Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

кафедра програмних засобів

Самостійна робота

з дисципліни «Алгоритми та структури даних»

Виконав:
ст.групи КНТ-113сп Іван ЩЕДРОВСЬКИЙ

Прийняв:
ст.викладач Валерій ЛЬОВКІН

2023

ЗМІСТ

1 ВСТУП

Вступна частина роботи має на меті забезпечити комплексне розуміння контексту, в якому вирішується важлива проблема. Сучасний стан досліджуваної проблеми визначається складністю вирішення завдань, частиною яких вже вдалося розв'язати. Знаходження практичних рішень в даній галузі здійснюється вже провідними фірмами і вченими. Однак існують прогалини в знаннях, що потребують додаткового дослідження.

Визначення світових тенденцій розв'язання проблеми стає ключовим етапом у вступі, що сприяє збагаченню та обґрунтуванню методів та підходів. Актуальність роботи обумовлена необхідністю вирішення конкретних завдань та важливістю її внеску у розвиток галузі. Зазначається мета роботи, визначається галузь застосування, а також встановлюється взаємозв'язок із схожими дослідженнями. Робота має особливий внесок в контексті існуючих досліджень і є необхідною для подальшого розвитку обраної галузі.

Таким чином, вступна частина роботи є необхідним елементом для формування повноцінного уявлення про стан галузі та важливість вирішення досліджуваної проблеми.

2 СТРУКТУРИ ДАНИХ

2.1 Алгоритм пірамідального сортування

Алгоритм пірамідального сортування є ефективним методом сортування, який базується на використанні структури даних "куча". Основна ідея полягає у побудові бінарної кучі з непосортованого масиву та подальшому впорядкуванні елементів шляхом видалення максимального (або мінімального) елементу із кучі. Цей процес повторюється до повного впорядкування всього масиву.

Індивідуальне завдання. Книжки в бібліотеці характеризуються наступними даними:

- автор;
- назва;
- жанр;
- видавництво;
- рік публікації;
- кількість сторінок;
- загальна кількість екземплярів;
- кількість екземплярів у читачів.

Визначити книжки, кількість наявних екземплярів яких у бібліотеці в поточний момент входить у перші 50 %. Обчислити сумарну кількість наявних екземплярів таких книжок

Гешування та B-дерева представляють собою важливі структури даних і супроводжуються відповідними алгоритмами для ефективного управління та пошуку даних.

Геш-таблиці використовують хеш-функції для визначення місця збереження даних. Основна ідея полягає у використанні хеш-функції для перетворення ключів у індекси, за якими можна швидко здійснювати доступ до

даних. Індивідуальне завдання може включати створення геш-таблиці для швидкого пошуку книг в бібліотеці за автором чи назвою.

B-дерева - це балансовані дерева, які забезпечують ефективний пошук та вставку даних. Вони особливо корисні для управління великими об'ємами даних. Індивідуальне завдання може включати створення B-дерева для швидкого доступу до книг в бібліотеці за роком публікації.

Програмне забезпечення для реалізації геш-таблиць і B-дерев повинне включати функції додавання, видалення та пошуку даних, а також можливість відображення результатів в екранних формах зручного інтерфейсу користувача.

Результати розв'язання завдання можуть включати швидкість виконання операцій вставки, видалення та пошуку для обох структур даних. Характеристики роботи алгоритмів можуть включати часові та пам'ятеві витрати.

Порівняння структур даних може бути здійснене на основі їхньої ефективності в розв'язанні конкретного завдання, враховуючи особливості і вимоги завдання.

Індивідуальне завдання. Створити геш-таблицю, що використовує метод ланцюжків для розв’язання колізій та геш-функцію множення. Геш-таблицю заповнити на основі виділення інформації з текстового файлу, в якому містяться прізвища, ім’я і по батькові співробітників фірми та займані ними посади. Визначити посаду заданого співробітника.

Індивідуальне завдання. Дані про власників автомобілів включають ідентифікаційний номер транспортного засобу, дату реєстрації та власника (прізвище, ім’я, по батькові). Сформувати дерево з інформації про власників автомобілів. Реалізувати пошук інформації про автомобіль за заданим ідентифікаційним номером транспортного засобу, визначення осіб, які володіють більше ніж одним автомобілем.

2.2 Удосконалені методи розроблення та аналізу

2.2.1 Жадібний алгоритм та Алгоритм Хаффмана:

Жадібні алгоритми вирішують проблеми шляхом прийняття локально оптимальних рішень на кожному кроці з надією на досягнення глобально оптимального рішення. Алгоритм Хаффмана є прикладом жадібного алгоритму, який використовується для стискання даних. Основна його ідея - призначити коротші коди більш часто вживаним символам, що дозволяє скоротити загальну довжину коду.

Індивідуальне завдання. На дачі стоїть велика діжка, яка вміщує задану кількість рідини. Хазяїн використовує її для поливу рослин, але не маючи централізованого водопостачання, має принести воду з річки. У його розпорядженні є відра заданого обсягу. Визначити, яку мінімальну кількість відер води хазяїну потрібно принести з річки, щоб заповнити діжку. Вважати, що кожне відро може бути принесене тільки повністю заповненим, адже хазяїн не хоче носити зайвий вантаж. Результати виводити, демонструючи кількість відер кожного обсягу

Результати розв'язання завдання можуть бути представлені у вигляді ефективності стиснення та швидкості обробки даних. Характеристики роботи алгоритму можуть включати час виконання та розмір стиснутого файлу.

2.2.2 Динамічне програмування:

Динамічне програмування є методом розв'язання складних задач шляхом розбиття їх на менші підзадачі та ефективного вирішення кожної з них, а потім складання результатів. Це застосовується для оптимізації завдань, які можна розбити на незалежні підзадачі.

Індивідуальне завдання 1. Послідовність складається з деякого набору цілих чисел. Елементи послідовності можуть бути, як від’ємними, так і невід’ємними цілими числами. Визначити найдовшу спадну підпослідовність

даної послідовності. Вивести на екран довжину такої послідовності та всі її члени.

Індивідуальне завдання 2. На одній з вулиць містечка будинки класифіковано за трьома типами: перший – звичайні житлові споруди, другий – промислові споруди, а третій – міські заклади (лікарні, школи тощо). У результаті вулиця схематично зображена набором літер, кожна з яких визначає тип будинку. У процесі збору інформації про місто була створена матриця – таблиця, в якій кожен стовпчик і рядок відповідають одному з типів будівель. Відповідно клітинка такої таблиці визначає, чи розташовані на даній вулиці міста поруч будівлі заданого типу. Матриця симетрична. Визначити, скільки існує способів взаємного розташування будинків даних типів між собою за заданою матрицею для заданої кількості будинків на вулиці, тобто кількість можливих наборів літер заданої довжини, що відповідають заданій матриці.

2.3 Алгоритми роботи з графами

Алгоритми обходу графів, такі як DFS (Depth-First Search) та BFS (Breadth-First Search), використовуються для вивчення графів та виявлення їхньої структури. DFS використовує глибинний підхід, досліджуючи граф по глибині, тоді як BFS використовує широкий підхід, рухаючись по рівнях графа. На рисунку 2.1 показаний приклад DFS. На рисунку 2.2 показаний приклад BFS.

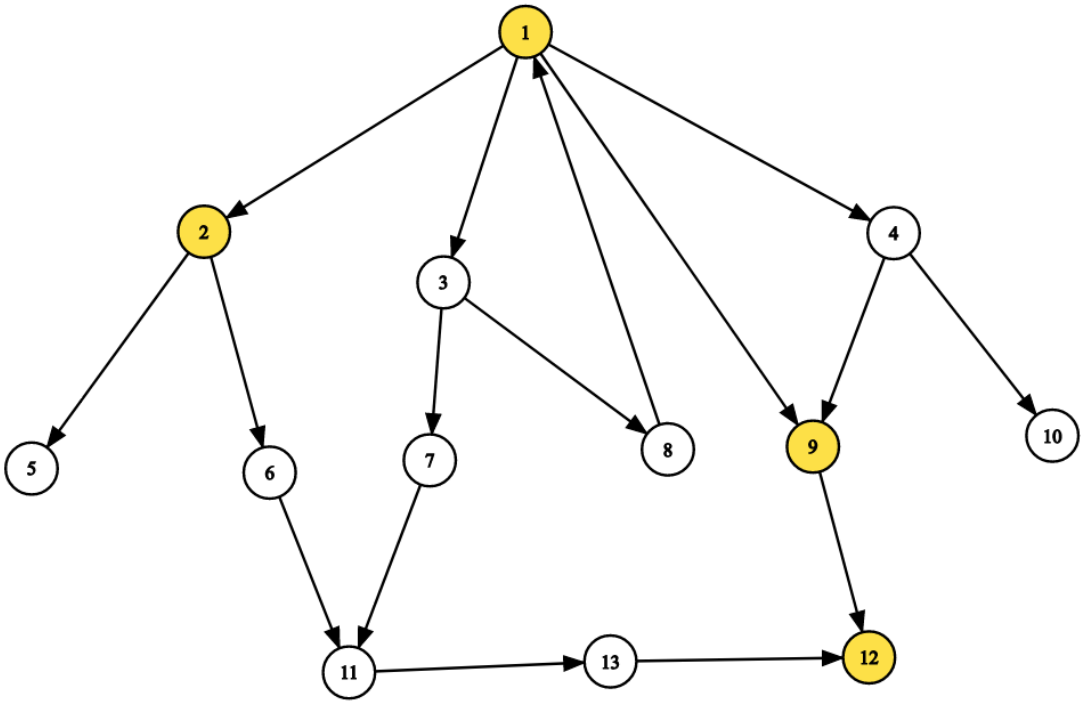

Рисунок 2.1 – Приклад DFS

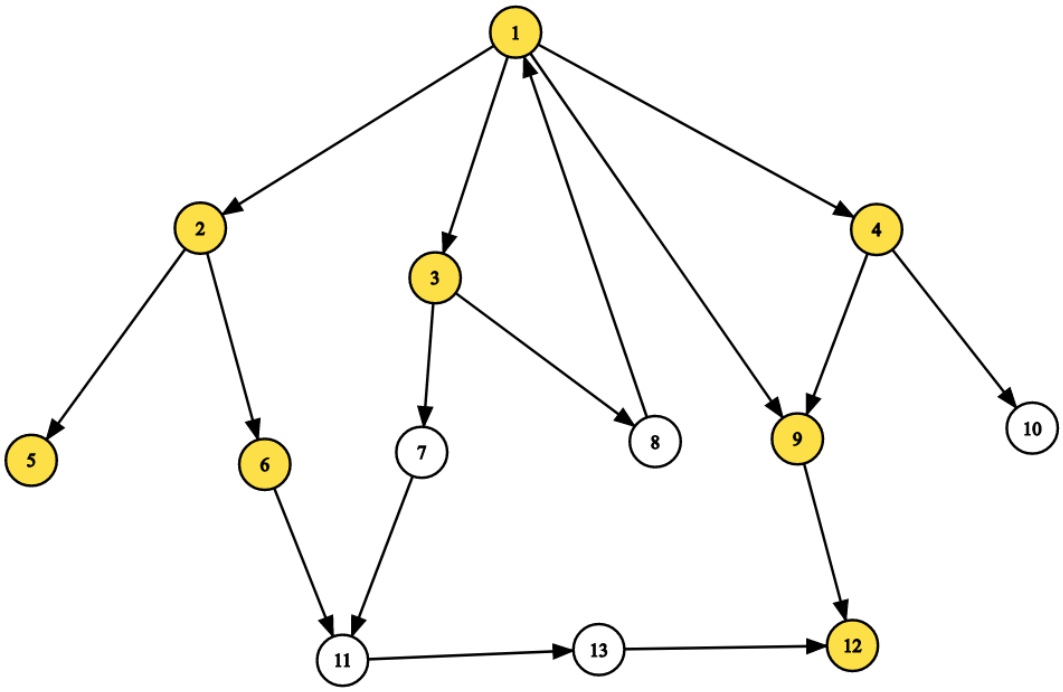

Рисунок 2.2 – Приклад BFS

Програмне забезпечення для реалізації цих алгоритмів повинно включати можливість відображення графа та результатів обходу у відповідних екранних формах. Характеристики роботи алгоритмів можуть включати швидкість обходу, витрати пам'яті та придатність для графів різного розміру та структури.

Алгоритми пошуку найкоротших шляхів, такі як Dijkstra та Алгоритм Беллмана-Форда, використовуються для знаходження оптимальних маршрутів між вузлами графа. Dijkstra працює для неорієнтованих графів з невід'ємними вагами, тоді як Беллман-Форд дозволяє обробляти графи з вагами, що можуть бути від'ємними.

Програмне забезпечення для реалізації цих алгоритмів повинно включати можливість відображення графа та знайдених найкоротших шляхів у відповідних екранних формах. Характеристики роботи алгоритмів можуть включати час виконання, точність та ефективність в умовах різних графів.

Алгоритм Форда-Фалкерсона використовується для знаходження максимального потоку в мережі. Його особливість полягає у використанні покращень алгоритму пошуку шляху виток-стік, щоб систематично збільшувати потік у графі.

Програмне забезпечення для реалізації цього алгоритму повинно включати можливість відображення мережі та результатів роботи алгоритму у відповідних екранних формах. Характеристики роботи алгоритму можуть включати час виконання, максимальний потік та ефективність в умовах різних мереж.

Індивідуальне завдання 1. Користувач визначає граф, задає вершину даного графа та деяку відстань. Визначити перелік всіх вершин графа, які знаходяться на заданій відстані від заданої вершини.

Індивідуальне завдання 2. Виконати класифікацію ребер графа на ребра дерева, зворотні ребра, прямі ребра та перехресні ребра. Визначити, чи є заданий орієнтований граф ациклічним.

Індивідуальне завдання 3. Перетворити задане користувачем слово А у слово В, створюючи при цьому ланцюжок перетворень. У кожному такому перетворенні змінюється тільки одна буква слова на іншу. При цьому всі слова мають існувати у відповідній мові. Використати перелік слів з орфографічного словника для побудови графа, у якому програмно визначити ребра, які мають з'єднувати вершини тільки у тому випадку, якщо з одного слова можна отримати інше заміною однією літери.

Індивідуальне завдання 4. Мапа визначає авіасполучення між містами Північної Америки. Кожний переліт з однієї точки на мапі в іншу має деяку мінімальну вартість, при чому зворотній рейс може коштувати іншу суму. Мандрівник хоче визначити авіапереліт між заданими містами, який має мінімальну вартість, розглядаючи зокрема і варіанти з пересадками. При цьому мандрівник має дисконтну програму з деякими авіаперевізниками, за якою ціна на деякі рейси може бути для нього знижена на деяку постійну суму (тобто вартість деяких рейсів може виявитися для нього прибутковою, в такому разі сума накопичується на окремому його рахунку).

Індивідуальне завдання 5. Мапа визначає автомобільні шляхи деякої частини міста Запоріжжя. Деякі вулиці мають односторонній рух, а на деяких можуть зустрічатися корки. Використовуючи дану інформацію та враховуючи обмеження швидкості на вулицях, визначити найкоротший шлях, яким можна дістатися з однієї заданої точки у Запоріжжі до іншої в даний момент часу.

