**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний**

**інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 5 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

**«Спискові структури даних»**

**Виконав** ІП-11Трикош Іван Володимирович

**Перевірила** Халус Олена Андріївна

Київ 2022

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації евристичних

алгоритмів і вирішення типових задач з їх допомогою.

2 ЗАВДАННЯ

Розробити алгоритм розв’язання задачі відповідно до варіанту. Виконати програмну реалізацію задачі. Не використовувати вбудовані спискові структури даних (контейнери). Зробити висновок по лабораторній роботі.

31. Реалізуйте структуру "черга з пріоритетом", яка підтримує наступні операції: додавання елемента в чергу; видалення з черги елемента з найбільшим пріоритетом; зміна пріоритету для довільного елемента, що знаходиться в черзі.

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

class Priority\_queue:

queue – список з елементами черги

Priority\_queue():

queue = []

end Priority\_queue

method add\_element(char element, int priority):

queue += [[element, priority]]

end method add\_element

method delete\_element():

maxpriority = +infinity

ind = +infinity

for i = 0 to queue.length do:

if maxpriority >= queue[i][1]:

maxpriority = queue[i][1]

ind = i

end if

end for

temp = queue.delete(ind)

end method delete\_element

method change(index, priority):

queue[index][1] = priority

end method change

method get\_queue():

return queue

end method get\_queue

method get\_length():

return queue.length

end method get\_length

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

class Priority\_queue:

def \_\_init\_\_(self):

self.\_\_queue = [] # Створюємо пусту чергу

def add\_element(self, element, priority):

self.\_\_queue += [[element, priority]] # Додаємо елемент до черги

def delete\_element(self):

maxpriority = float("inf") # Максимальний пріоритет

ind = float("inf") # Максимальний індекс

for i in range(len(self.\_\_queue)): # Проходимо по черзі

if maxpriority >= self.\_\_queue[i][1]: # Шукаємо елемент з найбільши пріоритетом

maxpriority = self.\_\_queue[i][1] # Запам'ятовуємо новий пріоритет

ind = i # Запам'ятовуємо індекс елемента з таким пріоритетом

temp = self.\_\_queue.pop(ind) # Видаляємо елемент з найбільшим пріоритетом

print(f"\nВидалено елемент \'{temp[0]}\' з пріоритетом {temp[1]}")

def change(self, index, priority): # Змінюємо пріоритет елемента за індексом

self.\_\_queue[index][1] = priority

print(f"\nНовий пріоритет елемента \'{self.\_\_queue[index][0]}\' = {self.\_\_queue[index][1]}")

def get\_queue(self): # Отримуємо чергу

return self.\_\_queue

def get\_length(self): # Отримуємо довжину черги

return len(self.\_\_queue)

def main():

queue = Priority\_queue() # Створюємо чергу

# Вводимо елементи черги

flag = True

while flag:

choice = int(input("Ви хочете додати елемент до черги? (1/0) - "))

if choice == 1:

symb = input(" Введіть елемент: ")

prior = int(input(" Введіть пріоритет елемента: "))

queue.add\_element(symb, prior)

else:

flag = False

print("\nЧерга:", queue.get\_queue())

# Видаляємо елемент з найбільшим пріоритетом

queue.delete\_element()

print("\nЧерга:", queue.get\_queue())

# Змінюємо пріоритет елемента за індексом

ind = int(input("\nВведіть індекс елемента, який хочете змінити: "))

if ind > queue.get\_length() or ind < 1:

print("Елемент з таким індексом відсутній!")

else:

newprior = int(input(" Введіть новий пріоритет: "))

queue.change(ind-1, newprior)

print("\nЧерга:", queue.get\_queue())

if \_\_name\_\_ == "\_\_main\_\_":

main()

3.2.2 Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

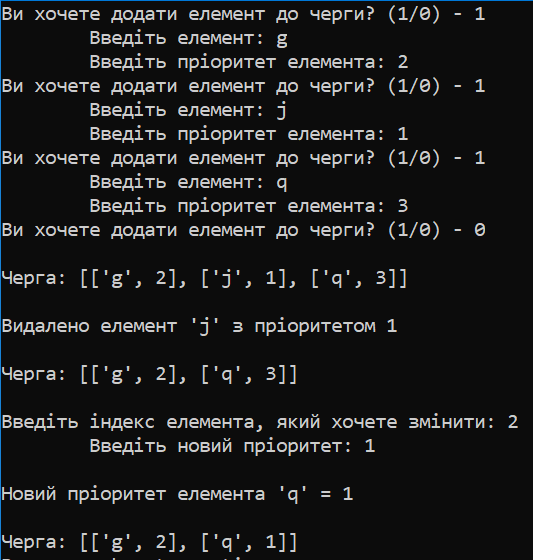


Рисунок 3.1

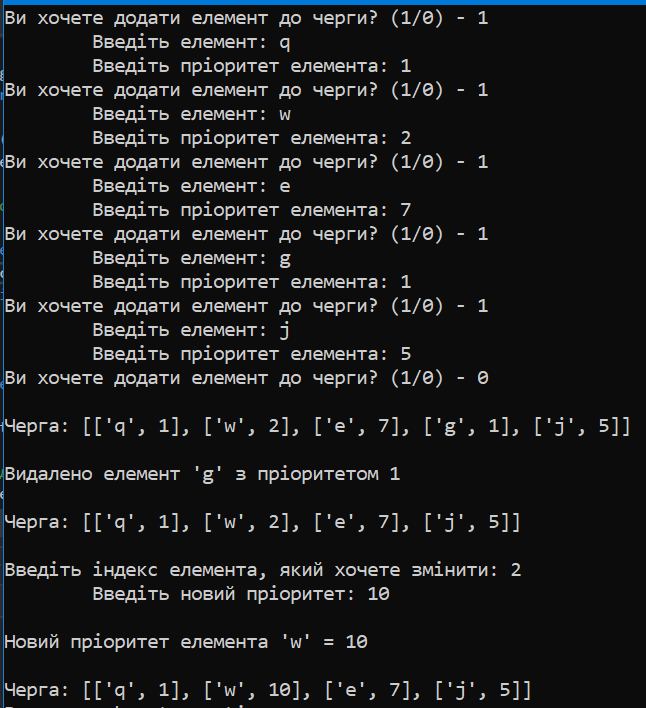


Рисунок 3.2

ВИСНОВОК

При виконанні даної лабораторної роботи я вивчив основні підходи формалізації евристичних алгоритмів і вирішення типових задач з їх допомогою; опанував структуру даних «черга з пріоритетом»; реалізував клас "черга з пріоритетом", яка підтримує наступні операції: додавання елемента в чергу; видалення з черги елемента з найбільшим пріоритетом; зміна пріоритету для довільного елемента, що знаходиться в черзі.