# 1 Анализ предметной области

*Программная реализация поиска пути в лабиринте* — это интересная и важная задача, которая используется во многих областях, таких как навигация, планирование транспортных маршрутов, сортировка, робототехника и искусственный интеллект. Существует множество алгоритмов, которые можно использовать для решения этой задачи. Наиболее популярные из них приведены ниже.

## Алгоритм Дейкстры

Алгоритм назван в честь голландского ученого Эдсгера Дейкстры, который разработал его в 1956 году. За основу алгоритма берется принцип жадного выбора: на каждом шаге выбирается вершина с наименьшим известным расстоянием от источника и проверяются все её соседние вершины. Если расстояние до соседней вершины через текущую вершину оказывается короче, то расстояние обновляется.

Однако этот алгоритм подходит только для графов без отрицательных ребер, так как при наличии отрицательных циклов может возникнуть бесконечный цикл обновления расстояний.

Алгоритм описывается следующим образом:

* в начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бо́льшим максимального возможного пути в графе);
* массив флагов заполняется нулями. Затем запускается основной цикл;
* на каждом шаге цикла мы ищем вершину v с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины u. Если в них (в u) расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его;
* цикл завершается, когда флаги всех вершин становятся равны 1, либо когда у всех вершин c флагом 0 Последний случай возможен тогда и только тогда, когда граф G несвязный.

Работа алгоритма показана на блок-схеме (Рисунок 1 ).

## 1.2 Алгоритм A\*

Является модификацией алгоритма Дейкстры, который использует эвристику для ускорения процесса поиска. Он работает путем оценки стоимости каждого возможного шага и выбора того, который, как предполагается, приведет к наиболее эффективному пути.

Основа алгоритма - это эвристическая функция “расстояние + стоимость” (обычно обозначается как f(x)), которая является суммой двух компонентов: стоимости достижения текущей вершины (x) из начальной (обычно обозначается как g(x) и может быть как эвристической, так и нет), и эвристической оценки расстояния от текущей вершины до конечной (обозначается как h(x)). Эта функция определяет порядок обхода вершин.

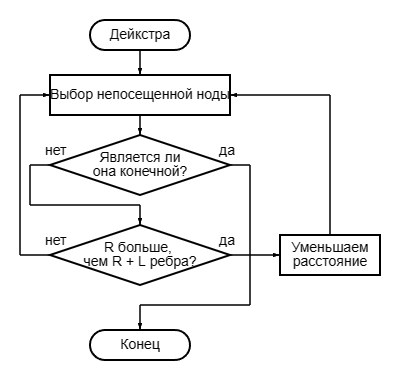


Рисунок 1 – Блок-схема алгоритма Дейкстры

Алгоритм описывается следующим образом:

* начинаем с начальной вершины и устанавливаем ее оценочную стоимость равной эвристической оценке расстояния до цели;
* пока есть не посещенные вершины выбираем вершину с наименьшей оценочной стоимостью и пометьте ее как посещенную. Если это целевая вершина, то путь найден;
* в противном случае для каждой соседней вершины обновляем ее оценочную стоимость, если текущая оценка больше суммы стоимости пути до выбранной вершины, веса ребра между ними и эвристической оценки до цели.

Работа алгоритма показана на блок-схеме (Рисунок 2 ).

## 1.3 Поиск в ширину (BFS)

Начинается с корневого узла и исследует все соседние узлы на данном уровне перед переходом к узлам следующего уровня. Ключевая идея заключается в том, что мы отслеживаем состояние расширяющегося кольца, которое называется границей. В сетке этот процесс иногда называется заливкой (flood fill), но та же техника применима и для карт без сеток.

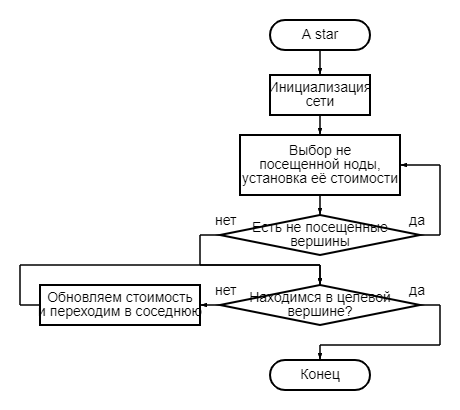


Рисунок 2 – Блок-схема алгоритма A\*

Алгоритм описывается следующим образом:

* берётся первый узел из очереди и помечается как посещенный;
* если он целевой, то завершаем работу программы, иначе проверяем преемников этого узла;
* если все узлы просмотрены, а целевой найден не был, то он недостижим из начального.

