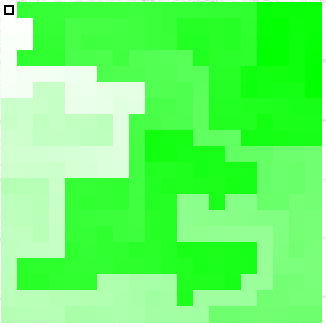


Лабиринты



**2020**

# Цель и задачи проекта

Цель: расширить горизонт знания о лабиринтах.

Задачи, которые необходимо решить для достижения поставленной цели:

1. Изучить способы представления лабиринтов;
2. Изучить алгоритмы генерации лабиринтов;
3. Изучить алгоритмы обхода лабиринтов;
4. Разработать программу сопровождения лабиринтов с возможностью:
   1. Генерации лабиринтов;
   2. Вычисления путей обхода лабиринтов;
   3. Расчета числовых характеристик лабиринтов.

# Классификация лабиринтов

Лабиринты в целом можно разбить по семи различным классификациям: размерности, гиперразмерности, топологии, тесселяции, маршрутизации, текстуре и приоритету [1].

**Размерность:** класс размерности определяет, сколько измерений в пространстве заполняет лабиринт, в частности, двумерные и трехмерные, более высокой размерности.

**Гиперразмерность:** классификация по гиперразмерности соответствует размерности объекта, двигающегося через лабиринт (точечный объект, линия, объемная фигура).

**Топология:** класс топологии описывает геометрию пространства лабиринта, в котором он существует как целое (плоский, края лабиринта соединены интересным образом).

**Тесселяция:** классификация геометрии отдельных ячеек, из которых состоит лабиринт. Существующие типы: ортогональные, треугольные, шестиугольные, с произвольными углами, фрактальные и т.д.

**Маршрутизация:** определяется типами проходов.

* **Идеальный:** «идеальным» называют лабиринт без петель или замкнутых цепей и без недостижимых областей. Также он называется лабиринтом с одиночным соединением (simply-connected Maze). Из каждой точки существует ровно один путь к любой другой точке. Лабиринт имеет только одно решение.
* **Плетёный (Braid):** «плетёный» означает, что в лабиринте нет тупиков. Также его называют лабиринтом с многократными соединениями (purely multiply connected Maze). В таком лабиринте используются проходы, замыкающиеся и возвращающиеся друг к другу (отсюда название «плетёный»), они заставляют тратить больше времени на ходьбу кругами вместо попадания в тупики.
* **Одномаршрутный (Unicursal):** под одномаршрутным подразумевается лабиринт без развилок. Одномаршрутный лабиринт содержит один длинный извивающийся проход, который меняет направление на всём протяжении лабиринта.
* **Разреженный:** разреженный лабиринт не прокладывает проходы через каждую ячейку, то есть некоторые из них не создаются. Это подразумевает наличие недостижимых областей, то есть он в некотором смысле противоположен плетёному лабиринту.
* **Частично плетёный:** частично плетёный лабиринт — это смешанный лабиринт, в котором есть и петли, и тупики.

**Текстура:** классификация по текстуре описывает стиль проходов при различной маршрутизации и геометрии.

* **Смещённость (Bias):** в лабиринте со смещёнными проходами прямые проходы склонны больше идти в одном направлении, чем в других.
* **Пролёты:** показатель «пролётов» определяет, как долго будут идти длинные проходы, прежде чем появятся вынужденные повороты. В лабиринте с низким показателем пролётов не будет прямых проходов длиннее трёх-четырёх ячеек, и он будет выглядеть очень случайным. В лабиринте с высоким показателем пролётов на протяжении лабиринта будет большой процент длинных проходов, из-за чего он будет походить на микрочип.

**Приоритет:** классификация показывает, что процессы создания лабиринтов можно разделить на два основных типа: добавляющие стены и вырезающие проходы. Обычно при генерации это сводится только к разнице в алгоритмах, а не к заметным отличиям лабиринтов. Один и тот же лабиринт часто генерируется обоими способами.

* **Добавление стен:** алгоритмы, для которых приоритетом являются стены, начинают с пустой области (или внешней границы), в процессе работы добавляя стены.
* **Вырезание проходов:** алгоритмы, приоритетом которых являются проходы, начинают со сплошного блока и в процессе работы вырезают в нём проходы.

