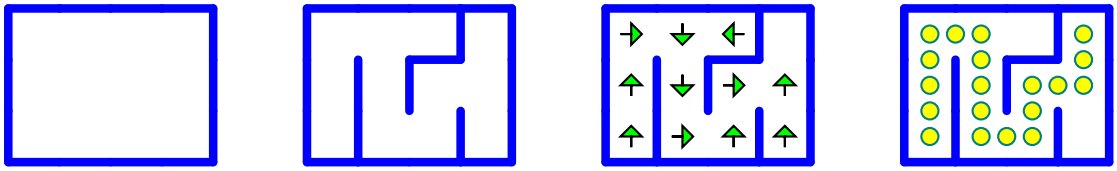
Рисование лабиринтов

Общая цель в этой тренировке — научиться генерировать случайный лабиринт, проходить его и рисовать результат в формате SVG.



Условие тренировки состоит из трёх частей. В первой дано общее описание всех этапов задачи. Во второй рассказано, как выводить ответ в формате SVG. В третьей части приведено разбиение общей задачи на конкретные части, которые можно сдать в систему проверки. Рекомендуется ознакомиться с условием полностью, прежде чем приступать к решению.

Общее описание

Итак, лабиринт состоит из квадратных клеток: h клеток в высоту (рядов) и w клеток в ширину (столбцов). Между каждой парой клеток, имеющих общую сторону — либо проход, либо стена. Кроме того, весь лабиринт окружён сплошной стеной. Внутри же лабиринта стены устроены так, что между каждыми двумя клетками есть ровно один простой путь — то есть путь, клетки в котором не повторяются. Ряды клеток лабиринта нумеруются с нуля сверху вниз, а столбцы — с нуля слева направо. Задача — найти путь в этом лабиринте из левой нижней клетки в правую верхнюю.

Лабиринт строится так.

Этап решения «shuffle»:

Запишем последовательность из всех возможных внутренних стен лабиринта — отрезков длиной в одну клетку, являющихся стороной каких-то двух клеток лабиринта. Сначала отсортируем эти возможные стены в порядке следования их центров сверху вниз, а при равенстве — слева направо. А именно: сначала слева направо все вертикальные стены в ряду 0, кроме двух крайних, затем слева направо все горизонтальные стены между рядом 0 и рядом 1, потом вертикальные стены в ряду 1, кроме крайних, и так далее.

Теперь случайно перемешаем эту последовательность стен. Для проверки требуется, чтобы алгоритм перемешивания был детерминированным, поэтому мы будем пользоваться следующим генератором псевдослучайных чисел.

```
RND_MULT =    1664525
RND_ADD  = 1013904223
P2_32    = 4294967296

rnd_cur = s

rnd_value (n):
    rnd_cur := (rnd_cur * RND_MULT + RND_ADD) mod P2_32
    return (rnd_cur * n) div P2_32
```

Другими словами, у генератора есть состояние `rnd_cur` — целое число от 0 до $2^{32}-1$ включительно. Изначально состояние — это некоторое заданное число s. Когда требуется очередное случайное целое число, равновероятно выбранное от 0 до $n-1$ включительно, сначала состояние умножается на `RND_MULT`, затем к нему прибавляется `RND_ADD`, после чего новым состоянием становится остаток от деления результата на 2^{32}. Наконец, требуемое случайное число получается умножением состояния на n, а затем делением нацело на 2^{32}. Можно убедиться, что распределение полученного числа близко к равновероятному.

Такой способ получения псевдослучайных чисел называют линейным конгруэнтным генератором. В реализации на языке C или C++ удобно хранить `rnd_cur` в 32-битном беззнаковом целом типе `unsigned`, а умножение `rnd_cur * n` производить в 64-битном целом типе `long long`.

Использовать этот генератор мы будем так:

```
rnd_shuffle (a, n):
    for i := 0, ..., n - 1:
        k := rnd_value (i + 1)
        a[i] <-> a[k]
```

Другими словами, чтобы перемешать последовательность $a_0, a_1, \ldots, a_{n-1}$, сначала сгенерируем псевдослучайное число `k := rnd_value (1)` (оно всегда равно нулю) и поменяем a_0 и a_k местами (то есть ничего не сделаем, кроме того, что вызов `rnd_value` поменял состояние генератора). Затем сгенерируем следующее псевдослучайное число `k := rnd_value (2)` (оно равновероятно окажется нулём или единицей) и поменяем a_1 и a_k местами, и так далее. Можно доказать, что, если `rnd_value` генерирует настоящие равномерно распределённые случайные числа, то этот алгоритм равновероятно перемешивает последовательность `a` одним из $n!$ способов.

Такой алгоритм перемешивания называется «тасование Фишера— Йетса».

Этап решения «decide»:

Итак, у нас есть случайно перемешанная последовательность возможных внутренних стен лабиринта. Будем рассматривать эти стены в полученном порядке. Для каждой стены найдём ответ на вопрос: если добавить её в лабиринт, можно ли будет из каждой клетки дойти до каждой, минуя стены? Если ответ утвердительный, добавим стену в лабиринт. Если же нет, пропустим её.