работа алгоритма показана на блок-схеме (Рисунок 3).

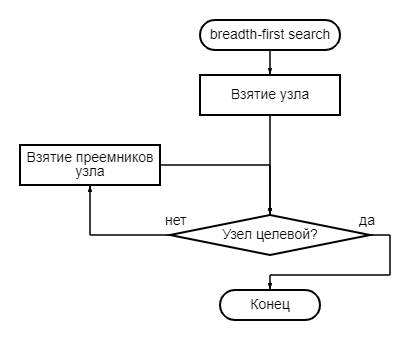


Рисунок 3 – Блок-схема BFS алгоритма

## 1.4 Поиск в глубину (DFS)

Основан на схожем принципе, что и поиск в ширину, однако распространяется в самую глубь графа, пока не достигнет конца, а затем возвращается назад, чтобы исследовать другие ветви. Возврат обратно к высшей вершине происходит только когда в выбранной вершине были рассмотрены все рёбра, в которых не осталось нерассмотренных вершин.

Алгоритм описывается следующим образом:

* начинаем с корневого узла и добавляем его в стек;
* пока стек не пуст извлекаем узел из стека и помечаем его как посещенный;
* добавляем все не посещенные соседние узлы в стек;
* если узел целевой или узлов не осталось (решение невозможно), то цикл заканчивается с соответствующим результатом.

Работа алгоритма показана на блок-схеме (Рисунок 4 ).

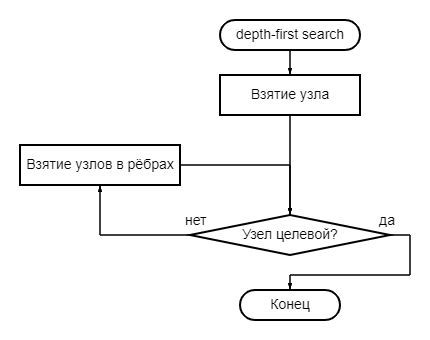


Рисунок 4 – Блок-схема DFS алгоритма

## 1.5 Jump Point Search (JPS) –

Является оптимизацией алгоритма поиска A\* для сеток с равномерной стоимостью. Он уменьшает симметрию в процедуре поиска путем обрезки графа, устраняя определенные узлы в сетке на основе предположений, которые можно сделать о соседях текущего узла, при условии, что выполняются определенные условия, относящиеся к сетке. Преимуществом JPS является то, что он не требует предобработки и поэтому тратит меньше памяти, также он является самым современным из перечисленных – он получил широкую огласку в 2012 году, а последние его крупные улучшения были опубликованы в 2014.

Алгоритм описывается следующим образом:

* берём корневой узел;
* если узел не целевой, то выбирается точка (узел графа), в которую совершится прыжок по прямой (Рисунок 5 ) или по диагонали (Рисунок 6 );
* если узел целевой или узлов не осталось (решение невозможно), то цикл заканчивается с соответствующим результатом.

Работа алгоритма показана на блок-схеме (Рисунок 7 ).

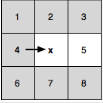


Рисунок 5 – Прямолинейный прыжок JPS алгоритма

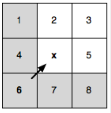


Рисунок 6 – Диагональный прыжок JPS алгоритма

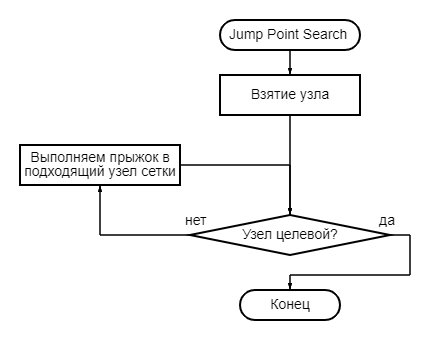


Рисунок 7 – Блок-схема JPS алгоритма

Пусть y является целевой (посадочной) точкой точки х, в направлении d, если y минимизирует значение k по закону: y = x + kd, и будет выполнено любое из условий:

Точка y – точка посадки.