Індивідуальне завдання 6. Визначити найкоротші шляхи між всіма точками на мапі міста Запоріжжя, використовуючи обмеження попереднього завдання.

ВИСНОВКИ

У результаті самостійної роботи були отримані значущі результати, які засвідчують успішність вирішення поставленої задачі. Аналізуючи одержані результати, варто відзначити їхню важливість та відповідність світовим тенденціям вирішення схожих завдань.

В процесі вивчення алгоритмів роботи з графами було реалізовано ефективне програмне забезпечення для обходу графів, пошуку найкоротших шляхів та використання алгоритму Форда-Фалкерсона для знаходження максимального потоку в мережі. Реалізація алгоритмів була детально відображена у відповідних екранних формах, що забезпечило зручний інтерфейс користувача.

Важливим аспектом є високий рівень достовірності отриманих результатів. Якісні та кількісні показники підтверджують ефективність використаних алгоритмів у конкретних умовах вирішення завдань з графами.

Зазначаючи галузі використання результатів роботи, слід відзначити їхню придатність для вирішення різноманітних завдань у сферах науки, техніки та інших галузях. Отримані алгоритми можуть бути використані для оптимізації процесів, пов'язаних з графовою моделлю даних.

Народногосподарська, наукова та соціальна значущість роботи визначається її внеском у розвиток області обробки графових структур та оптимізації задач, пов'язаних із мережами. Рекомендації щодо використання отриманих результатів можуть включати їхнє застосування в сучасних інформаційних системах, транспортних мережах, телекомунікаційних технологіях та інших областях.

Важливо підкреслити, що результати роботи є актуальним внеском у галузь та можуть служити основою для подальших досліджень та розробок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hatch, S.V. Computerized Engine Controls / S.V. Hatch. – Boston: Cengage Learning, 2016. – 688 p.

2. Czichos, H. Measurement, Testing and Sensor Technology. Fundamentals and Application to Materials and Technical Systems / H. Czichos. – Berlin: Springer, 2018. – 213 p.

3. Kaźmierczak, J. Data Processing and Reasoning in Technical Diagnostics / J. Kaźmierczak, W. Cholewa. – Warszawa: Wydawnictwa Naukowo-Techniczne, 1995. – 186 p.

4. Diagnostics as a Reasoning Process: From Logic Structure to Software Design / [M. Cristani, F. Olivieri, C. Tomazzoli, L. Vigano, M. Zorzi] // Journal of Computing and Information Technology. – 2018. – Vol. 27 (1). – P. 43-57.

5. Wieczorek, A.N. Analysis of the Possibility of Integrating a Mining Right-Angle Planetary Gearbox with Technical Diagnostics Systems / A.N. Wieczorek // Scientific Journal of Silesian University of Technology. Series Transport. – 2016. – Vol. 93. – P. 149-163.

6. Tso, B. Classification Methods for Remotely Sensed Data / B. Tso, P.M. Mather. – Boca Raton : CRC Press, 2016. – 352 p.

7. Oppermann, A. Regularization in Deep Learning – L1, L2, and Dropout [Electronic resource]. – Access mode: https://www.deeplearning-academy.com/p/ai-wiki-regularization.

8. Classic Regularization Techniques in Neural Networks [Electronic resource]. – Access mode: https://medium.com/@ODSC/classic-regularization-techniques-in-neural-networks-68bccee03764.

ДОДАТОК А - КОД ПРОГРАМИ

src\vite-env.d.ts
```
/// <reference types="vite/client" />
```

src\config\delay.ts
```
export const DELAY = 200
```

src\config\nodeColors.ts
```
export const NODE_COLORS = {
     default: 'white',
     checked: 'var(--color-red-300)',
     progress: 'var(--color-yellow-300)',
     done: 'var(--color-green-300)',
     passed: 'var(--color-slate-300)'
}
```

src\data\words.ts
```
export default [
     'aahs',
     'aals',
     'abac',
     'abas',
     'abba',
     'abbe',
     'abbs',
     'abed',
     'abet',
     'abid',
     'able',
     'ably',
     'abos',
     'abri',
     'abut',
     'abye',
     'abys',
     'acai',
     'acca',
     'aced',
     'acer',
     'aces',
     'ache',
     'achy',
     'acid',
     'acme',
     'acne',
     'acre',
     'acta',
     'acts',
     'acyl',
     'adaw',
     'adds',
     'addy',
     'adit',
     'ados',
     'adry',
     'adze',
     'aeon',
     'aero',
     'aery',
     'aesc',
     'afar',
     'affy',
```

```
'afro',
'agar',
'agas',
'aged',
'agee',
'agen',
'ager',
'ages',
'agha',
'agin',
'agio',
'aglu',
'agly',
'agma',
'agog',
'agon',
'ague',
'ahed',
'ahem',
'ahis',
'ahoy',
'aias',
'aida',
'aide',
'aids',
'aiga',
'ails',
'aims',
'aine',
'ains',
'airn',
'airs',
'airt',
'airy',
'aits',
'aitu',
'ajar',
'ajee',
'akas',
'aked',
'akee',
'akes',
'akin',
'alae',
'alan',
'alap',
'alar',
'alas',
'alay',
'alba',
'albe',
'albs',
'alco',
'alec',
'alee',
'alef',
'ales',
'alew',
'alfa',
'alfs',
'alga',
'alif',
'alit',
'alko',
```

```
'alky',
'alls',
'ally',
'alma',
'alme',
'alms',
'alod',
'aloe',
'aloo',
'alow',
'alps',
'also',
'alto',
'alts',
'alum',
'alus',
'amah',
'amas',
'ambo',
'amen',
'ames',
'amia',
'amid',
'amie',
'amin',
'amir',
'amis',
'amla',
'ammo',
'amok',
'amps',
'amus',
'amyl',
'anal',
'anan',
'anas',
'ance',
'ands',
'anes',
'anew',
'anga',
'anil',
'anis',
'ankh',
'anna',
'anno',
'anns',
'anoa',
'anon',
'anow',
'ansa',
'anta',
'ante',
'anti',
'ants',
'anus',
'apay',
'aped',
'aper',
'apes',
'apex',
'apod',
'apos',
'apps',
```

```
'apse',
'apso',
'apts',
'aqua',
'arak',
'arar',
'arba',
'arbs',
'arch',
'arco',
'arcs',
'ards',
'area',
'ared',
'areg',
'ares',
'aret',
'arew',
'arfs',
'argh',
'aria',
'arid',
'aril',
'aris',
'arks',
'arle',
'arms',
'army',
'arna',
'arow',
'arpa',
'arse',
'arsy',
'arti',
'arts',
'arty',
'arum',
'arvo',
'aryl',
'asar',
'asci',
'asea',
'ashy',
'asks',
'asps',
'atap',
'ates',
'atma',
'atoc',
'atok',
'atom',
'atop',
'atua',
'auas',
'aufs',
'auks',
'aula',
'auld',
'aune',
'aunt',
'aura',
'auto',
'aval',
'avas',
```

```
      'avel',
      'aver',
      'aves',
      'avid',
      'avos',
      'avow',
      'away',
      'awdl',
      'awed',
      'awee',
      'awes',
      'awfy',
      'awks',
      'awls',
      'awns',
      'awny',
      'awol',
      'awry',
      'axal',
      'axed',
      'axel',
      'axes',
      'axil',
      'axis',
      'axle',
      'axon',
      'ayah',
      'ayes',
      'ayin',
      'ayre',
      'ayus',
      'azan',
      'azon',
      'azym'
]
```

src\models\Edge.ts

```
import { Node } from './Node'

class Edge {
      adjacentNode: Node
      weight: number
      status: 'standart' | 'no-direction' = 'standart'
      type: 'default' | 'forward' | 'cross' | 'back' = 'default'

      constructor(
            adjacentNode: Node,
            weight: number,
            status: 'standart' | 'no-direction' = 'standart'
      ) {
            this.adjacentNode = adjacentNode
            this.weight = weight
            this.status = status
      }
}

export { Edge }
```

src\models\Graph.ts

```
import { Edge } from './Edge'
import { Node } from './Node'
```

```
export type GraphValue = string

export class Graph {
     graph = new Map<GraphValue, Node>()
     mode: 'directed' | 'undirected' = 'directed'
     weights: boolean = false

     addOrGetNode(
           graph: Map<GraphValue, Node>,
           value: GraphValue,
           x?: number,
           y?: number
     ) {
           if (value.length === 0) return null
           if (graph.has(value)) return graph.get(value) as Node

           const node: Node = new Node(value, x, y)
           graph.set(value, node)

           return node
     }

     toggleEdge(fromNode: Node, toNode: Node, weight: number = 1) {
           const findedEdgeFromTo = [...fromNode.edges].find(edge => {
                 return edge.adjacentNode === toNode
           })

           const findedEdgeToFrom = [...toNode.edges].find(edge => {
                 return edge.adjacentNode === fromNode
           })

           if (findedEdgeFromTo) {
                 if (findedEdgeFromTo.status === 'no-direction') {
                       findedEdgeFromTo.status = 'standart'
                       return
                 }

                 fromNode.edges.delete(findedEdgeFromTo)
                 toNode.parents.delete(fromNode)

                 if (findedEdgeToFrom) {
                       if (this.mode === 'directed') {
                             if (findedEdgeToFrom.status === 'no-
direction') {
                                   toNode.edges.delete(findedEdgeToFrom)
                                   fromNode.parents.delete(toNode)
                             }

                             if (findedEdgeToFrom.status === 'standart')
{
                                   const newEdge = new Edge(toNode, 1,
'no-direction')
                                   fromNode.edges.add(newEdge)
                                   toNode.parents.set(fromNode, newEdge)
                             }
                       }

                       if (this.mode === 'undirected') {
                             toNode.edges.delete(findedEdgeToFrom)
                             fromNode.parents.delete(toNode)
                       }
                 }
           } else {
                 const newEdge = new Edge(toNode, weight, 'standart')
```

```
                        fromNode.edges.add(newEdge)
                        toNode.parents.set(fromNode, newEdge)

                        if (!findedEdgeToFrom) {
                              if (this.mode === 'directed') {
                                    const newEdge = new Edge(fromNode, weight,
'no-direction')
                                    toNode.edges.add(newEdge)
                                    fromNode.parents.set(toNode, newEdge)
                              }

                              if (this.mode === 'undirected') {
                                    const newEdge = new Edge(fromNode, weight,
'standart')
                                    toNode.edges.add(newEdge)
                                    fromNode.parents.set(toNode, newEdge)
                              }
                        }
                  }
            }

            createGraph(
                  graphData: {
                        from: GraphValue
                        to: GraphValue
                        weight: number
                        x: number
                        y: number
                  }[]
            ) {
                  const newGraph = new Map<GraphValue, Node>()

                  for (const row of graphData) {
                        const node = this.addOrGetNode(newGraph, row.from)
                        if (!node) continue

                        node.x = row.x
                        node.y = row.y

                        const adjuacentNode = this.addOrGetNode(newGraph,
row.to)

                        if (adjuacentNode === null) continue

                        this.toggleEdge(node, adjuacentNode, row.weight)
                        // const edge = new Edge(adjuacentNode, row.weight)

                        // node?.edges.add(edge)
                  }

                  return newGraph
            }
      }


      src\models\Node.ts
      import { Edge } from './Edge'

      import { GraphValue } from './Graph'

      class Node {
           value: GraphValue
           edges = new Set<Edge>()
           parents = new Map<Node, Edge>()
```

```
	x: number | null = null
	y: number | null = null
	status: 'default' | 'progress' | 'done' | 'passed' = 'default'

	constructor(
		value: GraphValue,
		x: number | null = null,
		y: number | null = null
	) {
		this.value = value

		this.x = x
		this.y = y
	}

	toString() {
		return JSON.stringify({
			from: this.value,
			to: -1,
			weight: 1,
			x: this.x,
			y: this.y
		})
	}
}

export { Node }
```

src\pages\dynamic\FirstTask.class.ts

```
/*
В. Послідовність складається з деякого набору цілих чисел.
Елементи послідовності можуть бути, як від’ємними, так і
невід’ємними цілими числами.

Визначити найдовшу спадну підпослідовність даної послідовності.

Вивести на екран довжину такої послідовності та всі її члени.
*/

class FirstTask {
	resolve(
		nums: number[],
		index: number = 0,
		previous: number = Infinity
	): number[] {
		if (index === nums.length) return []

		const excludeCurrent = this.resolve(nums, index + 1, previous)

		if (nums[index] >= previous) return excludeCurrent

		const includeCurrent: number[] = [
			nums[index],
			...this.resolve(nums, index + 1, nums[index])
		]

		return includeCurrent.length > excludeCurrent.length
			? includeCurrent
			: excludeCurrent
	}
}
```

```
export { FirstTask }


src\pages\dynamic\main.ts
import { FirstTask } from './FirstTask.class'
import { SecondTask } from './SecondTask.class'

console.log('Lab 4 | Algoritms\n')

console.log('Lab 4 | First task\n')
const firstTask = new FirstTask()

const firstTaskOutput = document.querySelector('#first-task-output')

const firstTaskCheckData = [
     {
          input: [0, 1, 0, 3, 2, 3],
          expectedOutput: [3, 2]
     },
     {
          input: [10, 9, 2, 5, 3, 7, 101, 18],
          expectedOutput: [10, 9, 5, 3]
     },
     {
          input: [5, 8, 7, 1, 2, 10, 3],
          expectedOutput: [8, 7, 3]
     },
     {
          input: [1, 3, 6, 7, 9, 4, 10, 5, 6],
          expectedOutput: [10, 6]
     },
     {
          input: [3, 4, 2, 8, 10],
          expectedOutput: [4, 2]
     },
     {
          input: [1, 2, 3, 4, 5],
          expectedOutput: [5]
     },
     {
          input: [5, 4, 3, 2, 1],
          expectedOutput: [5, 4, 3, 2, 1]
     },
     {
          input: [1],
          expectedOutput: [1]
     },
     {
          input: [],
          expectedOutput: []
     },
     {
          input: [2, 2, 2, 2, 2],
          expectedOutput: [2]
     }
]

const testData = []

firstTaskCheckData.forEach(test => {
     const actual = firstTask.resolve(test.input)

     testData.push({
```

```
                passCheck:      JSON.stringify(test.expectedOutput)       ===
JSON.stringify(actual),
                input: JSON.stringify(test.input),
                expected: JSON.stringify(test.expectedOutput),
                actual: JSON.stringify(actual),
                length: actual.length
          })
     })

     firstTaskOutput.textContent = JSON.stringify(testData, null, 2)
     // console.table(testData)

     console.log('\nLab 4 | Second task\n')
     const secondTask = new SecondTask()

     const secondTaskOutput = document.querySelector('#second-task-output')

     const dataForSecond = [
          {
               matrix: [
                    [1, 0, 0],
                    [0, 1, 0],
                    [0, 0, 1]
               ],
               count: 6
          },
          {
               matrix: [
                    [1, 1, 0],
                    [1, 0, 1],
                    [0, 1, 1]
               ],
               count: 6
          },
          {
               matrix: [
                    [1, 1, 1],
                    [1, 1, 1],
                    [1, 1, 1]
               ],
               count: 6
          },
          {
               matrix: [
                    [1, 1, 1],
                    [1, 1, 1],
                    [1, 1, 1]
               ],
               count: 7
          }
     ]

     dataForSecond.forEach(data => {
          const result = secondTask.resolve(data.matrix, data.count)

          secondTaskOutput.textContent =
               secondTaskOutput.textContent +
               '\n' +
               `On matrix = ${JSON.stringify(data.matrix)}, count = ${
                    data.count
               } result = ${result.length}\n${JSON.stringify(result)}\n`
     })
```

