`	Скрипт для совместного запуска решения и проверяющей программы.
`InterRunner.exe`	Используемая в тестирующей системе программа для совместного запуска решения и проверяющей программы.
`WithInterStrict.bat`	Скрипт для совместного запуска решения и проверяющей программы.
`generator.exe`	Программа для генерации тестов.
`gen_tests.bat`	Скрипт для генерации тестов с теми параметрами, которые использовались при создании тестов жюри.
`tests1\*.in`	Все тесты жюри из группы 1.
`Task.cpp`	Очень глупая программа управления роботом на языке C++.
`Task.java`	Очень глупая программа управления роботом на языке Java.

`statement.pdf`	Это условие задачи.
`js-visualizer.exe`, `js-vis\*`	Интерактивный визуализатор “JS”.
`WithJsVis.bat`	Скрипт для совместного запуска решения и визуализатора “JS”.
`ne-vis\*`	Интерактивный визуализатор “noteye”.
`WithNeVis.bat`	Скрипт для совместного запуска решения и визуализатора “noteye”.
`TestSolution.bat`	Скрипт для локального тестирования решений.

2.1 Генератор

При запуске генератора (`generator.exe`) нужно указать следующие параметры командной строки:

1. `testNumber`: номер теста T, который будет передан программе управления роботом при запуске.
2. `width`: ширина W желаемого лабиринта.
3. `height`: высота H желаемого лабиринта.
4. `additionalWaysCount`: количество дополнительно убранных стенок γ (описано ниже).
5. `portalsCount`: количество порталов P в лабиринте.
6. `ifPortalsCanChangeDirection`: могут ли порталы изменять направление робота при прохождении (1 — да, 0 — нет).
7. `tasksCount`: количество заданий G для робота.

Эти параметры передаются программе управления роботом сразу после запуска.

Тест генерируется следующим образом. Сначала создаётся лабиринт из $W \times H$ клеток, в котором есть все возможные стенки. Затем случайные $W \cdot H - 1$ стенок убираются таким образом, чтобы из любой клетки лабиринта можно было добраться до любой другой. Потом дополнительно убирается ещё γ стенок среди оставшихся. Далее случайные P пар стенок соединяются порталами. При этом в зависимости от флага создаются либо произвольные порталы, либо порталы, не меняющие направление при прохождении. Наконец, начальное положение робота и положения G клеток-целей генерируются случайным образом.

Тест записывается в стандартный поток вывода в следующем формате. В первой строке записано целое число T — номер теста ($1 \le T \le 50$). Во второй строке записаны целые числа W и H — ширина и высота лабиринта ($5 \le W, H \le 100$), γ — количество дополнительно убранных стенок. Следующие $2H + 1$ строк содержат по $2W + 1$ символов каждая: в них описывается лабиринт в формате, аналогичном формату выдачи окрестности в ответе на команду о достижении цели. Далее записаны целые числа P — количество порталов в лабиринте ($0 \le P \le 100$), f — могут ли порталы разворачивать робота при прохождении ($f = 0$ или $f = 1$). В следующих P строках описываются порталы, для каждого из них указано два его входа. Каждый вход задан координатами x_i и y_i инцидентной клетки, и направлением, по которому из неё нужно пойти, чтобы войти в портал ($0 \le x_i < W$, $0 \le y_i < H$, направление — один из символов `D`, `U`, `R`, `L`). Далее записаны координаты x, y, определяющие начальное положение робота ($0 \le x < W$, $0 \le y < H$). Затем записано целое число G — количество целей ($G = 1000$). В оставшихся G строках записаны координаты клеток-целей в порядке их

выдачи роботу.

Клетка лабиринта с координатами (x, y) в текстовом представлении расположена в строке с номером $2y$ и в столбце с номером $2x$. Всюду в условии действует естественное соглашение о направлениях: при движении «вниз» увеличивается номер строки, при движении «вправо» увеличивается номер столбца, а направления «вверх» и «влево» противоположны направлениям «вниз» и «вправо» соответственно.

