Реализация и исследование алгоритмов генерации лабиринтов и поиска путей в них

Молошников Ф. А., студент 2 курса бакалавриата направления «Программная инженерия» СПбГУТ

Научный руководитель: доцент кафедры ПИВТ, кандидат технических наук Дагаев А.В.

*СПбГУТ*

*fmol00@mail.ru*

1. Цель работы

Реализация алгоритмов генерации и поиска пути в лабиринте оценка их эффективности. Были реализованы:

Алгоритмы генерации лабиринта:

1. Уилсона
2. Уилсона (модифицированный)
3. Олдоса-Бродера
4. Бинарного дерева

Алгоритмы поиска пути в лабиринте:

1. Ли
2. Ли (модификация с 2 волнами)
3. Дейкстры
4. А\* (AStar)

Первые два алгоритма поиска пути работают на ДРП (дискретном рабочем поле), где, с точки зрения прокладки путей, все ячейки одинаковы. 3 и 4 алгоритмы работают на графах. Мы используем их модификацию для работы на ДРП, где ячейки могут иметь разные веса. Вес означает путь в эту ячейку из соседней ячейки.

1. Область применения

Разработанная программа может использоваться в разработке игр, в программном обеспечении беспилотных систем (дроны и робототехника)

1. Описание Программы
   1. Описание программы и среды разработки

Язык: C++

Для исследования алгоритмов генерации лабиринтов [2] и поиска в них пути были разработаны классы maze и mazeWeighted, которые реализуют хранение лабиринтов и доступ к ним. Для сбора информации о времени генерации лабиринтов и о времени поиска в них путей были разработаны классы StatisticsMaze и StatisticsMazeSearch. Помимо двух основных целей, которые служат для исследования времени генерации лабиринтов и поиска путей в них соответственно, код содержит отдельные цели для демонстрации алгоритмов и разработанных классов. Также реализована консольная программа, осуществляющая генерацию лабиринтов и поиск пути в них. Программа имеет версии для Debian и Windows.

* 1. Разработка классов и функций
     1. maze

Лабиринт хранится внутри вектора BaseVector в виде символов. Нечетные строки хранят информацию о верхних и нижних стенах ячеек, четные – о боковых стенах ячеек и значения, хранящиеся в ячейках. Используются следующие символы:

# - стена между ячейками

? – отсутствие стены между ячейками

+ - угол ячейки (Служит для удобства представления. Никак не используется)

. – Значение ячейки по умолчанию.

Такой подход хотя и не является оптимальным по используемой памяти, но позволяет легко просматривать и редактировать лабиринт. Реализация на базе vector обеспечивает хорошую скорость при работе с лабиринтом.

1. Описание алгоритмов

В работе рассмотрены такие характеристики лабиринтов, как тип, фокус, смещенность, однородность, память.

**Тип** -способ построения идеальных (без петель и недостижимых областей) лабиринтов:

* 1. Алгоритм на основе дерева выращивает лабиринт подобно дереву.
  2. Алгоритм на основе множеств выполняет построения там, где ему хочется, отслеживая части лабиринта, соединённые друг с другом.

**Фокус** - тип реализации алгоритма генерации лабиринтов: с добавлением стен или вырезанием проходов.

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

**Смещенность** –изменение плотности в зависимости от лабиринта. В лабиринте со смещёнными проходами прямые проходы склонны больше идти в одном направлении, чем в других. Если смещенность отсутствует, то в независимости от направления сложность движения не изменяется.

**Однородность** – генерация алгоритмом всех возможных лабиринтов с равной вероятностью. «Да» означает, что алгоритм полностью однороден. «Нет» означает, что алгоритм потенциально может генерировать все возможные лабиринты в пределах любого пространства, но не с равной вероятностью. «Никогда» означает, что существуют возможные лабиринты, которые алгоритм никогда не сможет сгенерировать.

**Память** – объём потребляемой оперативной памяти или стека, необходимой для реализации алгоритма.

* 1. Алгоритмы генерации лабиринта

В данной работе реализованы алгоритмы генерации идеальных лабиринтов [3] методом вырезания проходов.