src\pages\dynamic\SecondTask.class.ts

```
/*
Д. На одній з вулиць містечка будинки класифіковано за трьома
типами: перший – звичайні житлові споруди, другий – промислові
споруди, а третій – міські заклади (лікарні, школи тощо).

У результаті вулиця схематично зображена набором літер, кожна з яких визначає
тип будинку. У процесі збору інформації про місто була створена
матриця – таблиця, в якій кожен стовпчик і рядок відповідають
одному з типів будівель.

Відповідно клітинка такої таблиці визначає,
чи розташовані на даній вулиці міста поруч будівлі заданого типу.
Матриця симетрична.

Визначити, скільки існує способів взаємного
розташування будинків даних типів між собою за заданою матрицею
для заданої кількості будинків на вулиці,

тобто кількість можливих
наборів літер заданої довжини, що відповідають заданій матриці.
*/

class SecondTask {
    checkItemOnValid(matrix: number[][], item: string) {
        for (let i = 0; i < matrix.length; i++) {
            for (let j = 0; j < matrix[i].length; j++) {
                if (matrix[i][j] === 0) continue

                if (!(item.includes(`${i}${j}`) || item.includes(`${j}${i}`))) {
                    return false
                }
            }
        }

        return true
    }

    resolve(matrix: number[][], count: number): string[] {
        const results: string[] = []

        const inner = (result: string = '', currentIndex: number = 0): void => {
            if (currentIndex === count) {
                if (this.checkItemOnValid(matrix, result)) {
                    results.push(result)
                }

                return
            }

            for (let type = 0; type < 3; type++) {
                if (matrix[type][result[currentIndex - 1]] === 0) {
                    continue
                }

                inner(result + type, currentIndex + 1)
            }
        }

        inner()
```

```
            return results
      }
}

export { SecondTask }
```

