

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины «Искусственный интеллект в профессиональной сфере»

Вариант _____

Выполнила:
Маньшина Дарья Алексеевна
2 курс, группа ИТС-б-о-22-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи»,
направленность (профиль)
«Инфокоммуникационные системы и
сети», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р. А.,
доцент департамента цифровых,
робототехнических систем и электроники

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

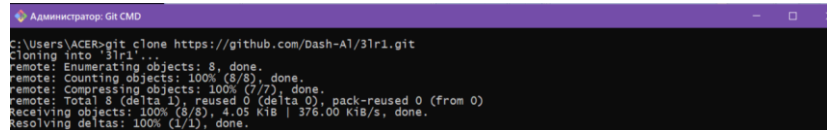
Тема: исследование методов поиска в пространстве состояний

Цель работы: приобретение навыков по работе с методами поиска в пространстве состояний с помощью языка программирования Python версии 3.x.

Ход работы:

Ссылка на репозиторий: <https://github.com/Dash-A1/3lr1/tree/main>

Копирование репозитория:

A screenshot of a Windows command prompt window titled "Администратор: Git CMD". The window shows the output of a git clone command. The text is as follows:

```
C:\Users\ACER>git clone https://github.com/Dash-A1/3lr1.git
Cloning into '3lr1'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (3/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (8/8), 4.05 KiB | 376.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
```

Рисунок 1 – Клонирование репозитория

Пример 1.

Код программы:

```
import random
import heapq
import math
import sys
from collections import defaultdict, deque, Counter
from itertools import combinations

class Problem:
    def __init__(self, initial=None, goal=None, **kwargs):
        self.__dict__.update(initial=initial, goal=goal, **kwargs)
    def actions(self, state): raise NotImplementedError
    def result(self, state, action): raise NotImplementedError
    def is_goal(self, state): return state == self.goal
    def action_cost(self, s, a, sl): return 1
    def h(self, node): return 0

    def __str__(self):
        return '{}({!r}, {!r})'.format(type(self).__name__, self.initial, self.goal)
```

Результат программы:

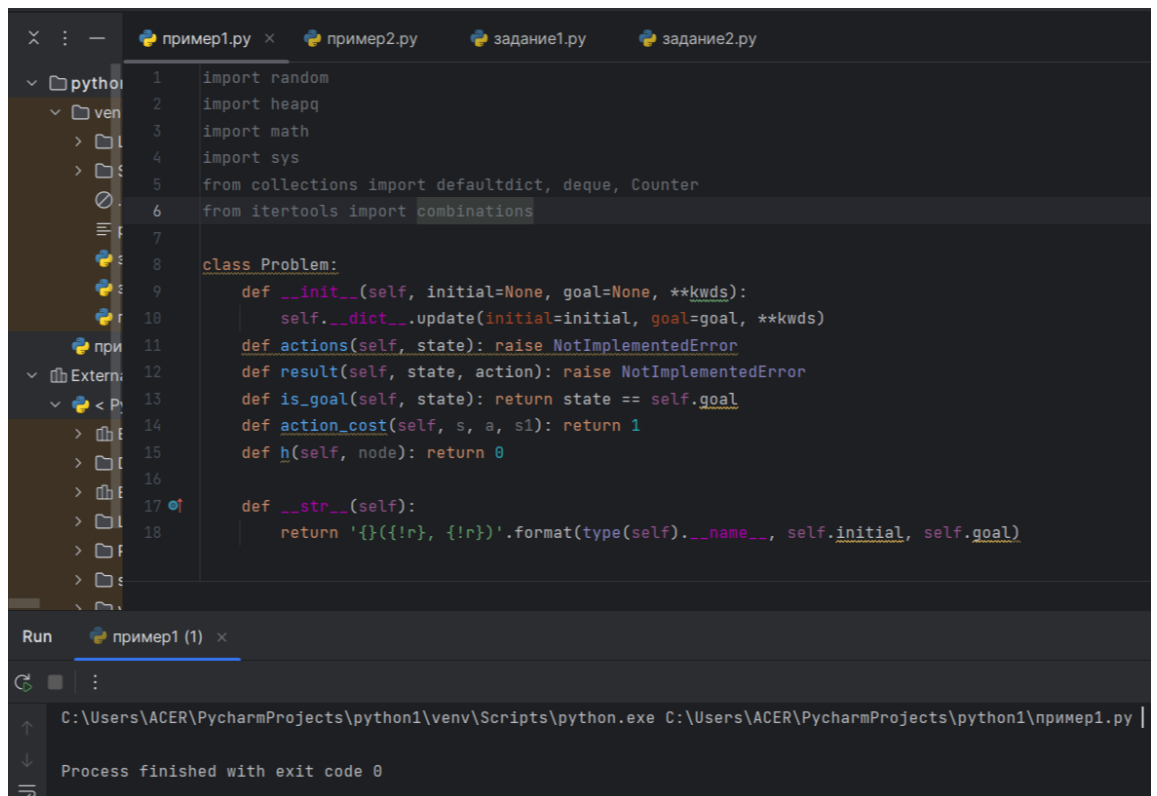


Рисунок 2 – Выполнение программы примера 1

Пример 2.

Код программы:

```
import random
import heapq
import math
import sys
from collections import defaultdict, deque, Counter
from itertools import combinations
FIFOQueue = deque
LIFOQueue = list
class PriorityQueue:
    #Очередь, в которой элемент с минимальным значением f(item) всегда выгружается
    #первым.
    def __init__(self, items=(), key=lambda x: x):
        self.key = key
        self.items = [] # a heap of (score, item) pairs
        for item in items:
            self.add(item)
    def add(self, item):
        #Добавляем элемент в очередь."
        pair = (self.key(item), item)
        heapq.heappush(self.items, pair)
    def pop(self):
        # Достаем и возвращаем элемент с минимальным значением f(item).
```

```

        return heap.heappop(self.items)[1]
    def top(self): return self.items[0][1]
    def __len__(self): return len(self.items)

```

Результат программы:

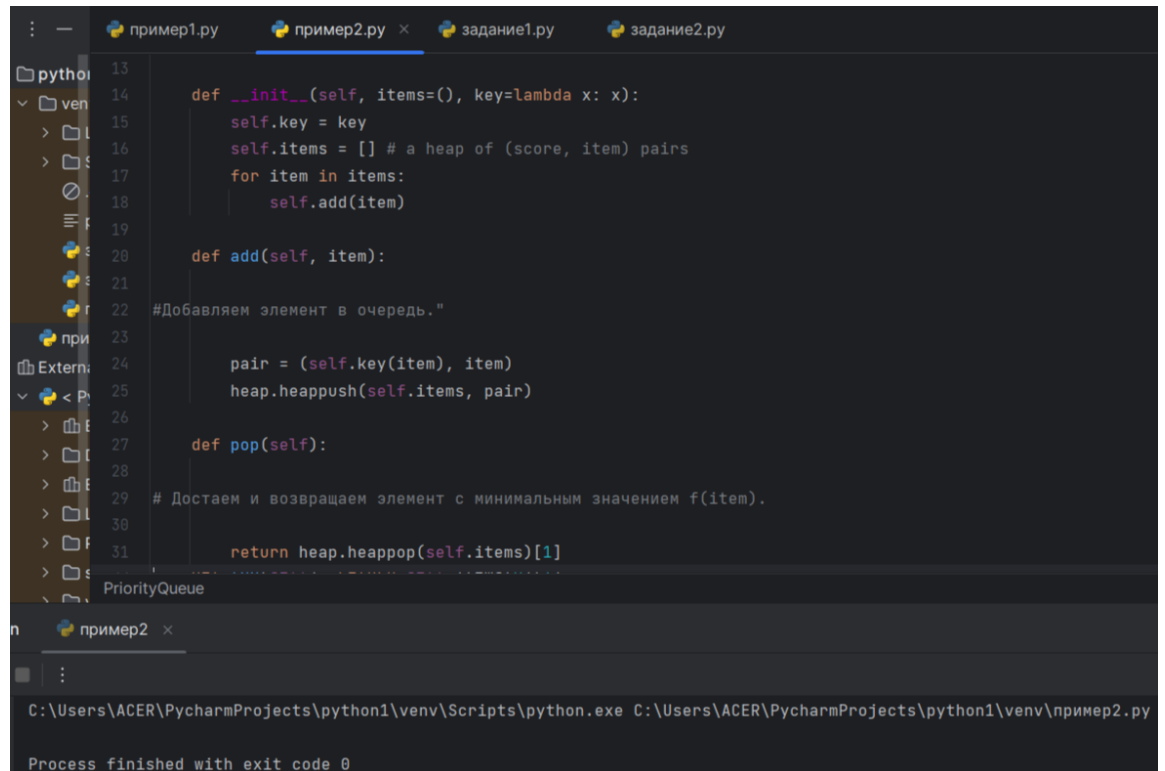


Рисунок 3 – Выполнение программы примера 2

Задание 1. Решим задачу коммивояжёра методом полного перебора.

Код программы:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import itertools

def calculate_total_distance(route, distance_matrix):
    total_distance = 0
    for i in range(len(route) - 1):
        total_distance += distance_matrix[route[i]][route[i + 1]]
    total_distance += distance_matrix[route[-1]][route[0]]
    return total_distance

def traveling_salesman(distance_matrix):
    num_cities = len(distance_matrix)
    cities = list(range(num_cities))

    min_distance = float('inf')
    best_route = None

    for route in itertools.permutations(cities):

```

```

        current_distance = calculate_total_distance(route, distance_matrix)
        if current_distance < min_distance:
            min_distance = current_distance
            best_route = route

    return best_route, min_distance

if __name__ == "__main__":
    # Матрица расстояний между городами для 20 узлов
    distance_matrix = [
        [0, 774, 246, 420, 198, 181, 853, 462, 457, 200],
        [774, 0, 349, 545, 732, 815, 1487, 196, 983, 833],
        [246, 349, 0, 174, 183, 100, 938, 708, 703, 284],
        [420, 545, 174, 0, 357, 274, 112, 882, 877, 458],
        [198, 732, 183, 357, 0, 83, 755, 560, 555, 101],
        [181, 815, 100, 274, 83, 0, 838, 643, 638, 184],
        [853, 1987, 938, 112, 755, 838, 0, 462, 674, 654],
        [462, 196, 708, 882, 560, 643, 462, 0, 212, 661],
        [457, 983, 703, 877, 555, 638, 674, 212, 0, 656],
        [200, 833, 284, 458, 101, 184, 654, 661, 656, 0],
    ]

    best_route, min_distance = traveling_salesman(distance_matrix)
    print(f"Лучший маршрут: {best_route}")
    print(f"Минимальное расстояние: {min_distance}")

```

Результат программы:

