Міністерство освіти і науки України

Національний університет «Запорізька Політехніка»

Кафедра програмних засобів

ЗВІТ

з лабораторної роботи №1

з дисципліни «Алгоритми та Структури Даних» на тему:

«Лінійні структури даних»

Виконав:

Студент групи КНТ-122 О. А. Онищенко

Прийняли:

Старший викладач: Л. Ю. Дейнега

2023

Лінійні структури даних

Мета роботи

Вивчити основні концепції побудови лінійних структур даних: зв'язних списків, стеків, куп та черг з пріоритетами.

Навчитися обирати та реалізовувати структури даних для сортування, вставки, видалення та пошуку елементів.

Навчитися реалізовувати та застосовувати алгоритм пірамідального сортування на практиці.

Завдання до роботи

Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

Розробити програмне забезпечення, що виконує базові операції з лінійними структурами даних.

Розроблюваний програмний проєкт має складатися з окремих класів, що реалізують структури даних двозв'язний список та купа (черга з пріоритетами). На найвищий рівень може бути передбачено графічну інтерфейсну взаємодію з користувачем для роботи зі створеними класами.

Клас, що реалізує двозв'язний список, має дозволяти виконувати наступні операції на основі окремих методів: додавання вузла в початок списку, додавання вузла після заданого, пошук вузла в списку, видалення вузла, виведення вузлів на екран з початку та з кінця.

Клас, що реалізує купу (чергу з пріоритетами), має дозволяти виконувати наступні операції на основі окремих методів: вставлення елементу, сортування елементів, побудова купи з невпорядкованого масиву, видалення елементу, сортування елементів із використанням купи, виведення елементів на екран.

Виконати тестування розробленого програмного забезпечення.

Розробити окремий модуль програмного забезпечення для реалізації пірамідального сортування на основі розробленого класу.

Розв’язати індивідуальне завдання за допомогою розробленої реалізації пірамідального сортування. Вважати, що масиви даних зберігаються в файлах.

Варіант №20

У відділі кадрів міститься інформація про захворювання співробітників, що включає:

– прізвище, ім’я, по батькові співробітника;

– відділ;

– посаду;

– вік;

– дату початку лікарняного;

– дату завершення лікарняного;

– хвороба.

Вивести інформацію про всі хвороби, якими хворіли співробітники, за зменшенням кількості випадків.

Порівняти одержані результати виконаних тестів, провести аналіз вірності, коректності та адекватності роботи розробленого програмного забезпечення.

Виконати порівняння пірамідального сортування з іншими відомими студенту алгоритмами сортування.

Результати виконання роботи

```
Welcome! Make your choice below

1. See a demo of a doubly linked list usage
2. See the raw data from your file
3. Sort the data
4. See the number of occurrences for each disease
5. Exit
> 1

---

Displaying list from head:
Node1 -> Node2 -> Node3 -> None

Displaying list from tail:
Node3 -> Node2 -> Node1 -> None

After adding a node after Node2:
Node1 -> Node2 -> Node2.5 -> Node3 -> None

Searching for Node2.5:
True

After deleting Node2.5:
Node1 -> Node2 -> Node3 -> None

---

1. See a demo of a doubly linked list usage
2. See the raw data from your file
3. Sort the data
4. See the number of occurrences for each disease
5. Exit
> 2

---

- Adele Henr - 45 years old, had Covid-19. Works in Sales as a Executive
- Jennie Jackson - 30 years old, had Flu. Works in Tech as a Engineer
- Philip Hudson - 35 years old, had Flu. Works in HR as a Manager
- Frank Blac - 40 years old, had Covid-19. Works in Marketing as a Analyst
- Floyd Patrick - 25 years old, had Flu. Works in Admin as a Clerk
- John Smith - 50 years old, had Covid-19. Works in Sales as a Manager
- Jane Doe - 42 years old, had Flu. Works in Marketing as a Director
- Bob Johnson - 35 years old, had Food poisoning. Works in Engineering as a Lead Engineer
- Sarah Lee - 28 years old, had Flu. Works in Sales as a Sales Rep
- Mike Williams - 32 years old, had Bronchitis. Works in IT as a Support Tech
- Emily Davis - 29 years old, had Bronchitis. Works in Marketing as a Copywriter

---
```

```
---

1. See a demo of a doubly linked list usage
2. See the raw data from your file
3. Sort the data
4. See the number of occurrences for each disease
5. Exit
> 4

---

- Covid-19: 3 times
- Flu: 5 times
- Food poisoning: 1 times
- Bronchitis: 2 times

---

1. See a demo of a doubly linked list usage
2. See the raw data from your file
3. Sort the data
4. See the number of occurrences for each disease
5. Exit
> 3

---

Data successfully sorted

---

1. See a demo of a doubly linked list usage
2. See the raw data from your file
3. Sort the data
4. See the number of occurrences for each disease
5. Exit
> 4

---

- Flu: 5 times
- Covid-19: 3 times
- Bronchitis: 2 times
- Food poisoning: 1 times

---

1. See a demo of a doubly linked list usage
2. See the raw data from your file
3. Sort the data
4. See the number of occurrences for each disease
5. Exit
> 5

---
```