src\pages\graph\main.ts

```
import './style.css'
import { Graph, GraphValue } from '../../models/Graph'
import { Edge } from '../../models/Edge'
import { Node } from '../../models/Node'
import { NODE_COLORS } from '../../config/nodeColors'
import { DELAY } from '../../config/delay'
import words from '../../data/words'

async function sleep(time: number) {
      await new Promise(resolve => {
            setTimeout(() => {
                  resolve(null)
            }, time)
      })
}

class TreeNode {
      value: GraphValue
      childrens: TreeNode[] = []

      constructor(value: GraphValue) {
            this.value = value
      }

      find(node: TreeNode, value: GraphValue): TreeNode | undefined {
            if (node.value === value) return node

            for (const child of node.childrens) {
                  const result = this.find(child, value)

                  if (result) {
                        return result
                  }
            }

            return undefined
      }

      add(node: TreeNode) {
            this.childrens.push(node)
      }
}

class Tree {
      root: TreeNode

      constructor(value: GraphValue) {
            this.root = new TreeNode(value)
      }

      add(node: GraphValue, value: GraphValue) {
            const prevNode = this.root.find(this.root, node)

            if (!prevNode) return
```

```
                prevNode.add(new TreeNode(value))
        }

        display() {
                console.log(this.root)
        }
}

const DEBUG = false

class App {
        graph: Graph

        offsetX = 0
        offsetY = -105
        lastGraph: 'default' | 'lb5' | 'lb61' | 'lb62' = 'default'

        mouseDownValues: {
                active: boolean
                target: HTMLElement | null
                innerOffsetX: number
                innerOffsetY: number
        } = {
                active: false,
                target: null,
                innerOffsetX: 0,
                innerOffsetY: 0
        }

        algorithmActiveId: number | null = -1
        currentClickedTarget: HTMLElement | null = null

        pressedKeyCode: string | null = null

        constructor() {
                this.graph = new Graph()
                // this.initializeGraph()
                this.initializeGraphForLB61()
                this.render()
        }

        onKeyDown(event: KeyboardEvent): void {
                this.pressedKeyCode = event.code

                if (event.code === 'Escape') {
                        this.currentClickedTarget = null
                        this.render()
                }
        }

        onKeyUp(event: KeyboardEvent): void {
                if (this.pressedKeyCode === event.code) {
                        this.pressedKeyCode = null
                        this.render()
                }
        }

        onMouseDown(e: MouseEvent) {
                this.mouseDownValues = {
                        active: true,
                        target: e.target as HTMLElement,
                        innerOffsetX:
                                e.clientX - (e.target as
HTMLElement).getBoundingClientRect().x - 20,
```

```
                        innerOffsetY:
                              e.clientY          -          (e.target          as
HTMLElement).getBoundingClientRect().y - 20
                  }
            }

            onMouseUp() {
                  this.mouseDownValues = {
                        active: false,
                        target: null,
                        innerOffsetX: 0,
                        innerOffsetY: 0
                  }
            }

            onMouseMove(e: MouseEvent) {
                  if (!this.mouseDownValues.active) return

                  if (this.pressedKeyCode === 'Space') {
                        console.log(e)
                        this.offsetX += e.movementX
                        this.offsetY += e.movementY

                        this.render()
                  } else {
                        if (!this.mouseDownValues.target) return
                        if      (!this.mouseDownValues.target.dataset.elementid)
return
                        const node = this.graph.graph.get(
                              this.mouseDownValues.target.dataset.elementid
                        )

                        if (!node) return

                        node.x     =     e.clientX     -     this.offsetX     -
this.mouseDownValues.innerOffsetX
                        node.y     =     e.clientY     -     this.offsetY     -
this.mouseDownValues.innerOffsetY

                        this.render()


      console.log(this.mouseDownValues.target.dataset.elementid)
                  }
            }

            onClick(e: MouseEvent) {
                  console.log('click')

                  if ((e.target as HTMLElement).tagName !== 'svg') {
                        if (!(e.target as HTMLElement).dataset.elementid) return

                        console.log('currentClikedTarget:                    ',
this.currentClickedTarget)

                        if (
                              this.currentClickedTarget !== null &&
                              this.currentClickedTarget !== e.target
                        ) {
                              const nodePrev = this.graph.graph.get(
                                    this.currentClickedTarget.dataset.elementid
|| ''
                              )
```

```
                              const nodeCurrent = this.graph.graph.get(
                                    (e.target as HTMLElement).dataset.elementid
|| ''
                              )

                              if (!nodePrev || !nodeCurrent) return

                              this.graph.toggleEdge(nodePrev, nodeCurrent)

                              //          const          findedEdge          =
[...nodePrev.edges].find(edge => {
                              //    return edge.adjacentNode === nodeCurrent
                              // })

                              // if (!findedEdge) {
                              //    nodePrev.edges.add(new Edge(nodeCurrent, 1))
                              // } else {
                              //    nodePrev.edges.delete(findedEdge)
                              // }

                              this.currentClickedTarget = null

                              this.render()
                              return
                        }

                        this.currentClickedTarget = e.target as HTMLElement

                        this.render()
                        return
                  }

                  console.log(e)
                  console.log({
                        x: e.x,
                        y: e.y,
                        offsetX: this.offsetX,
                        offsetY: this.offsetY
                  })

                  const                        lastElement                        =
[...this.graph.graph.values()].reduce((acc, node) => {
                        const asNumber = Number(node.value)

                        if (isNaN(asNumber)) {
                              return acc
                        }

                        return asNumber > acc ? asNumber : acc
                  }, -1)

                  this.graph.addOrGetNode(
                        this.graph.graph,
                        String(lastElement + 1),
                        e.clientX - this.offsetX,
                        e.clientY - this.offsetY
                  )

                  this.render()
            }

            onContextMenu(e: MouseEvent) {
                  e.preventDefault()
```

```
                if (!(e.target as HTMLElement).dataset.elementid) return

                const nodeId = (e.target as HTMLElement).dataset.elementid ||
''

                const node = this.graph.graph.get(nodeId)

                if (!node) return

                this.graph.graph.forEach(graphNode => {
                        graphNode.edges = new Set(
                                [...graphNode.edges].filter(edge => {
                                        return edge.adjacentNode !== node
                                })
                        )
                })

                this.graph.graph.delete(nodeId)

                this.render()
        }

        // #nodeStatusChanger(node: Node, newStatus: 'default' | 'progress'
| 'done') {
        //    node.status = newStatus
        // }

        #graphNodesStatusResetter(id: number) {
                if (this.algorithmActiveId !== id) return

                this.graph.graph.forEach(node => {
                        if (this.algorithmActiveId !== id) return

                        node.status = 'default'
                })

                this.render()
        }

        #graphEdgesTypeResetter(id: number) {
                if (this.algorithmActiveId !== id) return

                this.graph.graph.forEach(node => {
                        if (this.algorithmActiveId !== id) return

                        node.edges.forEach(edge => {
                                edge.type = 'default'
                        })
                })

                this.render()
        }

        async #setNodeStatus(
                node: Node,
                params: {
                        status?: Node['status']
                        sleep?: boolean
                        render?: boolean
                        statusForChange?: Node['status'] | 'any'
                        renderBeforeSleep?: boolean
                } = {}
        ) {
                const status = params.status ?? 'default'
```

```
                const sleepFlag = params.sleep ?? true
                const render = params.render ?? true
                const statusForChange = params.statusForChange ?? node.status
                const renderBeforeSleep = params.renderBeforeSleep ?? false

                if (statusForChange && statusForChange === node.status) {
                        node.status = status
                }

                if (render && renderBeforeSleep) {
                        this.render()
                }

                if (sleepFlag) {
                        await sleep(DELAY)
                }

                if (render && !renderBeforeSleep) {
                        this.render()
                }
        }

        /* Algorithms */

        // TODO: Move to another class
        async dfsWrapper(id: number) {
                const visited: Node[] = []

                for (const item of this.graph.graph.values()) {
                        if (!visited.includes(item)) {
                                if (this.algorithmActiveId !== id) {
                                        return
                                }

                                await this.dfs(item, visited, id)
                        }
                }

                this.render()
        }

        async dfs(node: Node, visited: Node[], id: number) {
                const jungle: Node[] = []
                const stack = [node]

                while (stack.length > 0) {
                        if (this.algorithmActiveId !== id) {
                                return
                        }

                        const item = stack.pop()
                        if (!item) return null

                        visited.push(item)
                        jungle.push(item)
                        ;[...item.edges].toReversed().forEach(edge => {
                                if (this.graph.mode === 'directed' && edge.status
=== 'no-direction') {
                                        return
                                }

                                const adjacentNode = edge.adjacentNode
```

```
                        if (!visited.includes(adjacentNode) &&
!stack.includes(adjacentNode)) {
                              stack.push(adjacentNode)
                        }
                  })

                  await this.#setNodeStatus(item, {
                        status: 'progress',
                        sleep: false
                  })
                  await sleep(DELAY)
            }

            this.render()

            console.log('DFS:', jungle.map(item => item.value).join(',
      '))
      }

      // TODO: Move to another class
      async bfsWrapper(id: number) {
            const visited: Node[] = []

            for (const item of this.graph.graph.values()) {
                  if (!visited.includes(item)) {
                        if (this.algorithmActiveId !== id) {
                              return
                        }

                        await this.bfs(item, visited, id)
                  }
            }

            this.render()
      }

      async bfs(node: Node, visited: Node[], id: number) {
            const tree = new Tree(node.value)

            const queue = [node]

            while (queue.length > 0) {
                  if (this.algorithmActiveId !== id) {
                        return
                  }

                  const item = queue.shift()
                  if (!item) return null

                  visited.push(item)

                  item.edges.forEach(edge => {
                        if (this.graph.mode === 'directed' && edge.status
=== 'no-direction')
                              return

                        const adjacentNode = edge.adjacentNode

                        if (!visited.includes(adjacentNode) &&
!queue.includes(adjacentNode)) {
                              tree.add(item.value ?? -1,
adjacentNode.value ?? -1)
                              queue.push(adjacentNode)
                        }
```

```
                        })

                        await this.#setNodeStatus(item, {
                              status: 'progress',
                              sleep: false
                        })
                        await sleep(DELAY)
                  }

                  tree.display()

                  this.render()
            }

            async lb5TaskOne(
                  node: Node,
                  visited: Node[],
                  id: number,
                  maxLevel = 2,
                  level = 0
            ) {
                  if (level > maxLevel) return []

                  visited.push(node)

                  node.status = 'progress'

                  this.render()

                  await sleep(DELAY)

                  const result: (number | null)[] = []

                  for (const edge of node.edges) {
                        if (this.graph.mode === 'directed' && edge.status ===
'no-direction') {
                              continue
                        }

                        if (!visited.includes(edge.adjacentNode)) {
                              if (this.algorithmActiveId !== id) {
                                    return
                              }

                              const r = await this.lb5TaskOne(
                                    edge.adjacentNode,
                                    visited,
                                    id,
                                    maxLevel,
                                    level + 1
                              )

                              if (r === undefined) {
                                    continue
                              }

                              result.push(r)
                        }
                  }

                  return [node.value, ...result].flat()
            }

            async lb5TaskSecond(id: number) {
```

```
      const visited: Node[] = []
      const startTime: Map<Node, number> = new Map()
      const endTime: Map<Node, number> = new Map()
      const state: { time: number } = { time: 0 }

      for (const item of this.graph.graph.values()) {
            if (!visited.includes(item)) {
                  if (this.algorithmActiveId !== id) {
                        return
                  }

                  await this.lb5TaskSecondInner(
                        item,
                        visited,
                        startTime,
                        endTime,
                        state,
                        id
                  )
            }
      }

      console.log(startTime.size, endTime.size)

      this.render()
}

compareStrings(firstString: string, secondString: string) {
      if (firstString === secondString) return true
      if (firstString.length !== secondString.length) return false

      let differences = 0

      for (let i = 0; i < firstString.length; i++) {
            if (firstString[i] !== secondString[i]) {
                  differences++

                  if (differences > 1) return false
            }
      }

      return true
}

async findPathThird(
      start: Node,
      end: Node,
      visited: Set<Node>,
      path: Map<Node, Node>
) {
      const queue = [start]

      while (queue.length > 0) {
            const item = queue.shift()
            if (!item) return null

            if (item === end) {
                  return item
            }

            visited.add(item)

            item.edges.forEach(edge => {
```

```
                        if (this.graph.mode === 'directed' && edge.status
=== 'no-direction')
                              return

                        const adjacentNode = edge.adjacentNode

                        if          (!visited.has(adjacentNode)          &&
!queue.includes(adjacentNode)) {
                              queue.push(adjacentNode)
                              path.set(adjacentNode, item)
                        }
                  })
            }

            return false

            // if (start === end) {
            //    paths.push([...visited, start])

            //    console.log(paths.at(-1))
            //    return
            // }

            // // if (visited.size > 50) {
            // //       return
            // // }

            // visited.add(start)

            // for (const edge of start.edges) {
            //    if (!visited.has(edge.adjacentNode)) {
            //          this.findPathThird(edge.adjacentNode,         end,
visited, paths)
            //    }
            // }

            // visited.delete(start)
      }

      async lb5TaskThird(wordFrom: string = 'abba', wordTo: string =
'alba') {
            if (wordFrom.length !== wordTo.length) return false

            const wordsWithLenght: string[] = words.filter(
                  word => word.length === wordFrom.length
            )

            if (wordsWithLenght.includes(wordFrom) === false) return false
            if (wordsWithLenght.includes(wordTo) === false) return false

            this.lastGraph = 'lb5'
            this.graph.graph = new Map()

            const newGraph = this.graph

            let x = 0
            let y = 0
            const maxRow = 30
            let row = 0

            wordsWithLenght.forEach(element => {
                  newGraph.addOrGetNode(newGraph.graph, element, x, y)
                  x += 100
```

```
                        if (row >= maxRow) {
                              y += 100
                              x = 0
                              row = 0
                        }

                        row++
                  })

                  this.render()

                  console.time('Start creating of graph')
                  let passed = 0

                  newGraph.graph.forEach(rootNode => {
                        newGraph.graph.forEach(node => {
                              if (rootNode === node) return

                              const  findedEdge  =  [...rootNode.edges].find(edge
=> {
                                    return edge.adjacentNode === node
                              })
                              if (findedEdge !== undefined) return

                              if            (this.compareStrings(rootNode.value,
node.value)) {
                                    newGraph.toggleEdge(rootNode, node)
                              }
                        })

                        passed++

                        console.log(
                              `${((passed     /     wordsWithLenght.length)     *
100).toFixed(2)}% passed`
                        )
                  })

                  console.timeEnd('Start creating of graph')

                  this.render()

                  // await sleep(5000)

                  const startNode = [...newGraph.graph.values()].find(
                        node => node.value === wordFrom
                  )
                  const endNode = [...newGraph.graph.values()].find(
                        node => node.value === wordTo
                  )
                  if (!startNode || !endNode) return

                  console.log('%c●', 'color: #733d00', startNode)
                  console.log('%c●', 'color: #00bf00', endNode)

                  const path = new Map()

                  const resultNode = await this.findPathThird(
                        startNode,
                        endNode,
                        new Set(),
                        path
                  )
```

```
                console.log(resultNode, path)

                let node = resultNode
                const path2: Node[] = []

                while (node !== startNode) {
                        console.log(node)

                        if (node == undefined || typeof node === 'boolean') {
                                break
                        }

                        path2.unshift(node)
                        node = path.get(node)
                }

                console.log(path2)

                await sleep(3000)

                path2.forEach(item => {
                        item.status = 'progress'
                })

                startNode.status = 'done'
                endNode.status = 'done'

                // this.graph.graph = new Map()

                path2.unshift(startNode)

                for (let i = 0; i < path2.length; i++) {
                        const element = path2[i]
                        // this.graph.graph.set(element.value, element)

                        const nextElement = path2[i + 1]

                        element.edges.clear()
                        if (nextElement) {
                                element.edges.add(new Edge(nextElement, 1))
                        }
                }

                this.graph.graph.forEach((item, key) => {
                        if (!path2.includes(item) && item !== startNode && item
!== endNode) {
                                this.graph.graph.delete(key)
                        }
                })

                this.render()
        }

        async lb5TaskSecondInner(
                node: Node,
                visited: Node[],
                startTime: Map<Node, number>,
                endTime: Map<Node, number>,
                state: { time: number },
                id: number
        ) {
                const jungle: Node[] = []

                if (this.algorithmActiveId !== id) {
```

```
                        return
                  }

                  if (!node) return null

                  startTime.set(node, state.time)
                  state.time++

                  visited.push(node)
                  jungle.push(node)

                  for (const edge of [...node.edges].toReversed()) {
                        if (this.graph.mode === 'directed' && edge.status ===
'no-direction') {
                              continue
                        }

                        const adjacentNode = edge.adjacentNode

                        // if (item.value === 8) debugger

                        if (!visited.includes(adjacentNode)) {
                              console.log(
                                    'Tree Edge: ' + node.value + '-->' +
adjacentNode.value + '<br>'
                              )

                              edge.type = 'default'

                              await this.lb5TaskSecondInner(
                                    adjacentNode,
                                    visited,
                                    startTime,
                                    endTime,
                                    state,
                                    id
                              )
                        } else {
                              // if parent node is traversed after the neighbour
node

                              const itemStartTime = startTime.get(node) ?? -1
                              const             adjacentStartTime             =
startTime.get(adjacentNode) ?? -1
                              const itemEndTime = endTime.get(node) ?? -1
                              const adjacentEndTime = endTime.get(adjacentNode)
?? -1

                              console.table({
                                    value: node.value,
                                    // startTime: startTime,
                                    // endTime: JSendTime,
                                    adjacentNode: adjacentNode.value,
                                    itemStartTime: itemStartTime,
                                    adjacentStartTime: adjacentStartTime,
                                    itemEndTime: itemEndTime,
                                    adjacentEndTime: adjacentEndTime
                              })

                              if (itemStartTime >= adjacentStartTime &&
adjacentEndTime === -1) {
                                    console.log(
                                          'Back Edge: ' + node.value + '-->' +
adjacentNode.value + '<br>'
```

```
                                    )
                                    edge.type = 'back'
                              }

                              // if the neighbour node is a  but not a part of
the tree
                              else  if  (itemStartTime  <  adjacentStartTime  &&
adjacentEndTime !== -1) {
                                    console.log(
                                          'Forward Edge: ' + node.value + '-->'
+ adjacentNode.value + '<br>'
                                    )
                                    edge.type = 'forward'
                              }

                              // if parent and neighbour node do not
                              // have any ancestor and descendant relationship
between them
                              else {
                                    console.log(
                                          'Cross Edge: ' + node.value + '-->' +
adjacentNode.value + '<br>'
                                    )
                                    edge.type = 'cross'
                              }
                        }
                  }

                  endTime.set(node, state.time)
                  state.time++

                  await this.#setNodeStatus(node, {
                        status: 'progress',
                        sleep: false
                  })

                  await sleep(DELAY)

                  this.render()

                  // console.log('DFS:', jungle.map(item => item.value).join(',
      '))
            }

            async initHashTables(
                  start: Node,
                  graph: Map<GraphValue, Node>,
                  unprocessedNodes: Set<Node>,
                  timeToNodes: Map<Node, number>
            ) {
                  for (const item of graph) {
                        const node = item[1]
                        unprocessedNodes.add(node)
                        timeToNodes.set(node, Infinity)
                  }

                  timeToNodes.set(start, 0)
            }

            async getNodeWithMinTime(
                  unprocessedNodes: Set<Node>,
                  timeToNodes: Map<Node, number>
            ) {
                  let nodeWithMinTime: Node | null = null
```

```
                let minTime = Infinity

                for (const node of unprocessedNodes) {
                        const time = timeToNodes.get(node)

                        if (time !== undefined && time < minTime) {
                                minTime = time
                                nodeWithMinTime = node
                        }
                }

                return nodeWithMinTime
        }

        async calculateTimeToEachNode(
                unprocessedNodes: Set<Node>,
                timeToNodes: Map<Node, number>
        ) {
                while (unprocessedNodes.size > 0) {
                        const              node              =              await
this.getNodeWithMinTime(unprocessedNodes, timeToNodes)
                        console.log(node)

                        if (!node) return
                        if (timeToNodes.get(node) === Infinity) return

                        await this.#setNodeStatus(node, {
                                status: 'progress',
                                statusForChange: 'default'
                        })

                        for (const edge of node.edges) {
                                if (this.graph.mode === 'directed' && edge.status
=== 'no-direction') {
                                        continue
                                }

                                const adjacentNode = edge.adjacentNode

                                if (unprocessedNodes.has(adjacentNode)) {
                                        const nodeTime = timeToNodes.get(node)
                                        if (nodeTime === undefined) continue

                                        const timeToCheck = nodeTime + edge.weight

                                        const           adjacentNodeTime           =
timeToNodes.get(adjacentNode)
                                        if (adjacentNodeTime === undefined) continue

                                        if (timeToCheck < adjacentNodeTime) {
                                                timeToNodes.set(adjacentNode,
timeToCheck)
                                        }
                                }
                        }

                        unprocessedNodes.delete(node)
                }
        }

        async getShortestPath(
                start: Node,
                end: Node,
```

```
                timeToNodes: Map<Node, number>
        ) {
                const path = []
                let node = end

                while (node !== start) {
                        const minTimeToNode = timeToNodes.get(node)
                        path.unshift(node)

                        for (const parentAndEdge of node.parents.entries()) {
                                const parent = parentAndEdge[0]
                                const parentEdge = parentAndEdge[1]

                                if (!timeToNodes.has(parent)) continue

                                const prevNodeFound =
                                        Number(parentEdge.weight                    +
(timeToNodes.get(parent) ?? 0)) ===
                                        minTimeToNode

                                if (prevNodeFound) {
                                        timeToNodes.delete(node)
                                        node = parent
                                        break
                                }
                        }
                }

                path.unshift(node)
                return path
        }

        async dijkstra(start: Node, end: Node) {
                const unprocessedNodes = new Set<Node>()
                const timeToNodes = new Map<Node, number>()

                await this.initHashTables(
                        start,
                        this.graph.graph,
                        unprocessedNodes,
                        timeToNodes
                )

                await          this.calculateTimeToEachNode(unprocessedNodes,
timeToNodes)
                console.log('%c●', 'color: #d90000', unprocessedNodes)
                console.log('%c●', 'color: #ffa640', timeToNodes)
                console.log(timeToNodes.get(end))

                if (timeToNodes.get(end) === Infinity) return null

                return await this.getShortestPath(start, end, timeToNodes)
        }

        /* LB 6 2*/

        async bellmanFord(startNode: Node) {
                const distances = new Map<Node, number>()

                for (const node of this.graph.graph.values()) {
                        distances.set(node, Infinity)
                }

                distances.set(startNode, 0)
```

```
                for (let i = 0; i < this.graph.graph.size; i++) {
                        for (const currentNode of this.graph.graph.values()) {
                                for (const edge of currentNode.edges) {
                                        if (
                                                this.graph.mode === 'directed' &&
                                                edge.status === 'no-direction'
                                        ) {
                                                continue
                                        }

                                        const              newDistance              =
distances.get(currentNode)! + edge.weight

                                        if               (newDistance               <
distances.get(edge.adjacentNode)!) {
                                                distances.set(edge.adjacentNode,
newDistance)
                                        }
                                }
                        }
                }

                for (const currentNode of this.graph.graph.values()) {
                        if (currentNode === startNode) {
                                continue
                        }

                        for (const edge of currentNode.edges) {
                                if (this.graph.mode === 'directed' && edge.status
=== 'no-direction') {
                                        continue
                                }

                                if (
                                        distances.get(currentNode)! + edge.weight <
                                        distances.get(edge.adjacentNode)!
                                ) {
                                        console.error('Graph  contains  a  negative
cycle.')
                                        return
                                }
                        }
                }

                return distances
        }

        floydWarshall(nodes: Node[]) {
                const numNodes = nodes.length

                // Initialize the distance matrix with Infinity
                const distances: number[][] = Array.from({ length: numNodes },
() =>
                        Array(numNodes).fill(Infinity)
                )

                // Initialize the distance matrix with actual edge weights
                nodes.forEach((node, i) => {
                        distances[i][i] = 0

                        node.edges.forEach(edge => {
                                if (this.graph.mode === 'directed' && edge.status
=== 'no-direction') {
```

```
                                    return
                              }

                              const j = nodes.indexOf(edge.adjacentNode)

                              distances[i][j] = edge.weight
                        })
                  })

                  // Floyd-Warshall algorithm
                  for (let k = 0; k < numNodes; k++) {
                        for (let i = 0; i < numNodes; i++) {
                              for (let j = 0; j < numNodes; j++) {
                                    if (distances[i][k] + distances[k][j] <
distances[i][j]) {
                                          distances[i][j] = distances[i][k] +
distances[k][j]
                                    }
                              }
                        }
                  }

                  nodes.forEach((nodeTop, i) => {
                        nodes.forEach((nodeBottom, j) => {
                              const result = distances[i][j]

                              if (result === Infinity) return

                              console.log(`${nodeTop.value}                    =>
${nodeBottom.value}: ${result}`)
                        })
                  })

                  return distances
            }

            async findPath(
                  start: Node,
                  end: Node,
                  visited: Node[],
                  path: Node[],
                  id: number
            ) {
                  if (start === end) {
                        path.push(start)

                        await this.#setNodeStatus(start, {
                              status: 'done',
                              sleep: false
                        })

                        return true
                  }

                  visited.push(start)

                  await this.#setNodeStatus(start, {
                        status: 'progress'
                  })

                  for (const edge of start.edges) {
                        if (this.algorithmActiveId !== id) {
                              return
                        }
```

```
                        if (!visited.includes(edge.adjacentNode)) {
                              if  (await  this.findPath(edge.adjacentNode,  end,
visited, path, id)) {
                                    if (this.algorithmActiveId !== id) {
                                          return
                                    }

                                    path.push(start)

                                    await this.#setNodeStatus(start, {
                                          status: 'done',
                                          sleep: false
                                    })

                                    return true
                              }
                        }
                  }

                  if (start.status === 'progress') {
                        start.status = 'passed'
                  }

                  this.render()
            }

            async findPathes(
                  start: Node,
                  end: Node,
                  visited: Set<Node>,
                  paths: Node[][],
                  id: number
            ) {
                  if (start === end) {
                        paths.push([...visited, start])

                        paths[paths.length - 1].forEach(item => {
                              item.status = 'done'
                        })

                        return
                  }

                  visited.add(start)

                  await this.#setNodeStatus(start, {
                        status: 'progress'
                  })

                  // if (start.status === 'default') {
                  //    start.status = 'progress'
                  // }

                  // await sleep(DELAY)

                  // this.render()

                  for (const edge of start.edges) {
                        if (this.algorithmActiveId !== id) {
                              return
                        }

                        if (!visited.has(edge.adjacentNode)) {
```

```
                        await this.findPathes(edge.adjacentNode, end,
visited, paths, id)
                  }
            }

            visited.delete(start)

            await this.#setNodeStatus(start, {
                  status: 'passed',
                  sleep: false,
                  statusForChange: 'progress'
            })

            // if (start.status === 'progress') {
            //    start.status = 'passed'
            // }
            // this.render()
      }

      private initializeGraph() {
            this.graph.graph = this.graph.createGraph([
                  { from: '1', to: '9', weight: 1, x: 751, y: 189 },
                  { from: '1', to: '2', weight: 1, x: 751, y: 189 },
                  { from: '1', to: '3', weight: 1, x: 751, y: 189 },
                  { from: '1', to: '4', weight: 1, x: 751, y: 189 },
                  { from: '2', to: '5', weight: 1, x: 516, y: 335 },
                  { from: '2', to: '6', weight: 1, x: 516, y: 335 },
                  {
                        from: '3',
                        to: '7',
                        weight: 1,
                        x: 691,
                        y: 372
                  },
                  {
                        from: '3',
                        to: '8',
                        weight: 1,
                        x: 691,
                        y: 372
                  },
                  { from: '4', to: '9', weight: 1, x: 1020, y: 336 },
                  { from: '4', to: '10', weight: 1, x: 1020, y: 336 },
                  { from: '5', to: '', weight: 1, x: 390, y: 508 },
                  {
                        from: '6',
                        to: '11',
                        weight: 1,
                        x: 564,
                        y: 511
                  },
                  { from: '7', to: '11', weight: 1, x: 681, y: 502 },
                  { from: '8', to: '1', weight: 1, x: 855, y: 494 },
                  { from: '9', to: '12', weight: 1, x: 961, y: 492 },
                  { from: '10', to: '', weight: 1, x: 1136, y: 484 },
                  {
                        from: '11',
                        to: '13',
                        weight: 1,
                        x: 622,
                        y: 657
                  },
                  { from: '12', to: '', weight: 1, x: 1002, y: 646 },
                  { from: '13', to: '12', weight: 1, x: 813, y: 649 }
```

```
                ])
        }

        #getNodeStatusForRender(node: Node) {
                return this.currentClickedTarget &&
                        this.currentClickedTarget.dataset.elementid         ===
node.value
                        ? 'checked'
                        : node.status
        }

        #getRenderedCircles() {
                //
console.log(JSON.stringify([...this.graph.graph.values()]))

                return [...this.graph.graph.entries()].map(
                        (
                                // eslint-disable-next-line
                                [_, node]
                        ) => {
                                const status = this.#getNodeStatusForRender(node)
                                const x = (node.x ?? 0) + this.offsetX
                                const y = (node.y ?? 0) + this.offsetY

                                return `<g
                                                fixed="false"
                                                style="cursor: pointer;"
                                        >
                                                <circle
                                                        class="content--circle"
                                                        stroke-width="2"
                                                        fill="${NODE_COLORS[status]}"
                                                        stroke="black"
                                                        r="19"
                                                        data-elementId="${node.value}"
                                                        cx="${x}"
                                                        cy="${y}"
                                                ></circle>
                                                <text
                                                        class="content--text"
                                                        font-size="14"
                                                        dy=".35em"
                                                        text-anchor="middle"
                                                        stroke-width="1"
                                                        fill="black"
                                                        stroke="black"
                                                        data-elementId="${node.value}"
                                                        x="${x}"
                                                        y="${y}"
                                                        style="user-select: none"
                                                >
                                                        ${node.value}
                                                </text>
                                        </g>`
                        }
                )
        }

        #getLinesForRender() {
                return (
                        [...this.graph.graph.entries()]
                                // eslint-disable-next-line
                                .map(([_, node]) => {
                                        return [...node.edges].map(edge => {
```

```
                                        const         adjacentNodeX         =
edge.adjacentNode.x ?? 0
                                        const         adjacentNodeY         =
edge.adjacentNode.y ?? 0
                                        const nodeX = node.x ?? 0
                                        const nodeY = node.y ?? 0

                                        const  vectorOne  =  [adjacentNodeX  -
nodeX, adjacentNodeY - nodeY]
                                        const    vectorOneProtectionToX    =
[Math.abs(adjacentNodeX - nodeX), 0]

                                        const top =
                                              vectorOne[0]                    *
vectorOneProtectionToX[0] +
                                              vectorOne[1]                    *
vectorOneProtectionToX[1]
                                        const bottom =
                                              Math.sqrt(vectorOne[0]  **  2  +
vectorOne[1] ** 2) *
                                              Math.sqrt(
                                                    vectorOneProtectionToX[0]
** 2 + vectorOneProtectionToX[1] ** 2
                                              )

                                        const arrowRotateDeg = (Math.acos(top
/ bottom) * 180) / Math.PI
                                        const arrowRotateDegWithReflection =
                                              vectorOne[1]   <   0   ?   360   -
arrowRotateDeg : arrowRotateDeg

                                        const distanceFromCenter = [
                                              19                              *
Math.cos((arrowRotateDegWithReflection * Math.PI) / 180),
                                              19                              *
Math.sin((arrowRotateDegWithReflection * Math.PI) / 180)
                                        ]

                                        if (
                                              edge.status  ===  'no-direction'
&&
                                              this.graph.mode !== 'undirected'
                                        )
                                              return

                                        const color =
                                              edge.status  ===  'no-direction'
&& DEBUG
                                                    ? 'green'
                                                    : edge.type === 'default'
                                                      ? 'black'
                                                      : edge.type === 'back'
                                                        ? 'lightblue'
                                                        :    edge.type    ===
'cross'
                                                            ? 'lightgreen'
                                                            : 'lightpink'

                                        const arrow = `<path stroke="${color}"
fill="${color}" d="M -15 5.5 L 0 0 L -15 -5.5 Z" transform="translate (${
                                              adjacentNodeX  +  this.offsetX  -
distanceFromCenter[0]
                                        } ${
```

```
                                                adjacentNodeY + this.offsetY -
distanceFromCenter[1]
                                          })
rotate(${arrowRotateDegWithReflection})"></path>`

                                          const textPosition = {
                                                x:
                                                      (nodeX + adjacentNodeX) /
2 +
                                                      distanceFromCenter[0] +
                                                      this.offsetX,
                                                y:
                                                      (nodeY + adjacentNodeY) /
2 +
                                                      distanceFromCenter[1] +
                                                      this.offsetY
                                          }

                                          const text = `
                                                <text      x="${textPosition.x}"
y="${textPosition.y}"                style="stroke:white;                stroke-
width:0.6em">${edge.weight}</text>
                                                <text      x="${textPosition.x}"
y="${textPosition.y}" style="fill:black">${edge.weight}</text> `

                                          return `<g>
                                                <path
                                                class="content--edge"
                                                      d="M       ${nodeX       +
this.offsetX} ${nodeY + this.offsetY} L ${
                                                            adjacentNodeX      +
this.offsetX
                                                      }     ${adjacentNodeY     +
this.offsetY}"
                                                      fill="none"
                                                      stroke-width="2"
                                                      stroke="${color}"
                                                ></path>
                                                <path
                                                class="content--edge"
                                                d="M   ${nodeX   +   this.offsetX}
${nodeY + this.offsetY} L ${
                                                      adjacentNodeX            +
this.offsetX
                                                }        ${adjacentNodeY       +
this.offsetY}"
                                                      opacity="0"
                                                      fill="none"
                                                      stroke-width="30"
                                                      stroke="${color}"
                                                ></path>
                                                ${this.graph.mode === 'directed'
|| DEBUG ? arrow : ''}
                                                ${this.graph.weights  ?  text  :
''}
                                          </g>`
                                    })
                              })
                              .flat()
                  )
            }

            render() {
                  const ourNodes = this.#getRenderedCircles()
```

```
                const ourEdges = this.#getLinesForRender()

                document.querySelector<HTMLDivElement>('#content')!.innerHTML
= `
                        <div class="graph__wrapper">
                                <svg
                                width="100%"
                                height="100%"
                                preserveAspectRatio="none"
                                cursor="${this.pressedKeyCode   ===   'Space'   ?
'grabbing' : 'default'}"
                                >
                                        <g>
                                                <g>
                                                        ${ourEdges.join(' ')}
                                                </g>
                                                <g>
                                                        ${ourNodes.join(' ')}
                                                </g>
                                        </g>
                                </svg>
                        </div>
                `
        }

        initializeApp() {
                this.#initilizeUserEvents()
                this.#initilizeMenu()
        }

        #initilizeUserEvents() {
                document.addEventListener('mousedown', (e: MouseEvent) =>
                        this.onMouseDown(e)
                )
                document.addEventListener('mouseup', () => this.onMouseUp())
                document.addEventListener('mousemove', (e: MouseEvent) =>
                        this.onMouseMove(e)
                )

                document.addEventListener('contextmenu', (e: MouseEvent) =>
                        this.onContextMenu(e)
                )
                document.addEventListener('click',    (e:    MouseEvent)    =>
this.onClick(e))

                document.addEventListener('keydown', (e: KeyboardEvent) =>
                        this.onKeyDown(e)
                )
                document.addEventListener('keyup',   (e:   KeyboardEvent)   =>
this.onKeyUp(e))
        }

        localState:
                | {
                        opened: false
                  }
                | {
                        opened: true
                        algorithm: string
                        activeElement: HTMLElement
                  } = {
                opened: false
        }
```

```
initializeGraphForLB61() {
      this.graph.graph = this.graph.createGraph([
            { from: '1', to: '6', weight: 22, x: 500, y: 500 },
            { from: '1', to: '5', weight: 31, x: 500, y: 500 },
            { from: '1', to: '4', weight: 31, x: 500, y: 500 },
            { from: '1', to: '3', weight: 52, x: 500, y: 500 },
            {
                  from: '2',
                  to: '9',
                  weight: 37,
                  x: 494,
                  y: 216
            },
            {
                  from: '3',
                  to: '2',
                  weight: 52,
                  x: 984,
                  y: 336
            },
            {
                  from: '3',
                  to: '9',
                  weight: 89,
                  x: 984,
                  y: 336
            },
            {
                  from: '3',
                  to: '4',
                  weight: 52,
                  x: 984,
                  y: 336
            },
            {
                  from: '4',
                  to: '2',
                  weight: 13,
                  x: 668,
                  y: 346
            },
            {
                  from: '5',
                  to: '6',
                  weight: 65,
                  x: 123,
                  y: 333
            },
            {
                  from: '5',
                  to: '2',
                  weight: 73,
                  x: 123,
                  y: 333
            },
            {
                  from: '6',
                  to: '4',
                  weight: 68,
                  x: 396,
                  y: 338
            },
            {
                  from: '6',
```

```
                        to: '2',
                        weight: 40,
                        x: 396,
                        y: 338
                    },
                    { from: '7', to: '', weight: 18, x: 633, y: 6, status:
'default' },
                    {
                        from: '8',
                        to: '',
                        weight: 81,
                        x: 888,
                        y: 70
                    },
                    {
                        from: '9',
                        to: '',
                        weight: 60,
                        x: 800,
                        y: 214
                    }
                ])
                this.render()
            }

            initializeGraphForLB62() {
                this.graph.graph = this.graph.createGraph([
                    { from: '1', to: '6', weight: 22, x: 500, y: 500 },
                    { from: '1', to: '5', weight: -31, x: 500, y: 500 },
                    { from: '1', to: '4', weight: -31, x: 500, y: 500 },
                    { from: '1', to: '3', weight: 52, x: 500, y: 500 },
                    {
                        from: '2',
                        to: '9',
                        weight: 37,
                        x: 494,
                        y: 216
                    },
                    {
                        from: '3',
                        to: '2',
                        weight: -52,
                        x: 984,
                        y: 336
                    },
                    {
                        from: '3',
                        to: '9',
                        weight: 89,
                        x: 984,
                        y: 336
                    },
                    {
                        from: '3',
                        to: '4',
                        weight: -52,
                        x: 984,
                        y: 336
                    },
                    {
                        from: '4',
                        to: '2',
                        weight: 13,
                        x: 668,
```

```
                        y: 346
                    },
                    {
                        from: '5',
                        to: '6',
                        weight: -65,
                        x: 123,
                        y: 333
                    },
                    {
                        from: '5',
                        to: '2',
                        weight: 73,
                        x: 123,
                        y: 333
                    },
                    {
                        from: '6',
                        to: '4',
                        weight: 68,
                        x: 396,
                        y: 338
                    },
                    {
                        from: '6',
                        to: '2',
                        weight: -40,
                        x: 396,
                        y: 338
                    },
                    { from: '7', to: '', weight: 18, x: 633, y: 6, status:
'default' },
                    {
                        from: '8',
                        to: '',
                        weight: 81,
                        x: 888,
                        y: 70
                    },
                    {
                        from: '9',
                        to: '',
                        weight: 60,
                        x: 800,
                        y: 214
                    }
                ])
                this.render()
            }

            #initilizeMenu() {
                const mainMenu = document.querySelector('#main-menu')
                const panel = document.querySelector('#panel')
                const form = document.querySelector('#form')
                const  formHeading  =  document.querySelector('.panel__form-
heading')
                const  formCodeOutput  =  document.querySelector('#form-code-
output')

                mainMenu?.addEventListener('click', async e => {
                    if                    (!(e.target                    as
HTMLElement).className.includes('menu__link')) return
```

```
                        const      targetDataId      =      (e.target      as
HTMLElement).dataset.id

                        if (!targetDataId) {
                              return
                        }

                        if (targetDataId === 'bfs' || targetDataId === 'dfs') {
                              ;(e.target                                        as
HTMLElement).classList.add('menu__link--active')

                              const activeId = new Date().getTime()
                              this.algorithmActiveId = new Date().getTime()

                              this.#graphNodesStatusResetter(activeId)

                              if (targetDataId === 'bfs') {
                                    await this.bfsWrapper(activeId)
                              }

                              if (targetDataId === 'dfs') {
                                    await this.dfsWrapper(activeId)
                              }

                              ;(e.target                                        as
HTMLElement).classList.remove('menu__link--active')
                              this.#graphNodesStatusResetter(activeId)

                              return
                        }

                        if (targetDataId === 'reset') {
                              this.algorithmActiveId = -1


      this.#graphNodesStatusResetter(this.algorithmActiveId)

      this.#graphEdgesTypeResetter(this.algorithmActiveId)
                              return
                        }

                        if (targetDataId === 'mode') {
                              if (this.graph.mode === 'directed') {
                                    this.graph.mode = 'undirected'
                              } else {
                                    this.graph.mode = 'directed'
                              }

                              ;(e.target    as    HTMLElement).textContent    =
this.graph.mode[0]
                              this.render()
                              return
                        }

                        if (targetDataId === 'change_graph') {
                              if (this.lastGraph === 'default') {
                                    // lb61
                                    this.initializeGraphForLB61()
                                    ;(e.target  as  HTMLElement).textContent  =
'lb61'
                                    this.lastGraph = 'lb61'
                                    this.render()
                                    return
                              }
```

```
                        if (this.lastGraph === 'lb5') {
                              this.initializeGraph()
                              this.lastGraph = 'default'
                              ;(e.target as HTMLElement).textContent = 'd'
                              this.render()
                              return
                        }

                        if (this.lastGraph === 'lb61') {
                              this.initializeGraphForLB62()
                              this.lastGraph = 'lb62'
                              ;(e.target  as  HTMLElement).textContent  =
'lb62'
                              this.render()
                              return
                        }

                        if (this.lastGraph === 'lb62') {
                              this.initializeGraph()
                              this.lastGraph = 'default'
                              ;(e.target as HTMLElement).textContent = 'd'
                              this.render()
                              return
                        }

                        return
                  }

                  if (targetDataId === 'weight') {
                        if (this.graph.weights === true) {
                              this.graph.weights = false
                        } else {
                              this.graph.weights = true
                        }

                        ;(e.target    as    HTMLElement).textContent    =
this.graph.weights ? 'w' : 'nw'

                        this.render()
                        return
                  }

                  if (this.localState.opened) {
                        if (this.localState.algorithm === targetDataId) {
                              panel?.classList.remove('panel--opened')
                              ;(e.target                                    as
HTMLElement).classList.remove('menu__link--active')

                              this.localState = {
                                    opened: false
                              }
                        } else {
                              this.localState.algorithm = targetDataId
                              ;(e.target                                    as
HTMLElement).classList.add('menu__link--active')

      this.localState.activeElement.classList.remove('menu__link--active')

                              this.localState.activeElement = e.target as
HTMLElement
                        }
                  } else {
                        panel?.classList.add('panel--opened')
```

```
                              ;(e.target                                    as
HTMLElement).classList.add('menu__link--active')

                              this.localState = {
                                    opened: true,
                                    algorithm: targetDataId,
                                    activeElement: e.target as HTMLElement
                              }
                        }

                        if (formHeading) {
                              formHeading.textContent = targetDataId
                        }
                  })

                  form?.addEventListener('submit', async e => {
                        e.preventDefault()

                        formCodeOutput.textContent = ''

                        const  start  =  document.querySelector('#panel__form--
from')
                        const to = document.querySelector('#panel__form--to')

                        // @ts-expect-error TODO
                        const algorithm = this.localState.algorithm

                        if (algorithm === 'lb5third') {
                              const activeId = new Date().getTime()
                              this.algorithmActiveId = activeId

                              // this.#graphNodesStatusResetter(activeId)

                              this.lb5TaskThird(start?.value, to?.value)
                              // .then(async value => {
                              //    console.log(value)

                              //    this.#graphNodesStatusResetter(activeId)
                              // })

                              return
                        }

                        // @ts-expect-error TODO
                        const startNode = this.graph.graph.get(start.value)
                        // @ts-expect-error TODO
                        const endNode = this.graph.graph.get(to.value)

                        if (!startNode || !endNode) return

                        if (algorithm === 'lb5first') {
                              const activeId = new Date().getTime()
                              this.algorithmActiveId = activeId

                              this.#graphNodesStatusResetter(activeId)

                              startNode.status = 'done'

                              const maxLevel = Number(to?.value) || 2

                              this.lb5TaskOne(startNode,      [],      activeId,
maxLevel).then(async value => {
                                    console.log(value)
```

```
                        this.#graphNodesStatusResetter(activeId)
                    })
                }

                if (algorithm === 'lb5second') {
                    const activeId = new Date().getTime()
                    this.algorithmActiveId = activeId

                    this.#graphNodesStatusResetter(activeId)

                    startNode.status = 'done'

                    // const maxLevel = Number(to?.value) || 2

                    this.lb5TaskSecond(activeId).then(async value => {
                        console.log(value)

                        this.#graphNodesStatusResetter(activeId)
                    })
                }

                if (algorithm === 'lb6one') {
                    const activeId = new Date().getTime()
                    this.algorithmActiveId = activeId

                    this.#graphNodesStatusResetter(activeId)

                    startNode.status = 'done'
                    this.render()

                    const distances = await
this.bellmanFord(startNode)

                    if (distances) {
                        const result =
[...distances.entries()].reduce(
                            (acc, [node, distance]) => {
                                return {
                                    ...acc,
                                    [node.value]: distance
                                }
                            },
                            {}
                        )

                        formCodeOutput.textContent =
JSON.stringify(result, null, 2)
                    } else {
                        console.log('Paths not found')
                    }
                }

                if (algorithm === 'lb6two') {
                    const activeId = new Date().getTime()
                    this.algorithmActiveId = activeId

                    this.#graphNodesStatusResetter(activeId)

                    startNode.status = 'done'
                    endNode.status = 'done'

                    const result = await this.dijkstra(startNode,
endNode)
```

```
                        formCodeOutput.textContent = result
                              ?.map(item => item.value)
                              .join(' -> ')
                        // .then(async () => {
                        //    console.log('asdsd')
                        //    // await sleep(DELAY)

                        //    this.#graphNodesStatusResetter(activeId)
                        // })
                  }

                  if (algorithm === 'lb6three') {
                        const activeId = new Date().getTime()
                        this.algorithmActiveId = activeId

                        this.#graphNodesStatusResetter(activeId)

                        const                  nodes                  =
Array.from(this.graph.graph.values()).sort((a, b) => {
                              return Number(a.value) - Number(b.value)
                        })
                        console.log('%c●', 'color: #731d1d', nodes)

                        const result = await this.floydWarshall(nodes)

                        formCodeOutput.textContent = JSON.stringify(
                              result.map(item => {
                                    const tmp: string[] = []

                                    item.map(subItem => {
                                          let s = String(subItem)

                                          if (subItem === Infinity) {
                                                s = '-'
                                          }

                                          if (s.length < 2) {
                                                s = ' ' + s
                                          }

                                          if (s.length < 3 && s.length ===
2) {
                                                s = ' ' + s
                                          }

                                          tmp.push(s)
                                    })

                                    return tmp.join(',')
                              }),
                              null,
                              2
                        )
                        // .then(async () => {
                        //    console.log('asdsd')
                        //    // await sleep(DELAY)

                        //    this.#graphNodesStatusResetter(activeId)
                        // })
                  }

                  // if (algorithm === 'find-one-path') {
                  //    const path: Node[] = []
```

```
                        //    const activeId = new Date().getTime()
                        //    this.algorithmActiveId = activeId

                        //    this.#graphNodesStatusResetter(activeId)

                        //    startNode.status = 'done'
                        //    endNode.status = 'done'

                        //    this.findPath(startNode,   endNode,   [],   path,
activeId).then(async () => {
                        //          console.log('asdsd')
                        //          // await sleep(DELAY)

                        //          this.#graphNodesStatusResetter(activeId)
                        //    })
                        // }

                        // if (algorithm === 'find-all-paths') {
                        //    const path: Node[][] = []

                        //    const activeId = new Date().getTime()
                        //    this.algorithmActiveId = activeId

                        //    this.#graphNodesStatusResetter(activeId)

                        //    startNode.status = 'done'
                        //    endNode.status = 'done'

                        //    console.log(
                        //          await  this.findPathes(startNode,  endNode,
new Set(), path, activeId)
                        //    )

                        //    await sleep(DELAY)

                        //    this.#graphNodesStatusResetter(activeId)

                        //    console.log(path)
                        // }
                  })
            }
      }

      const app = new App()

      app.initializeApp()

      /*

      Ориентированный:
      - У нас есть направления по которому мы проходим

      Не ориентированый:
      - У нас нет направления и мы можем идти куда хотим

      Варианты:
      - Добавить в edges создание не 1 edge, а 2, только у 1 будет status -
standart, а у второй - no-direction
        А при создании из 2 в 1 - заменять у второй с no-direction на `standart`
           При удалении - изменять на no-direction

      */
```

```
src\pages\greedy\encodeAndDecode.ts
class TreeNode {
    letter: string | null
    weight: number
    right: TreeNode
    left: TreeNode

    constructor(letter: string | null, weight: number) {
        this.letter = letter
        this.weight = weight
    }
}

class EncodeAndDecode {
    encodeFromFileToFiles(textForEncode: string) {
        const fileText = textForEncode

        return this.encode(fileText)
    }

    decodeFromFilesToFile(textForEncode: string, treeJson: string) {
        const treeRoot = this.treeFromObject(JSON.parse(treeJson))

        const decoded = this.decode(textForEncode, treeRoot)

        return decoded
    }

    encode(text: string) {
        const lettersCount = this.lettersCountInText(text)

        const list = [...lettersCount.entries()].map(([letter,
weight]) => {
            return new TreeNode(letter, weight)
        })

        const huffmanBinaryTreeRoot = this.huffman(list)

        const codes = {}
        this.printCodesFromBinaryTree(huffmanBinaryTreeRoot, codes)

        const resultArray = Array.from(text).map(char => codes[char])

        const treeOutput = this.treeToJson(huffmanBinaryTreeRoot)

        return {
            tree: treeOutput,
            string: resultArray.join('')
        }
    }

    decode(encodedText: string, tree: TreeNode) {
        const result = []

        let node = tree
        for (let i = 0; i <= encodedText.length; i++) {
            node = encodedText.charAt(i) == '0' ? node.left :
node.right

            if (node.letter !== null) {
                result.push(node.letter)
                node = tree
            }
        }
```

```
            return result.join('')
        }

        // eslint-disable-next-line
        private treeFromObject(object: any) {
            const node = new TreeNode(object.letter, 0)

            if (object.letter === null) {
                node.right = this.treeFromObject(object.right)
                node.left = this.treeFromObject(object.left)
            }

            return node
        }

        private treeToJson(node: TreeNode) {
            const result = {
                letter: node.letter
            }

            if (node.letter === null) {
                result['left'] = this.treeToJson(node.left)
                result['right'] = this.treeToJson(node.right)
            }

            return result
        }

        private printCodesFromBinaryTree(
            node: TreeNode,
            result: Record<string, string>,
            c = ''
        ) {
            if (node.letter !== null) {
                result[node.letter] = c
                return
            }

            this.printCodesFromBinaryTree(node.right, result, c + '1')
            this.printCodesFromBinaryTree(node.left, result, c + '0')
        }

        private huffman(list: TreeNode[]) {
            while (list.length > 1) {
                list.sort((a, b) => {
                    return b.weight - a.weight
                })

                const right = list.pop()
                const left = list.pop()

                const newTreeNode = new TreeNode(null, left.weight +
right.weight)
                newTreeNode.left = left
                newTreeNode.right = right

                list.push(newTreeNode)
            }

            return list[0]
        }

        private lettersCountInText(text: string): Map<string, number> {
```

```
                const lettersCount = new Map<string, number>()

                for (const letter of text) {
                      lettersCount.set(letter, (lettersCount.get(letter) ?? 0)
+ 1)
                }

                return lettersCount
          }
    }

    export { EncodeAndDecode }
```