* + 1. Олдоса-Бродера
       1. Теоретические сведения

Тип: алгоритм на основе дерева

Фокус: присутствует возможность вырезать/добавлять стены

Смещенность: отсутствует.

Однородность: присутствует.

Объём дополнительной памяти для реализации алгоритма: 0

* + - 1. Алгоритм

1. Лабиринт состоит из изолированных стенами ячеек
2. Выбирается случайная ячейка и отмечается как посещенная
3. Счетчик посещенных вершин k устанавливается равным 1
4. Пока k не равен числу ячеек в лабиринте
   1. Выбирается случайное направление движения
   2. Если это направление не направлено в стену, то
      1. проверяем, была ли уже посещена эта ячейка. Если нет, то прорезаем стену к ней, переходим к ней и помечаем её как посещенную
   3. В противном случае просто переходим к ней
5. Алгоритм завершен
   * 1. Уилсона
        1. Теоретические сведения

Тип: алгоритм на основе дерева

Фокус: присутствует возможность вырезать/добавлять стены

Смещенность: отсутствует.

Однородность: присутствует.

Объём дополнительной памяти для реализации алгоритма: N^2 (пропорционально количеству ячеек)

* + - 1. Алгоритм

1. Лабиринт состоит из изолированных стенами ячеек
2. Выбрать случайную ячейку и добавить её в множество UST
3. Выбрать случайную ячейку и совершить от неё случайную прогулку, пока не будет встречена какая-нибудь ячейка из UST. Во время прогулки в ячейки записываются путевые координаты. Причем если во время такой прогулки мы наткнемся на ячейку, в которой уже были, т.е. сделаем петлю, то её путевые координаты перезапишутся, т.е. эта петля не попадет в итоговый путь (по-видимому, это одна из причин, по которой этот алгоритм такой медленный)
4. Проходим по путевым координатам от ячейки, с которой мы начали прогулку, до ячейки из UST по путевым координатам, прорезая стены (сразу это нельзя сделать из-за петель, которые срезает алгоритм во время первого прохода). Пока есть ячейки, не входящие в UST – повторяем шаг 3).
   * 1. Уилсона (модификация)

Отличается от предыдущего методом выбора ячейки, которой ещё нет в множестве UST.

* + 1. Двоичным деревом
       1. Теоретические сведения

Тип: алгоритм на основе множеств

Фокус: присутствует.

Смещенность: присутствует.

Однородность: отсутствует

Объём дополнительной памяти для реализации алгоритма: 0

* + - 1. Алгоритм

1. Лабиринт состоит из изолированных стенами ячеек
2. В данной реализации сразу вырезается один из столбцов (такой же результат можно было бы получить, двигаясь построчно, пробегая все столбцы)
3. Выбираем 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;
   1. Алгоритмы поиска пути в лабиринте
      1. Подготовка лабиринта

При поиске путей более интересной, на мой взгляд, является ситуация, когда между ячейками могут существовать альтернативные пути, причем стоимость (вес) пути между различными ячейками может быть различным.

Поэтому был создан класс mazeWeighted и написаны функции для разряжения лабиринта и для генерации весов лабиринта в виде кругов со случайным радиусом.

* + 1. Ли
       1. Теоретические сведения

Алгори́тм Ли [4] — алгоритм поиска кратчайшего пути на графе.

Состоит из трех этапов: инициализации, распространения волны и восстановления пути.

* + - 1. Алгоритм

Стоит отметить, что в данной реализации не используется очередь, характерная для алгоритмов поиска в ширину.

1. Вводится матрица весов путей (стоимости путей от стартовой ячейки до текущей)
2. Матрица так же используется для отличия посещенных ячеек от непосещенных (для непосещенных ячеек устанавливается вес, максимальный для int на данной платформе)
3. Начинается распространение волны от стартовой ячейки: ей присваивается нулевой вес.
4. Далее идет распространение волны: для каждой ячейки фронта волны просматриваются соседи. Если соседи ещё не были посещены, то им присваивается вес. Также отслеживается, достигнута ли финишная ячейка и удалось ли продвинутся фронту волны.
5. Если финишная ячейка достигнута, либо же фронт волны больше не может распространятся, то происходит завершение распространения волны. Если завершение было связано с достижением финишной ячейки, то начинается восстановление пути, в противном случае выдается сообщение о несуществовании пути, и происходит выход из функции.
6. Восстановление пути идет из финишной в начальную ячейку. Для того, чтобы восстановить путь нужно двигаться из финишной в начальную ячейку так, чтобы при движении вес пути уменьшался на единицу. После полного восстановления пути алгоритм завершается.
   * 1. Ли (модификация с двумя волнами)

Более сложная реализация алгоритма Ли [4]. В этой реализации помимо матрицы весов путей вводится матрица идентификаторов волн.

