МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 8
З дисципліни: *"Алгоритми та структури даних"*
На тему: *"Лінійні структури даних"*

Лектор:
доц. каф. ПЗ
Коротєєва Т.О.

Виконав:
ст. гр. ПЗ – 22
Ясногородський Н.В.

Прийняв:
асист. каф. ПЗ
Франко А.В.

« ____ » _____ 2022 р.
∑= _____ _____.

Львів – 2022

Тема роботи: Лінійні структури даних.
Мета роботи: познайомитися з лінійними структурами даних (стек, черга, дек, список) та отримати навички програмування алгоритмів, що їх обробляють.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Стек, черга, дек, список відносяться до класу лінійних динамічних структур.

Зі стеку (stack) можна видалити тільки той елемент, який був у нього доданий останнім: стек працює за принципом «останнім прийшов – першим пішов» (last-in, first-out – LIFO).

З черги (queue), навпаки, можна видалити тільки той елемент, який знаходився в черзі довше за всіх: працює принцип «першим прийшов – першим пішов» (first-in, first-out – FIFO).

Дек - це впорядкована лінійна динамічно змінювана послідовність елементів, у якій виконуються такі умови: 1) новий елемент може приєднуватися з обох боків послідовності; 2) вибірка елементів можлива також з обох боків послідовності. Дек називають реверсивною чергою або чергою з двома боками.

У зв’язаному списку (або просто списку; linked list) елементи лінійно впорядковані, але порядок визначається не номерами, як у масиві, а вказівниками, що входять до складу елементів списку. Списки є зручним способом реалізації динамічних множин.

Елемент двобічно зв’язаного списку (doubly linked list) – це запис, що містить три поля: key (ключ) і два вказівники next (наступний) і prev (попередній). Крім цього, елементи списку можуть містити додаткові дані.

У кільцевому списку (circular list) поле prev голови списку вказує на хвіст списку, а поле next хвоста списку вказує на голову списку.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Розробити програму, яка читає з клавіатури послідовність даних, жодне з яких не повторюється, зберігає їх до структури даних (згідно з варіантом) та видає на екран такі характеристики:

- кількість елементів;
- мінімальний та максимальний елемент (для символів за кодом);
- третій елемент з початку послідовності та другий з кінця послідовності;
- елемент, що стоїть перед мінімальним елементом та елемент, що стоїть після максимального;
- знайти позицію елемента, значення якого задається з клавіатури;
- об'єднати дві структури в одну.

Всі характеристики потрібно визначити із заповненої структури даних.
Використовувати готові реалізації структур даних (наприклад, STL) заборонено.

Варіант 15, черга з пріоритетом (символи)

ВИКОНАННЯ РОБОТИ

Код програми:

```
class ListNode:
    def __init__(self, val, prio):
        self.val = val
        self.prio = prio
        self.next = None

    def __str__(self):
        return f"Node(value={self.val}, priority={self.prio})"



class PriorityQueue:
    def __init__(self):
        self.root = None
        self.length = 0
```

```
def pop_left(self):
    if not self.root:
        return None
    old_root = self.root
    self.root = self.root.next
    self.length -= 1
    return old_root

def push(self, val, priority):
    new_node = ListNode(val, priority)
    self.length += 1

    # edge case: root is None
    if not self.root:
        self.root = new_node
        return

    # edge case: swap with root
    if self.root.prio < priority:
        self.root, new_node.next = new_node, self.root
        return

    it = self.root
    while it and it.next and it.prio ⩾ priority:
        it = it.next
    it.next, new_node.next = new_node, it.next

def merge(self, pq2):
    self.length += len(pq2)
    pq1 = self.root
    pq2 = pq2.root
```

```
    dummy_new_root = ListNode(0, 0)
    it = dummy_new_root
    while pq1 and pq2:
        if pq1.prio > pq2.prio:
            it.next = pq1
            pq1 = pq1.next
        else:
            it.next = pq2
            pq2 = pq2.next
        it = it.next

    while pq1:
        it.next = pq1
        it = it.next
        pq1 = pq1.next
    while pq2:
        it.next = pq2
        it = it.next
        pq2 = pq2.next

    self.root = dummy_new_root.next
    return self.root

def __iter__(self):
    self.it = self.root
    return self

def __next__(self):
    curr = self.it
    if not curr:
        raise StopIteration()
```

```
        self.it = self.it.next
        return curr

    def __len__(self):
        return self.length


import pprint
import random
import string

from lab8.priority_queue_list import PriorityQueue


def gen_input_data(n):
    def get_random_char():
        return random.choice(string.ascii_letters)

    def get_random_prio():
        return random.choice(range(1, 101))

    return [(get_random_char(), get_random_prio()) for _ in range(n)]


def show_statistics(pq):
    min_el = max_el = pq.root
    pre_min_el = None
    second_from_end = third_el = None
    n = len(pq)

    for idx, el in enumerate(pq):
        if idx == 2:
            third_el = el
```

```
        if idx == n - 2:
            second_from_end = el

        if min_el.val > el.val:
            pre_min_el = min_el
            min_el = el

        if max_el.val < el.val:
            max_el = el

    print(
        f"""\
          Elements count: {len(pq)}
          Min element: {min_el}
          Max element: {max_el}
          3rd element from the start: {third_el}
          2nd element from the end: {second_from_end}
          Pre-min element: {pre_min_el}
          Post-max element: {max_el.next if max_el else None}
          """
    )


def populate_pq(n):
    pq = PriorityQueue()
    data = gen_input_data(n)
    for char, prio in data:
        pq.push(char, prio)
    print("Created Priority Queue (based on LinkedList)")
    pp = pprint.PrettyPrinter(width=60, compact=True)
    print(f"From data:\n {pp.pformat(data)}\n")
    return pq
```

```
def main():
    pq = populate_pq(100)
    pq2 = populate_pq(200)

    print("Statistics:")
    show_statistics(pq)

    pq.merge(pq2)
    print("Statistics After merge:")
    show_statistics(pq)


main()
```

ПРОТОКОЛ РОБОТИ

Спочатку виводиться стандартна інформація про задану чергу, що вимагається в пунктах до індивідуального завдання:

```
Statistics:
          Elements count: 100
          Min element: Node(value=A, priority=97)
          Max element: Node(value=z, priority=41)
          3rd element from the start: Node(value=g, priority=94)
          2nd element from the end: Node(value=B, priority=4)
          Pre-min element: Node(value=M, priority=98)
          Post-max element: Node(value=S, priority=35)


Statistics After merge:
          Elements count: 300
          Min element: Node(value=A, priority=97)
          Max element: Node(value=z, priority=76)
          3rd element from the start: Node(value=s, priority=92)
          2nd element from the end: Node(value=r, priority=3)
          Pre-min element: Node(value=M, priority=98)
          Post-max element: Node(value=u, priority=84)
```

ВИСНОВКИ

У цій лабораторній роботі я ознайомився з лінійною структурою даних “пріоритетна черга” та отримав навички програмування алгоритмів, що її обробляють