src\pages\greedy\main.ts

```
    import { BringWater } from './water'
    import { EncodeAndDecode } from './encodeAndDecode'

    const bringWater = new BringWater(0, [])

    const waterForm = document.querySelector('#water-form')
    const waterBarrel = document.querySelector('#water-barrel')
    const waterBuckets = document.querySelector('#water-buckets')
    const waterOutput = document.querySelector('#water-output')

    waterForm?.addEventListener('submit', e => {
          e.preventDefault()

          bringWater.buckets = waterBuckets.value.split(',').map(Number) ?? []
          bringWater.barrelCapacity = Number(waterBarrel.value)

          waterOutput.textContent = JSON.stringify(bringWater.result, null, 2)
    })

    const encodeAndDecode = new EncodeAndDecode()

    const haffmanForm = document.querySelector('#haffman-form')
    const haffmanInput = document.querySelector('#haffman-input')
    const haffmanEncoded = document.querySelector('#haffman-encoded')
    const haffmanTree = document.querySelector('#haffman-tree')
    const haffmanDecoded = document.querySelector('#haffman-decoded')

    haffmanForm?.addEventListener('submit', e => {
          e.preventDefault()

          const                                res                                =
encodeAndDecode.encodeFromFileToFiles(haffmanInput.value)
          haffmanEncoded.textContent = res.string
          haffmanTree.textContent = JSON.stringify(res.tree, null, 2)

          haffmanDecoded.textContent = encodeAndDecode.decodeFromFilesToFile(
                haffmanEncoded.textContent,
                haffmanTree.textContent
          )
    })
```