# Представление лабиринта

|  |  |
| --- | --- |
| Лабиринт с тонкими стенами  Каждая ячейка имеет 4 стены.  Горизонтальный проход между двумя ячейками приводит к удалению правой стены левой ячейки и левой стены правой ячейки.  Вертикальный проход между двумя ячейками приводит к удалению нижней стены у верхней ячейки и верхней стены у нижней ячейки. | |
|  |  |
|  |  |
| Лабиринт с толстыми стенами  Каждая ячейка является или стеной или проходом. | |
|  |  |

# Ограничения, накладываемые на реализацию лабиринтов

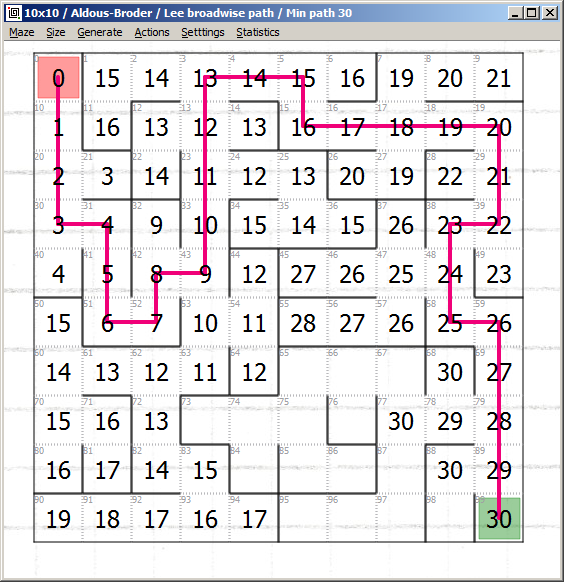
* Рассматриваются идеальные лабиринты, то есть, лабиринты без петель или замкнутых цепей и без недостижимых областей. Из каждой точки существует ровно один путь к любой другой точке. Лабиринт имеет только одно решение. С точки зрения программирования такой лабиринт можно описать как дерево, связующее множество ячеек или вершин.
* Рассматриваются алгоритмы генерации лабиринтов с разными приоритетами: с добавлением стен и с вырезанием стен
* Реализована поддержка лабиринтов произвольного размера по горизонтали и вертикали до 100 ячеек максимум в обоих направлениях. Однако для простоты визуализации используются квадратные лабиринты.
* Рассматриваются лабиринты с тонкими стенками.
* В волновом алгоритме поиска кратчайшего пути соседние ячейки классифицируются в смысле окрестности фон Неймана (4 ячейки по вертикали и горизонтали, в отличие от окрестности Мура, которая учитывает еще и диагональные ячейки).

# Программная реализация

Функции программы:

1. Генерация лабиринтов:
   * Выбор размера
   * Выбор алгоритма генерации
   * Сохранение лабиринта в файл
   * Загрузка лабиринта из файла
   * Редактирование лабиринта (установка/удаление стен)
2. Поиск пути между двумя ячейками
   * Выбор алгоритма поиска (волновой алгоритм, волновой алгоритм вглубь, алгоритм Флойда-Уоршелла)
   * Построение минимального остовного дерева (алгоритм Краскала)
   * Включение/выключение нумерации ячеек при проходе
3. Вычисление характеристик лабиринта

* Построение тепловой карты: для заданной ячейки А рассчитывается минимальный путь от А до каждой ячейки лабиринта. Чем длиннее путь тем более темным цветом закрашивается ячейка, чем короче, тем светлее.
* Расчет характеристик для лабиринтов, сгенерированных разными алгоритмами с вариацией по размеру с сохранением результатов в файлы. Допускается выбрать, какие характеристики рассчитываются:
  + Время создания
  + Количество тупиков
  + Количество проходов
  + Средняя длина пути при прохождении лабиринта сверху вниз
  + Среднее время нахождения пути при прохождении лабиринта сверху вниз
* Расчет отдельных характеристик для текущего лабиринта (количество тупиков и проходов)



*Общий вид программы*

# Алгоритмы генерации идеальных лабиринтов

## Двоичное дерево

Суть: проложить путь в случайном направлении из каждой клетки поля, например, либо наверх, либо вправо. За единицу времени обрабатывается только 1 клетка, следовательно, можно генерировать лабиринты бесконечного размера, сохраняя лишь конечный результат (лабиринт) без необходимости хранить какую-либо побочную информацию [2].