Данная реализация позволяет посещать примерно в два раза меньшее количество ячеек, чем вариант с одной волной.

* + 1. Дейкстры
       1. Алгоритм

Основное отличие алгоритма Дейкстры от алгоритма поиска в ширину – использование очереди с приоритетами.

1. Устанавливаем путевой вес стартовой ячейки равным 0.
2. Помещаем стартовую ячейку в очередь
3. Извлекаем ячейку из очереди
4. Проверяем всех её соседей (всего 4 соседа): Если сосед ни разу не был посещен, то устанавливаем для него путевые координатыи путевые веса. Если сосед уже был посещен, то находим для него новый путевой вес и сравниваем с текущим. Если новый вес оказался не меньше текущего, то не делаем ничего, если же новый вес оказался меньше текущего, то посещенному соседу присваиваем новый вес, новые путевые координаты и помещаем в очередь
5. Продолжаем выполнять пункты 3-4, пока извлеченный элемент не окажется финишной ячейкой, либо пока в очереди не останется элементов.
6. Узнав причину остановки, делаем вывод о том, существует ли путь или нет.
7. Если путь существует, то восстанавливаем его, двигаясь по путевым координатам от финиша к старту.
   * 1. А\* (AStar; А со звездой)
        1. Теоретические сведения

Поиск A\* [1] — алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины к другой.

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость».

* + - 1. Алгоритм

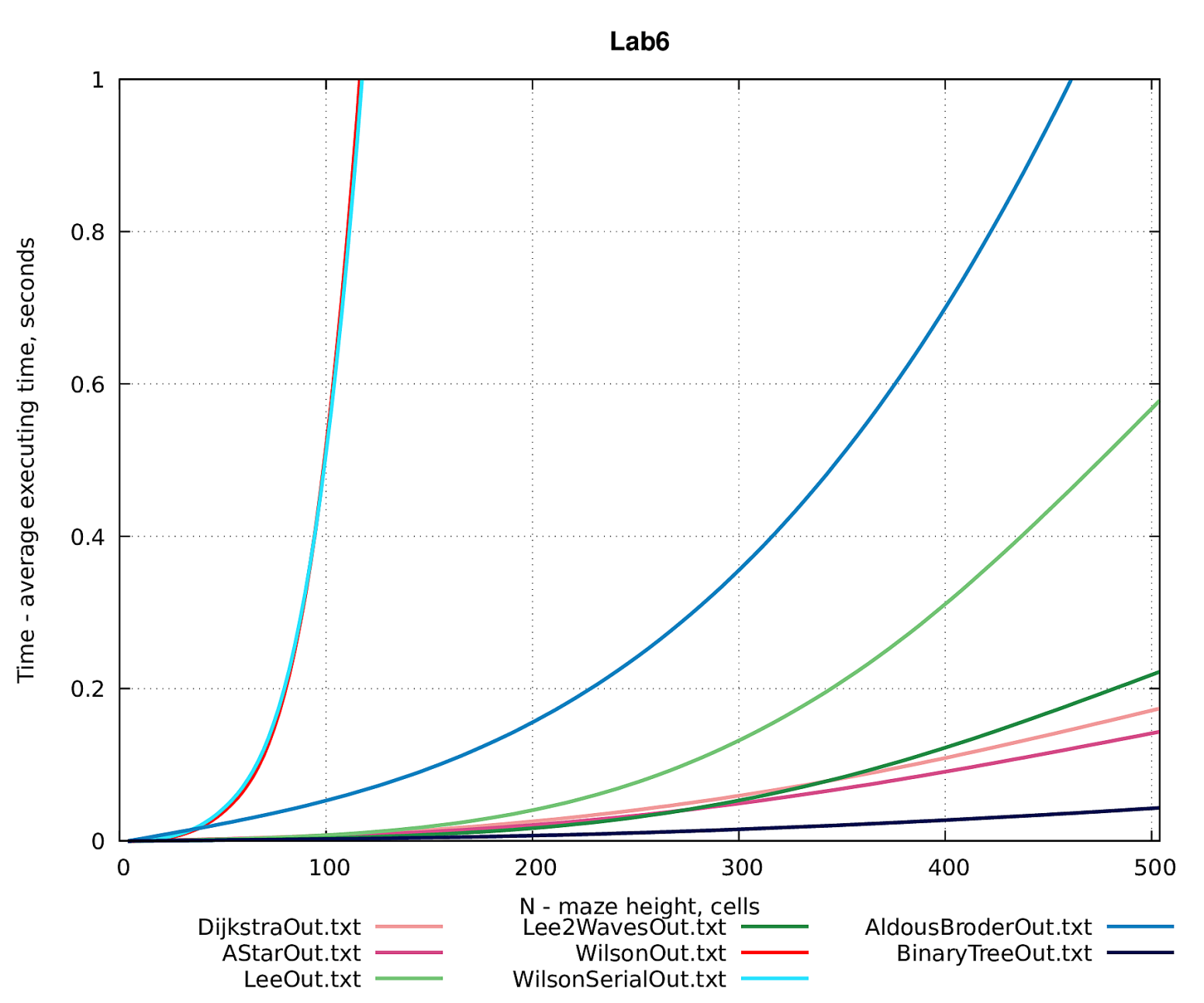
Реализация данного алгоритма мало отличается от реализации алгоритма Дейкстры.

