МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Криворізький національний університет

Кафедра моделювання та програмного забезпечення

ЛАБОРАТОРНА РОБОТА №2

З дисципліни «Бази даних»

Тема: «Розробка системи управління нереляційними базами даних.

Об`єктний підхід.»

Виконав студент групи ІПЗ-21-2

Губарєв Р.В.

Перевірив викладач

Білашенко С.В.

Кривий Ріг

2023

1. **Основні відомості про структури даних та основні алгоритми обробки дерев.**

**Двійкове дерево** – це скінченна множина вершин, яка або порожня, або складається з кореня з двома окремими двійковими деревами, які називають лівим і правим піддеревом кореня.

Процес обходу розбивається на три частини: відвідання кореня, обхід лівого піддерева та обхід правого піддерева. Різні обходи відрізняються порядком виконання цих кроків.

1. **Основні відомості про принципи організації баз даних: призначення, структури, методи керування**

База даних – це засіб збирання та впорядкування інформації. Бази даних можуть зберігати відомості про людей, продукти, замовлення або будь-що інше. Багато баз даних починаються зі списку в текстовому редакторі або електронній таблиці.

1. **Загальна інформація про файлову систему, файли, типи даних та типи файлів**

**Файлова** **система** - спосіб організації даних, який використовується операційною системою для збереження інформації у вигляді файлів на носіях інформації. Також цим поняттям позначають сукупність файлів та директорій, які розміщуються на логічному або фізичному пристрої.

В залежності від організації файлів на носії даних, файлові системи можуть поділятись на:

* ієрархічні файлові системи — дозволяють розміщувати файли в каталоги;
* плоскі файлові системи — не використовують каталогів;

Типи файлів

* **Файли Microsoft Office:** .doc, .docx, .xls, .xlsx, .ppt (лише перегляд), .pptx (лише перегляд).
* **Медіафайли:** .3gp, .avi, .mov, .mp4, .m4v, .m4a, .mp3, .mkv, .ogv, .ogm, .ogg, .oga, .webm, .wav
* **Зображення:** .bmp, .gif, .jpg, .jpeg, .png, .webp
* **Стиснені файли:** .zip, .rar
* **Інші файли:** .txt, .pdf

1. **Основні відомості про програмні засоби роботи з потоками введення-виведення в мовах програмування**

Робота з потоками введення-виведення (I/O) є важливою частиною розробки програм. Вона дозволяє програмі взаємодіяти з користувачем, читати та записувати дані з різних джерел і призначень. Нижче наведено основні відомості про програмні засоби роботи з потоками введення-виведення в різних мовах програмування:

**Python:**

* Введення: Функція **input()** використовується для отримання введення від користувача зі стандартного введення (консолі).
* Виведення: Функція **print()** використовується для виведення даних на стандартний вивід (консоль).
* Робота з файлами: Для роботи з файлами використовуються функції **open(), read(), write(), close()** та інші, які дозволяють відкривати файли, читати та записувати дані у файл, а також закривати файл.

**Java:**

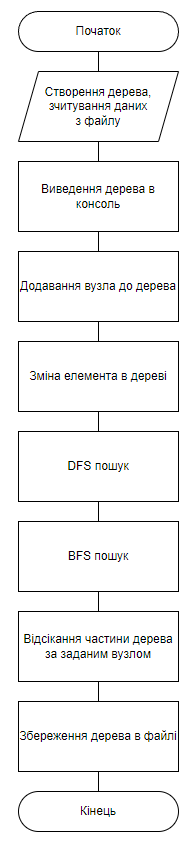
* Введення: Для введення даних з консолі використовується клас **Scanner**. Наприклад, **Scanner scanner = new Scanner(System.in);** дозволяє отримувати введення від користувача.
* Виведення: Для виведення даних на консоль використовується клас **System.out** та метод **println()** або **print().**
* Робота з файлами: Для роботи з файлами використовуються класи **File, FileReader, FileWriter, BufferedReader, BufferedWriter** та інші, які дозволяють відкривати файли, читати та записувати дані у файл, а також закривати файл.

**C++:**

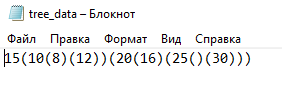
* Введення: Для введення даних з консолі використовуються об'єкти **cin** та операції **>>**. Наприклад, **cin >> variable;** дозволяє отримувати введення від користувача.
* Виведення: Для виведення даних в консоль використовуються об'єкт **cout** та операція **<<**. Наприклад, **cout << "Hello, World!";** виведе рядок в консоль.
* Робота з файлами: Для роботи з файлами використовуються об'єкти **ifstream** (для читання), **ofstream** (для запису)та **fstream** (для читання і запису), які дозволяють відкривати файли, читати та записувати дані у файл, а також закривати файл.

