БУ ВО «Сургутский государственный университет»

Политехнический институт

Кафедра автоматизированных систем обработки информации и управления

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

ПО ТЕМЕ «Алгоритм A\* с различными эвристиками для поиска пути в лабиринте»

ПО ДИСЦИПЛИНЕ «Поисковые алгоритмы в информационном пространстве»

Выполнил: студент группы №606-12,

Демьянцев Виталий Владиславович

Принял: ст. преподаватель,

Гавриленко Анна Владимировна

Сургут 2024

**Введение**

**Данная лабораторная работа посвящена изучению и сравнению алгоритма A\* с различными эвристическими функциями для поиска пути в лабиринте. Основная цель работы — исследовать влияние эвристик (манхэттенской и евклидовой) и возможности диагонального движения на производительность алгоритма, провести эксперименты на лабиринтах различного размера и визуализировать результаты.**

**Алгоритм A\* является одним из ключевых методов поиска пути в графах и сетках, широко применяемым в робототехнике, видеоиграх и системах навигации. Эффективность алгоритма зависит от выбора эвристической функции и параметров движения.**

**Алгоритмы и методы**

**Алгоритм A\***

Алгоритм A\* находит кратчайший путь от начальной точки до целевой, используя комбинацию текущей стоимости пути (g) и эвристической оценки расстояния до цели (h).

* **Особенности:**
  + Использует приоритетную очередь для выбора узла с минимальным значением f = g + h.
  + Гарантирует оптимальность пути при соблюдении свойств эвристики (монотонности и неравенства треугольника).
  + Временная сложность: O(b^d), где b — фактор ветвления, d — длина пути.
* **Пример работы:**  
  Для сетки 3x3 с началом в (0,0) и целью в (2,2) алгоритм исследует узлы, выбирая следующий по минимальному f.

**Эвристические функции**

1. **Манхэттенская эвристика:**
   * Вычисляет расстояние как сумму абсолютных разностей координат:  
     h(a, b) = |a[0] - b[0]| + |a[1] - b[1]|.
   * Подходит для сеток без диагонального движения.
2. **Евклидова эвристика:**
   * Вычисляет прямое расстояние:  
     h(a, b) = sqrt((a[0] - b[0])^2 + (a[1] - b[1])^2).
   * Используется с диагональным и бездиагональным движением.

**Параметры движения**

* Без диагонального движения: 4 направления (вверх, вниз, вправо, влево), стоимость шага = 1.
* С диагональным движением: 8 направлений, стоимость диагонального шага = √2 ≈ 1.414.

**Требования**

Для выполнения работы использован Python версии 3.x с библиотеками:

* json — для загрузки данных лабиринта;
* heapq — для реализации приоритетной очереди;
* math — для вычислений в эвристиках;
* time — для замера времени;
* matplotlib и numpy — для визуализации.

Требуются файлы лабиринтов в формате JSON (например, maze\_10x10.json). Результаты выводятся в консоль, визуализация отображается в окнах.

Пример файла

{

    "width": 10,

    "height": 10,

    "start": [0,0],

    "goal": [6,9],

    "maze": [

        [0,1,0,0,0,0,1,0,0,0],

        [0,1,0,0,0,0,1,0,1,0],

        [0,0,0,1,0,0,0,0,1,0],

        [1,1,1,1,0,0,0,0,1,0],

        [0,0,0,0,0,1,0,0,1,0],

        [0,1,1,1,1,1,0,1,1,0],

        [0,0,0,0,0,0,0,0,0,0],

        [0,1,1,1,1,1,1,1,1,0],

        [0,0,0,0,0,0,0,0,0,0],

        [0,0,0,0,0,0,0,0,0,0]

    ]

}

**Реализация и эксперименты**

**Структура кода**

1. **Эвристики:**
   * heuristic\_manhattan(a, b) — манхэттенское расстояние.
   * heuristic\_euclidean(a, b) — евклидово расстояние.
2. **Основной алгоритм:**
   * a\_star\_search(maze\_data, heuristic, allow\_diagonal) — реализация A\*.
3. **Вспомогательные функции:**
   * load\_maze(file\_path) — загрузка лабиринта.
   * visualize\_maze(maze\_data, results, file\_path) — визуализация пути.
   * compare\_heuristics(file\_paths) — сравнение эвристик.

**Тестовые данные**

* Лабиринт 10x10 (maze\_10x10.json).
* Лабиринт 20x20 (maze\_20x20.json).
* Лабиринт произвольного размера (maze\_nxm.json).

