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»**

варіант №2

Виконав:

ст. гр. КНТ-113сп Артем Бедських

Прийняв:

Старший викладач Лариса ДЕЙНЕГА

2023

# Мета роботи:

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

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

1.1.3 Навчитися реалізовувати та застосовувати алгоритм  пірамідального сортування на практиці.

# Завдання до лабораторної роботи:

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

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

1.3.2.1 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.

1.3.2.2 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.

1.3.2.3 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.

1.3.3 Виконати тестування розробленого програмного  забезпечення.

1.3.4 Розробити окремий модуль програмного забезпечення для  реалізації пірамідального сортування на основі розробленого класу. 1.3.5 Розв’язати індивідуальне завдання за допомогою  розробленої реалізації пірамідального сортування.

Варіант № 2

Фінансова діяльність підприємства представлена записами, що  містять:

– назву підприємства;

– адресу;

– місяць;

– рік;

– прибуток за даний місяць.

Визначити перелік підприємств, які за вказаний рік входять у  число 25 % найприбутковіших за даний рік. У випадку  неоднозначності віддавати пріоритет підприємству, яке має більший  максимальний прибуток протягом всіх місяців даного року.

# Текст розробленого програмного забезпечення з коментарями:

**Doubly-linked-list.js**

class Node {

  constructor(data, prev = null, next = null) {

    this.data = data;

    this.prev = prev;

    this.next = next;

  }

  toString() {

    return `{prev: ${this.prev?.data}, data: ${this.data}, next: ${this.next?.data}}`;

  }

}

class DoublyLinkedList {

  constructor(data) {

    this.head = new Node(data);

  }

  addToStart(data) {

    const newNode = new Node(data, null, this.head);

    this.head.prev = newNode;

    this.head = newNode;

  }

  insert(nodeData, data) {

    const newNode = new Node(data);

    let currentNode = this.head;

    while (currentNode) {

      if (currentNode.data === nodeData) {

        newNode.prev = currentNode;

        newNode.next = currentNode.next;

        if (currentNode.next) {

          currentNode.next.prev = newNode;

        }

        currentNode.next = newNode;

        return;

      }

      currentNode = currentNode.next;

    }

  }

  search(data) {

    let currentNode = this.head;

    while (currentNode) {

      if (currentNode.data === data) {

        return currentNode;

      }

      currentNode = currentNode.next;

    }

    return null;

  }

  remove(data) {

    let currentNode = this.head;

    while (currentNode) {

      if (currentNode.data === data) {

        if (currentNode.prev === null) {

          this.head = currentNode.next;

          this.head.previous = null;

          return;

        }

        currentNode.prev.next = currentNode.next;

        if (currentNode.next) {

          currentNode.next.prev = currentNode.prev;

        }

        return;

      }

      currentNode = currentNode.next;

    }

  }

  getFromStart() {

    let currentNode = this.head;

    const data = [];

    while (currentNode) {

      data.push(currentNode.data);

      currentNode = currentNode.next;

    }

    return data;

  }

  getFromEnd() {

    return this.getFromStart().toReverse();

  }

}

export { DoublyLinkedList };

**financialRecord.js**

/\*

  Фінансова діяльність підприємства представлена записами, що

  містять:

  – назву підприємства;

  – адресу;

  – місяць;

  – рік;

  – прибуток за даний місяць

\*/

class FinancialRecord {

  constructor(name, address, month, year, profit) {

    this.name = name;

    this.address = address;

    this.month = month;

    this.year = year;

    this.profit = profit;

  }

  valueOf() {

    return this.profit;

  }

  clone() {

    return new FinancialRecord(

      this.name,

      this.address,

      this.month,

      this.year,

      this.profit,

    );

  }

  toString() {

    return `${this.profit} - ${this.name}`;

  }

}

export { FinancialRecord };

**heap.js**

import { FinancialRecord } from "./FinancialRecord.js";

class HeapSort {

  constructor(heap) {

    this.heap = heap;

  }

  getSorted() {

    const heapCopy = this.heap.\_heap.map(item => {

      if (item instanceof FinancialRecord) {

        return item.clone();