Код

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None
        self.prev = None


class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def prepend(self, data):
        # check if the list is empty
        if self.head is None:
            # then the new node is head
            self.head = Node(data)
        else:
            # declare new node
            newNode = Node(data)
            # set current head's previous pointer to the new node
            self.head.prev = newNode
            # set new node's next pointer to current head
            newNode.next = self.head
            # set head to be the new node, since we're adding to the
front
            self.head = newNode

    def append(self, data):
        # if the list is empty
        if self.head is None:
            # set new node to be a head
            self.head = Node(data)
        else:
            # set a temporary node to be a head
```

```
                currentNode = self.head
                # loop through all the nodes and reach the end of a list
                while currentNode.next is not None:
                    currentNode = currentNode.next
                # declare a new node
                newNode = Node(data)
                # set last node's next pointer to new node
                currentNode.next = newNode
                # set new node's previous pointer to last node
                newNode.prev = currentNode

        def appendAfterNode(self, nodeData, data):
            # set temporary node to be head
            currentNode = self.head
            # loop through all the nodes
            while currentNode is not None:
                # check if we reached the wanted node by comparing the
data with input
                if currentNode.data == nodeData:
                    # declare new node
                    newNode = Node(data)
                    # set new node's previous pointer to the found node
                    newNode.prev = currentNode
                    # set new node's next pointer to the node after
found one
                    newNode.next = currentNode.next
                    # check if the node after current exists
                    if currentNode.next is not None:
                        # if it does, set its previous pointer to our
new node
                        currentNode.next.prev = newNode
                    # set found node's next pointer to the new node and
break out
                    currentNode.next = newNode
                    return
                # keep iterating until found
                currentNode = currentNode.next
            # if our function continues running after the loop, it means
we didn't return once the node was found, meaning the node was not found
```

```
            print(f"Node with data of {str(nodeData)} was not found.")

        def search(self, data) -> bool:
            # set temporary node to head
            currentNode = self.head
            # iterate through every node in the list
            while currentNode is not None:
                # check if node's data corresponds to the one we're
looking for
                if currentNode.data == data:
                    # then return true
                    return True
                # keep iterating until found
                currentNode = currentNode.next
            # if not found return false
            return False

        def delete(self, data) -> bool:
            # set temporary node to head
            currentNode = self.head
            # iterate over all the nodes in a list
            while currentNode is not None:
                # check if the node's data corresponds to the one we
wanna delete
                if currentNode.data == data:
                    # check if there is a previous node
                    if currentNode.prev is not None:
                        # then set previous node's next pointer to the
node after our current
                        currentNode.prev.next = currentNode.next
                    # check if there is a next node
                    if currentNode.next is not None:
                        # set next node's previous pointer to our
previous node
                        currentNode.next.prev = currentNode.prev
                    # check if our node is the first node
                    if currentNode == self.head:
                        # set head to the next element
                        self.head = currentNode.next
```

```
                    # return true indicating the successful deletion
                    return True
                # keep iterating until found the one we're looking for
                currentNode = currentNode.next
            # if we exit out of a loop, it means we haven't found a node we wanted to delete, so we return false, indicating the node was not found and deleted
            return False

        def displayFromHead(self):
            # set temporary node to head
            currentNode = self.head
            # iterate through all the nodes
            while currentNode is not None:
                # print node's data and move onto the next one via the next pointer
                print(currentNode.data, end=" → ")
                currentNode = currentNode.next
            print("None")

        def displayFromTail(self):
            # if the list is empty
            if self.head is None:
                # then print none and return
                print("None")
                return
            # set temporary node to head
            currentNode = self.head
            # iterate to the end of a list
            while currentNode.next is not None:
                currentNode = currentNode.next
            # iterate from the end of the list
            while currentNode is not None:
                # print current node and move onto the node before it by moving via a previous pointer
                print(currentNode.data, end=" → ")
                currentNode = currentNode.prev
            print("None")
```

