**Лабораторная работа №2. Алгоритмы поиска в глубину и поиска в ширину**

## Цель: Рассмотреть реализацию алгоритмов поиска в глубину и ширину на задаче поиска выхода из лабиринта.

**Ход работы**

В лабораторной работе необходимо выполнить следующие задачи:

1. Используйте файл лабиринта из своей папки, в котором точка входа на верхней границе, а точка выхода - на нижней.
2. Ввести с клавиатуры точку (указать в каких пределах) расположения "сокровища", найти к нему путь (любой, поиск в глубину).
3. От точки сокровища узнать длину кратчайшего пути (поиск в ширину) к выходу (вывести) и построить его.
4. Сохранить в файл 'maze-for-me-done.txt', в котором точка сокровища будет указана как '\*', а сам маршрут построен точками к сокровищу и запятыми от него к выходу.

Ниже приведен листинг кода с комментариям, который при помощи алгоритмов поиска в глубину и ширину ищет точку выхода из лабиринта.

*from* collections *import* deque

maze\_o = *open*('maze-for-u.txt',"r")

opened\_maze = []

#открываем файл с лабиринтом и считываем данные в список

*for* line *in* maze\_o:

    opened\_maze.append(line[:-1])

POSSIBLE\_WAYS = ('N', 'S', 'W', 'E')

treja\_is\_found = False

#функция, которая проверяет, что заданные точки сокровища не выходят за пределы лабиринта

*def* coord\_in(opened\_maze, coord):

    len\_y, len\_x = *len*(opened\_maze), *len*(opened\_maze[0])

*if* coord[0]<0 or coord[0]>len\_x :

*return* False

*if* coord[1]<0 or coord[1]>len\_y:

*return* False

*return* True

#функция, которая выводит точки нахождения сокровища

*def* treasure\_clean(coord):

*global* treasure\_here

*if* coord[0] == treasure\_here[0] and coord[1] == treasure\_here[1]:

*print*('treasure\_is\_here!')

*return* True

*else*:

*return* False

#функция, которая возвращает

*def* split(s):

*return* [char *for* char *in* s]

*def* *exit*(coord):

*if* coord[0]>9:

*return* False

*return* True

#функция, которая проверяет, что заданная точка сокровища не попадает в координату с '#'

*def* path\_clean(opened\_maze, coord):

*if* opened\_maze[coord[0]][coord[1]] == '#':

*return* False

*return* True

#функция, которая осуществляет шаги в заданных направлениях

*def* step(coord, direction):

*if* direction == 'N':

*return* step\_N(coord)

*elif* direction == 'S':

*return* step\_S(coord)

*elif* direction == 'E':

*return* step\_E(coord)

*elif* direction == 'W':

*return* step\_W(coord)

*def* step\_N(coord):

*return* [coord[0]-1, coord[1]]

*def* step\_E(coord):

*return* [coord[0], coord[1]+1]

*def* step\_S(coord):

*return* [coord[0]+1, coord[1]]

*def* step\_W(coord):

*return* [coord[0], coord[1]-1]

функция, которая осуществляет по диагонали

*def* the\_way\_back(direction):

*if* direction == "N":

*return* ("N","E","W")

*if* direction == "S":

*return* ("S","E","W")

*if* direction == "E":

*return* ("N","E","S")

*if* direction == "W":

*return* ("N","S","W")

Алгоритм поиска в ширину позволяет найти кратчайшие пути из одной вершины невзвешенного (ориентированного или неориентированного) графа до всех остальных вершин. Под кратчайшим путем подразумевается путь, содержащий наименьшее число ребер.

Шаги алгоритма поиска в ширину для поиска путей лабиринта:

1. Поместить координату текущей точки лабиринта, с которого начинается поиск, в изначально пустую очередь.
2. Извлечь из начала очереди координату {\displaystyle u}.
   * Если координата  {\displaystyle u}является целевым точкой, т. е. из нее нет путей, то завершить поиск с результатом «успех».
   * В противном случае, в конец очереди добавляются все координаты в {\displaystyle u}которые можно перейти из текущей.
3. Если очередь пуста, то все точки были просмотрены, следовательно, целевая точка недостижима из начальной; завершить поиск с результатом «неудача».
4. Вернуться к п. 2.