Для оценки расстояния от текущей ячейки до финишной вводится эвристическая функция, состоящая из одного оператора:

return abs(finishi-Currenti)+abs(finishj-Currentj);

Отличие от алгоритма Дейкстры в том, что в качестве приоритета в очереди берутся не путевые веса, а сумма путевого веса и эвристической оценки расстояния до финишной ячейки.

1. Результаты работы



Сравнение времени генерации и времени поиска в лабиринтах

* 1. Анализ графиков
     1. Алгоритмы генерации лабиринтов

Алгоритм Уилсона оказался самым медленным из всех, сильно уступая даже алгоритму Олдоса-Бродера, который является более «глупым» (за 10 секунд алгоритм Олдоса-Бродера успевает сгенерировать примерно в 25 раз больше ячеек, чем алгоритм Уилсона). Это, по-видимому, связано с тем, что срезая петли, алгоритм многократно проходится по одним и тем же ячейкам, в отличие от Олдоса-Бродера, который добавляет в лабиринт все ячейки, на которые наткнется.

Его модификация с последовательным выбором ячеек, не входящих в UST, как оказалось, практически не влияет на время его работы.

Алгоритм Двоичного дерева оказался намного быстрее алгоритмов Уилсона (за 10 секунд он успевает сгенерировать примерно в 900 раз больше ячеек!) и Олдоса-Бродера(за 10 секунд в 36 раз больше ячеек). Но, как известно, лабиринты, сгенерированные им, имеют сильную смещенность и неоднородность, что заметно невооруженным взглядом.

По итогу, самым сложным по времени, при всех рассмотренных высотах лабиринта, оказался алгоритм Уилсона, а самым быстрым – двочиное дерево.

* + 1. Алгоритмы поиска в лабиринтах

Модификация Ли с двумя волнами работает примерно в 2 раза быстрее, чем вариант с одной волной. Причем при больших значениях лабиринта выигрыш во времени увеличивается.

Алгоритм А\* быстрее алгоритма Дейкстры в 1,19 раз для высоты лабиринта в 104 ячейки и в 1,21 раз для высоты лабиринта в 504 ячейки.

Алгоритм Ли быстрее алгоритма А\* на лабиринтах с высотой до 80 ячеек и быстрее алгоритма Дейкстры на лабиринтах с высотой до 100 ячеек. Причем при высоте в 504 ячейки алгоритм Ли становится медленнее в 3,32 раза.