Пример вывода для maze\_10x10.json

Лабиринт: maze\_10x10.json

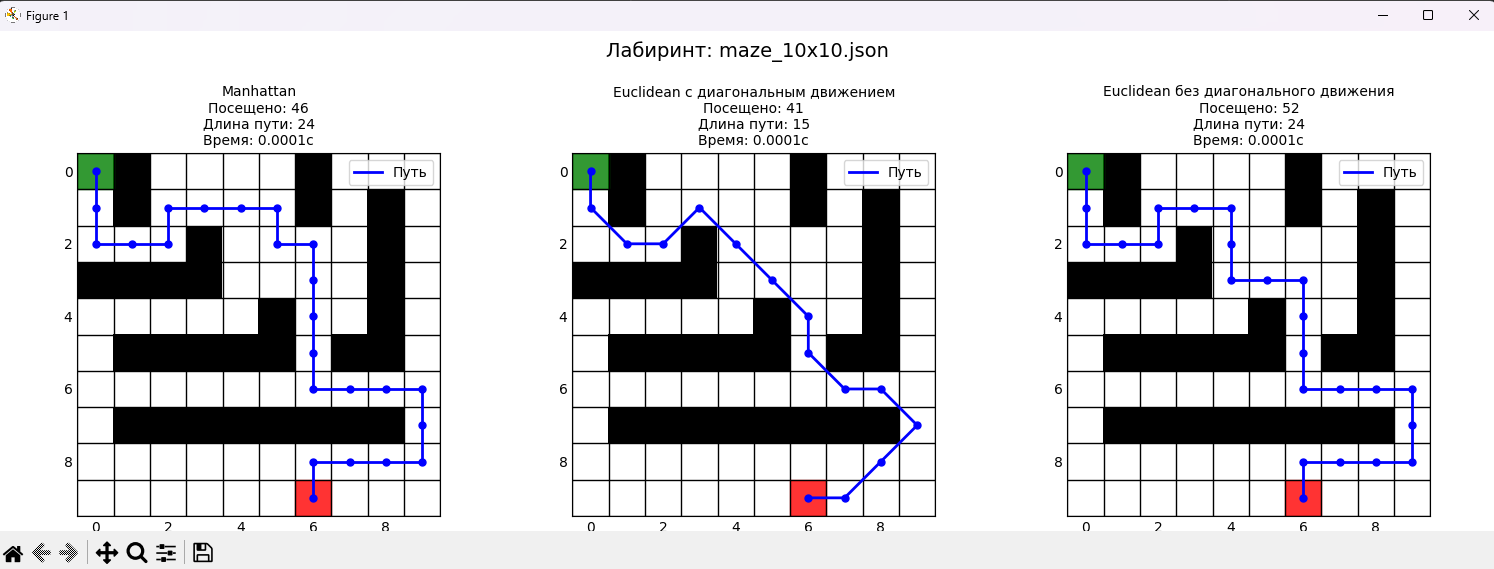
Эвристика | Длина пути | Посещено узлов | Время (с)

----------------------------------------------------------------------

Manhattan | 24 | 46 | 0.000085

Euclidean с диагональным движением | 15 | 41 | 0.000124

Euclidean без диагонального движения | 24 | 52 | 0.000076



**Анализ результатов**

**Производительность**

* **Manhattan:**
  + Эффективна для сеток без диагоналей, но путь длиннее при наличии диагональных возможностей.
  + Посещает больше узлов из-за строгой оценки.
* **Euclidean с диагональным движением:**
  + Находит более короткий путь за счет диагоналей.
  + Меньше посещенных узлов благодаря точной эвристике.
* **Euclidean без диагонального движения:**
  + Сравнима с манхэттенской по длине пути, но чаще медленнее из-за вычислений корня.

**Визуализация**

Графики показывают:

* Зеленый квадрат — стартовая точка.
* Красный квадрат — цель.
* Синий путь — найденный маршрут.
* Заголовки содержат длину пути, число посещенных узлов и время выполнения.

**Экспериментальные данные**

* На малых лабиринтах (10x10) разница во времени минимальна (порядка 10⁻⁴ сек).
* На больших лабиринтах (20x20) диагональное движение сокращает длину пути и число узлов.

**Выводы**

1. **Алгоритм A**\* с евклидовой эвристикой и диагональным движением демонстрирует лучшую производительность на лабиринтах, где возможны диагональные шаги, обеспечивая более короткий путь и меньшее число посещенных узлов.
2. **Манхэттенская эвристика** предпочтительна для простых сеток без диагоналей благодаря скорости вычислений.
3. **Евклидова эвристика без диагоналей** менее эффективна, так как совмещает сложность вычислений с ограничениями движения.
4. Выбор эвристики зависит от структуры лабиринта и требований к оптимальности пути.

Таким образом, работа подтверждает влияние эвристических функций на эффективность A\*, демонстрируя их применимость в задачах поиска пути.

import json

import heapq

import math

import time

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.patches import Rectangle

def heuristic\_manhattan(a, b):

    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def heuristic\_euclidean(a, b):

    return math.sqrt((a[0] - b[0]) \*\* 2 + (a[1] - b[1]) \*\* 2)

def load\_maze(file\_path):

    with open(file\_path, 'r', encoding='utf-8') as file:

        return json.load(file)

def a\_star\_search(maze\_data, heuristic, allow\_diagonal=False):

    width, height = maze\_data['width'], maze\_data['height']

    start, goal = tuple(maze\_data['start']), tuple(maze\_data['goal'])

    maze = maze\_data['maze']

    open\_list = []

    heapq.heappush(open\_list, (0, 0, start))

    came\_from = {}

    g\_score = {start: 0}

    f\_score = {start: heuristic(start, goal)}

    visited\_nodes = 0

    tiebreaker = 0

    while open\_list:

        \_, \_, current = heapq.heappop(open\_list)

        visited\_nodes += 1

        if current == goal:

            path = []

            while current in came\_from:

                path.append(current)

                current = came\_from[current]

            path.append(start)

            path.reverse()

            return path, visited\_nodes

        x, y = current

        neighbors = [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

        if allow\_diagonal:

            neighbors.extend([(x+1, y+1), (x-1, y-1), (x+1, y-1), (x-1, y+1)])

        for neighbor in neighbors:

            nx, ny = neighbor

            if 0 <= nx < width and 0 <= ny < height and maze[ny][nx] == 0:

                cost = 1.414 if allow\_diagonal and (abs(nx - x) + abs(ny - y)) == 2 else 1

                tentative\_g\_score = g\_score[current] + cost

                if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:

                    came\_from[neighbor] = current

                    g\_score[neighbor] = tentative\_g\_score

                    f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, goal)

                    tiebreaker += 1

                    heapq.heappush(open\_list, (f\_score[neighbor], tiebreaker, neighbor))

    return None, visited\_nodes

def visualize\_maze(maze\_data, results, file\_path):

    maze = np.array(maze\_data['maze'])

    heuristics = list(results.keys())

    num\_heuristics = len(heuristics)

    fig, axes = plt.subplots(1, num\_heuristics, figsize=(num\_heuristics \* 5, 5))

    fig.suptitle(f"Лабиринт: {file\_path.split('/')[-1]}", fontsize=14)

    if num\_heuristics == 1:

        axes = [axes]

    for ax, heuristic\_name in zip(axes, heuristics):

        path = results[heuristic\_name]["path"]

        ax.imshow(maze, cmap='Greys', origin='upper')

        ax.set\_xticks(np.arange(-0.5, maze\_data['width'], 1), minor=True)

        ax.set\_yticks(np.arange(-0.5, maze\_data['height'], 1), minor=True)

        ax.grid(which='minor', color='black', linestyle='-', linewidth=1)

        ax.tick\_params(which='both', size=0)

        ax.add\_patch(Rectangle((maze\_data['start'][0] - 0.5, maze\_data['start'][1] - 0.5), 1, 1, color='green', alpha=0.8))

        ax.add\_patch(Rectangle((maze\_data['goal'][0] - 0.5, maze\_data['goal'][1] - 0.5), 1, 1, color='red', alpha=0.8))

        if path:

            path\_x, path\_y = zip(\*path)

            ax.plot(path\_x, path\_y, 'b-', linewidth=2, label='Путь')

            for x, y in path:

                ax.plot(x, y, 'bo', markersize=5)

        ax.set\_title(f"{heuristic\_name}\nПосещено: {results[heuristic\_name]['visited\_nodes']}\nДлина пути: {results[heuristic\_name]['path\_length']}\nВремя: {results[heuristic\_name]['time']:.4f}с", fontsize=10)

        ax.legend(loc='upper right')

    plt.tight\_layout(rect=[0, 0, 1, 0.95])

    plt.show()

def compare\_heuristics(file\_paths):

    heuristics = {

        "Manhattan": (heuristic\_manhattan, False),

        "Euclidean с диагональным движением": (heuristic\_euclidean, True),

        "Euclidean без диагонального движения": (heuristic\_euclidean, False)

    }

    all\_results = {}

    for file\_path in file\_paths:

        maze\_data = load\_maze(file\_path)

        results = {}

        for name, (heuristic, allow\_diagonal) in heuristics.items():

            start\_time = time.perf\_counter()

            path, visited\_nodes = a\_star\_search(maze\_data, heuristic, allow\_diagonal)

            elapsed\_time = time.perf\_counter() - start\_time

            results[name] = {

                "path\_length": len(path) if path else 0,

                "visited\_nodes": visited\_nodes,

                "time": elapsed\_time,

                "path": path

            }

        all\_results[file\_path] = results

        visualize\_maze(maze\_data, results, file\_path)

    print("\n=== Результаты сравнения эвристик ===")

    for file\_path in file\_paths:

        print(f"\nЛабиринт: {file\_path.split('/')[-1]}")

        print(f"{'Эвристика':<30} | {'Длина пути':<12} | {'Посещено узлов':<15} | {'Время (с)':<10}")

        print("-" \* 70)

        for name in heuristics:

            r = all\_results[file\_path][name]

            print(f"{name:<30} | {r['path\_length']:<12} | {r['visited\_nodes']:<15} | {r['time']:.6f}")

    return all\_results

if \_\_name\_\_ == "\_\_main\_\_":

    file\_paths = [

        "maze\_10x10.json",

        "maze\_20x20.json",

        "maze\_nxm.json"

    ]

    results = compare\_heuristics(file\_paths)