Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**НЕЙРОННЫЕ СЕТИ**

**ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ**

студентки 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Стыценковой Валерии Сергеевны

|  |  |  |
| --- | --- | --- |
| Научный руководитель  Доцент | \_\_\_\_\_\_\_\_\_\_ | И. И. Слеповичев |
|  | подпись, дата |  |
|  |  |  |

Саратов 2023

**2.1 Создание ориентированного графа**

На входе: текстовый файл с описанием графа в виде списка дуг:

(a\_1, b\_1, n\_1), (a\_2, b\_2, n\_2), ..., (a\_k, b\_k, n\_k)

где a\_i – начальная вершина дуги i, b\_i – конечная вершина дуги i, n\_i – порядковый номер дуги в списке всех заходящих в вершину b\_i дуг.

На выходе:

а) Ориентированный граф с именованными вершинами и линейно упорядоченными дугами (в соответствии с порядком из текстового файла).

б) Сообщение об ошибке в формате файла, если ошибка присутствует.

Способ проверки результата:

a) Сериализованная структура графа в формате XML или JSON.

Пример:

<graph>

<vertex>v1</vertex>

<vertex>v2</vertex>

<vertex>v3</vertex>

<arc>

<from>v1</from>

<to>v3</to>

<order>1</order>

</arc>

<arc>

<from>v2</from>

<to>v3</to>

<order>2</order>

</arc>

</graph>

б) Сообщение об ошибке с указанием номера строки с ошибкой во входном файле.

Листинг программы, реализованной на языке Python, представлен в приложении А.

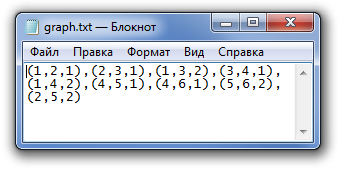


Рисунок 2.1.1 – Входной файл, описывающий ребра графа

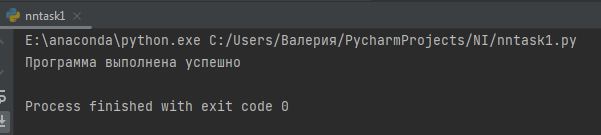


Рисунок 2.1.2 – Завершение работы программы

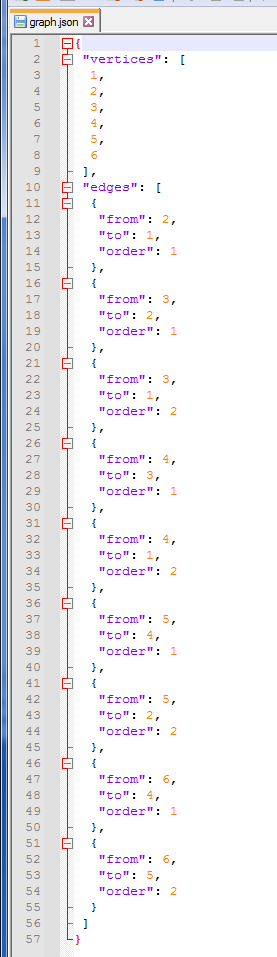


Рисунок 2.1.3 – Результат работы программы

**2.2 Создание функции по графу**

На входе: ориентированный граф с именованными вершинами как описано в задании 1.

На выходе: линейное представление функции, реализуемой графом в префиксной скобочной записи:

A1(B1(C1(...),..., Cm(...)),..., Bn(...))

Способ проверки результата:

a) выгрузка в текстовый файл результата преобразования графа в имя функции.

б) сообщение о наличии циклов в графе, если они присутствуют.

Листинг программы, реализованной на языке Python, представлен в приложении Б.

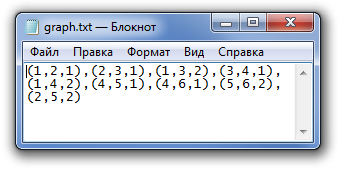


Рисунок 2.2.1 – Входной файл, описывающий ребра графа

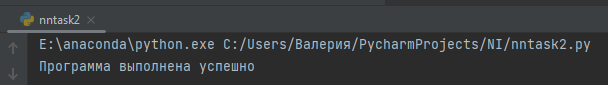


Рисунок 2.2.2 – Завершение работы программы

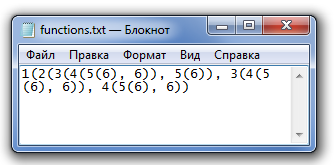


Рисунок 2.2.3 – Линейное представление функции, реализуемой графом

**2.3 Вычисление значения функции на графе**

На входе:

а) Текстовый файл с описанием графа в виде списка дуг (см. задание 1).

