СРСП №3

WIN-1-22

Графовые алгоритмы и структуры данных

Выполнил:

Бактыбеков Н.Б.

Проверил:

Картанова А.Д.

May 9, 2023

Contents

Индивидуальное задание

Решить две задачи:

Задача 1

Составить программу, которая содержит текущую информацию о книгах в библиотеке.

Сведения о книгах содержат:

- Номер удк;
- Фамилию и инициалы автора;
- Название;
- Год издания;
- Количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать:

- Начальное формирование данных о всех книгах в библиотеке в виде списка;
- При взятии каждой книги вводится номер УДК, и программа уменьшает
- Значение количества книг на единицу или выдает сообщение о том, что
- Требуемой книги в библиотеке нет, или требуемая книга находится на руках;
- При возвращении каждой книги вводится номер УДК, и программа
- Увеличивает значение количества книг на единицу;
- По запросу выдаются сведения о наличии книг в библиотеке.

Задача 2

Написать 4 представления графа.

- Матрица смежности;
- Матрица инцидентности;
- Списки смежности;
- Списки Рёбер.

.А также реализивать алгоритм Дейкстры

1 Задача 1

1.1 Реализация программы на python:

Пояснение программы:

Для реализации данного задания на Python, в коде были созданы 2 класса **Book** и **Library** их методы и конструкторы классов.

А также ниже написан пример использования данной программы

Код программы:

```python
from prettytable import PrettyTable


class Book:
    def __init__(self, udk, author, title, year, quantity) -> None:
        self.udk: int = udk
        self.author: str = author
        self.title: str = title
        self.publishing_year: int = year
        self.book_quantity: int = quantity
        self.taken: int = 0

    def take(self) -> None:
        if self.book_quantity >= 1:
            self.book_quantity -= 1
            self.taken += 1
            print(f"You have take the book {self.title}")
        else:
            print("You can't take book, all of them already taken")

    def back(self) -> None:
        if self.taken >= 1:
            self.book_quantity += 1
            self.taken -= 1
            print(f"You have back the book {self.title}")
        else:
            print("You can't back this book, all of them already here")

    def get_udk(self) -> int:
        return self.udk

    def get_info(self) -> list[str]:
        return [
            str(i)
            for i in [
                self.udk,
                self.author,
                self.title,
                self.publishing_year,
                self.book_quantity,
            ]
        ]
```

```
43
44      def __str__(self) -> str:
45          return str(
46              [
47                  self.udk,
48                  self.author,
49                  self.title,
50                  self.publishing_year,
51                  self.book_quantity,
52              ]
53          )
54
55
56  class Library:
57      def __init__(self) -> None:
58          self.books: list[Book] = []
59
60      def add_book(self, book: Book) -> None:
61          self.books.append(book)
62
63      def take_book(self, udk: int) -> str:
64          for book in self.books:
65              if book.get_udk() == udk:
66                  book.take()
67                  print("You successfully took book")
68                  return "success"
69          return "fail"
70
71      def back_book(self, udk: int) -> str:
72          for book in self.books:
73              if book.get_udk() == udk:
74                  book.back()
75                  print("You successfully back book")
76                  return "success"
77          return "fail"
78
79      def search_book(self, udk: int) -> str:
80          for book in self.books:
81              if book.get_udk() == udk:
82                  print(book)
83                  return str(book)
84          print("book not found")
```

```
        return "not found"

    def show_books(self) -> None:
        table = PrettyTable()
        table.field_names = ["udk", "author", "title", "year", "quantity"]
        for book in self.books:
            table.add_row(book.get_info())

        print(table)


bishkek_state_library = Library()

# udk, author, title, year, quantity
bishkek_state_library.add_book(Book(111, "Aitamatov.Ch.T", "Jamila", 1999,
    11))
bishkek_state_library.add_book(Book(112, "Aitamatov.Ch.T", "Plaha", 1979,
    10))
bishkek_state_library.add_book(Book(113, "Aitamatov.Ch.T", "White ship",
    1969, 18))
bishkek_state_library.add_book(
    Book(114, "Aitamatov.Ch.T", "The day longer that thecentury", 1975, 9)
)
bishkek_state_library.add_book(
    Book(115, "Aitamatov.Ch.T", "Zeitunget Juidon", 1984, 21)
)
bishkek_state_library.add_book(Book(116, "Aitamatov.Ch.T", "Face to face",
    1971, 31))


bishkek_state_library.show_books()

bishkek_state_library.search_book(111)
bishkek_state_library.take_book(111)
bishkek_state_library.search_book(111)
bishkek_state_library.back_book(111)
bishkek_state_library.search_book(111)
```