```
    class Heap:
        def __init__(self):
            self.heap = []

        def insert(self, value):
            # append element to our array
            self.heap.append(value)
            # heapify it in order to set in the correct place
            self._heapifyUp(len(self.heap) - 1)

        def sort(self):
            sortedItems = []
            size = len(self.heap)
            # for each element in the heap
            for _ in range(size):
                # append the largest element to the temporary array
                sortedItems.append(self.delete())
            return sortedItems

        def buildHeap(self, arr):
            # set our heap to be the given array
            self.heap = arr
            # start in the middle of it cause the other half are leaves
- nodes without children
            start = len(arr) // 2
            # for each element from the start to the first element in
reversed order
            for i in reversed(range(start + 1)):
                # heapify it down to set in the correct place
                self._heapifyDown(i)
            return self

        def delete(self):
            # if the heap is empty, return null
            if len(self.heap) == 0:
                return None
            # swap root with the last element
            self._swap(0, len(self.heap) - 1)
```

```
                # assign root to temporary variable
                root = self.heap.pop()
                # heapify the new root, which previously was the last
element, to set it in a correct position
                self._heapifyDown(0)
                # return the original root
                return root

            def display(self):
                for item in self.heap:
                    print(item, end=" ")
                print()

            def _heapifyUp(self, index):
                parentIndex = (index - 1) // 2
                # checking if parent element exists and if our current
element is larger than its parent
                if parentIndex >= 0 and self.heap[index] >
self.heap[parentIndex]:
                    # then swapping those two and continuing the process for
parent
                    self._swap(parentIndex, index)
                    self._heapifyUp(parentIndex)

            def _heapifyDown(self, index):
                leftChildIndex = 2 * index + 1
                rightChildIndex = 2 * index + 2
                largest = index

                # checking if left child exists and its more than our
largest element, which is currently our current element
                if (
                    leftChildIndex < len(self.heap)
                    and self.heap[leftChildIndex] > self.heap[largest]
                ):
                    # then reassigning largest element to left child
                    largest = leftChildIndex
```

```
            # checking if right child exists and if its larger than the
current largest element
            if (
                rightChildIndex < len(self.heap)
                and self.heap[rightChildIndex] > self.heap[largest]
            ):
                # then reassigning largest element to the right child
                largest = rightChildIndex

            # checking if our current element is not the largest one
            if largest != index:
                # if so, swapping them two
                self._swap(index, largest)
                # continuing the process for the current largest element
                self._heapifyDown(largest)

        def _swap(self, i, j):
            self.heap[i], self.heap[j] = self.heap[j], self.heap[i]


    def heapSort(arr):
        return Heap().buildHeap(arr).sort()


    from json import load


    def getEmployeesData():
        # opening file for read and reading JSON data from a file into
the variable
        with open("employees.json", "r") as inputFile:
            data = load(inputFile)
        return data


    def countDiseaseCases(employeeData):
        diseasesCount = dict()
        for employee in employeeData["employees"]:
            # if employee's disease doesn't yet exist in the dictionary
```

```
            if employee["disease"] not in diseasesCount:
                # add it with a count of 0
                diseasesCount[employee["disease"]] = 0
            # else just increment the count
            diseasesCount[employee["disease"]] += 1
        return diseasesCount


    def demoDoublyLinkedList():
        linkedListDemo = DoublyLinkedList()

        linkedListDemo.append("Node1")
        linkedListDemo.append("Node2")
        linkedListDemo.append("Node3")

        print("Displaying list from head:")
        linkedListDemo.displayFromHead()

        print("\nDisplaying list from tail:")
        linkedListDemo.displayFromTail()

        linkedListDemo.appendAfterNode("Node2", "Node2.5")
        print("\nAfter adding a node after Node2:")
        linkedListDemo.displayFromHead()

        print("\nSearching for Node2.5:")
        print(linkedListDemo.search("Node2.5"))

        linkedListDemo.delete("Node2.5")
        print("\nAfter deleting Node2.5:")
        linkedListDemo.displayFromHead()


    def main():
        # getting raw employee data from a JSON file
        employeeData = getEmployeesData()
        # counting number of occurrences of each disease
        diseasesCount = countDiseaseCases(employeeData)
        # converting disease occurrences to a list of tuples
```