Ниже представлен код функция поиска в ширину, который работает путём последовательного просмотра координат точек отдельных путей, начиная с заданной точки сокровища {\displaystyle u}.

*def* bfs(opened\_maze,coord,t):

    n, m = *len*(opened\_maze), *len*(opened\_maze[0])

    INF =  0

    p = [[None]\*m *for* \_ *in* *range*(n)]

    d = [[INF]\*m *for* \_ *in* *range*(n)]

    used = [[False]\*m *for* \_ *in* *range*(n)]

    queue = deque()

    delta = [(0,-1),(0,1),(1,0),(-1,0)]

    d[coord[0]][coord[1]] = 0

    used[coord[0]][coord[1]] = True

    queue.append(coord)

*while* *len*(queue)!= 0:

        x,y = queue.popleft()

*for* dx,dy *in* delta:

            nx,ny = x + dx, y + dy

*if* 0 <= nx <n and 0 <= ny < m and not used[nx][ny] and opened\_maze[nx][ny] != "#":

                d[nx][ny] = d[x][y] + 1

                p[nx][ny] = (x,y)

                used[nx][ny] = True

                queue.append((nx,ny))

*print*(d[t[0]][t[1]])

    cur = t

    path = []

*while* cur is not None:

        path.append(cur)

        cur = p[cur[0]][cur[1]]

    path.reverse()

    count = 0

*for* i *in* path:

        ys = path[count][1]

        xs = path[count][0]

        b = split(opened\_maze[xs])

        b[ys] = ","

        opened\_maze[xs] = ''.join(b)

        count+=1

Алгоритм поиска (или обхода) в глубину (англ. depth-first search, DFS) позволяет построить обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины.

Отличие поиска в глубину от поиска в ширину заключается в том, что (в случае неориентированного графа) результатом алгоритма поиска в глубину является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины. Этим он принципиально отличается от поиска в ширину, где одновременно обрабатывается множество вершин, в поиске в глубину в каждый момент исполнения алгоритма обрабатывается только одна вершина. С другой стороны, поиск в глубину не находит кратчайших путей, зато он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.

Если же граф ориентированный, то поиск в глубину строит дерево путей из начальной вершины во все доступные из нее.

Обход в глубину можно представить себе следующим образом. Пусть исследователь находится в некотором лабиринте (графе) и он хочет обойти весь лабиринт (посетить все доступные вершины в графе). Исследователь находится в некоторой вершине и видит ребра, исходящие из этой вершины. Очевидная последовательность действий исследователя такая:

1. Пойти в какую-нибудь смежную вершину.
2. Обойти все, что доступно из этой вершины.
3. Вернуться в начальную вершину.
4. Повторить алгоритм для всех остальных вершин, смежных из начальной.

Ниже представлен код функция поиска в глубину, начиная с заданной точки сокровища.

*def* find\_way(opened\_maze, coord,possible\_ways):

    len\_y, len\_x = *len*(opened\_maze), *len*(opened\_maze[0])

*global* path\_to\_exit, current\_path , treja\_is\_found

*if* not coord\_in(opened\_maze, coord):

*print*('not in bounds!')

*return*

*if* *len*(current\_path) > *len*(path\_to\_exit):

*print*('too large path')

*return*

*for* direction *in* possible\_ways:

*if* treasure\_clean(step(coord, direction)) or treja\_is\_found:

            treja\_is\_found = True

            b = split(opened\_maze[coord[0]])

            b[coord[1]] = "."

            opened\_maze[coord[0]] = ''.join(b)

*return*

*else*:

*if* path\_clean(opened\_maze, step(coord, direction)) :

                current\_path.append(direction)

                find\_way(opened\_maze, step(coord, direction),the\_way\_back(direction))

                current\_path.pop()