1.2 Результат выполнения программы

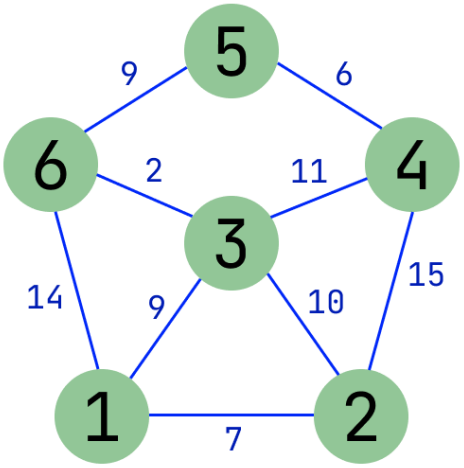
```
> py .\library.py
+-----+---------------+------------------------------+------+----------+
| udk |     author    |             title            | year | quantity |
+-----+---------------+------------------------------+------+----------+
| 111 | Aitamatov.Ch.T |             Jamila           | 1999 |    11    |
| 112 | Aitamatov.Ch.T |             Plaha            | 1979 |    10    |
| 113 | Aitamatov.Ch.T |           White ship         | 1969 |    18    |
| 114 | Aitamatov.Ch.T | The day longer that thecentury | 1975 |    9     |
| 115 | Aitamatov.Ch.T |        Zeitunget Juidon      | 1984 |    21    |
| 116 | Aitamatov.Ch.T |          Face to face        | 1971 |    31    |
+-----+---------------+------------------------------+------+----------+
[111, 'Aitamatov.Ch.T', 'Jamila', 1999, 11]
You have take the book Jamila
You successfully took book
[111, 'Aitamatov.Ch.T', 'Jamila', 1999, 10]
You have back the book Jamila
You successfully back book
[111, 'Aitamatov.Ch.T', 'Jamila', 1999, 11]
```

Picture. 1: Результат работы программы

2 Задача 2

2.1 Представления графа

Необходимо представить граф:



Picture. 2: Граф

Все четыре четыре способа способа представления графа можно реализовать на любом языке программирования используя 2-х или 3-х мерные массивы.

Table 1: Матрица смежности

№	1	2	3	4	5	6
1	0	7	9	0	0	14
2	7	0	10	15	0	0
3	9	10	0	11	0	2
4	0	15	11	0	6	0
5	0	0	0	6	0	9
6	14	0	2	0	9	0

Table 2: Матрица инцидентности

№	1-2	1-3	1-6	2-3	2-4	3-4	3-5	3-6	4-5	5-6
1	1	1	1	0	0	0	0	0	0	0
2	1	0	0	1	1	0	0	0	0	0
3	0	1	0	1	0	1	0	1	0	0
4	0	0	0	0	1	1	0	0	1	0
5	0	0	0	0	0	0	1	0	1	1
0	0	1	0	0	0	0	0	1	0	1

Table 3: Списки смежности

Номер вершины	Смежные вершины
1	2, 3, 6
2	1, 3, 4
3	1, 2, 4, 5
4	2, 3, 5
5	3, 4, 5
6	1, 3, 5

Table 4: Списки рёбер

Номер ребра	Вершины, соединенные этим ребром
1	1, 6
2	1, 3
3	1, 2
4	2, 3
5	2, 4
6	3, 4
7	3, 6
8	4, 5
9	5, 6

2.2 Алгоритм дейкстры

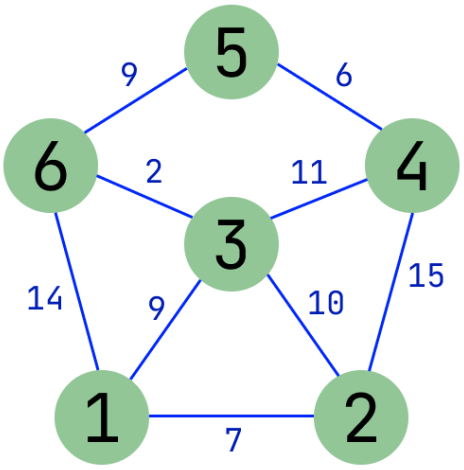
Словестное описание алгоритма:

В простейшей реализации для хранения чисел *d[i]* можно использовать массив чисел, а для хранения принадлежности элемента множеству U — массив булевых переменных.