      }

      return { ...item };

    });

    const result = [];

    while (heapCopy.length >= 1) {

      result.push(this.heap.extractTop(heapCopy));

    }

    return result;

  }

}

class Heap {

  \_heap = [];

  add(element) {

    const index = this.\_heap.push(element);

    this.siftup(index);

    return this;

  }

  fromArray(array) {

    this.\_heap = array;

    this.\_heap.forEach((\_, index) => {

      this.siftdown(this.\_heap, this.\_heap.length - 1 - index);

    });

    return this;

  }

  extractTop(heap = this.\_heap) {

    const last = heap.length - 1;

    [heap[0], heap[last]] = [heap[last], heap[0]];

    const element = heap.pop();

    this.siftdown(heap, 0);

    return element;

  }

  getPrint() {

    console.log(

      "Heap:",

      this.\_heap.map(item => item.toString()),

    );

  }

  siftup(i) {

    let parent = Math.floor(i - 1 / 2);

    while (i !== 0 && Number(this.\_heap[i]) > Number(this.\_heap[parent])) {

      [this.\_heap[i], this.\_heap[parent]] = [this.\_heap[parent], this.\_heap[i]];

      i = parent;

      parent = Math.floor(i - 1 / 2);

    }

  }

  siftdown(heap, i) {

    let left = i \* 2 + 2;

    let right = i \* 2 + 1;

    while (

      (left < heap.length && Number(heap[i]) < Number(heap[left])) ||

      (right < heap.length && Number(heap[i]) < Number(heap[right]))

    ) {

      let smallest = right;

      if (right >= heap.length || Number(heap[left]) > Number(heap[right])) {

        smallest = left;

      }

      [heap[i], heap[smallest]] = [heap[smallest], heap[i]];

      i = smallest;

      left = i \* 2 + 2;

      right = i \* 2 + 1;

    }

  }

}

export {Heap, HeapSort};

**main.js**

import { DoublyLinkedList } from "./doubly-linked-list.js";

import { Heap, HeapSort } from "./heap.js";

import { FinancialRecord } from "./FinancialRecord.js";

/\*

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.

\*/

const list = new DoublyLinkedList(1);

list.insert(1, 2);

list.insert(2, 6);

list.insert(1, 7);

list.insert(1, 9);

console.log("DoublyLinkedList: ");

console.log(list.getFromStart());

console.log("Search data = 7: ", String(list.search(7)));

console.log("Remove 9");

list.remove(9);

console.log(list.getFromStart());

console.log("");

/\*

    Фінансова діяльність підприємства представлена записами, що

    містять:

    – назву підприємства;

    – адресу;

    – місяць;

    – рік;

    – прибуток за даний місяць

\*/

// prettier-ignore

const records = [

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 1, 2023, 98929),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 2, 2023, 54537),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 3, 2023, 2737),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 4, 2023, 28807),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 5, 2023, 87794),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 6, 2023, 52163),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 7, 2023, 71813),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 8, 2023, 13101),

    new FinancialRecord('Cartwright LLC', 'Port Zenabury', 9, 2023, 22664),

    new FinancialRecord('Erdman - Waters', 'Port Omari', 1, 2022, 65817),

    new FinancialRecord('Erdman - Waters', 'Port Omari', 2, 2023, 5472),

    new FinancialRecord('Erdman - Waters', 'Port Omari', 3, 2023, 75018),

    new FinancialRecord('Erdman - Waters', 'Port Omari', 4, 2022, 62397),

    new FinancialRecord('Erdman - Waters', 'Port Omari', 5, 2022, 58051),

    new FinancialRecord('Schaefer - Kautzer', 'Raushire', 6, 2022, 5726),

    new FinancialRecord('Zemlak, Volkman and Morissette', 'Surprise', 1, 2023, 68899),

    new FinancialRecord('Zemlak, Volkman and Morissette', 'Surprise', 2, 2023, 83181),

    new FinancialRecord('Zemlak, Volkman and Morissette', 'Surprise', 3, 2022, 61105),