```

python1  Version control
-
  пример1.py  пример2.py  задание1.py  задание2.py x
python1
22
23     min_distance = float('inf')
24     best_route = None
25
26     # Генерация всех возможных маршрутов
27     for route in itertools.permutations(cities):
28         current_distance = calculate_total_distance(route, distance_matrix)
29         if current_distance < min_distance:
30             min_distance = current_distance
31             best_route = route
32
33     return best_route, min_distance
34
35
36 # Пример использования
37 if __name__ == "__main__":
38     # Матрица расстояний между городами для 20 узлов
39     distance_matrix = [
40
41 if __name__ == "__main__"
задание2 x
:
C:\Users\ACER\PycharmProjects\python1\venv\Scripts\python.exe C:\Users\ACER\PycharmProjects\python1\venv\задание2.py
Лучший маршрут: (0, 2, 1, 7, 8, 6, 3, 5, 4, 9)
Минимальное расстояние: 2447
Process finished with exit code 0

```

Рисунок 4 – Выполнение программы задания 1

Вывод: в ходе лабораторной работы были приобретены навыки по работе с методами поиска в пространстве состояний с помощью языка программирования Python версии 3.x.

Контрольные вопросы

1. Что представляет собой метод "слепого поиска" в искусственном интеллекте?

Метод «слепого поиска» в искусственном интеллекте представляет собой алгоритм поиска, который не использует информацию о структуре или свойствах исследуемого пространства для определения наилучшего пути к цели.

2. Как отличается эвристический поиск от слепого поиска?

Эвристический поиск отличается от слепого поиска тем, что он использует дополнительную информацию о структуре и свойствах исследуемого пространства для определения наилучшего пути к цели.

3. Какую роль играет эвристика в процессе поиска?

Эвристика играет роль в процессе поиска, предоставляя дополнительные знания о структуре и свойствах исследуемого пространства, которые могут помочь алгоритму определить наилучший путь к цели.

4. Приведите пример применения эвристического поиска в реальной задаче.

Пример применения эвристического поиска — использование алгоритма A* для поиска кратчайшего пути между двумя точками на карте.

5. Почему полное исследование всех возможных ходов в шахматах затруднительно для ИИ?

Полное исследование всех возможных ходов в шахматах затруднительно для ИИ из-за огромного количества возможных комбинаций и высокой вычислительной сложности задачи.

6. Какие факторы ограничивают создание идеального шахматного ИИ?

Факторы, ограничивающие создание идеального шахматного ИИ, включают сложность правил игры, большое количество возможных ходов и высокую вычислительную сложность задачи.

7. В чем заключается основная задача искусственного интеллекта при выборе ходов в шахматах?

Основная задача искусственного интеллекта при выборе ходов в шахматах заключается в оценке позиции фигур на доске и определении оптимального хода. Алгоритмы ИИ используют различные критерии оценки, такие как количество материала и контроль над полями, чтобы определить лучший ход.

8. Как алгоритмы ИИ балансируют между скоростью вычислений и нахождением оптимальных решений?

Алгоритмы ИИ балансируют между скоростью вычислений и нахождением оптимальных решений, используя недетерминированные алгоритмы, такие как глубокие нейронные сети, которые могут адаптироваться к обучающим примерам и менять свои внутренние параметры.

9. Каковы основные элементы задачи поиска маршрута по карте?

Основные элементы задачи поиска маршрута по карте включают начальную и конечную точки маршрута, а также промежуточные пункты назначения.

10. Как можно оценить оптимальность решения задачи маршрутизации на карте Румынии?

Оптимальность решения оценивается по длине маршрута, времени в пути и затратам на перемещение между пунктами.

11. Что представляет собой исходное состояние дерева поиска в задаче маршрутизации по карте Румынии?

Исходное состояние дерева поиска в задаче маршрутизации по карте Румынии представляет собой граф дорог с указанием начальной и конечной точек, а также промежуточных пунктов назначения.

12. Какие узлы называются листовыми в контексте алгоритма поиска по дереву?

Листовые узлы в контексте алгоритма поиска по дереву — это узлы, которые находятся на самом низком уровне иерархии и не имеют дочерних узлов.

13. Что происходит на этапе расширения узла в дереве поиска?

На этапе расширения узла в дереве поиска происходит анализ соседних узлов и выбор наиболее подходящего для продолжения поиска.

14. Какие города можно посетить, совершив одно действие из Арада в примере задачи поиска по карте?

Города, которые можно посетить, совершив одно действие из Арада, включают Бухарест, Брашов, Сибиу, Сигишоара и другие города Румынии.

15. Как определяется целевое состояние в алгоритме поиска по дереву?

Целевое состояние в алгоритме поиска по дереву определяется как состояние, в котором достигается желаемый результат, например, нахождение кратчайшего пути или оптимального решения задачи маршрутизации.