б) Текстовый файл соответствий арифметических операций именам вершин:

{

a\_1 : операция\_1

a\_2 : операция\_2

...

a\_n : операция\_n

}

где a\_i - имя i-й вершины, операция\_i - символ операции, соответствующий вершине a\_i.

Допустимы следующие символы операций:

* + – cумма значений,
* – произведение значений,
* exp – экспонирование входного значения,
* число – любая числовая константа.

На выходе: значение функции, построенной по графу а) и файлу б).

Способ проверки результата: результат вычисления, выведенный в файл.

Листинг программы, реализованной на языке Python, представлен в приложении В.

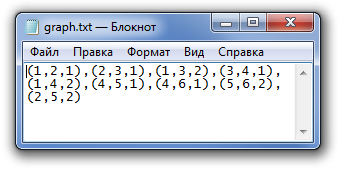


Рисунок 2.3.1 – Входной файл, описывающий ребра графа

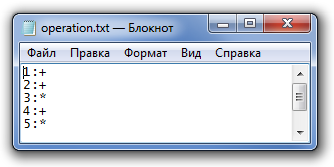


Рисунок 2.3.2 – Входной файл соответствий арифметических операций именам вершин

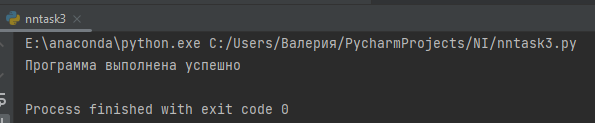


Рисунок 2.3.3 – Завершение работы программы

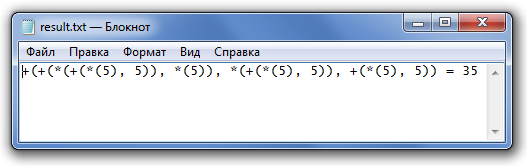


Рисунок 2.3.4 – Значение функции, построенное по графу и заданному соответствию

ПРИЛОЖЕНИЕ А

import argparse

import json

def read\_graph(inp):

try:

input\_graph = open(inp, 'r')

line = input\_graph.read()

except Exception:

print("Файл не найден")

exit()

text = ''

for i in line.split():

text += i

tmp = text[1:-1].split('),(')

edges = []

for i in tmp:

edges.append(list(map(int, i.split(','))))

maximum = 0

for i in range(len(edges)):

maximum = max(maximum, edges[i][0], edges[i][1])

graph = [[] for \_ in range(maximum)]

count = [0 for \_ in range(maximum)]

x = 0

for i in edges:

try:

graph[i[1] - 1].find(i[0] - 1)

print("Такое ребро уже существует", "Ошибка в строке", x + 1)

exit()

except Exception:

graph[i[1] - 1].append(i[0] - 1)

if count[i[1] - 1] == i[2] - 1:

count[i[1] - 1] += 1

else:

print("Нарушен порядок перечисления рёбер", "Ошибка в строке", x + 1)

exit()

return graph

def write\_to\_json(graph, out):

vertex = []

for i in range(len(graph)):

vertex.append(i + 1)

edges = []

for i in graph:

if i != []:

for j in i:

edges.append({'from': graph.index(i) + 1, 'to': j + 1, 'order': i.index(j) + 1})

ans = {'vertices': vertex, 'edges': edges}

with open(out, 'w') as outfile:

json.dump(ans, outfile, indent=1)

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('-inp', default='graph.txt')

parser.add\_argument('-out', default='graph.json')

args = parser.parse\_args()

values = write\_to\_json(read\_graph(args.inp), args.out)

print("Программа выполнена успешно")

main()