Результат работы алгоритма – случайное двоичное дерево, в котором из каждой клетки (вершины) есть ровно 1 путь по направлению к корню (родительской вершине), и, соответственно, ровно 1 путь к любой другой клетке. Как следствие, любая клетка имеет не более 3 соединений со своими соседями.

### Формальный алгоритм (для северо-восточного смещения)

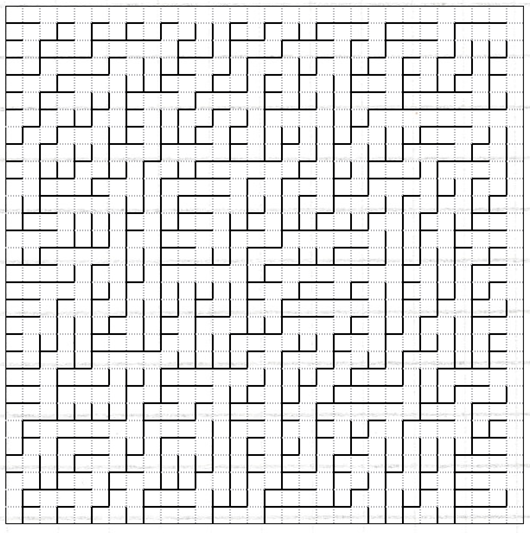
1. Начальное положение: все клетки со стенами;
2. Выбрать начальную клетку;
3. Выбрать случайное направление для прокладывания пути. Если соседняя клетка в этом направлении выходит за границы поля, прокопать клетку в единственно возможном направлении;
4. Перейти к следующей клетке;
5. Повторять 3-4 до тех пор, пока не будут обработаны все клетки;

### Плюсы

* Простая реализация;
* Высокая скорость работы;
* Возможность генерировать бесконечные лабиринты;

### Минусы

* Низкая сложность рисунка;
* Сильное смещение по диагонали;
* Однообразность сгенерированных лабиринтов;
* Два пустых коридора по сторонам лабиринта.



*Бинарное дерево*

## Sidewinder

Суть: алгоритм похож на алгоритм двоичного дерева, но летки рассматриваются не по отдельности, а множествами. Лабиринты получаются с преимущественно вертикальным или горизонтальным смещением (в зависимости от реализации), с отсутствием тупиков в их направлении. Смещение не так заметно и больше похоже на «спираль», которая плавно сменяет вертикальные и горизонтальные коридоры [2].

### Формальный алгоритм

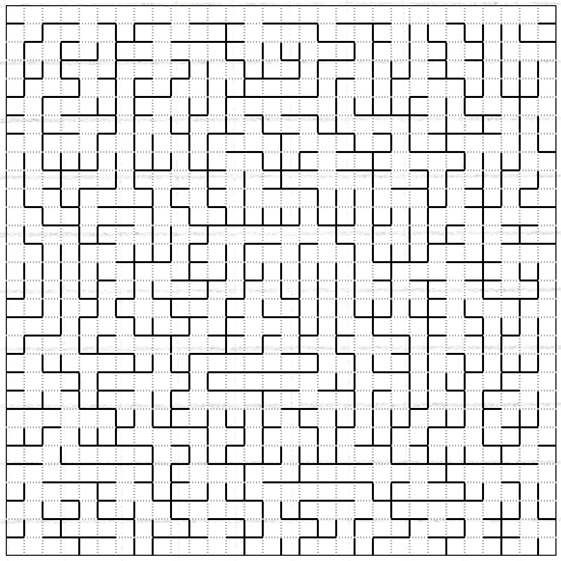
1. Выбрать начальный ряд;
2. Выбрать начальную клетку ряда и сделать её текущей;
3. Инициализировать пустое множество;
4. Добавить текущую клетку в множество;
5. Решить, прокладывать ли путь направо;
6. Если проложили, то перейти к новой клетке и сделать её текущей. Повторить шаги 4-6;
7. Если не проложили, выбираем случайную клетку множества и прокладываем оттуда путь наверх. Переходим на следующий ряд и повторяем 2-7;
8. Продолжать, пока не будет обработан каждый ряд;