    new FinancialRecord('Zemlak, Volkman and Morissette', 'Surprise', 4, 2023, 34930),

    new FinancialRecord('Hirthe - Grimes', 'Killeen', 1, 2023, 15143),

    new FinancialRecord('Cole - Bins', 'East Rowan', 1, 2023, 97187),

    new FinancialRecord('Gleichner - Dibbert', 'South Alvah', 8, 2023, 33077),

    new FinancialRecord('Waelchi Group', 'Cheektowaga', 1, 2023, 16515),

    new FinancialRecord('Waelchi Group', 'Cheektowaga', 2, 2023, 21019),

    new FinancialRecord('Waelchi Group', 'Cheektowaga', 3, 2023, 91653),

    new FinancialRecord('Waelchi Group', 'Cheektowaga', 4, 2023, 78815),

    new FinancialRecord('Waelchi Group', 'Cheektowaga', 5, 2023, 37380),

    new FinancialRecord('Waelchi Group', 'Cheektowaga', 6, 2023, 61544),

    new FinancialRecord('Miller, White and Stark', 'New Ryan', 1, 2023, 75505),

    new FinancialRecord('Miller, White and Stark', 'New Ryan', 2, 2023, 74164),

    new FinancialRecord('Miller, White and Stark', 'New Ryan', 3, 2023, 21339),

    new FinancialRecord('Miller, White and Stark', 'New Ryan', 4, 2023, 91388),

    new FinancialRecord('Miller, White and Stark', 'New Ryan', 5, 2023, 53429),

    new FinancialRecord('Miller, White and Stark', 'New Ryan', 6, 2023, 15504),

    new FinancialRecord('Miller, White and Stark', 'New Ryan', 7, 2022, 94464)

]

console.log("Індивідуальне завдання варіант 2:");

/\*

Визначити перелік підприємств, які за вказаний рік входять у

число 25 % найприбутковіших за даний рік. У випадку

неоднозначності віддавати пріоритет підприємству, яке має більший

максимальний прибуток протягом всіх місяців даного року.

\*/

const heap = new Heap().fromArray(records);

heap.getPrint();

const sortedCompanies = new HeapSort(heap).getSorted();

const currentYear = sortedCompanies.filter(item => item.year === 2023);

const byYear = currentYear.reduce((acc, item) => {

  return {

    ...acc,

    [item.name]: (acc[item.name] || 0) + item.profit,

  };

}, {});

const entries = Object.entries(byYear).sort((a, b) => b[1] - a[1]);

const items = entries.slice(0, Math.ceil(entries.length \* 0.25));

console.log();

console.log("Сумми компаній за 2023 рік:", byYear);

console.log();

console.log(

  "Топ 25% компаній за 2023 рік:\n",

  items.map(item => {

    return `${item[1]} - ${item[0]}`;

  }),

);

# Результати роботи програмного забезпечення:

На рисунку 4.1 показано виконання програми:

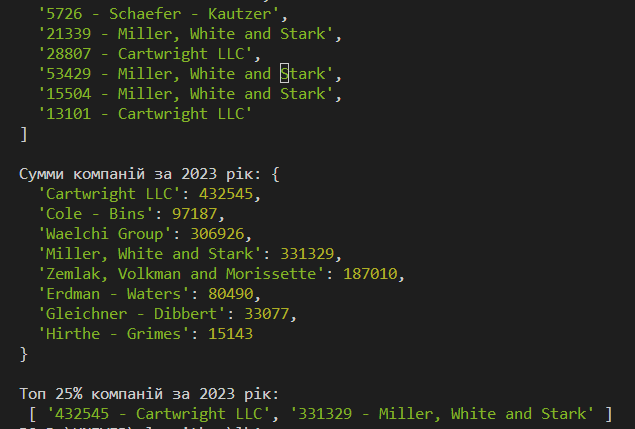
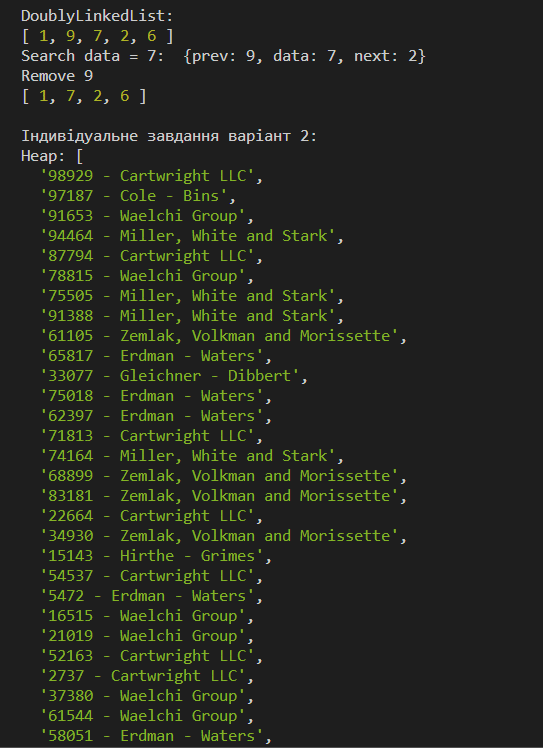