Модификация Ли с двумя волнами оказывается быстрее алгоритма Дейкстры (приблизительно до 340 ячеек по высоте).

Это связано с отличиями реализации этих алгоритмов. Алгоритм Дейкстры использует очередь с приоритетами, работа с которой требует затрат времени. Очереди – обычный подход к реализации алгоритмов, основанных на поиске в ширину. В этой работе алгоритм Ли реализован без использования очередей. При каждой итерации распространения фронта волны ячейки фронта волны ищутся в прямоугольной области поля лабиринта. Эта область ограничена наиболее удаленными друг от друга ячейками фронта волны. Такой подход оказывается быстрее для относительно небольших лабиринтов (до 640 ячеек для Ли и А\*) и значительно более медленным при больших лабиринтах. В прочем, для сравнения нужна реализация алгоритма Ли с очередью. Может оказаться, что очередь без приоритетов, которая нужна для реализации Ли окажется быстрее очереди с приоритетами настолько, что реализация без очередей не будет иметь преимуществ.

По итогу, самым быстрым оказался алгоритм A\* при большой высоте лабиринта (начиная с высоты равной, приблизительно, 270 ячеек). Также в данном случае самый медленный – алгоритм Ли с одной волной. Однако на небольшой высоте (до ста ячеек по высоте) самым быстрым оказался алгоритм Ли с двумя волнами, а самым медленным – Дейкстра.

* + 1. Сравнение алгоритмов генерации и поиска

Скорость генерации и поиска сопоставимы друг с другом. Из рассмотренных алгоритмов генерации быстрее алгоритмов поиска оказался только алгоритм генерации Двоичным деревом.

Так, алгоритм Олдоса-Бродера медленнее алгоритма Ли примерно в 2 раза для лабиринта с высотой 400 ячеек.

1. Выводы
2. В ходе поделанный работы были изучены алгоритмы Уилсона, Уилсона(модификация), Олдоса-Бродера, Бинарного дерева для генерации лабиринтов, алгоритмы Ли, Ли с двумя волнами, Дейкстры, А\* для поиска пути в лабиринтах.
3. Был освоен класс-шаблон адаптера контейнера priority\_queue в стандартной библиотеке шаблонов (STL) языка C++, способы вывода данных с использованием псевдографики Unicode.
4. Были разработаны классы для хранения, доступа и изменения лабиринтов без весов ячеек и с весами ячеек. Реализован вывод лабиринтов в пригодном для редактирования вида, а так же в декоративном виде в двух вариантах: с мелкими и крупными ячейками. Были разработаны классы для сбора времени выполнения алгоритмов генерации и поиска в лабиринтах
5. Анализ показал, что алгоритмы поиска на лабиринтах по времени сопоставимы с алгоритмами генерации лабиринтов. Самым медленным является алгоритм Уилсона, самым быстрым – алгоритм Бинарного дерева. Использование эвристики уменьшает время поиска пути в лабиринте приблизительно в 1,2 раз.
6. Было разработано приложение, запускающееся в терминале и реализующее генерацию лабиринтов и поиск путей в них.
7. Заключение

В ходе данной работы были исследованы существующие алгоритмы генерации лабиринтов и решения задач поиска оптимального пути в пространстве. К каждому алгоритму приведено описание и пример его работы. Также был произведен сравнительный анализ всех алгоритмов с оценкой эффективности, наглядно продемонстрировавшей их сложность по времени.

1. Список используемых источников
2. Введение в алгоритм A\*. URL: <https://habr.com/ru/post/331192/> (дата обращения 2022 г.)
3. Неприлично простая реализация неприлично простого алгоритма генерации лабиринта. URL: <https://habr.com/ru/post/319532/> (дата обращения 2022 г.)
4. Лабиринты: классификация, генерирование, поиск решений. URL: <https://habr.com/ru/post/445378/> (дата обращения 2022 г.)
5. Овчинников В.А., Васильев А.Н., Лебедев В.В // Проектирование печатных плат: учебное пособие -  1-е изд. Тверь: ТГТУ, 2005. 116 с