### Плюсы

* Возможность генерировать бесконечные лабиринты;
* Только один пустой коридор;
* Более сложный рисунок, в отличии от алгоритма двоичного дерева;

### Минусы

* Сильное вертикальное смещение;

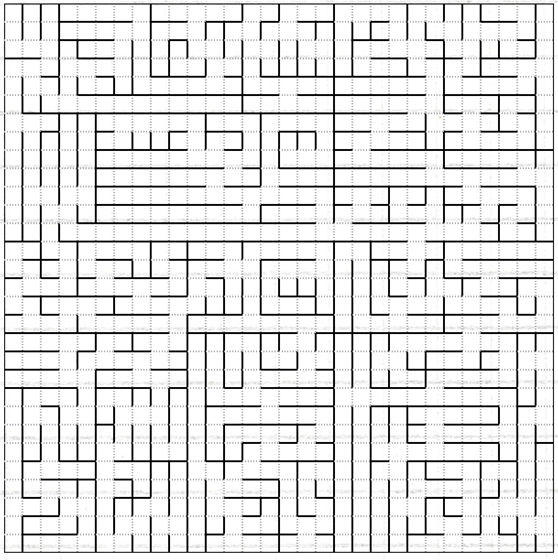


*Sidewinder*

## Рекурсивное деление

Суть: начинают пустого лабиринта, в котором есть только внешние стены. Лабиринт делится на 4 комнаты парой перпендикулярных стен. В трех из четырех стен пробиваются проходы. Каждая из комнат опять делится на 4. Разбиение на комнаты повторяется до тех пор, пока один из размеров комнаты не станет равен 1[4].

|  |  |
| --- | --- |
| Шаг 1. Деление на 4 части | Шаг 2. Проходы в трех из четырех комнат |
| Шаг 3. Деление следующей комнаты на 4 части | Шаг 4. Проходы в трех из четырех комнат |



*Recursive divide*

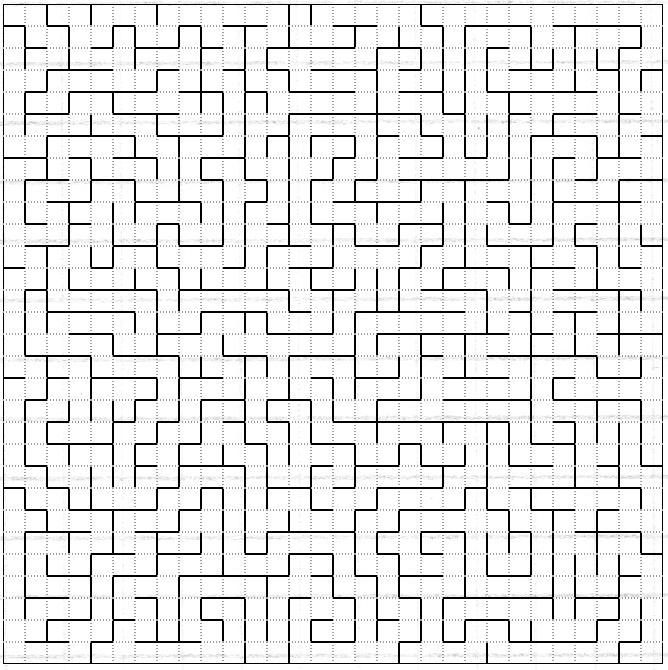
## Backtraking

Алгоритм позволяет создавать ветвистые лабиринты с очень длинными тупиковыми ответвлениями [3].

Предполагается, что изначально у каждой клетки есть стенки со всех четырех сторон, которые отделяют ее от соседних клеток.