Пример запуска генератора из командной строки:

```
generator.exe -1 6 5 3 1 0 1000 >input.txt
```

При этом создаётся лабиринт размера 6×5 с 3-мя дополнительно удалёнными стенками, 1-им порталом и 1000-ей клеток-целей. Порталы не меняют направление, а номер теста записывается, равный -1. Добавление `>input.txt` в конце командной строки перенаправляет стандартный поток вывода генератора в файл в именем `input.txt`.

2.2 Интерактор и запуск

Проверяющая программа (`interactor.exe`) принимает команды робота через стандартный поток ввода и выдаёт ответы на эти команды в стандартный поток вывода. Программа принимает следующие параметры командной строки при запуске:

1. `labyrinth-file`: имя входного файла теста, сгенерированного генератором (`generator.exe`).
2. `score-file`: файл, в который проверяющая программа записывает набранные роботом очки (указывать необязательно).

Вы можете запускать проверяющую программу саму по себе, писать команды руками в консоль и читать ответы на консоли.

Скрипт `WithInterSimple.bat` запускает вместе проверяющую программу и решение (т.е. программу управления роботом). Стандартный поток ввода каждой программы перенаправляется в стандартный поток вывода другой программы, благодаря чему обеспечивается автоматическое общение решения и проверяющей программы. Скрипт основывается на `runner.py`, и *отличается от используемого в системе тестирования*. Обратите внимание, что при этом *время работы не замеряется*.

Скрипт `WithInterStrict.bat` тоже предназначен для совместного запуска проверяющей программы и решения. Основная разница заключается в том, что он основан на программе `InterRunner.exe`, которая *используется в системе тестирования* для проверки решений участников. При запуске при помощи данного скрипта накладываются ограничения на время работы и используемую память, указанные выше. По завершению работы выводится код возврата и полученный вердикт тестирования.

Оба скрипта принимают один или два параметра:

1. Имя исполняемого файла вашего решения.
2. Имя входного файла теста (если не указано, то используется `input.txt`).

Количество баллов, набранное решением, выводится в `result.txt`.

К примеру, можно запустить проверку решения на входном файле `input.txt`.
Командная строка для проверки решения на C++:

```
WithInterStrict Task.exe
```

Командная строка для проверки решения на Java:

```
WithInterStrict "java Task"
```

2.3 Визуализаторы

В материалах есть два визуализатора для задачи. Оба основаны на интерактивном принципе: они работают точно так же, как проверяющая программа `interactor.exe`, но дополнительно показывают происходящие с роботом события визуально.

Визуализатор "JS" позволяет просматривать происходящее в браузере. Страница в браузере открывается автоматически при запуске. Для запуска рекомендуется использовать скрипт `WithJsVis.bat`. Скрипт принимает те же два параметра, что и скрипты для запуска проверяющей программы (например `WithInterSimple.bat`). Кроме того, третьим необязательным параметром скрипту может быть передан номер команды робота, с которого визуализатор начнёт показ (нумерация шагов с нуля).

Например, для просмотра решения на Java можно использовать командную строку:

```
WithJsVis "java Task"
```

Можно начать визуализацию с 10000-ой команды, используя командную строку:

```
WithJsVis "java Task" input.txt 10000
```

Визуализатор "noteye" основан на одноимённой графической библиотеке для roguelike игр. Для запуска рекомендуется использовать скрипт `WithNeVis`. Скрипт принимает те же параметры, что и скрипты для запуска проверяющей программы (вроде `WithInterSimple.bat`).

Например, для просмотра решения на C++ можно использовать командную строку:

```
WithNeVis Task.exe
```

2.4 Локальное тестирование

Вы можете тестировать ваши решения локально на любом наборе тестов, который вам нравится. Разумеется, результаты такого тестирования не имеют никакой силы в соревновании, кроме того, они могут отличаться от получаемых результатов в системе тестирования.

Для локального тестирования можно запустить скрипт `TestSolution.bat` с одним параметром: исполняемым файлом решения, которые требуется проверить. Все файлы с расширением `.in` в данной директории будут использованы в качестве входных файлов тестов. Результаты проверки складываются в файл `AllResults.txt` в текущей директории.