Це лише загальна інформація про роботу з потоками введення-виведення у деяких популярних мовах програмування. Кожна мова має свої власні бібліотеки та методи роботи з I/O, які можуть бути більш розширеними та містити додаткові функціональні можливості.

1. **Блок-схеми алгоритмів роботи функцій і програм**

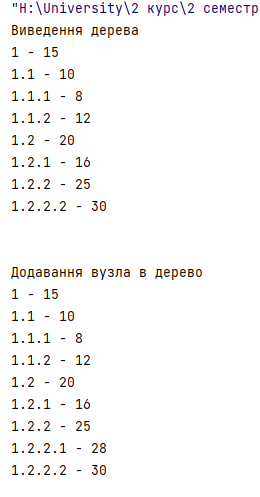
****

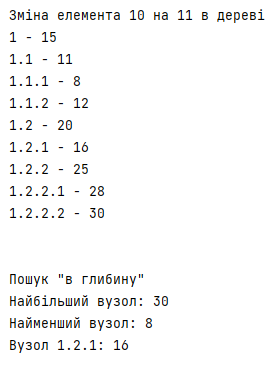
1. **Вміст файлів вихідних даних**

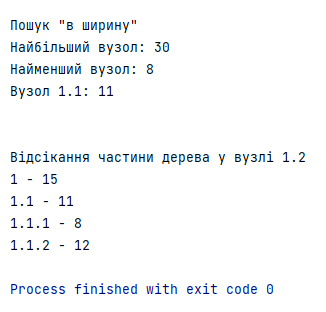
****

|  |
| --- |
| **tree\_data.txt**  15(10(8)(12))(20(16)(25()(30))) |

1. **Скріншот екрану програми з результатом роботи програми**







1. **Текст вихідних кодів програм**

|  |
| --- |
| **Python**  from collections import deque  *# Клас для зберігання вузла бінарного дерева.* class Node:  def \_\_init\_\_(self, data=None, left=None, right=None):  self.data = data  self.left = left  self.right = right  def build\_tree\_from\_file(filename):  with open(filename, 'r') as file:  data = file.readline().strip() *# Зчитування рядка з файлу* return build\_tree(data)  def build\_tree(data):  if not data:  return None   *# Рекурсивна функція для побудови дерева* def build\_subtree(data\_list):  if not data\_list:  return None   value = ""  while data\_list and data\_list[0] not in "()":  value += data\_list.pop(0)   if value == "":  return None   node = Node(int(value))  if data\_list and data\_list[0] == "(":  data\_list.pop(0) *# Видалення лівої дужки* node.left = build\_subtree(data\_list)  data\_list.pop(0) *# Видалення правої дужки* if data\_list and data\_list[0] == "(":  data\_list.pop(0) *# Видалення лівої дужки* node.right = build\_subtree(data\_list)  data\_list.pop(0) *# Видалення правої дужки* return node   *# Розбиття рядка на список символів* data\_list = list(data)  return build\_subtree(data\_list)   *# Збереження дерева у файл* def save\_tree\_to\_file(root, filename):  with open(filename, 'w') as file:  data = serialize\_tree(root) *# Серіалізація дерева* file.write(data)  def serialize\_tree(root):  if root is None:  return ""   data = str(root.data)   if root.left is not None or root.right is not None:  data += "(" + serialize\_tree(root.left) + ")(" + serialize\_tree(root.right) + ")"   return data       *# Обхід дерева в попередньому порядку та збереження вузлів у словнику, # що відповідає їх рівню* def preorder(root, level, d):  *# Базовий випадок: пусте дерево* if root is None:  return  def traverse(node, path):  if node is None:  return   print(path, '-', node.data)   if node.left:  traverse(node.left, path + ".1")   if node.right:  traverse(node.right, path + ".2")   traverse(root, "1")  def replace(root, old\_value, new\_value):  if root is None:  return   if root.data == old\_value:  root.data = new\_value   replace(root.left, old\_value, new\_value)  replace(root.right, old\_value, new\_value)  def cut\_subtree(root, target):  if root is None:  return None   if root == target:  return None   root.left = cut\_subtree(root.left, target)  root.right = cut\_subtree(root.right, target)   return root  def dfs(node, criterion):  if node is None:  return None   *# Пошук найбільшого вузла* if criterion == 'maximum':  if node.right:  return dfs(node.right, criterion)  else:  return node.data   *# Пошук найменшого вузла* elif criterion == 'minimum':  if node.left:  return dfs(node.left, criterion)  else:  return node.data   return None def dfs\_find(node, value, path="1"):  if node is None:  return None   if node.data == value:  return (node.data, path)   left\_result = dfs\_find(node.left, value, path + ".1")  if left\_result:  return left\_result   right\_result = dfs\_find(node.right, value, path + ".2")  if right\_result:  return right\_result   return None *# Функція для виведення номера вузла з його ієрархією* def print\_node\_with\_path(node\_value, node\_path):  if node\_path:  print(f"Вузол {node\_path}: {node\_value}")  else:  print(f"Кореневий вузол: {node\_value}")   def bfs(node, criterion, target\_level=None):  if node is None:  return None   queue = deque([(node, 0)])  result = []   while queue:  current\_node, level = queue.popleft()   *# Пошук найбільшого вузла* if criterion == 'найбільший':  if not result or current\_node.data > result[0]:  result = [current\_node.data]   *# Пошук