У точки y есть хотя бы один сосед, у которого более 1 соседа.

d –движение по диагонали и существует точка z = y + kidi, которая лежит в ki шагах в направлении di ∈ {d1, d2}, таких что z – точка прыжка из y при условии 1 или 2.

Рассмотрим пример одного из прыжков (Рисунок 8 ). Пусть мы начинаем в точке х и заканчиваем движение по диагонали, пока не наткнёмся на точку у. Из у в точку z можно попасть с ki по горизонтали. Таким образом, z является преемником точки для прыжка x, а это в свою очередь определяет y как преемник для прыжка точки x.

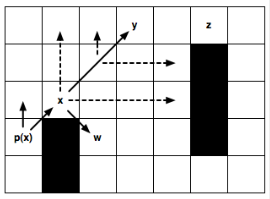


Рисунок 8 – Пример прыжка JPS алгоритма

## 1.6 Вывод

Представляя лабиринт в виде неориентированного графа, можно использовать множество разнообразных алгоритмов, которые могут быть эффективны в конкретном спектре задач. Большинство из них используют схожий принцип последовательного исследования графа с проверкой текущей вершины (точки лабиринта) на то, является ли она целевой (выходом из лабиринта). Однако они кардинально различаются в подходе к порядку выбора рассматриваемых вершин.

# 2 Описание программы

## 2.1 Общая характеристика программы

Название программы - «Изучение быстродействия методов поиска путей в лабиринте».

Программа, написанная в ходе выполнения курсовой работы направлена на изучение быстродействия и особенностей отдельных алгоритмов поиска путей в лабиринтах. Программа способна строить лабиринты тремя методами, находить в них путь и выводить лабиринты на экран с применением форматирования (отрисовкой стен).

Для написания программы был выбран C++, поскольку он имеет более высокое быстродействие по сравнению с python.

Минимальные системные требования:

* процессор: 2 ядра и частотой от 2 ГГц;
* оперативная память: 4 Гб;
* свободное дисковое пространство: 5Мб;
* операционная система: Windows 10 со средой разработки Visual Studio

## 2.2 Структура программы и данных

В разработке программы применялся метод ООП и был разработан класс «Лабиринт» с полями матрицы лабиринта и посещённых клеток и класс «Точка» с полями координат для заполнения списков очереди в алгоритмах поиска. В каждом из классов были реализованы конструкторы (по умолчанию, с параметрами, копий) и деструкторы.

В классе «Лабиринт» также были созданы методы отрисовки лабиринта на матрице с помощью трёх методов (Алгоритмом северо-восточного смещения, комбинированным алгоритмом северо-восточного и северо-западного смещения и алгоритмом «Sidewinder») и три метода поиска путей (BFS, DFS и A star).

Блок схема представлена на рисунке (Рисунок 9).

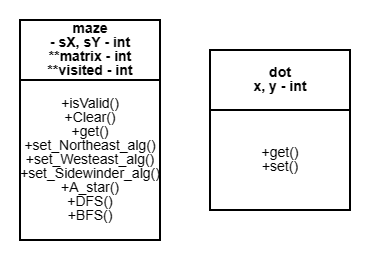


Рисунок 9 – Блок схема классов программы

## 2.3 Интерфейс программы

Использование консоли в качестве взаимодействия с пользователем накладывает некоторые ограничения в отрисовке изображений на экране, однако лабиринты всё равно можно вывести с помощью символов. Для отрисовки стен применять, например, крупные объемные буквы, а для пустых мест ставить пробелы или точки. Я использовал для вывода стен-границ символ “H”, для вывода внутренних стен символ “X”, а для проходов - пробелы. Для обозначения финиша используется звёздочка “\*” (находится в правом нижнем углу лабиринта).

Пример вывода лабиринта, заполненного северо-восточным алгоритмом размером 31 x 31 на рисунке (Рисунок 10 ).