*if* treja\_is\_found == False:

                    b = split(opened\_maze[coord[0]])

                    b[coord[1]] = "."

                    opened\_maze[coord[0]] = ''.join(b)

*return*

path\_to\_exit = []

*for* i *in* *range*(*len*(opened\_maze)\**len*(opened\_maze[0])):

    path\_to\_exit.append(' ')

current\_path = []

start\_point = [0, 1]

x = int(*input*("коорд х трежи: "))

y = int(*input*("коорд y трежи: "))

coord\_point = [y,x]

t\_coord = [118,141]

treasure\_here = (y,x)

find\_way(opened\_maze, start\_point,POSSIBLE\_WAYS)

bfs(opened\_maze,coord\_point,t\_coord)

b = split(opened\_maze[y])

b[x] = "\*"

opened\_maze[y] = ''.join(b)

*with* *open*("opened\_maze-for-me-done.txt", "w") *as* txt\_file:

*for* line *in* opened\_maze:

        txt\_file.write(" ".join(line) + "\n")

После выполнения алгоритмов поиска в ширину и глубину сформирован файл с построенным маршрутом от заданной точки сокровища и до выхода из лабиринта.

На рисунках 1-3 приведен лабиринт и найденный путь.

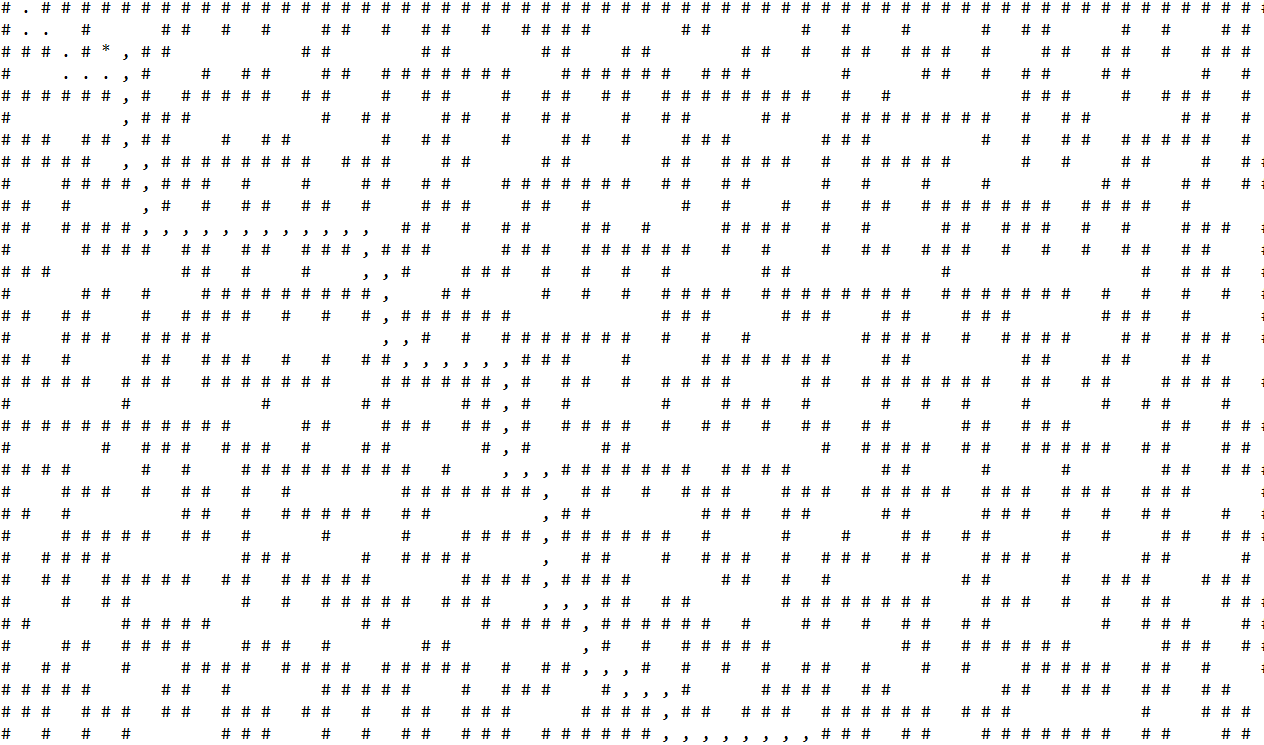


Рисунок 1

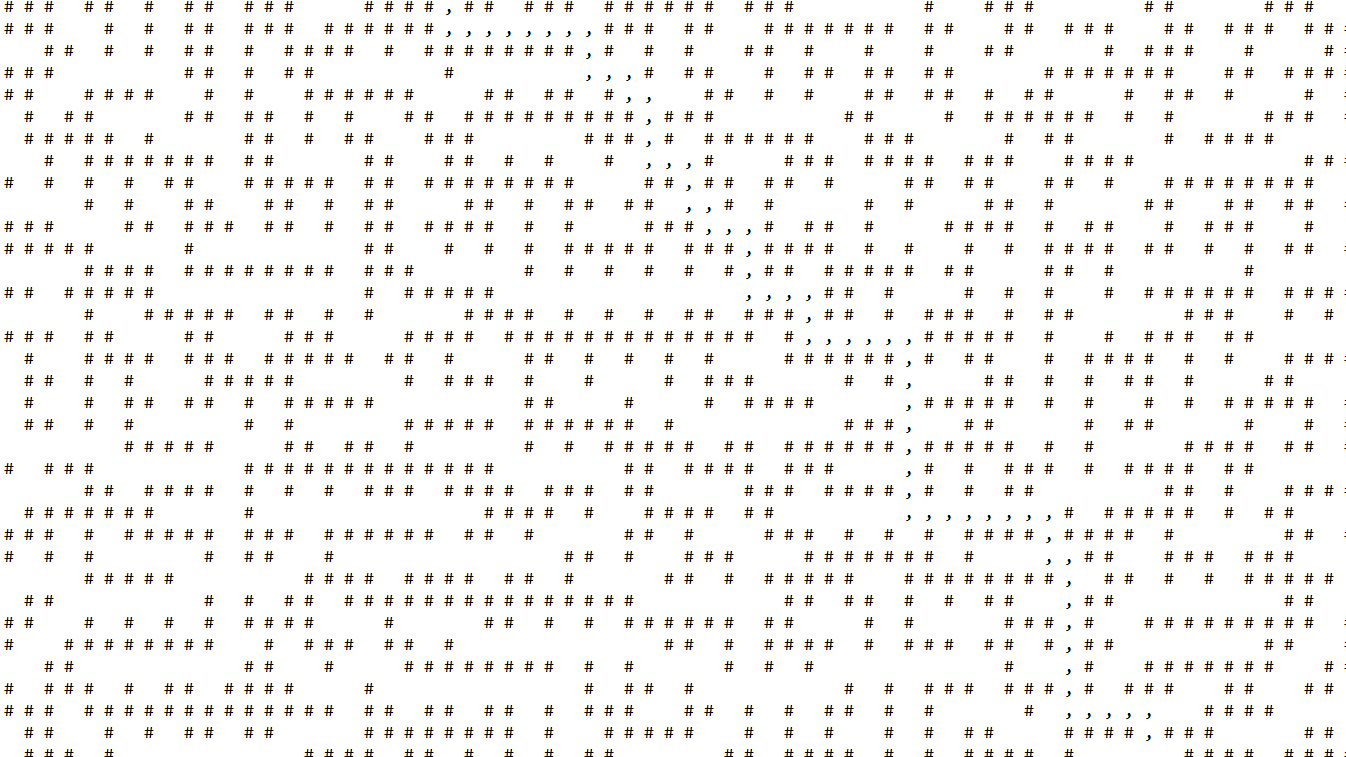


Рисунок 2



Рисунок 3

Вывод: В лабораторной работе рассмотрены и реализованы алгоритмы поиска в глубину и ширину, для кратчайшего нахождения пути в лабиринте.