```
            diseasesCountList = [
                (occurences, name) for name, occurences in
diseasesCount.items()
            ]

            print("Welcome! Make your choice below\n")

            while True:
                print("1. See a demo of a doubly linked list usage")
                print("2. See the raw data from your file")
                print("3. Sort the data")
                print("4. See the number of occurrences for each disease")
                print("5. Exit")
                choice = input("> ")

                print("\n---\n")
                if choice == "1":
                    demoDoublyLinkedList()
                elif choice == "2":
                    for employee in employeeData["employees"]:
                        print(
                            f"- {employee['name']} - {employee['age']} years
old, had {employee['disease']}. Works in {employee['department']} as a
{employee['position']}"
                        )
                elif choice == "3":
                    diseasesCountList = heapSort(diseasesCountList)
                    print("Data successfully sorted")
                elif choice == "4":
                    for occurences, name in diseasesCountList:
                        print(f"- {name}: {occurences} times")
                else:
                    break
                print("\n---\n")


        if __name__ == "__main__":
            print()
            main()
```

```
        print()
```

```
{
  "employees": [
    {
      "name": "Adele Henr",
      "department": "Sales",
      "position": "Executive",
      "age": 45,
      "start_sick_leave": "2021-02-15",
      "end_sick_leave": "2021-02-25",
      "disease": "Covid-19"
    },
    {
      "name": "Jennie Jackson",
      "department": "Tech",
      "position": "Engineer",
      "age": 30,
      "start_sick_leave": "2021-01-10",
      "end_sick_leave": "2021-01-20",
      "disease": "Flu"
    },
    {
      "name": "Philip Hudson",
      "department": "HR",
      "position": "Manager",
      "age": 35,
      "start_sick_leave": "2021-04-15",
      "end_sick_leave": "2021-04-25",
      "disease": "Flu"
    },
    {
      "name": "Frank Blac",
      "department": "Marketing",
      "position": "Analyst",
      "age": 40,
      "start_sick_leave": "2021-03-10",
```

```
    "end_sick_leave": "2021-03-15",
    "disease": "Covid-19"
  },
  {
    "name": "Floyd Patrick",
    "department": "Admin",
    "position": "Clerk",
    "age": 25,
    "start_sick_leave": "2021-07-01",
    "end_sick_leave": "2021-07-10",
    "disease": "Flu"
  },
  {
    "name": "John Smith",
    "department": "Sales",
    "position": "Manager",
    "age": 50,
    "start_sick_leave": "2021-05-01",
    "end_sick_leave": "2021-05-10",
    "disease": "Covid-19"
  },
  {
    "name": "Jane Doe",
    "department": "Marketing",
    "position": "Director",
    "age": 42,
    "start_sick_leave": "2021-06-15",
    "end_sick_leave": "2021-06-25",
    "disease": "Flu"
  },
  {
    "name": "Bob Johnson",
    "department": "Engineering",
    "position": "Lead Engineer",
    "age": 35,
    "start_sick_leave": "2021-08-01",
    "end_sick_leave": "2021-08-07",
    "disease": "Food poisoning"
  },
```

```
    {
      "name": "Sarah Lee",
      "department": "Sales",
      "position": "Sales Rep",
      "age": 28,
      "start_sick_leave": "2021-09-10",
      "end_sick_leave": "2021-09-17",
      "disease": "Flu"
    },
    {
      "name": "Mike Williams",
      "department": "IT",
      "position": "Support Tech",
      "age": 32,
      "start_sick_leave": "2021-11-15",
      "end_sick_leave": "2021-11-22",
      "disease": "Bronchitis"
    },
    {
      "name": "Emily Davis",
      "department": "Marketing",
      "position": "Copywriter",
      "age": 29,
      "start_sick_leave": "2021-12-01",
      "end_sick_leave": "2021-12-07",
      "disease": "Bronchitis"
    }
  ]
}
```


Висновки

Таким чином, ми вивчили основні концепції побудови лінійних структур даних: зв’язних списків, стеків, куп та черг з пріоритетами; навчилися обирати та реалізовувати структури даних для сортування,

вставки, видалення та пошуку елементів; навчилися реалізовувати та застосовувати алгоритм пірамідального сортування на практиці.

Контрольні питання

Який принцип роботи стеку?

Стек - це структура даних, яка базується на принципі "першим прийшов, останнім пішов" (Last-In-First-Out, LIFO). Це означає, що елементи додаються і видаляються зі стеку з одного боку, який зазвичай називають вершиною стеку. Коли ми додаємо новий елемент до стеку, він стає активним верхнім елементом і є доступним для подальшого використання. Коли ми видаляємо елемент зі стеку, видаляється саме цей верхній елемент.

Що таке дек?

Дек (або двостороння черга) - це структура даних, яка подібна до стеку та черги одночасно. У дека можна додавати та видаляти елементи як з початку, так і з кінця. Це дозволяє виконувати операції вставки та вилучення на обох кінцях дека. Дек може бути корисним для різних завдань, де потрібно працювати з елементами з обох боків.

Які операції можна виконувати зі стеком?

Основні операції, які можна виконувати зі стеком, включають додавання елементу (push) і видалення елементу (pop). Також можна виконувати операцію перевірки верхнього елемента (peek), яка дозволяє переглядати верхній елемент без його видалення.