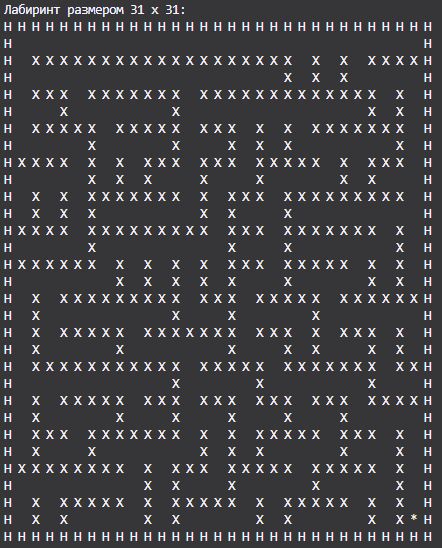


Рисунок 10 – Пример вывода лабиринта в консоль

Пример вывода лабиринта, заполненного алгоритмом Sidewinder размером 31 x 31 на рисунке (Рисунок 11 ).

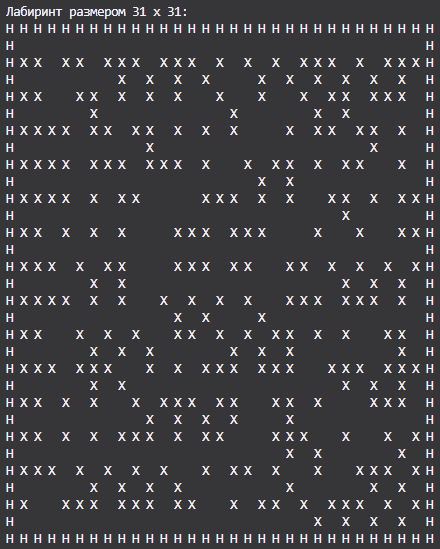


Рисунок 11 – Пример вывода лабиринта в консоль

**Описание меню программы:**

Визуально, меню отображается в виде пронумерованных пунктов с описанием каждой опции. Это обеспечивает понятность и удобство взаимодействия с пользователем. Каждый пункт меню отображается с новой строки, что улучшает читаемость и структурированность вывода, делая его более интуитивно понятным для пользователя.

Меню представлено на рисунке Рисунок 12

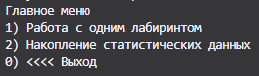


Рисунок 12 – Вид главного меню

Выбор режима: позволяет выбрать режим работы (работа с одним лабиринтом, накопление статистических данных).

*Режим работы с одним лабиринтом*: пользователь вводит с консоли размеры исследуемого лабиринта, далее открывается меню функций по работе с одним лабиринтом: построение (по выбору), решение (с выбором метода) и вывод на экран.

*Режим работы с накоплением статистических данных*: пользователь вводит диапазон тестирования и шаг, далее выбирается метод заполнения и тестируемый метод решения лабиринта. По окончанию решения каждого из N лабиринтов результаты выводятся на экран и файл results.txt.

UML – диаграмма прецедентов (Рисунок 13 ).

UML – диаграмма интерфейса представлена на рисунке (Рисунок 14 ).

