Санкт-Петербургский Государственный Технический Университет
(Технологический институт)

Кафедра системного анализа и информационных технологий

Лабораторная работа №6

Выполнили:

Лелякин А. А., Люцай В.Н.

Проверил

Мусаев А.А.

Санкт-Петербург,

2022

Ход работы

Задание 1

“Написать программу, которая определяет, является ли введенная скобочная структура правильной.

Примеры правильных скобочных выражений: (), (())(), ()(), ((())), неправильных —)(, ())((), (,))))), (((.

Найдите порядковый номер первого символа (скобки), нарушающего правильность расстановки скобок.”

Выполнение:

Мы написали программу, которая определяет, является ли введенная скобочная структура правильной. Под правильной скобочной структурой подразумевается такая структура, которая имеет смысл при построении в предложении. К примеру: ()(()()), ((())), ()((()()())(())) и т.д. Код программы (рис.1):

```
a = input().split()
c = []
k = []
f = False
for i in range (len(a)):
    if a[i] == '(':
        c.append(a[i])
        k.append(i)
    if a[i] == ')':
        if len(c) == 0:
            print('Ну не повезло, лишняя закрывающая скобка на', i+1,'позиции')
            f = True
            break
        if len(c) > 0:
            if c[-1] == '(':
                c.pop()
                k.pop()

if len(c) == 0 and f == False:
    print('Ура!!! Последовательность корректная')
if len(c)>0 and f == False:
    print('Эх, не повезло, скобочка открылась, однако не закрылась на позиции', k[-1]+1)
```

рис.1

Также программа находит порядковый номер первого символа (скобки), нарушающего правильную последовательность расстановки.

Задание 2

“Придумайте и решите задачу для алгоритма поиска в глубину. Придумайте и решите задачу для алгоритма поиска в ширину. Объясните, почему для решения поставленных задач были выбраны именно эти алгоритмы поиска (подразумевается возможность выбора и других алгоритмов для решения поставленной задачи).”

Выполнение:

Мы написали программу, которая использует алгоритм поиска в ширину, используя произвольный граф (рис.2):

```
graph = {'1': ['2', '5'],
         '2': ['1', '3', '6', '7'],
         '3': ['2', '4', '8'],
         '4': ['3', '8'],
         '5': ['1', '7'],
         '6': ['2'],
         '7': ['2', '5', '8'],
         '8': ['3', '4', '7', '9'],
         '9': ['8']}
```

рис.2

Сама функция поиска в ширину выглядит так (рис. 3):

```
from collections import deque
def shirina(start, end, graph):
    queue = deque([start])
    visited = {start: None}
    while queue:
        a = queue.popleft()
        if a == end:
            break
        b = graph[a]
        for c in b:
            if c not in visited:
                queue.append(c)
                visited[c] = a
    return visited
```

рис.3

Зададим начальную и конечную точку (рис.4):

```
start = '1'
end = '9'
```

рис.4

Завершение кода (рис.5):

```
visited = shirina(start, end, graph)
a = end
print(f'Путь от {end} до {start}: {end} ', end='')
while a != start:
    a = visited[a]
    print(f'-> {a} ', end='')
```

рис.5

Итого, консоль выводит следующее (рис.6):

```
Путь от 9 до 1: 9 -> 8 -> 3 -> 2 -> 1
```

рис.6

Мы написали программу, которая использует алгоритм поиска в глубину (рис. 7):

```
def glubina(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for next in graph[start] - visited:
        glubina(graph, next, visited)
    return visited


graph = {'0': set(['1', '2']),
         '1': set(['0', '3', '4']),
         '2': set(['0']),
         '3': set(['1']),
         '4': set(['2', '3'])}

print(glubina(graph, '0'))
```

рис.7

Итого имеем (рис.8):

```
{'1', '0', '2', '4', '3'}
```

рис.8

Итак, рассмотрим два метода поиска, рассмотренные выше. Поиск в ширину выгодно использовать, когда важно оптимизировать маршрут, тем самым сделал его наиболее коротким, однако анализ такого маршрута займет достаточно много времени, особенно, если речь идет о большом количестве точек. Поиск в глубину, в свою очередь, зачастую используется в том случае, если нам не важно, каким путем мы попадем в пункт назначения, при этом поиск такого любого маршрута займет куда меньше времени, чем анализ всех возможных путей в поиске в ширину.

Задание 3

“Дана случайная квадратная матрица, заполненная нулями и единицами. Предположив, что 0 – это проход, а 1 – это стена, напишите алгоритм, который найдет выход из лабиринта”

Выполнение:

Мы написали программу, которая находит выход из лабиринта, состоящего из нулей и единиц, где 0 – проход, а 1 – стена (рис.9-10):

```
a = [
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1],
    [1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1],
    [1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
]
zoom = 20
borders = 6
start = 1,1
end = 5,19

def make_step(k):
    for i in range(len(m)):
        for j in range(len(m[i])):
            if m[i][j] == k:
                if i>0 and m[i-1][j] == 0 and a[i-1][j] == 0:
                    m[i-1][j] = k + 1
                if j>0 and m[i][j-1] == 0 and a[i][j-1] == 0:
                    m[i][j-1] = k + 1
                if i<len(m)-1 and m[i+1][j] == 0 and a[i+1][j] == 0:
                    m[i+1][j] = k + 1
                if j<len(m[i])-1 and m[i][j+1] == 0 and a[i][j+1] == 0:
                    m[i][j+1] = k + 1

def print_m(m):
    for i in range(len(m)):
        for j in range(len(m[i])):
            print(str(m[i][j]).ljust(2), end=' ')
        print()

m = []
for i in range(len(a)):
    m.append([])
    for j in range(len(a[i])):
```

рис.9

```
        m[-1].append(0)
i,j = start
m[i][j] = 1

k = 0
while m[end[0]][end[1]] == 0:
    k += 1
    make_step(k)

i, j = end
k = m[i][j]
the_path = [(i,j)]
while k > 1:
    if i > 0 and m[i - 1][j] == k-1:
        i, j = i-1, j
        the_path.append((i, j))
        k-=1
    elif j > 0 and m[i][j - 1] == k-1:
        i, j = i, j-1
        the_path.append((i, j))
        k-=1
    elif i < len(m) - 1 and m[i + 1][j] == k-1:
        i, j = i+1, j
        the_path.append((i, j))
        k-=1
    elif j < len(m[i]) - 1 and m[i][j + 1] == k-1:
        i, j = i, j+1
        the_path.append((i, j))
        k -= 1

print_m(m)
print(the_path)
```

рис.10

Данная программа использует метод поиска в ширину.