src\pages\greedy\water.ts

```
    /*
    Варіант No 3.
    На дачі стоїть велика діжка, яка вміщує задану кількість рідини.
    Хазяїн використовує її для поливу рослин, але не маючи
    централізованого водопостачання, має принести воду з річки. У його
    розпорядженні є відра заданого обсягу. Визначити, яку мінімальну
```

```
кількість відер води хазяїну потрібно принести з річки, щоб заповнити
діжку. Вважати, що кожне відро може бути принесене тільки повністю
заповненим, адже хазяїн не хоче носити зайвий вантаж. Результати
виводити, демонструючи кількість відер кожного обсягу.
*/

class BringWater {
     barrelCapacity = 0
     private _buckets: number[] = []

     constructor(barrelCapacity: number, buckets: number[]) {
          this.barrelCapacity = barrelCapacity
          this.buckets = buckets
     }

     get buckets() {
          return this._buckets
     }

     set buckets(newBuckets) {
          this._buckets = newBuckets.sort((a, b) => a - b)
     }

     get result() {
          let notFilledCapacity = this.barrelCapacity

          const usedBuckets = {}

          if (this.buckets.length === 0) {
               return 'Buckets must be!'
          }

          while (notFilledCapacity > 0) {
               for (let i = 0; i < this.buckets.length; i++) {
                    const bucket = this.buckets[i]

                    if (notFilledCapacity <= bucket || i ===
this.buckets.length - 1) {
                         if (!(bucket in usedBuckets)) {
                              usedBuckets[bucket] = 0
                         }

                         usedBuckets[bucket]++
                         notFilledCapacity -= bucket
                         break
                    }
               }
          }

          return usedBuckets
     }
}

export { BringWater }


src\pages\heap\Book.ts
/*
Книжки в бібліотеці характеризуються наступними даними:
- автор;
- назва;
- жанр;
- видавництво;
- рік публікації;
```

```
- кількість сторінок;
- загальна кількість екземплярів;
- кількість екземплярів у читачів.
*/

class Book {
     author: string
     title: string
     genre: string
     publisher: string
     publicationYear: number
     pageCount: number
     totalCopies: number
     copiesCheckedOut: number

     constructor(
           author: string,
           title: string,
           genre: string,
           publisher: string,
           publicationYear: number,
           pageCount: number,
           totalCopies: number,
           copiesCheckedOut: number
     ) {
           this.author = author
           this.title = title
           this.genre = genre
           this.publisher = publisher
           this.publicationYear = publicationYear
           this.pageCount = pageCount
           this.totalCopies = totalCopies
           this.copiesCheckedOut = copiesCheckedOut
     }

     valueOf() {
           return this.totalCopies - this.copiesCheckedOut
     }

     clone() {
           return new Book(
                 this.author,
                 this.title,
                 this.genre,
                 this.publisher,
                 this.publicationYear,
                 this.pageCount,
                 this.totalCopies,
                 this.copiesCheckedOut
           )
     }
}

export { Book }


src\pages\heap\doubly-linked-list.ts
class DoublyLinkedListItem<T> {
     previous: DoublyLinkedListItem<T> | null = null
     value: T
     next: DoublyLinkedListItem<T> | null = null

     constructor(
           value: T,
```

```
            previous: DoublyLinkedListItem<T> | null = null,
            next: DoublyLinkedListItem<T> | null = null
        ) {
            this.value = value
            this.previous = previous
            this.next = next
        }
    }

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

    class DoublyLinkedList<T> {
        head: DoublyLinkedListItem<T>

        constructor(value: T) {
            this.head = new DoublyLinkedListItem(value)
        }

        addAsHead(value: T) {
            const newHead = new DoublyLinkedListItem(value, null,
this.head)
            this.head.previous = newHead

            this.head = newHead
        }

        addAfter(node: DoublyLinkedListItem<T>, value: T) {
            const newNode = new DoublyLinkedListItem(value, node,
node.next)

            node.next = newNode

            if (newNode.next) {
                newNode.next.previous = newNode
            }

            return this
        }

        find(value: T): DoublyLinkedListItem<T> | null {
            let current: DoublyLinkedListItem<T> | null = this.head

            while (current !== null) {
                if (current.value === value) {
                    return current
                }

                current = current.next
            }

            return null
        }

        delete(node: DoublyLinkedListItem<T>) {
            if (node.previous === null && node.next === null) return
```

```
                if (node === this.head) {
                      if (this.head.next === null) {
                            throw new Error('List must have minimum one node -
head')
                      }

                      if (this.head.next !== null) {
                            this.head = this.head.next
                            this.head.previous = null

                            node.next = null
                            return
                      }
                }

                if (node.previous === null) {
                      console.log('What? How?!')
                      return
                }

                node.previous.next = node.next

                if (node.next) {
                      node.next.previous = node.previous
                }
          }

          getPrint(valueToString: (value: T) => string) {
                const result: string[] = []

                let current: DoublyLinkedListItem<T> | null = this.head

                while (current !== null) {
                      result.push(valueToString(current.value))

                      current = current.next
                }

                return result.join(' ⇄ ')
          }
    }

    export { DoublyLinkedList }


    src\pages\heap\heap.ts
    /*
    Клас, що реалізує купу (чергу з пріоритетами), має
    дозволяти виконувати наступні операції на основі окремих методів:
    [x] вставлення елементу
    [x] побудова купи з невпорядкованого масиву
    [x] видалення елементу
    [x] сортування елементів
    [ ] виведення елементів на екран
    */

    import { Book } from './Book'
    import { logHeap } from './helpers/logHeap'

    class Heap<T> {
          private heap: T[] = []

          add(element: T) {
                const index = this.heap.push(element)
```

```
            this.siftup(index)

            return this
        }

        fromArray(array: T[]) {
            this.heap = array

            this.heap.forEach((_, index) => {
                this.siftdown(this.heap, this.heap.length - 1 - index)
            })

            return this
        }

        extractTop(heap: T[] = this.heap) {
            const last = heap.length - 1

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

            const element = heap.pop() as T

            this.siftdown(heap, 0)

            return element
        }

        getSortedArray() {
            const heapCopy = this.heap.map(item => {
                if (item instanceof Book) {
                    return item.clone()
                }

                return { ...item }
            }) as T[]

            const result: T[] = []

            while (heapCopy.length >= 1) {
                result.push(this.extractTop(heapCopy))
            }

            return result
        }

        getPrint(toString: (value: T) => string) {
            return logHeap<T>(this.heap, toString)
        }

        private siftup(i: number) {
            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)
            }
        }

        private siftdown(heap: T[], i: number) {
```

```
                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 default Heap
```

