МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика» Кафедра №806 «Вычислительная математика и программирование»

Курсовая работа по курсу “Дискретный анализ”

A* Алгоритм

Студент: Деньгов Илья Андреевич

Группа: М8О-307Б-22

Преподаватель: Макаров Никита Константинович

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/vgbhj/MAI/tree/main/DA/KP

Постановка задачи

Реализуйте алгоритм A* для графа на решетке

Формат ввода

первая строка содержит два целых числа n и m (1 <= n, m <= 1000) следующие n строк содержат описание клетчатого поля. каждая из этих строк имеет длину m и состоит только из символов "." и "#". Символ "." соответствует свободной клетке, а символ "#" клетке с препятствием. Следующая строка содержит целое число q (1 <=q <= 200) - количество запросов ДАлее следует q строк. Каждая из этих q строк содержит по четыре целых числа x1, y1, x2, y2 (1 <= x1, x2 <= n, 1 <= y1, y2 <= m) - координаты начальной и конечной клетки. гарантируется, что каждая клетка из запроса свободна.

Формат вывода

в ответе на каждый запрос выведите единственное число - длина кратчайшего пути между клетками из запроса. Если пути между клетками нет, выведите -1

Метод решения

Код реализует **поиск кратчайшего пути на решётке** (где . — свободная клетка, # — препятствие) с помощью **оптимизированного алгоритма A***.

1. Представление графа

- Игровое поле задаётся в виде **двумерного массива символов (vector<string> grid)**, где:
  - '.' — проходимая клетка
  - '#' — стена (нельзя пройти)
- Каждая клетка (x, y) рассматривается как **вершина графа**, а возможные шаги в **четырёх направлениях (вверх, вниз, влево, вправо)** — как **рёбра**.

2. Использование алгоритма A*

- A* находит **оптимальный путь**, комбинируя:
  - **Фактическую длину пути** от начала (g(x, y)).
  - **Эвристическую оценку расстояния до цели** (h(x, y)).
  - **Общую оценку приоритета** (f(x, y) = g(x, y) + h(x, y)).

3. Оптимизация с помощью deque (двухсторонней очереди)

Вместо priority_queue используется **deque**, что ускоряет обработку узлов:

- Если новая клетка **улучшает путь** и ближе к цели (по эвристике), она добавляется в **начало очереди (push_front)** → обрабатывается раньше.
- В остальных случаях она добавляется **в конец очереди (push_back)** → обрабатывается позже.
- Это помогает быстрее находить оптимальный путь в большинстве случаев.

Эвристика: В данной задаче используется манхэттенское расстояние, которое вы- числяется как

$$h(x, y) = |x - x_g| + |y - y_g|.$$

Описание алгоритма с оптимизацией

Для оптимизации работы алгоритма используется двусторонняя очередь (deque) вместо стандартной приоритетной очереди. Это позволяет эффективно управлять вершинами с одинаковым значением приоритета, добавляя их в начало или в конец очереди, в зависимости от величины приоритета.

1. **Инициализация:**

- Создаётся структура данных для Open List в виде двусторонней очереди (deque). Вершины хранятся в порядке увеличения значения $f(x)$. Когда два соседа имеют одинаковое значение $f(x)$, один из них добавляется в начало очереди, а другой — в конец.
- Для каждой вершины на карте задаются начальные значения $g(x) = \infty$, а $h(x)$ вычисляется с использованием манхэттенской эвристики.

2. **Основной цикл:**

- Извлекается вершина с минимальным $f(x)$ из Open List. Если эта вершина — конечная точка, алгоритм завершается.
- Для каждого соседа текущей вершины:
  - Если сосед не является проходимой клеткой (стена), он пропускается.
  - Если путь через текущую вершину улучшает значение g для соседа, обновляются его значения g, h, f, и сосед добавляется в Open List:
    * Если новое значение $f(x)$ соседней вершины равно текущему значению, сосед добавляется в начало очереди.
    * Если новое значение $f(x)$ соседней вершины больше на 2, он добавляется в конец очереди.

3. **Завершение:**

- Если конечная точка была добавлена в dq и обработана, выводится длина кратчайшего пути.
- Если dq становится пустым, выводится -1, так как путь не существует.

Описание программы

Программа реализует **алгоритм A*** для поиска кратчайшего пути на решётке с препятствиями.

Чтение входных данных:

- Считываются размеры поля n × m и его содержимое (. — свободные клетки, # — препятствия).
- Считывается число запросов q, затем координаты начальной и конечной точек для каждого запроса.

Поиск пути (функция astar):

- Используется **массив расстояний** dist и **массив посещённых клеток** visited.
- Применяется **двухочередная стратегия** (deque), позволяющая ускорить обработку узлов.
- В качестве **эвристики** используется **Манхэттенское расстояние**, чтобы направлять поиск к цели.
- Обрабатываются **четыре направления движения** (вверх, вниз, влево, вправо).