Алгоритм:

1. Сделайте начальную клетку текущей и отметьте ее как посещенную.
2. Пока есть не посещённые клетки:
   1. Если текущая клетка имеет не посещённых «соседей»;
      1. Протолкните текущую клетку в стек;
      2. Выберите случайную клетку из соседних;
      3. Уберите стенку между текущей клеткой и выбранной;
      4. Сделайте выбранную клетку текущей и отметьте ее как посещенную.
   2. Иначе, если стек не пуст:
      1. Выдерните клетку из стека;
      2. Сделайте ее текущей;
3. Иначе:
   1. Выберите случайную не посещённую клетку, сделайте ее текущей и отметьте как посещенную.



*Backtracking*

## Алгоритм Олдоса-Бродера

Суть: случайное блуждание по полю с целью найти еще не посещённую клетку [5].

Все лабиринты, получаемые с помощью алгоритма, абсолютно случайны и не похожи друг на друга. Алгоритм не имеет предпочтений по направленности, запутанности или ещё каким-либо характеристикам. Результирующие лабиринты случайны и равновероятны.

### Формальный алгоритм

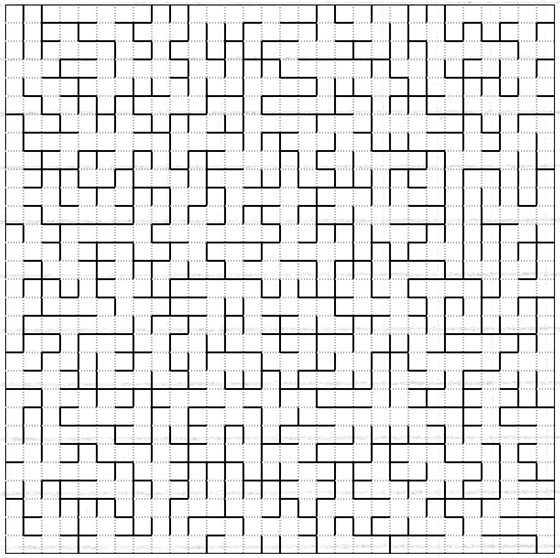
1. Выбрать случайную клетку, отметить ее посещенной;
2. Выбрать случайную соседнюю клетку и перейти в неё. Если она не была посещена, соединить ее с предыдущей, убрав между ними стену, отметить ее посещенной;
3. Повторять шаг 2, пока все клетки не будут посещены.

### Плюсы

* Отсутствует какое-либо смещение;
* Лабиринты абсолютно случайны, поэтому невозможно создать определенный алгоритм их решения;
* Сложность решения для человека;
* Простая реализация;

### Минусы

* Скорость;
* Не позволяет генерировать бесконечные лабиринты;
* Сильное падение эффективности под конец генерации;



*Aldous-Broder*

## Сравнение алгоритмов генерации по быстродействию

# Алгоритмы поиска кратчайшего пути

## Представление лабиринта в виде графа

Каждая ячейка – узел графа, ребро – проход межу парой ячеек (на рисунке ниже красная окружность – узел).

Размерность матрицы смежности – (ШиринаЛабиринат X ВысотаЛабиринта)2. Для снижения матрицы смежности можно использовать свертку графа. Снижение размерности матрицы смежности дает выигрыш при реализации алгоритмов поиска минимального пути.

В качестве веса ребра берется минимальный из номеров ячеек, которые соединяет ребро.

|  |  |
| --- | --- |
| Исходный граф (16 вершин) | Свертка графа (10 вершин) |

## Остовное дерево

Остовное дерево графа состоит из минимального подмножества рёбер графа, таких, что из одной из вершин графа можно попасть в любую другую вершину, двигаясь по этим рёбрам.

Для идеального лабиринта минимальное остовное дерево совпадает с самим графом.

В программе реализован алгоритм Краскала – эффективный алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа [6].

### Суть алгоритма Краскала

В начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

## Волновой алгоритм

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно [7].

Работа алгоритма включает в себя три этапа: **инициализацию**, **распространение волны** и **восстановление пути**.

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки.