src\pages\heap\HeapSort.ts

```
import Heap from './heap'

class HeapSort<T> {
        heap: Heap<T>

        constructor(heap: Heap<T>) {
                this.heap = heap
        }

        getSorted() {
                return this.heap.getSortedArray()
        }
}

export { HeapSort }
```

src\pages\heap\main.ts

```
import { HeapSort } from './HeapSort'
// import { DoublyLinkedList } from './doubly-linked-list'
import Heap from './heap'
import { Book } from './Book'

// const list = new DoublyLinkedList<number>(1)

// list.addAfter(list.head, 2)
// list.addAfter(list.head, 6)
// list.addAfter(list.head, 7)
// list.addAfter(list.head, 9)

// console.log(list.getPrint((value: number) => String(value)))

const books = [
        new Book(
                'Camille Predovic',
                'Armenian Gampr dog',
```

```
      'Electronics',
      'weepy-status.org',
      2022,
      213,
      11,
      5
),
new Book(
      'Chris Von',
      'Black Norwegian Elkhound',
      'Automotive',
      'illiterate-antigen.biz',
      2001,
      308,
      24,
      6
),
new Book(
      'Alonzo Fahey II',
      'Pekingese',
      'Outdoors',
      'concrete-dashboard.org',
      2021,
      399,
      50,
      43
),
new Book(
      'Fred Buckridge',
      'Fila Brasileiro',
      'Outdoors',
      'glamorous-relative.org',
      2011,
      208,
      11,
      7
),
new Book(
      'Colin O`Connell',
      'Armant',
      'Beauty',
      'graceful-territory.com',
      2018,
      365,
      38,
      30
),
new Book(
      'Shelly Greenfelder',
      'Hygen Hound',
      'Games',
      'quirky-mother-in-law.com',
      2010,
      355,
      24,
      6
),
new Book(
      'Margarita Franecki Jr.',
      'Bracco Italiano',
      'Home',
      'direct-lymphocyte.name',
      2021,
      302,
```

```
                25,
                12
        ),
        new Book(
                'Margaret Hills',
                'Montenegrin Mountain Hound',
                'Baby',
                'attached-wake.name',
                2002,
                423,
                41,
                33
        ),
        new Book(
                'Julio Nikolaus',
                'Silky Terrier',
                'Electronics',
                'ragged-jelly.name',
                2006,
                321,
                46,
                37
        ),
        new Book(
                'Alan Gibson I',
                'Lancashire Heeler',
                'Sports',
                'definite-garbage.com',
                2006,
                356,
                44,
                4
        )
]

const heapBook = new Heap<Book>().fromArray(books)

const bookToString = (value: Book, index?: number) => {
        return `${index !== undefined ? index + ' | ' : ''}${value.author}
${
                value.title
        }: ${value.totalCopies - value.copiesCheckedOut} | ${
                value.copiesCheckedOut
        }/${value.totalCopies}`
}

/*
– програмного модуля, який реалізує графічний інтерфейс
відповідної вкладки і дозволяє додавати нові елементи до купи на
основі полів, що відповідають індивідуальному завданню, та на
основі підключення файлів з масивами даних, вилучати існуючі
елементи, виконувати пірамідальне сортування та два інші алгоритми
сортування, визначені індивідуальним завданням, з виведенням
результатів наочним способом (отриманого порядку елементів та часу,
витраченого на сортування).
*/

function renderHeap() {
        heapOutput.textContent = heapBook.getPrint(bookToString)

        console.log(heapBook.getPrint(bookToString))
}

function removeTop() {
```

```
    const el = heapBook.extractTop()

    alert(JSON.stringify(el))
    renderHeap()
    // heapOutputTopElement.textContent = el
}

function addElement() {
    const NAMES = [
        'Bartell - Harris',
        'Schoen Inc',
        'Labadie - Rodriguez',
        'Weimann LLC',
        'Veum - Tillman',
        'Willms Inc',
        'Heller, Deckow and Funk',
        'Buckridge, Gutmann and Gaylord',
        'Boehm and Sons',
        'Rempel - Bruen',
        'Boyer, Wisoky and Altenwerth',
        'Steuber, Kovacek and Huels',
        'Ruecker, Jacobs and Daniel',
        'Abbott, Gutkowski and Waelchi',
        'Cartwright - Kuhlman',
        'Maggio - Zboncak',
        'Waelchi Group',
        'Hilll - Bode',
        'Marks - Stroman'
    ]

    const TITLES = [
        'Virtual non-volatile toolset',
        'Cloned neutral functionalities',
        'Digitized cohesive flexibility',
        'Face to face systemic utilisation',
        'Operative real-time application',
        'Vision-oriented intermediate collaboration',
        'Versatile modular circuit',
        'Switchable fault-tolerant conglomeration',
        'Implemented methodical matrices',
        'Open-architected bi-directional data-warehouse',
        'Inverse intangible conglomeration',
        'Sharable intangible migration',
        'Re-contextualized system-worthy adapter',
        'Diverse upward-trending core',
        'Streamlined well-modulated framework',
        'Devolved background standardization',
        'Advanced content-based time-frame',
        'Cross-platform local groupware',
        'User-centric foreground middleware'
    ]

    const GENRE = [
        'motivating',
        'system-worthy',
        'maximized',
        'client-server',
        'systematic',
        'fresh-thinking',
        'bi-directional',
        'multimedia',
        'global',
        'tertiary',
        'didactic',
```

```
                'leading edge',
                'client-driven',
                'empowering',
                'discrete',
                'user-facing',
                'non-volatile',
                'actuating',
                'impactful'
            ]

            heapBook.add(
                new Book(
                    NAMES[Math.floor(Math.random() * NAMES.length)],
                    TITLES[Math.floor(Math.random() * TITLES.length)],
                    GENRE[Math.floor(Math.random() * GENRE.length)],
                    'SITE.com',
                    Math.abs(Math.floor(Math.random() * 5000)),
                    Math.abs(Math.floor(Math.random() * 1000)),
                    Math.abs(Math.floor(Math.random() * 50)),
                    Math.abs(Math.floor(Math.random() * 50))
                )
            )

            renderHeap()
        }

        function sort() {
            const heapSort = new HeapSort<Book>(heapBook)

            const a = heapSort.getSorted()

            let summ = 0
            const message = [
                'Визначити книжки, кількість наявних екземплярів яких у
бібліотеці в поточний момент входить у перші 50 %:\n'
            ]

            a.slice(0, a.length / 2).map(item => {
                // Визначити книжки, кількість наявних екземплярів яких у
бібліотеці в поточний момент входить у перші 50 %.
                message.push(bookToString(item))

                summ += Number(item)
            })

            // Обчислити сумарну кількість наявних екземплярів таких книжок.
            message.push('\nSumm: ' + summ)

            alert(message.join('\n'))
        }

        const heapOutput = document.querySelector('#output')
        const heapOutputTopElement = document.querySelector('#output-element')

        document.querySelector('#extract-top')?.addEventListener('click', () =>
{
            removeTop()
        })

        document.querySelector('#add-element')?.addEventListener('click', () =>
{
            addElement()
        })
```

```
document.querySelector('#sort')?.addEventListener('click', () => {
    sort()
})

renderHeap()
```