Можно доказать, что после рассмотрения всех возможных стен между каждыми двумя клетками лабиринта останется ровно один простой путь.

Когда лабиринт построен, нужно найти путь из левого нижнего угла (старт) в правый верхний (финиш). Сделаем это в два этапа.

Этап решения «arrows»:

Наш лабиринт таков, что из каждой клетки есть ровно один простой путь до финиша. Рассмотрим все эти пути. В каждом из них есть первый шаг: чтобы приблизиться к финишу, нужно пойти вверх, вниз, влево или вправо. Отметим этот факт стрелочкой в соответствующую сторону. Исключение составляет сама клетка финиша — на ней не будет никакой стрелочки.

Этап решения «path»:

Когда стрелочки в каждой клетке уже стоят, найти путь от старта до финиша не составляет труда. Начнём из клетки старта и будем двигаться по стрелочкам, пока не придём на клетку финиша.

Вывод ответа

Формат SVG (Scalable Vector Graphics) — это способ рисовать простые картинки так, чтобы их можно было:

- редактировать в текстовой форме,
- увидеть в любом современном браузере,
- масштабировать без потери качества.

Полное описание формата SVG было бы слишком длинным для этого условия. Да и использоваться будет лишь небольшое его подмножество, как можно увидеть в примерах к задачам. Для более подробного ознакомления с форматом можно взглянуть в какой-нибудь учебник, например, `https://www.w3schools.com/graphics/svg_intro.asp`.

Вывод в каждой задаче — одна картинка в формате SVG. В ней могут понадобиться следующие группы элементов: стены, стрелочки и путь.

Каждый отрезок стены длиной в одну клетку, включая сплошную стену вокруг лабиринта, выводится отдельно. Эти отрезки выводятся в порядке сортировки их центров сверху вниз, а при равенстве — слева направо. А именно: сначала слева направо выводятся все горизонтальные стены над рядом 0, затем слева направо все вертикальные стены в ряду 0, далее слева направо все горизонтальные стены между рядом 0 и рядом 1, потом вертикальные стены в ряду 1 и так далее. При рисовании расстояние между соседними клетками равно 24 (здесь и далее все размеры указаны в условных единицах). Стена имеет толщину 4. На картинке должно быть ровно столько места, чтобы поместились все клетки и все стены.

Стрелочки выводятся в порядке сортировки сверху вниз, а при равенстве — слева направо. А именно: сначала слева направо выводятся все стрелочки в ряду 0, затем все стрелочки в ряду 1, и так далее. Центр стрелочки совпадает с центром клетки. Для стрелочек расстояние углов от центра равно 5, а толщина линий равна 1.

Круги, которыми отмечается путь, выводятся в порядке следования от старта к финишу. А именно: сначала выводится круг на клетке старта, затем круг посередине между клеткой старта и второй клеткой пути, далее круг во второй клетке пути до финиша, и так далее. Последним выводится круг на клетке финиша. Круги имеют радиус 4 и толщину линии 1.

Посмотрите, как именно нужно выводить элементы лабиринта, в примерах:
`http://acm.math.spbu.ru/trains/171024_b17`.

Поскольку ваши ответы проверяются автоматически, соблюдайте порядок элементов и их форматирование как можно точнее!

Разбиение на задачи

Далее идут краткие описания задач. Каждая задача — это шаг в какую-то сторону на пути к цели. В задаче под буквой «`X`» нужно читать входные данные из файла «`x.txt`» и выводить в файл «`x.html`». Такие расширения выбраны для удобства локального тестирования: файл «`x.txt`» можно создать в любом текстовом редакторе, а файл «`x.html`» можно открыть в любом современном браузере и посмотреть на получившуюся картинку. Первый тест по каждой задаче и ответ к нему выложены здесь:
`http://acm.math.spbu.ru/trains/171024_b17`.

Ограничения на входные данные следующие: $2 \leqslant h \leqslant 50$, $2 \leqslant w \leqslant 50$, $0 \leqslant s \leqslant 10^9$. Во всех задачах ограничение по времени — 2 секунды, ограничение по памяти — 256 мебибайт.

A. (border)

В первой строке заданы h и w.

Выведите внешние стены лабиринта.

B. (border, walls)

В первой строке заданы h, w и количество стен n. В следующих n строках заданы внутренние стены в случайном порядке. Гарантируется, что после установки этих стен между каждыми двумя клетками лабиринта будет ровно один простой путь. Каждая стена задана числами r_1, c_1, r_2, c_2 — координатами (ряд и столбец) двух своих концов. Ряды и столбцы стен нумеруются с нуля, второй конец всегда правее или ниже первого.

Поставьте все заданные стены.

Выведите получившийся лабиринт.