16. Какие основные шаги выполняет алгоритм поиска по дереву?

Основные шаги алгоритма поиска по дереву включают: обход дерева (поиск в глубину или в ширину); выполнение операций прямого, обратного или централизованного обхода для каждого узла дерева; посещение левого поддерева, правого поддерева и самого узла.

17. Чем различаются состояния и узлы в дереве поиска?

Состояния и узлы в дереве поиска различаются следующим образом: состояния представляют собой вершины дерева, а узлы — элементы дерева, содержащие данные и ссылки на своих потомков.

18. Что такое функция преемника и как она используется в алгоритме поиска?

Функция преемника определяет переход от текущего состояния к следующему состоянию в процессе поиска. Она используется для определения следующего узла или вершины для обработки в алгоритме поиска.

19. Какое влияние на поиск оказывают такие параметры, как b (разветвление), d (глубина решения) и m (максимальная глубина)?

Параметры b , d и m влияют на глубину поиска и разветвление дерева. Они определяют максимальную глубину поиска, глубину решения и максимальное количество уровней ветвления соответственно.

20. Как алгоритмы поиска по дереву оцениваются по критериям полноты, временной и пространственной сложности, а также оптимальности?

Алгоритмы поиска по дереву оцениваются по критериям полноты, временной и пространственной сложности, а также оптимальности. Полнота означает, что алгоритм должен находить все возможные решения, временная сложность отражает скорость работы алгоритма, а пространственная сложность указывает на использование памяти. Оптимальность означает, что алгоритм находит оптимальное решение за минимальное время.

21. Какую роль выполняет класс `Problem` в приведенном коде?

Класс `Problem` выполняет роль абстрактного класса, определяющего общую структуру и поведение проблемы, которую нужно решить. Он содержит методы, необходимые для выполнения поиска и обработки результатов.

22. Какие методы необходимо переопределить при наследовании класса `Problem`?

При наследовании класса `Problem` необходимо переопределить методы, связанные с операциями поиска, такие как `is_goal` и `action_cost`.

23. Что делает метод `is_goal` в классе `Problem`?

Метод `is_goal` проверяет, достигнуто ли целевое состояние, и возвращает `true`, если да, иначе `false`.

24. Для чего используется метод `action_cost` в классе `Problem`?

Метод `action_cost` используется для вычисления стоимости перехода от одного состояния к другому. Он применяется в алгоритмах поиска по дереву для определения оптимального пути или решения.

25. Какую задачу выполняет класс `Node` в алгоритмах поиска?

Класс `Node` в алгоритмах поиска выполняет задачу предоставления универсального узла для связанного списка. Каждый узел содержит данные (ссылку на объект типа `E`) и ссылку (ссылку на следующий узел списка).

26. Какие параметры принимает конструктор класса `Node`?

Конструктор класса `Node` принимает два параметра: начальные данные (типа `E`) и ссылку на следующий узел (типа `Node`).

27. Что представляет собой специальный узел `failure`?

Специальный узел `failure` представляет собой узел, который указывает на сбой в алгоритме поиска. Он используется для обозначения того, что искомый элемент не найден.

28. Для чего используется функция `expand` в коде?

Функция `expand` используется для расширения пути, заданного в виде строки, до массива путей.

29. Какая последовательность действий генерируется с помощью функции `path_actions` ?

Функция `path_actions` генерирует последовательность действий, которые необходимо выполнить для достижения цели, заданной в виде пути.

30. Чем отличается функция `path_states` от функции `path_actions`?

Функция `path_states` создаёт список состояний пути, а функция `path_actions` создаёт список действий для достижения этих состояний.

31. Какой тип данных используется для реализации `FIFOQueue`?

Для реализации `FIFOQueue` используется тип данных «список».

32. Чем отличается очередь `FIFOQueue` от `LIFOQueue`?

Очередь FIFOQueue отличается от LIFOQueue тем, что в ней элементы удаляются из очереди только после того, как будут извлечены все предыдущие элементы. В LIFOQueue элементы удаляются после добавления нового элемента.

33. Как работает метод add в классе PriorityQueue?

Метод add в классе PriorityQueue работает аналогично методу add в обычном списке: добавляет элемент в конец очереди, сохраняя порядок приоритетов элементов.

34. В каких ситуациях применяются очереди с приоритетом?

Очереди с приоритетом применяются в ситуациях, когда необходимо обрабатывать элементы в определённом порядке, например, сортировать данные или выполнять операции с элементами в соответствии с их приоритетом.

35. Как функция heappop помогает в реализации очереди с приоритетом?

Функция heappop помогает извлекать элемент с наименьшим приоритетом из приоритетной очереди, удаляя его и возвращая значение этого элемента.