Рисунок 4.1 – Виконання програми

# Висновки:

Під час виконання лабораторної роботи, я вивчив основні концепції побудови лінійних структур даних, таких як зв’язні списки, стеки, купи та черги з пріоритетами. Я розумію, що кожна з цих структур має свої унікальні особливості та використання.

Далі, я навчився обирати та реалізовувати відповідні структури даних для виконання операцій сортування, вставки, видалення та пошуку елементів. Це дозволяє мені ефективно працювати з даними, враховуючи їхню природу та вимоги завдання.

Окрему увагу я приділив алгоритму пірамідального сортування. Я навчився його реалізовувати та застосовувати на практиці. Розумію, що цей алгоритм є ефективним для сортування даних у вигляді купи.

У цілому, завдяки вивченню цих концепцій та алгоритмів, я здатен ефективно працювати з лінійними структурами даних та використовувати їх для вирішення практичних завдань.

**Контрольні запитання:**

**Які операції можна виконувати зі стеком?**

З стеком можна виконувати три основних операції:

* push - додавання елемента в стек. Новий елемент поміщається в верхівку стека.
* pop - видалення елемента зі стека. Елемент, який був останнім в стеку, видаляється з нього.
* peek - читання елемента зі стека без його видалення. Елемент, який був останнім в стеку, читається, але не видаляється.

**Що таке дек?**

Дек (англ. deque - double-ended queue) - це структура даних, яка схожа на стек, але елементи можуть бути додані або видалені з обох кінців. Дек також називається двосторонньою чергою.

**Який принцип роботи стеку?**

Стек - це структура даних, яка працює за принципом LIFO (last in, first out). Це означає, що елемент, який був останнім доданий в стек, буде першим видалений з нього.

Стек можна уявити як стопку тарілок. Щоб додати тарілку в стопку, ми кладемо її зверху. Щоб видалити тарілку зі стопки, ми знімаємо її зверху.

Стек можна використовувати для реалізації різних алгоритмів, таких як:

* Рекурсія
* Обробка винятків
* Пісування виразів
* Обробка історії

Приклади використання стеку

* Рекурсія. Рекурсія - це алгоритм, який викликає сам себе. Для того, щоб рекурсія працювала коректно, вона повинна мати умову виходу. Стек можна використовувати для зберігання поточної інформації про рекурсивне заклик, щоб не втратити її при виході з рекурсії.
* Обробка винятків. Виключення - це подія, яка може порушити виконання програми. Стек можна використовувати для зберігання інформації про виклик функцій, щоб у разі виникнення виключення можна було визначити, де воно виникло.
* Пісування виразів. Пісування виразів - це процес перекладу виразу з одного представлення в інше. Стек можна використовувати для зберігання операторів і операндів виразу, щоб потім їх можна було обробити в правильному порядку.
* Обробка історії. Історія - це список дій, які користувач виконав в програмі. Стек можна використовувати для зберігання інформації про недавні дії користувача, щоб користувач міг їх відтворити.