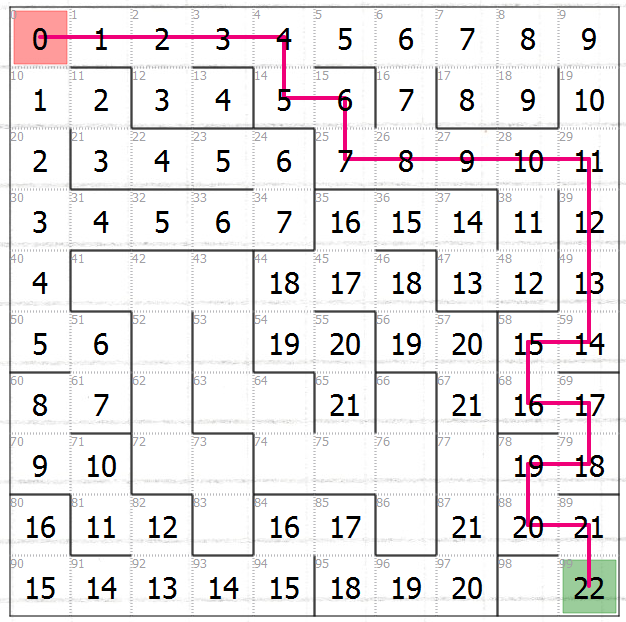
Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченная числом шагов от стартовой ячейки, становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек.

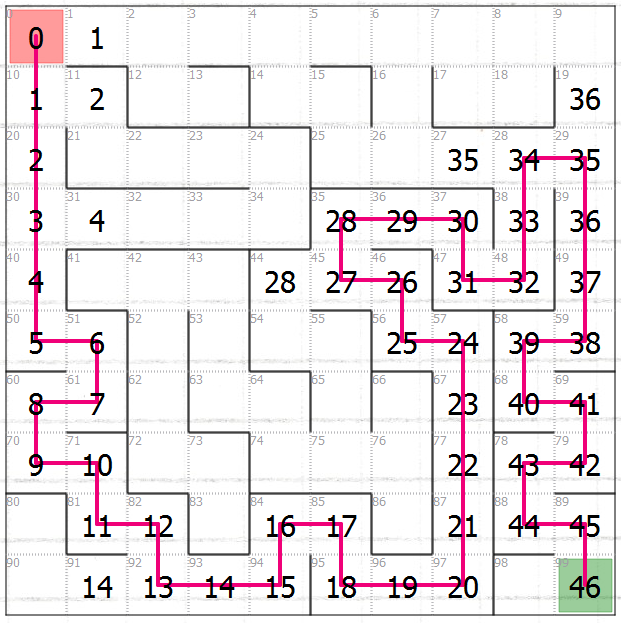
Для идеальных лабиринтов найденный путь является единственным.

При реализации при обходе ячеек соседние ячейки сохраняются в контейнер типа очередь (первый пришел, первый ушел). В результате реализуется обход в ширину.



*Неидеальный лабиринт. Волновой алгоритм (вширь)*

Если реализации волнового алгоритма при обходе ячеек соседние ячейки сохраняются в контейнер типа стек (первый пришел, последний ушел), то реализуется обход вглубь. В результате будет найден первый подходящий маршрут, не обязательно кратчайший (в случае неидеального лабиринта).



*Неидеальный лабиринт. Волновой алгоритм (вглубь)*

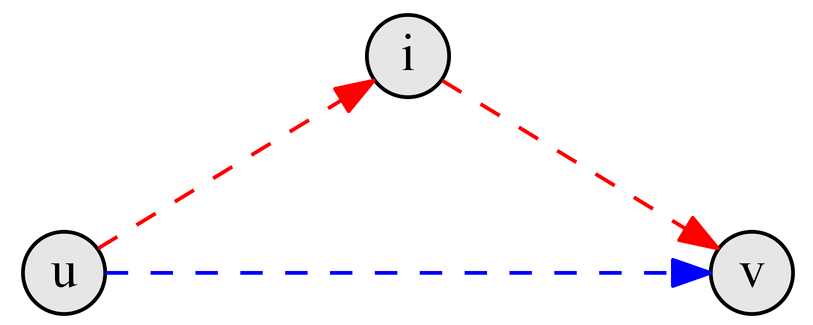
## Алгоритм Флойда-Уоршелла

**Алгоритм Флойда (алгоритм Флойда–Уоршелла)** — алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе. Работает корректно, если в графе нет циклов отрицательной величины, а в случае, когда такой цикл есть, позволяет найти хотя бы один такой цикл. Время работы алгоритма ~n3 , использование памяти ~ n2 [8].