Вывод результата:

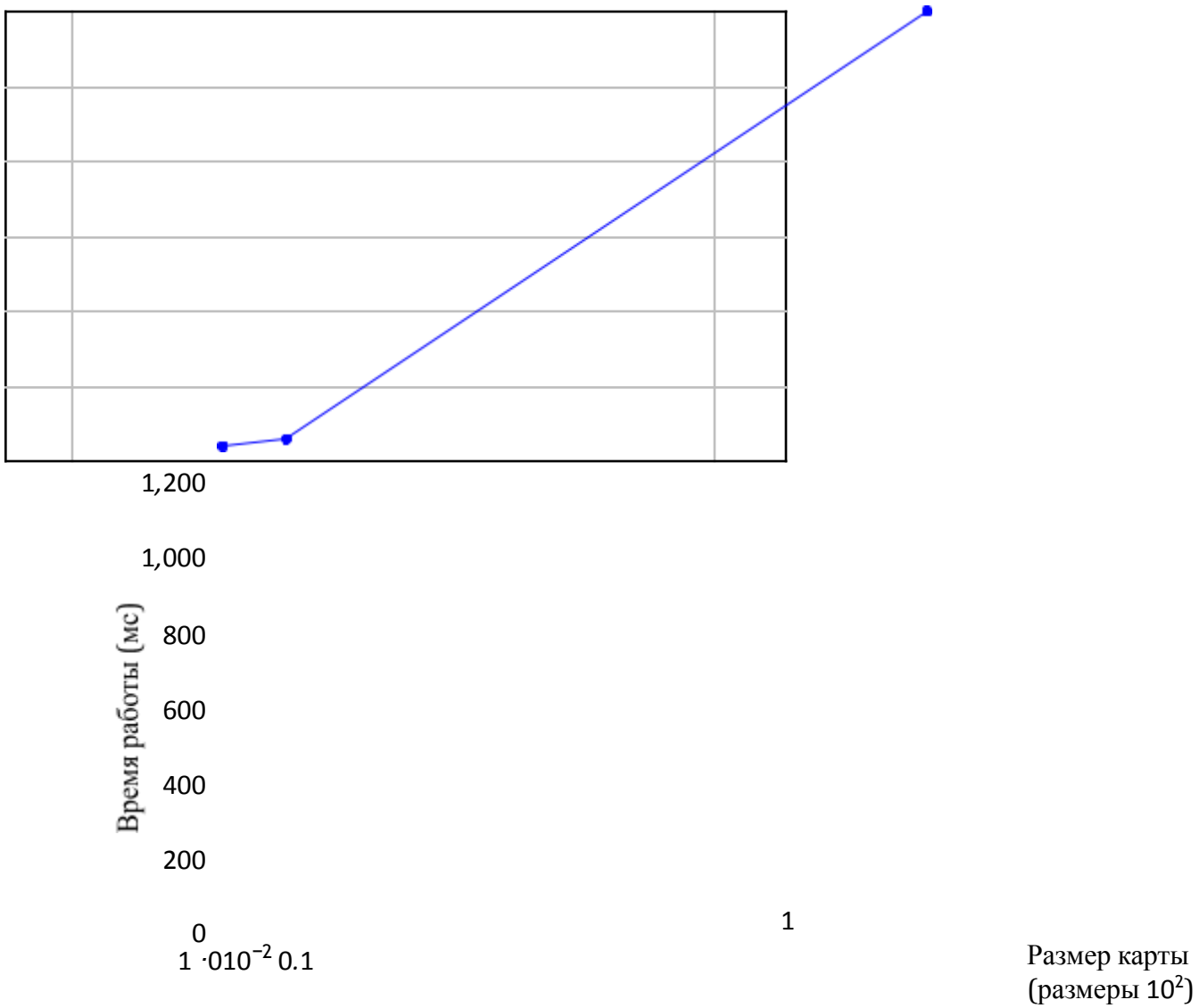
- Для каждого запроса программа выводит длину кратчайшего пути или -1, если путь невозможен.

Оптимизации:
deque вместо priority_queue (ускоряет обработку узлов)
Манхэттенская эвристика (быстрая оценка расстояния)
vector<int> вместо map (экономия памяти и времени)

Тест производительности

Для оценки производительности программы были использованы карты различных раз- меров (от 10 × 10 до 1000 × 1000) с различным количеством препятствий. Результаты:

- Среднее время выполнения на карте 100 × 100: 0.16129 мс.
- Максимальное время выполнения на карте 1000 × 1000: 20.0682 мс.
- Время выполнения на карте 10000 × 10000: 1163.42 мс.
- Алгоритм успешно находит кратчайший путь, если он существует, и корректно возвращает −1 в случае отсутствия пути.



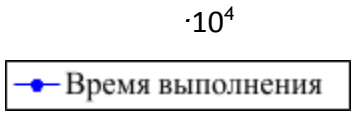


Рис. 1: График времени выполнения алгоритма в зависимости от размера карты

Выводы

В ходе данной курсовой работы был реализован алгоритм A* для поиска кратчайшего пути на карте, представленной в виде сетки. Алгоритм сочетает в себе преимущества жадного поиска и поиска по ширине, что позволяет эффективно находить оптимальные решения в задачах, связанных с навигацией и маршрутизацией.

В процессе работы над проектом была разработана оптимизированная версия алгоритма, использующая двустороннюю очередь для управления открытыми вершина- ми. Это значительно улучшило производительность алгоритма, особенно при работе с большими картами. Эвристическая функция, основанная на манхэттенском расстоянии, обеспечила быструю оценку расстояний до цели, что также способствовало ускорению поиска.

Тестирование производительности показало, что алгоритм способен обрабатывать карты размером до 1000×1000 с приемлемым временем выполнения. Среднее время выполнения на карте 100×100 составило всего 0.16129 мс, в то время как максимальное время на карте 1000×1000 достигло 20.0682 мс. Эти результаты подтверждают эффективность алгоритма в условиях, требующих быстрого поиска кратчайшего пути. Кроме того, алгоритм успешно справляется с задачами, где необходимо учитывать препятствия на пути, корректно возвращая результат в случае отсутствия пути. Это делает его подходящим для применения в различных областях, таких как робототехника, игры и системы навигации.

Таким образом, реализация алгоритма A* с использованием оптимизаций и эвристик позволила достичь высокой производительности и надежности в решении задачи поиска кратчайшего пути, что открывает возможности для дальнейших исследований и улучшений в данной области.