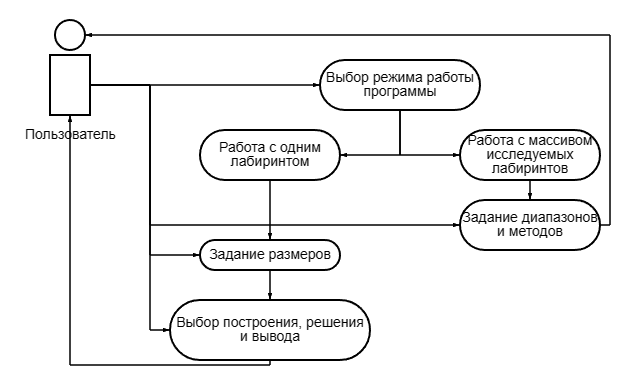


Рисунок 13 – UML диаграмма прецедентов

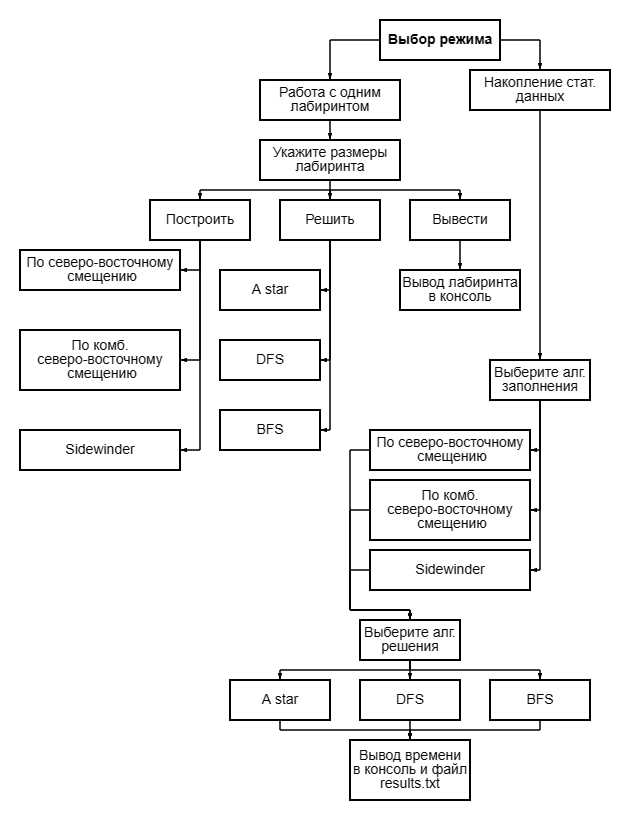


Рисунок 14 – UML диаграмма интерфейса программы

## 2.4 Тестирование программы

Для тестирования работы алгоритмов решения лабиринтов создавался лабиринт размером 51 x 51 элемент. Работа алгоритма начиналась с клетки (1,1) левого верхнего угла и заканчивалась, когда звёздочка “\*” становилась текущей клеткой.

Для тестирования был сгенерирован лабиринт 15 x 15, он примечателен тем, что у него имеется быстрый путь к финишу только после трёх глубоких перекрёстков.

Тестируемый лабиринт изображен на рисунке (Рисунок 15 ).

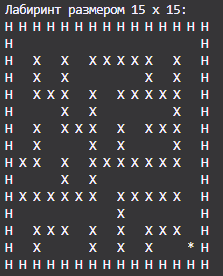


Рисунок 15 – Тестируемый лабиринт

DFS алгоритм (поиск в глубину) по итогам тестирования прошелся почти по каждой клетке лабиринта, за исключением тех, что находятся за целевой точкой (на рисунке цифрами показаны те места, в которые зашел алгоритм и оценил стоимость точки). Такое поведение напрямую связано с логикой его работы – метод выбирает куда пойти только “на ощупь”, т.е. не имеет эвристики, на перекрёстках идёт по правилу левой руки (в ту клетку, что была первее в стеке) и идёт до тех пор, пока не упрётся в тупик, затем выбирает следующую сторону на перекрёстке.

Результат работы DFS алгоритма представлен на рисунке (Рисунок 16 ).

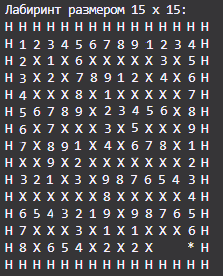


Рисунок 16 – Результат работы DFS алгоритма

BFS алгоритм (поиск в ширину) по итогам тестирования показал себя также как и DFS, поскольку целевая точка находится в самом дальнем углу. Этот метод может показать себя эффективнее, чем DSF, если известно, что целевая точка находится не “на глубине”.

Результат работы BFS алгоритма представлен на рисунке (Рисунок 17 ).

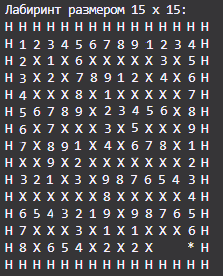


Рисунок 17 – Результат работы BFS алгоритма

Алгоритм A star показывает наиболее оптимальный путь решения, не заходит в заведомо более далёкие от цели пути. Его логика основывается на DSF поиске, за исключением того, что у него присутствует, помимо анализа стоимости длины g(n), ещё и функция h(n), которая вычисляет удалённость точки до цели. Благодаря этому, на развилках, A star выберет дорогу согласно её направлению к цели.

Результат работы A star алгоритма представлен на рисунке (Рисунок 18 ).

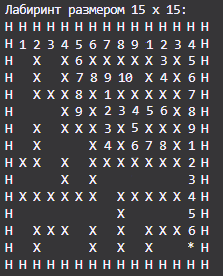


Рисунок 18 – Результат работы A star алгоритма

## 2.5 Вывод

В данном разделе проведен обзор программы, ее структуры, интерфейса и результатов тестирования. Выявлено, что использование алгоритма A\* в программе обеспечивает более эффективный поиск пути в сравнении с методами BFS и DFS, что подтверждает не только теоретические предположения, но и результаты практических испытаний.