Ключевая идея алгоритма — идея динамического программирования, а именно, разбиение процесса поиска кратчайших путей на **фазы**.

Перед k-ой фазой (k = 1…n) считается, что в матрице расстояний d[][] сохранены длины таких кратчайших путей, которые содержат в качестве внутренних вершин только вершины из множества { 1, 2, …, k-1}.

Работа, которую требуется произвести на k-ой фазе — это перебрать все пары вершин и пересчитать длину кратчайшего пути между ними. В результате после выполнения n-ой фазы в матрице расстояний d[i][j] будет записана длина кратчайшего пути между i и j, либо -1, если пути между этими вершинами не существует.



*Попытка улучшения длины между узлами*

На k-ой фазе требуется пересчитать длины кратчайших путей между всеми парами вершин i и j следующим образом:

new\_d[i][j] = min (d[i][j], d[i][k] + d[k][j]);

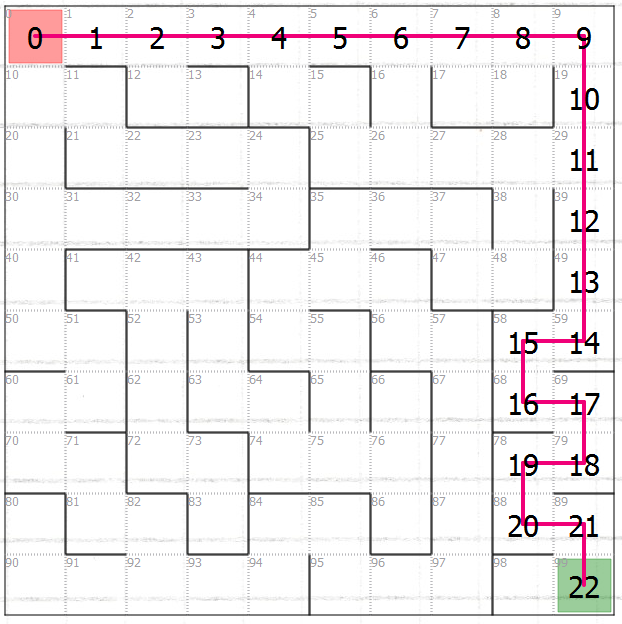
Реализация имеет вид: На вход программе подаётся граф, заданный в виде матрицы смежности — двумерного массива d[][] размера n2, в котором каждый элемент задаёт длину ребра между соответствующими вершинами. Требуется, чтобы выполнялось d[i][i] = 0 для любых i.

for (int k=0; k<n; ++k)

for (int i=0; i<n; ++i)

for (int j=0; j<n; ++j)

d[i][j] = min (d[i][j], d[i][k] + d[k][j]);



*Неидеальный лабиринт. Путь по алгоритму Флойда-Уоршелла*

## Сравнение времени поиска пути для лабиринтов различной генерации

Сравнивается скорость работы волнового алгоритма для лабиринтов различной генерации при поиске пути от ячеек верхнего ряда к ячейкам нижнего ряда.

## Сравнение средней длины пути для лабиринтов различной генерации

Сравнивается средняя длина пути от ячеек верхнего ряда к ячейкам нижнего ряда для лабиринтов различной генерации.

# Числовые характеристики

## Тупики и проходы

## Тепловая карта

Тепловая карта лабиринта для заданной ячейки отображает динамику изменения длины пути от заданной ячейки до всех ячеек лабиринта [9].

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Алгоритм Олдоса-Бродера* | | |
|  |  |  |
| *Двоичное дерево* | | |
|  |  |  |
| *Backtracking* | | |
|  |  |  |
| *Рекурсивное деление* | | |
|  |  |  |
| *Sidewinder* | | |

# Источники

1. <https://habr.com/ru/post/445378/>
2. https://habr.com/ru/post/320140/
3. <https://habr.com/ru/post/262345/>
4. <https://ru.qwe.wiki/wiki/Maze_generation_algorithm#Simple_algorithms>
5. https://habr.com/ru/post/321210/
6. <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%80%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D0%B0>
7. <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B8>
8. <https://e-maxx.ru/algo/floyd_warshall_algorithm>