src\pages\heap\helpers\logHeap.ts

```
type TreeNode<T = string> = {
    name: T
    children?: Array<TreeNode>
}

function logTree(
    tree: TreeNode<string> | TreeNode<string>[],
    level = 0,
    parentPre = '',
    treeStr = ''
) {
    if (!Array.isArray(tree)) {
        const children = tree['children']

        treeStr = `${tree['name']}\n`

        if (children) {
            treeStr += logTree(children, level + 1)
        }

        return treeStr
    }

    tree.forEach((child, index) => {
        const hasNext = tree[index + 1] ? true : false
        const children = child['children']

        treeStr += `${parentPre}${hasNext ? '├' : '└'}── ${child['name']}\n`

        if (children) {
            treeStr += logTree(
                children,
                level + 1,
                `${parentPre}${hasNext ? '│' : ' '}   `
            )
        }
    })

    return treeStr
}

function heapToTree<T>(
    heap: T[],
    toString: (value: T) => string,
    index: number = 0
): TreeNode<string> | null {
    if (index >= heap.length) return null

    const left = heapToTree(heap, toString, index * 2 + 1)
    const right = heapToTree(heap, toString, index * 2 + 2)

    return {
        name: toString(heap[index]),
        children: [left, right].filter(item => item !== null) as TreeNode<string>[]
```

```
    }
}

function logHeap<T>(heap: T[], toString: (value: T) => string) {
    return logTree(heapToTree(heap, toString) as TreeNode)
}

export { logHeap }


src\pages\tree-and-hash\AVLTree.class.ts
import { LogTreeNode } from './helper/logTree'

/*
- [x] створення порожнього дерева
- [x] відображення структури дерева
- [x] пошук у дереві
- [x] вставлення ключа
- [x] видалення ключа
*/

class TreeNode<T> {
    key: number
    value: T
    left: TreeNode<T> | null = null
    right: TreeNode<T> | null = null

    height = 0

    constructor(key: number, value: T) {
        this.key = key
        this.value = value
    }

    insert(node: TreeNode<T>, key: number, value: T) {
        if (key < node.key) {
            if (node.left === null) {
                node.left = new TreeNode<T>(key, value)
            } else {
                this.insert(node.left, key, value)
            }
        } else if (node.right === null) {
            node.right = new TreeNode(key, value)
        } else {
            this.insert(node.right, key, value)
        }

        this.updateHeight(node)
        this.balance(node)
    }

    search(node: TreeNode<T>, key: number) {
        if (node === null) return null
        if (node.key === key) return node
        return key < node.key
            ? this.search(node.left, key)
            : this.search(node.right, key)
    }

    getMin(node: TreeNode<T>): TreeNode<T> | null {
        if (node === null) return null
        if (node.left === null) return node

        return this.getMin(node.left)
```

```
        }

        getMax(node: TreeNode<T>): TreeNode<T> | null {
            if (node === null) return null
            if (node.right === null) return node

            return this.getMax(node.right)
        }

        delete(node: TreeNode<T>, key: number) {
            if (node === null) return null
            if (key < node.key) node.left = this.delete(node.left, key)
            else if (key > node.key) node.right = this.delete(node.right,
key)
            else {
                if (node.left === null || node.right === null) {
                    node = node.left === null ? node.right : node.left
                } else {
                    const maxInLeft = this.getMax(node.left)

                    node.key = maxInLeft.key
                    node.value = maxInLeft.value

                    node.left = this.delete(node.left, maxInLeft.key)
                }
            }

            if (node !== null) {
                this.updateHeight(node)
                this.balance(node)
            }

            return node
        }

        treeForOutput(node: TreeNode<T>) {
            if (node === null) return

            const left = this.treeForOutput(node.left)
            const right = this.treeForOutput(node.right)

            const EMPTY_NAME = ''
            let children = [left, right].map(item => {
                if (item !== undefined) return item

                return {
                    name: EMPTY_NAME
                }
            })

            if (children[0].name === EMPTY_NAME && children[1].name ===
EMPTY_NAME) {
                children = []
            }

            const result: LogTreeNode = {
                name: String(node.key)
            }

            if (children.length > 0) {
                result.children = children
            }

            return result
```

```
        }

        // Симетричний обхід
        inorderTreeWalkPrint(node: TreeNode<T>) {
              if (node === null) return

              const result = {}

              const left = this.inorderTreeWalkPrint(node.left)
              const right = this.inorderTreeWalkPrint(node.right)

              if (left) result['left'] = left
              result['root'] = node.value
              if (right) result['right'] = right

              return result
        }

        // Зворотній обхід
        preorderTreeWalkPrint(node: TreeNode<T>) {
              if (node === null) return
              this.preorderTreeWalkPrint(node.left)
              this.preorderTreeWalkPrint(node.right)
              console.log(node.value)
        }

        // Прямий обхід
        postorderTreeWalkPrint(node: TreeNode<T>) {
              if (node === null) return
              console.log(node.value)
              this.postorderTreeWalkPrint(node.left)
              this.postorderTreeWalkPrint(node.right)
        }

        updateHeight(node: TreeNode<T>) {
              node.height =
                    Math.max(this.getHeight(node.left),
this.getHeight(node.right)) + 1
        }

        getHeight(node: TreeNode<T>) {
              return node === null ? -1 : node.height
        }

        getBalance(node: TreeNode<T>) {
              return node === null
                    ? 0
                    : this.getHeight(node.right) - this.getHeight(node.left)
        }

        swap(a: TreeNode<T>, b: TreeNode<T>) {
              const a_key = a.key
              a.key = b.key
              b.key = a_key

              const a_value = a.value
              a.value = b.value
              b.value = a_value
        }

        rightRotate(node: TreeNode<T>) {
              this.swap(node, node.left)
              const buffer = node.right
              node.right = node.left
```

```
                node.left = node.right.left
                node.right.left = node.right.right
                node.right.right = buffer

                this.updateHeight(node.right)
                this.updateHeight(node)
        }

        leftRotate(node: TreeNode<T>) {
                this.swap(node, node.right)
                const buffer = node.left

                node.left = node.right
                node.right = node.left.right
                node.left.right = node.left.left
                node.left.left = buffer

                this.updateHeight(node.left)
                this.updateHeight(node)
        }

        balance(node: TreeNode<T>) {
                const balance = this.getBalance(node)
                if (balance === -2) {
                        if (this.getBalance(node.left) === 1)
this.leftRotate(node.left)
                        this.rightRotate(node)
                } else if (balance === 2) {
                        if (this.getBalance(node.right) === -1)
this.rightRotate(node.right)
                        this.leftRotate(node)
                }
        }
}

class AVLTree<T> {
        headNode: TreeNode<T> | null = null

        insert(key: number, value: T) {
                if (this.headNode === null) {
                        this.headNode = new TreeNode<T>(key, value)
                        return
                }

                this.headNode.insert(this.headNode, key, value)
        }

        search(key: number) {
                return this.headNode.search(this.headNode, key)?.value
        }

        delete(key: number) {
                return this.headNode.delete(this.headNode, key)
        }

        showStructure(type: 'inorder' | 'preorder' | 'postorder' = 'inorder')
{
                if (type === 'inorder')
                        return
this.headNode.inorderTreeWalkPrint(this.headNode)
                if (type === 'preorder')
                        return
this.headNode.preorderTreeWalkPrint(this.headNode)
                if (type === 'postorder')
```

```
                        return
this.headNode.postorderTreeWalkPrint(this.headNode)
            }

            treeForOutput() {
                  return this.headNode.treeForOutput(this.headNode)
            }
      }

      export default AVLTree


      src\pages\tree-and-hash\HashTable.class.ts
      /*
      - [x] Вставлення елементу
      - [x] Видалення елементу
      - [x] Пошук елементу
      - [x] Відображення структури геш-таблиці на основі використання
параметрів, обраних у відповідності з варіантом індивідуального завдання з п.
2.3.4.
      */
      type KeyType = number | string

      class List {
            private key: KeyType
            private value: unknown
            public next: List | null

            addOrUpdate(key: KeyType, value: unknown) {
                  if (this.key == undefined || this.value == undefined ||
this.key === key) {
                        this.key = key
                        this.value = value
                        return this
                  }

                  if (this.next) {
                        this.next.addOrUpdate(key, value)
                        return this
                  }

                  this.next = new List().addOrUpdate(key, value)
            }

            get(key: KeyType): unknown | null {
                  if (this.key === key) {
                        return this.value
                  }

                  if (!this.next) return null

                  return this.next.get(key)
            }

            remove(key: KeyType): List | null {
                  const dummy = new List()
                  dummy.next = this.next

                  let prev = dummy

                  let current = new List()

                  if (this.key === key) {
                        prev.next = current.next
```

```
                        current = current.next
                    } else {
                        prev = current
                        current = current.next
                    }

                    while (current) {
                        if (current.key === key) {
                            prev.next = current.next

                            current = current.next
                        } else {
                            prev = current
                            current = current.next
                        }
                    }

                    return dummy.next
                }

                __getInfoWithRemove(): null | [KeyType, unknown, List | null] {
                    this.remove(this.key)

                    return [this.key, this.value, this.next]
                }

                getShow(): string[] {
                    if (this.next == null) return [String(this)]

                    return [String(this), ...this.next.getShow()]
                }

                toString() {
                    return `${this.key}: ${this.value}`
                }
            }

            class HashTable {
                private readonly sizes: number[] = [
                    5, 11, 23, 47, 97, 193, 389, 769, 1543, 3072, 3079, 12289,
24593, 49157,
                    98317, 196613, 393241, 786433, 1572869, 3145739, 6291469,
12582917,
                    25165843, 50331653, 100663319, 201326611, 402653189,
805306457, 1610612736,
                    2147483629
                ]
                private sizes_index: number = 0
                private factor = 0.75
                private count: number = 0

                private values: List[] = Array(this.sizes[this.sizes_index])

                addOrUpdate(key: KeyType, value: unknown) {
                    if (this.checkMemory()) {
                        this.addMemory()
                    }

                    const index = this.getIndex(key)

                    if (this.values[index] == null) {
                        this.values[index] = new List()
                        this.count++
                    }
```

```
                this.values[index].addOrUpdate(key, value)
        }

        get(key: KeyType) {
                const index = this.getIndex(key)

                if (this.values[index] == null) {
                        return null
                }

                return this.values[index].get(key)
        }

        remove(key: KeyType) {
                const index = this.getIndex(key)

                if (this.values[index] == null) {
                        return null
                }

                const removeResult = this.values[index].remove(key)

                if (removeResult !== null) {
                        this.values[index] = removeResult
                        return null
                }

                this.values[index] = null
        }

        show() {
                return this.values
                        .map((item, index) => {
                                if (item) return { index: index, value:
item.getShow().join(' => ') }
                        })
                        .filter(item => item != undefined)
        }

        private getHash(key: KeyType): number {
                const value = typeof key === 'number' ? key :
this.stringToNumber(key)

                const w = 10
                const A = Math.sqrt(5) / 2 ** w
                const M = 2 ** 16

                const resultOfMultiple = value * A

                return Math.ceil(M * (resultOfMultiple % 1))
        }

        private stringToNumber(key: string) {
                let hash = 0

                for (let i = 0; i < key.length; i++) {
                        hash = (hash << 5) - hash + key.charCodeAt(i)
                }

                return Math.abs(hash)
        }

        private getIndex(key: KeyType): number {
```

```
                const hash = this.getHash(key)

                return hash % this.sizes[this.sizes_index]
          }

          private checkMemory() {
                return    this.count    /    this.sizes[this.sizes_index]    >=
this.factor
          }

          private addMemory() {
                this.count = 0
                const oldValues = [...this.values]

                this.sizes_index += 1

                this.values = Array(this.sizes[this.sizes_index])

                for (let i = 0; i < this.sizes[this.sizes_index - 1]; i++) {
                      if (!oldValues[i]) continue

                      let node = oldValues[i].__getInfoWithRemove()

                      while (node != null) {
                            this.addOrUpdate(node[0], node[1])

                            if (node[2] == null) {
                                  break
                            }

                            node = node[2].__getInfoWithRemove()
                      }
                }
          }
    }

    export default HashTable


    src\pages\tree-and-hash\main.ts
    import AVLTree from './AVLTree.class'
    import HashTable from './HashTable.class'
    import { logTree } from './helper/logTree'

    console.log('Var 8 => Task 1 B)')
    /*
    B. Створити геш-таблицю, що використовує метод ланцюжків
    для розв’язання колізій та геш-функцію множення. Геш-таблицю
    заповнити на основі виділення інформації з текстового файлу, в якому
    містяться прізвища, ім’я і по батькові співробітників фірми та займані
    ними посади. Визначити посаду заданого співробітника.
    */

    const hashData = [
          ['Loy Graham Zboncak', 'Directives'],
          ['Roy Breitenberg Runte', 'Accountability'],
          ['Erika Emard Feest', 'Security'],
          ['Vance Flatley Thiel', 'Research'],
          ['Mallory Hoppe O`Hara', 'Applications'],
          ['Dulce Douglas Boyer', 'Interactions'],
          ['Dedrick Jerde Kozey', 'Accounts'],
          ['Hudson Langosh Mayert', 'Applications'],
          ['Georgianna Bergstrom VonRueden', 'Solutions'],
          ['Marjorie Rolfson Bashirian', 'Integration'],
```

```
    ['Mitchell O`Keefe Shanahan', 'Branding'],
    ['Filiberto Gottlieb Marquardt', 'Accountability'],
    ['Murphy Cassin Franey', 'Configuration'],
    ['Javier Kilback Rodriguez', 'Branding'],
    ['Kayley Powlowski Kuphal', 'Assurance'],
    ['Liliana Johnston Ebert', 'Metrics'],
    ['Everette Little Cartwright', 'Accounts'],
    ['Otilia Fadel Spinka', 'Implementation'],
    ['Watson Schuppe Lowe', 'Web'],
    ['Duane Emmerich Rohan', 'Paradigm'],
    ['Linwood Huel VonRueden', 'Marketing'],
    ['Toni Johns Wiegand', 'Accounts'],
    ['Taylor Kreiger Kihn', 'Functionality']
]

const hashTableOutput = document.querySelector('#hash-table')
const hashOutput = document.querySelector('#hash-output')
const hashAdd = document.querySelector('#hash-add')

const hashTable = new HashTable()

hashData.map(([name, job], index) => {
    hashTable.addOrUpdate(`${index} - ${name}`, job)
})

function renderData() {
    hashTableOutput.textContent = JSON.stringify(
        hashData.map((item, index) => `${item[0]}: ${item[1]}`),
        null,
        2
    )
}

function renderHash() {
    hashOutput.textContent = JSON.stringify(
        hashTable.show().map(item => `${item.index} | ${item.value}`),
        null,
        2
    )
}

/*
програмного модуля, що реалізує графічний інтерфейс
відповідної вкладки і дозволяє виконувати формування геш-таблиці та
B-дерева, додавання і видалення елементів, пошук, оброблення
результатів, виведення їх у відповідні поля для виконання
індивідуального завдання, при цьому розв’язуючи одне з завдань за
допомогою обох структур даних.
*/

renderData()
renderHash()

hashAdd?.addEventListener('click', () => {
    addHash()
})

function addHash() {
    const TITLES = [
        'Virtual non-volatile toolset',
        'Cloned neutral functionalities',
        'Digitized cohesive flexibility',
        'Face to face systemic utilisation',
        'Operative real-time application',
```

```
            'Vision-oriented intermediate collaboration',
            'Versatile modular circuit',
            'Switchable fault-tolerant conglomeration',
            'Implemented methodical matrices',
            'Open-architected bi-directional data-warehouse',
            'Inverse intangible conglomeration',
            'Sharable intangible migration',
            'Re-contextualized system-worthy adapter',
            'Diverse upward-trending core',
            'Streamlined well-modulated framework',
            'Devolved background standardization',
            'Advanced content-based time-frame',
            'Cross-platform local groupware',
            'User-centric foreground middleware'
        ]

        const GENRE = [
            'motivating',
            'system-worthy',
            'maximized',
            'client-server',
            'systematic',
            'fresh-thinking',
            'bi-directional',
            'multimedia',
            'global',
            'tertiary',
            'didactic',
            'leading edge',
            'client-driven',
            'empowering',
            'discrete',
            'user-facing',
            'non-volatile',
            'actuating',
            'impactful'
        ]

        const data = [
            `${TITLES[Math.floor(Math.random() * TITLES.length)]}`,
            GENRE[Math.floor(Math.random() * GENRE.length)]
        ]

        hashData.push(data)

        hashTable.addOrUpdate(data[0], data[1])

        renderHash()
        renderData()
    }

    // hashTable.addOrUpdate(`24 - Tara Cremin Skiles`, 'Accountability')

    // console.log(
    //   'How save in HashTable after add "24 - Tara Cremin Skiles: Accountability" people:'
    // )
    // console.table(hashTable.show())
    // console.log()

    // console.log("Search '13 - Javier Kilback Rodriguez' value: ")
    // console.log('value = ', hashTable.get('13 - Javier Kilback Rodriguez'))

    // console.log('\n ====================\n')
```

```
// console.log('Var 8 => Task 2 Б)')

// /*
//   2 Б) Дані про власників автомобілів включають ідентифікаційний
//   номер транспортного засобу, дату реєстрації та власника (прізвище,
//   ім’я, по батькові). Сформувати дерево з інформації про власників
//   автомобілів. Реалізувати пошук інформації про автомобіль за заданим
//   ідентифікаційним номером транспортного засобу, визначення осіб, які
//   володіють більше ніж одним автомобілем.
// */

class CarOwner {
     id: number
     registerDate: Date
     owner: string

     constructor(id: number, registerDate: Date, owner: string) {
          this.id = id
          this.registerDate = registerDate
          this.owner = owner
     }
}

const carOwners = [
     new CarOwner(67324, new Date('Sun Jan 06 2075 23:37:10'), 'Alexandra
Becker'),
     new CarOwner(29497, new Date('Tue May 03 2016 23:40:18'), 'Mr. Lela
Kessler'),
     new CarOwner(22486, new Date('Tue Apr 28 2020 02:28:00'), 'Roosevelt
Crooks'),
     new CarOwner(85849, new Date('Tue Jan 13 2054 10:55:12'), 'Cory
Schowalter'),
     new CarOwner(74389, new Date('Sat Aug 05 2017 19:04:01'), 'Wendell
Hessel'),
     new CarOwner(44563, new Date('Tue Oct 05 2094 03:34:48'), 'Joe
Lesch'),
     new CarOwner(61297, new Date('Fri Dec 21 2012 16:56:12'), 'Karla
Simonis'),
     new CarOwner(53376, new Date('Mon Nov 05 2074 12:16:53'), 'Marty
Beahan'),
     new CarOwner(5613, new Date('Sat Jan 26 2069 01:07:46'), 'Kim
Lockman'),
     new CarOwner(15435, new Date('Sun Dec 17 1995 18:56:09'), 'Mr. Victor
Kunze'),
     new CarOwner(15673, new Date('Wed Sep 17 2053 01:54:01'), 'Arturo
Robel IV'),
     new CarOwner(63325, new Date('Fri Sep 02 2033 01:25:31'), 'Ms. Donna
Kessler')
]

const treeOutput = document.querySelector('#tree-output')

const binaryTree = new AVLTree<CarOwner>()

carOwners.map(item => {
     binaryTree.insert(item.id, item)
})

function treeRender() {
     treeOutput.textContent = logTree(binaryTree.treeForOutput())
}

treeRender()
```

```
document.querySelector('#tree-delete')?.addEventListener('click', () =>
{
    binaryTree.delete(binaryTree.headNode?.value)

    treeRender()
})

document.querySelector('#tree-add')?.addEventListener('click', () => {
    const id = Math.floor(Math.random() * 500)

    const GENRE = [
        'motivating',
        'system-worthy',
        'maximized',
        'client-server',
        'systematic',
        'fresh-thinking',
        'bi-directional',
        'multimedia',
        'global',
        'tertiary',
        'didactic',
        'leading edge',
        'client-driven',
        'empowering',
        'discrete',
        'user-facing',
        'non-volatile',
        'actuating',
        'impactful'
    ]

    binaryTree.insert(
        id,
        new CarOwner(
            id,
            new Date(),
            GENRE[Math.floor(Math.random() * GENRE.length)]
        )
    )

    treeRender()
})

// console.log('Tree structure after delete 4 items:')
// console.log(logTree(binaryTree.treeForOutput()))

// console.log("Find 5613, 61297 and 44563 id's:")
// console.table([
//   binaryTree.search(5613),
//   binaryTree.search(61297),
//   binaryTree.search(44563)
// ])


src\pages\tree-and-hash\helper\logTree.ts
export type LogTreeNode = {
    name: string
    children?: Array<LogTreeNode>
}

function logTree(
    tree: LogTreeNode | LogTreeNode[],
    level = 0,
```

```
            parentPre = '',
            treeStr = ''
      ) {
            if (!Array.isArray(tree)) {
                  const children = tree['children']

                  treeStr = `${tree['name']}\n`

                  if (children) {
                        treeStr += logTree(children, level + 1)
                  }

                  return treeStr
            }

            tree.forEach((child, index) => {
                  const hasNext = tree[index + 1] ? true : false
                  const children = child['children']

                  treeStr   +=   `${parentPre}${hasNext   ?   '├'   :   '└'}──
${child['name']}\n`

                  if (children) {
                        treeStr += logTree(
                              children,
                              level + 1,
                              `${parentPre}${hasNext ? '│' : ' '}   `
                        )
                  }
            })

            return treeStr
      }

      export { logTree }
```