Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

Выполнил:

студент группы 20ВВ2

Беляков А.М.

Проверили:

Митрохин М.А.

Юрова О.В.

Пенза 2021

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного графа G. Выведите матрицу на экран.

2. Для сгенерированного графа осуществите процедуру обхода в ширину,

реализованную в соответствии с приведенным выше описанием. При

реализации алгоритма в качестве очереди используйте класс queue из

стандартной библиотеки С++.

3.\* Реализуйте процедуру обхода в ширину для графа, представленного

списками смежности.

Задание 2\*

1. Для матричной формы представления графов реализуйте алгоритм

обхода в ширину с использованием очереди, построенной на основе

структуры данных «список», самостоятельно созданной в лабораторной

работе № 3.

2. Оцените время работы двух реализаций алгоритмов обхода в ширину

(использующего стандартный класс queue и использующего очередь,

реализованную самостоятельно) для графов разных порядков.

Изображение выглядит как текст

Автоматически созданное описание

Main.py

from Graph import Graph  
  
graph = Graph(name="v")  
  
for i in range(5):  
 graph.add\_node(i)  
  
graph.connect(0,1)  
  
graph.connect(2,1)  
graph.connect(3,1)  
graph.connect(4,0)  
  
graph.print()  
print(graph.bfs(graph.nodes[0]))

Graph.py

def bfs(self, node):  
 return [self.\_matrix.bfs(node), self.\_adjacency\_list.bfs(node)]

Matrix.py

def bfs(self, node):  
 visits = []  
 queue = Queue()  
 queue.add(node)  
 while len(queue):  
 item = queue.pop()  
 visits.append(item.index)  
 for i in range(len(self.\_matrix[item.index])):  
 if len(self.\_matrix[item.index][i]) >= 1 and i not in visits:  
 queue.add(self.\_nodes[i])  
 return visits

AdjacencyList.py

def bfs(self, node):  
 visits = []  
 queue = Queue()  
 queue.add(node)  
 while len(queue):  
 item = queue.pop()  
 visits.append(item.index)  
 for i in range(len(self.\_adj\_list[item])):  
 if self.\_adj\_list[item][i][0].index not in visits:  
 queue.add(self.\_adj\_list[item][i][0])  
 return visits

Queue.py

class Queue():  
 def \_\_init\_\_(self):  
 self.\_first = None  
 self.\_last = None  
 self.\_len = 0  
  
 def add(self, data):  
 node = QueueNode(data)  
 if self.\_first is None:  
 self.\_first = node  
 self.\_last = node  
 else:  
 self.\_last.next = node  
 self.\_last = node  
 self.\_len += 1  
  
 def pop(self):  
 to\_ret = self.\_first  
 self.\_first = self.\_first.next  
 self.\_len -= 1  
 return to\_ret.data  
  
 def \_\_len\_\_(self):  
 return self.\_len  
  
  
class QueueNode():  
 def \_\_init\_\_(self, data):  
 self.\_data = data  
 self.\_next = None  
  
 @property  
 def data(self):  
 return self.\_data  
  
 @data.setter  
 def data(self, new):  
 self.\_data = new  
  
 @property  
 def next(self):  
 return self.\_next  
  
 @next.setter  
 def next(self, new):  
 self.\_next = new

Сравнение производительности:

Для графов из 1000 нод и 10000 связей.

Изображение выглядит как текст

Автоматически созданное описание

1. Время для обхода в глубину на классе Queue и матрице
2. Время для обхода в глубину на классе list и матрице
3. Время для обхода в глубину по списку смежности

Как мы можем увидеть обход по списку смежности работает быстрее чем по матрице, и то что класс list работает быстрее чем Queue.

Вывод: Мы научились реализовывать обход графа в глубину и создали класс очереди Queue