найменшого вузла* elif criterion == 'найменший':  if not result or current\_node.data < result[0]:  result = [current\_node.data]   *# Пошук вузлів на певному рівні* elif criterion == 'рівень':  if level == target\_level:  result.append(current\_node.data)   if current\_node.left:  queue.append((current\_node.left, level + 1))  if current\_node.right:  queue.append((current\_node.right, level + 1))   return result  def bfs\_find(node, value, path="1"):  if node is None:  return None   if node.data == value:  return (node.data, path)   left\_result = dfs\_find(node.left, value, path + ".1")  if left\_result:  return left\_result   right\_result = dfs\_find(node.right, value, path + ".2")  if right\_result:  return right\_result   return None  *# Рекурсивна функція для друку обходу заданого бінарного дерева за рівнями* def levelOrderTraversal(root):  *# створює порожній словник для зберігання вузлів між заданими рівнями* d = {}   *# проходить по дереву і вставляє його вузли у словник  # відповідний їхньому рівню* preorder(root, 1, d)   *# виконує ітерацію за словником та друкує всі вузли між заданими рівнями* for i in range(1, len(d) + 1):  print(f'Level {i}:', d[i])   if \_\_name\_\_ == '\_\_main\_\_':  *# Ім'я файлу з даними про дерево* filename = "tree\_data.txt"  *# Зчитування дерева з файлу* root = build\_tree\_from\_file(filename)    *# Виведення дерева* print("Виведення дерева")  levelOrderTraversal(root)   *# Додавання вузла* print('\n')  print("Додавання вузла в дерево")  root.right.right.left = Node(28)  levelOrderTraversal(root)   *# Зміна елемента дерева* print('\n')  print("Зміна елемента 10 на 11 в дереві")  replace(root, 10, 11)  levelOrderTraversal(root)    print('\n')  *# Пошук "в глибину" (DFS)  # Пошук найбільшого вузла* print("Пошук \"в глибину\"")  maximum\_node = dfs(root, 'maximum')  print("Найбільший вузол:", maximum\_node)   *# Пошук найменшого вузла* minimum\_node = dfs(root, 'minimum')  print("Найменший вузол:", minimum\_node)   *# Пошук вузла зі значенням 16* found\_node = dfs\_find(root, 16)  if found\_node:  value, path = found\_node  print\_node\_with\_path(value, path)  else:  print("Вузол не знайдено")   print('\n')  *# Пошук "в ширину" (ВFS)  # Пошук найбільшого вузла* print("Пошук \"в ширину\"")  max\_node = bfs(root, 'найбільший')  print("Найбільший вузол:", max\_node[0])  *# Пошук найменшого вузла* min\_node = bfs(root, 'найменший')  print("Найменший вузол:", min\_node[0])  *# Пошук вузла зі значенням 11* found\_node = bfs\_find(root, 11)  if found\_node:  value, path = found\_node  print\_node\_with\_path(value, path)  else:  print("Вузол не знайдено")   print('\n')  print("Відсікання частини дерева у вузлі 1.2")  cut\_subtree(root, root.right)  levelOrderTraversal(root)   *# Ім'я файлу для збереження даних* filename = "new\_tree\_data.txt"  *# Збереження дерева у файл* save\_tree\_to\_file(root, filename) |

1. **Короткі висновки**

В цій лабораторній роботі я навчився створювати та працювати з структурою бінарне дерево, а також навчився робити DFS (пошук в глибину) та BFD (пошук в ширину).

1. **Перелік використаних джерел**

* [**https://support.google.com/chromebook/answer/183093?hl=uk**](https://support.google.com/chromebook/answer/183093?hl=uk)
* [**https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0\_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0**](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)
* [**https://support.microsoft.com/uk-ua/office/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96-%D0%BF%D1%80%D0%BE-%D0%B1%D0%B0%D0%B7%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85-a849ac16-07c7-4a31-9948-3c8c94a7c204**](https://support.microsoft.com/uk-ua/office/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96-%D0%BF%D1%80%D0%BE-%D0%B1%D0%B0%D0%B7%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85-a849ac16-07c7-4a31-9948-3c8c94a7c204)
* [**http://elcat.pnpu.edu.ua/docs/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8%20%D1%96%20%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%B8%D1%85/lab8-9\_tree.html**](http://elcat.pnpu.edu.ua/docs/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8%20%D1%96%20%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%B8%D1%85/lab8-9_tree.html)