# 1 Программная реализация поиска пути в лабиринте

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 ).

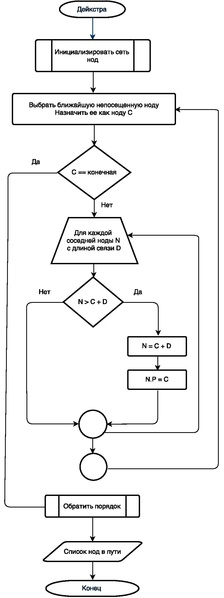


Рисунок 1 – Блок-схема алгоритма Дейкстры.

## 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.

Алгоритм описывается следующим образом:

* Начинаем с начальной вершины и установливаем 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 ).

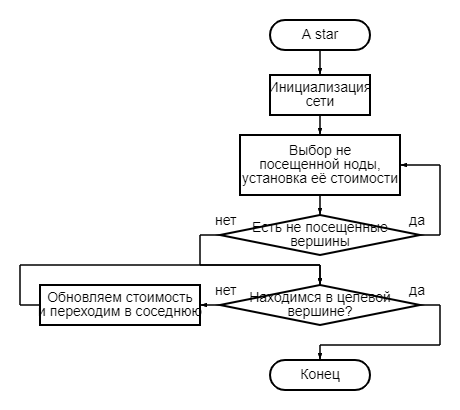


Рисунок 2 – Блок-схема алгоритма A\*.

## 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.

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).

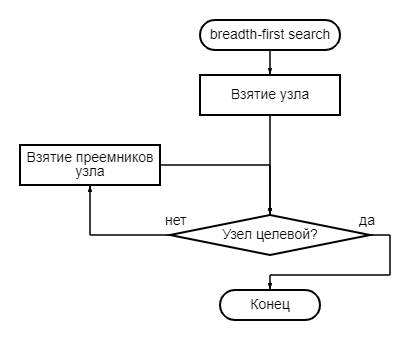


Рисунок 3 – Блок-схема BFS алгоритма.

## 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:

* Начнинаем 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 ).

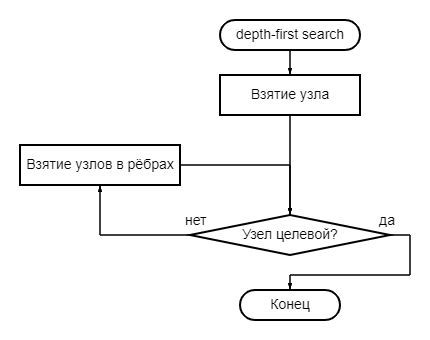


Рисунок 4 – Блок-схема DFS алгоритма.

## 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.

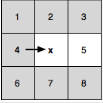


Рисунок 5 – Прямолинейный прыжок JPS алгоритма

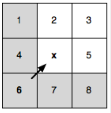


Рисунок 6 – Диагональный прыжок JPS алгоритма

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

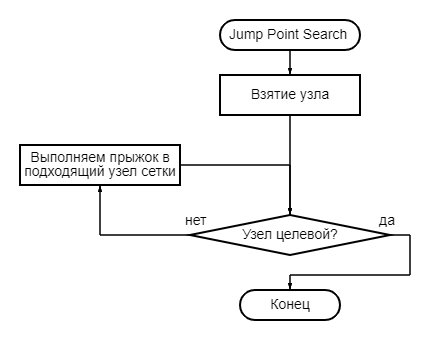


Рисунок 7 – Блок-схема JPS алгоритма

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.

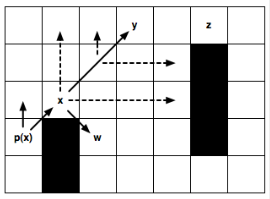


Рисунок 8 – Пример прыжка JPS алгоритма.

## 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.