C. (border, walls, arrows)

В первой строке заданы h, w и количество стен n. В следующих n строках заданы внутренние стены в случайном порядке. Гарантируется, что после установки этих стен между каждыми двумя клетками лабиринта будет ровно один простой путь. Каждая стена задана числами r_1, c_1, r_2, c_2 — координатами (ряд и столбец) двух своих концов. Ряды и столбцы стен нумеруются с нуля, второй конец всегда правее или ниже первого.

Поставьте все заданные стены. Затем в каждой клетке, кроме финиша, поставьте стрелочку: в какую сторону идти, чтобы прийти к финишу.

Выведите получившийся лабиринт, а также стрелочки в его клетках.

D. (decide, border, walls)

В первой строке заданы h, w и количество стен m. В следующих m строках заданы все возможные внутренние стены в случайном порядке. Каждая стена задана числами r_1, c_1, r_2, c_2 — координатами (ряд и столбец) двух своих концов. Ряды и столбцы стен нумеруются с нуля, второй конец всегда правее или ниже первого.

Рассматривая стены в заданном порядке, ставьте те из них, после установки которых между каждыми двумя клетками лабиринта по-прежнему существует путь.

Выведите получившийся лабиринт.

E. (decide, border, walls, arrows)

В первой строке заданы h, w и количество стен m. В следующих m строках заданы все возможные внутренние стены в случайном порядке. Каждая стена задана числами r_1, c_1, r_2, c_2 — координатами (ряд и столбец) двух своих концов. Ряды и столбцы стен нумеруются с нуля, второй конец всегда правее или ниже первого.

Рассматривая стены в заданном порядке, ставьте те из них, после установки которых между каждыми двумя клетками лабиринта по-прежнему существует путь. Затем в каждой клетке, кроме финиша, поставьте стрелочку: в какую сторону идти, чтобы прийти к финишу.

Выведите получившийся лабиринт, а также стрелочки в его клетках.

F. (shuffle, decide, border, walls)

В первой строке заданы h, w и начальное значение s для генератора псевдослучайных чисел.

Перемешайте все возможные внутренние стены генератором псевдослучайных чисел. Затем, рассматривая стены в получившемся порядке, ставьте те из них, после установки которых между каждыми двумя клетками лабиринта по-прежнему существует путь.

Выведите получившийся лабиринт.

G. (decide, border, walls, arrows, path)

В первой строке заданы h, w и количество стен m. В следующих m строках заданы все возможные внутренние стены в случайном порядке. Каждая стена задана числами r_1, c_1, r_2, c_2 — координатами (ряд и столбец) двух своих концов. Ряды и столбцы стен нумеруются с нуля, второй конец всегда правее или ниже первого.

Рассматривая стены в заданном порядке, ставьте те из них, после установки которых между каждыми двумя клетками лабиринта по-прежнему существует путь. Затем в каждой клетке, кроме финиша, поставьте стрелочку: в какую сторону идти, чтобы прийти к финишу. После этого найдите путь от старта к финишу по стрелочкам.

Выведите получившийся лабиринт, а также путь от старта к финишу.

H. (shuffle, decide, border, walls, arrows)

В первой строке заданы h, w и начальное значение s для генератора псевдослучайных чисел.

Перемешайте все возможные внутренние стены генератором псевдослучайных чисел. Затем, рассматривая стены в получившемся порядке, ставьте те из них, после установки которых между каждыми двумя клетками лабиринта по-прежнему существует путь. После этого в каждой клетке, кроме финиша, поставьте стрелочку: в какую сторону идти, чтобы прийти к финишу.

Выведите получившийся лабиринт, а также стрелочки в его клетках.

I. (shuffle, decide, border, walls, arrows, path)

В первой строке заданы h, w и начальное значение s для генератора псевдослучайных чисел.

Перемешайте все возможные внутренние стены генератором псевдослучайных чисел. Затем, рассматривая стены в получившемся порядке, ставьте те из них, после установки которых между каждыми двумя клетками лабиринта по-прежнему существует путь. После этого в каждой клетке, кроме финиша, поставьте стрелочку: в какую сторону идти, чтобы прийти к финишу. Наконец, найдите путь от старта к финишу по стрелочкам.

Выведите получившийся лабиринт, а также путь от старта к финишу.

J. (shuffle, decide, border, walls, arrows, path, fancy)

Формально по задаче нужно послать код, решающий предыдущую задачу (`I`). Однако, чтобы получить зачётные баллы по этой задаче, нужно выполнить два дополнительных условия:

- Если при сборке решения указать версию `FANCY_MAZE` (то есть, в случае C или C++, компилировать с флагом `-DFANCY_MAZE`, чтобы активировать код между `#ifdef FANCY_MAZE` и соответствующим `#endif`), оно должно выдавать украшенную картинку (например, изобразить стены кирпичными или путь в виде тропинки между деревьями) и при этом задействовать части формата SVG, не использованные в предыдущих задачах.
- Код решения должен быть читаемым и понятным.

Эти критерии трудно формализовать, поэтому фактические решения будут обсуждаться индивидуально с теми, кто будет сдавать эту задачу.