**ПРИЛОЖЕНИЕ Б**

import argparse

import json

def read\_graph(inp):

try:

input\_graph = open(inp, 'r')

line = input\_graph.read()

except Exception:

print("Файл не найден")

exit()

text = ''

for i in line.split():

text += i

tmp = text[1:-1].split('),(')

edges = []

for i in tmp:

edges.append(list(map(int, i.split(','))))

maximum = 0

for i in range(len(edges)):

maximum = max(maximum, edges[i][0], edges[i][1])

graph = [[] for \_ in range(maximum)]

count = [0 for \_ in range(maximum)]

x = 0

for i in edges:

try:

graph[i[1] - 1].find(i[0] - 1)

print("Такое ребро уже существует", "Ошибка в строке", x + 1)

exit()

except Exception:

graph[i[1] - 1].append(i[0] - 1)

if count[i[1] - 1] == i[2] - 1:

count[i[1] - 1] += 1

else:

print("Нарушен порядок перечисления рёбер", "Ошибка в строке", x + 1)

exit()

return graph

def dfs(graph, v, marks):

marks[v] = 1

for i in graph[v]:

if marks[i] == 0:

marks = dfs(graph, i, marks)

if marks[i] == 1:

print("В графе есть циклы")

exit()

marks[v] = 2

return marks

def source\_list(v, graph):

ans = [v]

for i in range(len(graph)):

for j in graph[i]:

if j == v:

l = source\_list(i, graph)

if l != []:

ans.append(l)

return ans

def list\_to\_string(l):

ans = ''

for i in range(len(l)):

if type(l[i]) == int:

f = False

for j in range(i + 1, len(l)):

if type(l[j]) == list:

f = True

if f:

ans += str(l[i] + 1) + '('

for j in range(i + 1, len(l)):

ans += list\_to\_string(l[j])

if j != len(l) - 1:

ans += ', '

ans += ')'

else:

for j in range(i, len(l)):

ans += str(l[j] + 1)

if j != len(l) - 1:

ans += ', '

return ans

def write\_func(inp, out):

graph = read\_graph(inp)

marks = []

for \_ in graph:

marks.append(0)

v = 0

while True:

if v == len(marks):

break

if marks[v] == 0:

marks = dfs(graph, v, marks)

v += 1

f = []

for i in range(len(graph)):

if graph[i] == []:

f.append(i)

ans = []

for i in f:

ans.append(list\_to\_string(source\_list(i, graph)))

output\_graph = open(out, 'w')

for i in ans:

output\_graph.write(i + '\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('-inp', default='graph.txt')

parser.add\_argument('-out', default='functions.txt')

args = parser.parse\_args()

values = write\_func(args.inp, args.out)

print("Программа выполнена успешно")

main()

**ПРИЛОЖЕНИЕ В**

import argparse

import json

import math

import warnings

warnings.filterwarnings("ignore")

def read\_graph(inp):

try:

input\_graph = open(inp, 'r')

line = input\_graph.read()

except Exception:

print("Файл не найден")

exit()

text = ''

for i in line.split():

text += i

tmp = text[1:-1].split('),(')

edges = []

for i in tmp:

edges.append(list(map(int, i.split(','))))

maximum = 0

for i in range(len(edges)):

maximum = max(maximum, edges[i][0], edges[i][1])

graph = [[] for \_ in range(maximum)]

count = [0 for \_ in range(maximum)]

x = 0

for i in edges:

try:

graph[i[1] - 1].find(i[0] - 1)

print("Такое ребро уже существует", "Ошибка в строке", x + 1)

exit()

except Exception:

graph[i[1] - 1].append(i[0] - 1)

if count[i[1] - 1] == i[2] - 1:

count[i[1] - 1] += 1

else:

print("Нарушен порядок перечисления рёбер", "Ошибка в строке", x + 1)

exit()

return graph

def dfs(graph, v, marks):

marks[v] = 1

for i in graph[v]:

if marks[i] == 0:

marks = dfs(graph, i, marks)

if marks[i] == 1:

print("В графе есть циклы")

exit()

marks[v] = 2

return marks

def source\_list(v, graph):

ans = [v]

for i in range(len(graph)):

for j in graph[i]:

if j == v:

l = source\_list(i, graph)

if l != []:

ans.append(l)

return ans

def list\_to\_string(l):

ans = ''

for i in range(len(l)):

if type(l[i]) == int:

f = False

for j in range(i + 1, len(l)):

if type(l[j]) == list:

f = True

if f:

ans += str(l[i] + 1) + '('

for j in range(i + 1, len(l)):

ans += list\_to\_string(l[j])

if j != len(l) - 1:

ans += ', '

ans += ')'

else:

for j in range(i, len(l)):

ans += str(l[j] + 1)

if j != len(l) - 1:

ans += ', '

return ans

def to\_string(inp):

graph = read\_graph(inp)

marks = []

for \_ in graph:

marks.append(0)

v = 0

while True:

if v == len(marks):

break

if marks[v] == 0:

marks = dfs(graph, v, marks)

v += 1

f = []

for i in range(len(graph)):

if graph[i] == []:

f.append(i)

ans = []

for i in f:

ans.append(list\_to\_string(source\_list(i, graph)))

return ans

def my\_exp(x):

return math.exp(x)

def my\_sum(\*args):

ans = 0

for i in args:

ans += i

return ans

def my\_mult(\*args):

ans = 1

for i in args:

ans \*= i

return ans

def caluculate(inp1, inp2, out):

string = to\_string(inp1)

calc\_string = []

try:

operations = open(inp2, 'r')

old\_line = operations.read().split('\n')

line = []

for i in old\_line:

line.append(i.split(':'))

except Exception:

print("Файл не найден")

exit()

while True:

for i in range(len(line)):

if line[i] == ['']:

break

if i != len(line) - 1 or (i == len(line) - 1 and line[i] == ['']):

line.pop(i)

else:

break

for i in range(len(string)):

for j in line:

try:

index = 0

while True:

index = string[i].find(j[0], index)

if index == -1:

break

if index == 0:

string[i] = string[i].replace(j[0], '\_' + j[1])

elif string[i][index - 1] != '\_' and index != -1:

string[i] = string[i].replace(string[i][index - 1] + j[0], string[i][index - 1] + '\_' + j[1])

index += 1

except Exception:

do\_nothing = 0

index = 0

while True:

index = string[i].find('\_')

if index != -1:

string[i] = string[i][:index] + string[i][index + 1:]

else:

break

calc\_string.append(string[i])

index = 0

while True:

index = calc\_string[i].find('\*', index)

if index == 0:

calc\_string[i] = calc\_string[i].replace('\*', 'my\_mult')

elif calc\_string[i][index - 1] != '\_' and index != -1:

calc\_string[i] = calc\_string[i].replace(calc\_string[i][index - 1] + '\*',

calc\_string[i][index - 1] + 'my\_mult')

else:

break

index += 1

index = 0

while True:

index = calc\_string[i].find('+', index)

if index == 0:

calc\_string[i] = calc\_string[i].replace('+', 'my\_sum')

elif calc\_string[i][index - 1] != '\_' and index != -1:

calc\_string[i] = calc\_string[i].replace(calc\_string[i][index - 1] + '+',

calc\_string[i][index - 1] + 'my\_sum')

else:

break

index += 1

index = 0

while True:

index = calc\_string[i].find('exp', index)

if index == 0:

calc\_string[i] = calc\_string[i].replace('exp', 'my\_exp')

elif calc\_string[i][index - 1] != '\_' and index != -1:

calc\_string[i] = calc\_string[i].replace(calc\_string[i][index - 1] + 'exp',

calc\_string[i][index - 1] + 'my\_exp')

else:

break

index += 1

ans = []

for i in calc\_string:

try:

ans.append(eval(i))

except Exception:

ans.append('Ошибка')

with open(out, 'w') as calculated\_functions:

for i in range(len(calc\_string)):

calculated\_functions.write(string[i] + ' = ' + str(ans[i]) + '\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('-inp1', default='graph.txt')

parser.add\_argument('-inp2', default='operation.txt')

parser.add\_argument('-out', default='result.txt')

args = parser.parse\_args()

values = caluculate(args.inp1, args.inp2, args.out)

print("Программа выполнена успешно")

main()