В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бо□льшим 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 с флагом 0 $d[i] = \infty$. Последний случай возможен тогда и только тогда, когда граф G несвязный.

Блок схема алгоритма дейкстры



Picture. 3: Блок схема алгоритм Дейкстры

Реализация на языке python

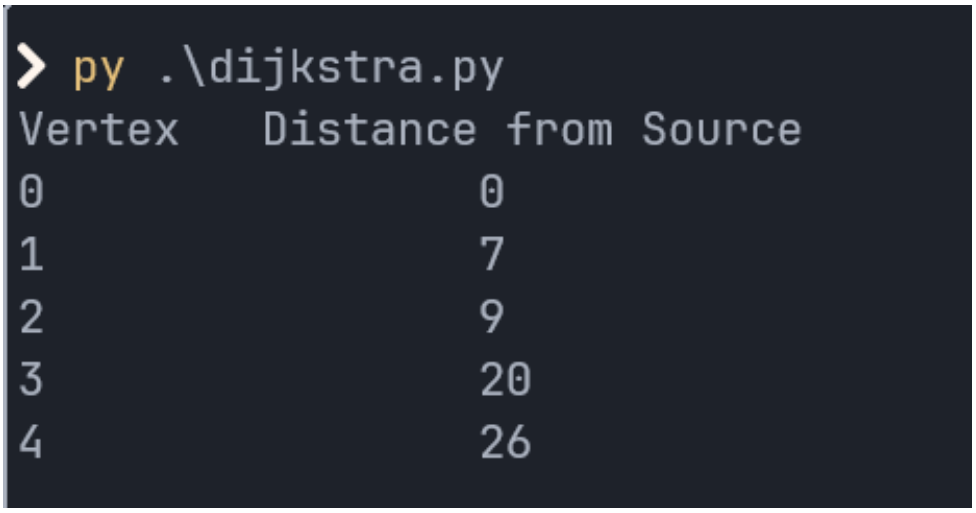
```python
class Graph:
    def __init__(self, vertices: int) -> None:
        self.vertices = vertices
        self.graph = [[0] * vertices for _ in range(vertices)]

    def print_solution(self, distances_from_source: list[int]) -> None:
        """
        Vertex   Distance from Source
        """
        print("Vertex \t Distance from Source")
        for vertex in range(self.vertices):
            print(vertex, "\t\t", distances_from_source[vertex])

    def minimum_distance(
        self, distances_from_source: list[int], visited: list[bool]
    ) -> int:
        """
        A utility function to find the vertex with minimum distance value,
    from the set
        of vertices not yet included in shortest path tree.
        """

        # Initialize minimum distance for next node
```

```
        minimum = 1e7
        min_index = 0

        # Search not nearest vertex not in the shortest path tree
        for vertex in range(self.vertices):
            if distances_from_source[vertex] < minimum and visited[vertex]
    is False:
                minimum = distances_from_source[vertex]
                min_index = vertex
        return min_index

    def dijkstra(self, source: int) -> None:
        """
        Function that implements Dijkstra's single source shortest path
    algorithm for a
        graph represented using adjacency matrix representation.
        """

        distances = [int(1e7)] * self.vertices  # distances from the source
        distances[source] = 0
        visited = [False] * self.vertices

        for _ in range(self.vertices):
            u = self.minimum_distance(distances, visited)
            visited[u] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shortest path tree
            for v in range(self.vertices):
                if (
                    self.graph[u][v] > 0
                    and visited[v] is False
                    and distances[v] > distances[u] + self.graph[u][v]
                ):
                    distances[v] = distances[u] + self.graph[u][v]

        self.print_solution(distances)


graph = Graph(5)
```

```
graph.graph = [
    [
        0,
        7,
        9,
        0,
        0,
        14,
    ],
    [
        7,
        0,
        10,
        15,
        0,
        0,
    ],
    [
        9,
        10,
        0,
        11,
        0,
        2,
    ],
    [
        0,
        15,
        11,
        0,
        6,
        0,
    ],
    [
        0,
        0,
        0,
        6,
        0,
        9,
    ],
    [
```

```
        14,
        0,
        2,
        0,
        9,
        0,
    ],
]
graph.dijkstra(0)
```

Результат выполнения программы



Picture. 4: Результат работы программы

Комплексность

Комплексность данного алгоритма $O(N^2)$

3 Ссылки

Ссылка на весь код представленный в этом документе, а также исходный код самого документа вы можете найти по этой ссылке

https://github.com/orenvadi/LAB3