**УДК 004.422.5**

*Пацула А. О., здобувач 2 курсу*
*спеціальності 122 Комп'ютерні науки,*
*Потапова Н. А., канд. екон. наук, доцент,*
*доцент кафедри інформаційних технологій*

# РЕКУРСІЯ В ПРОГРАМУВАННІ: ПОНЯТТЯ, ПРИНЦИП РОБОТИ ТА ЗАСТОСУВАННЯ

*Донецький національний університет імені Василя Стуса, м. Вінниця*

У програмуванні рекурсія – це метод розв'язання задачі, коли функція викликає сама себе, поступово спрощуючи задачу до базового випадку. Інакше кажучи, розв'язання задачі будується на розв'язанні її менших випадків [1; 2]. Рекурсія забезпечує природний опис багатьох алгоритмів: під час виклику функції кожен рекурсивний крок обчислює частину задачі, доки не буде досягнуто умову завершення (базовий випадок), після чого за всіма рівнями рекурсії формується результат. Рекурсивні процедури часто застосовують для обчислень, заданих рекурентними співвідношеннями [3]. Наприклад, для обчислення факторіалу або чисел Фібоначчі функція викликає сама себе з меншими параметрами, доки не досягне відомих початкових значень [3]. Проаналізуємо приклади використання рекурсивних алгоритмів:

***1. Генерація всіх підмножин / комбінацій – «subsets»***

```
using System;
using System.Collections.Generic;
static List<List<int>> GetSubsets(int[] nums, int index = 0) {
List<List<int>> all = new List<List<int>>();
if (index >= nums.Length) {
all.Add(new List<int>()); // порожня підмножина
return all;
}
int current = nums[index];
List<List<int>> subsetsWithoutCurrent = GetSubsets(nums, index + 1);

// для кожної підмножини без current – повернути копію + поточний елемент
foreach (var subset in subsetsWithoutCurrent) {
all.Add(new List<int>(subset)); // без current
var subsetWith = new List<int>(subset);
subsetWith.Add(current);
all.Add(subsetWith); // з current
}
return all;
}
```

Ця функція рекурсивно формує всі підмножини масиву, що корисно для задач на комбінації, перебори станів, підмножини множин тощо.

Рекурсивний обхід структури даних – дерево чи вкладені колекції. У багатьох задачах є вкладені структури (дерево, списки списків, папки файлів), і рекурсія – природне рішення для їх обходу.

```
using System;
using System.Collections.Generic;
class Node {
public int Value;
public List<Node> Children = new List<Node>();
}
static void Traverse(Node node) {
if (node == null) return;
Console.WriteLine(node.Value);
foreach (var child in node.Children) {
Traverse(child); // рекурсивний виклик для кожного нащадка
}
}
```

Цей підхід легко масштабується на будь-яку глибину вкладення, що ідеально для дерев, графів (якщо контролювати відвідані вершини), JSON / XML / DOM / вкладених колекцій тощо.

Рекурсія має свої переваги та недоліки. Вона дає змогу записувати компактний і зрозумілий код: алгоритм часто описується простіше, ніж еквівалентний ітеративний варіант [4; 5]. Завдяки рекурсії алгоритми стають більш компактними, і «рекурсивна процедура показує, що потрібно робити» [4]. За допомогою рекурсії можливо природно відобразити математичні формули та ієрархічні структури даних, полегшуючи розробку і доведення коректності. Вона широко застосовується у функціональному програмуванні і під час роботи з деревами, графами, фракталами та іншими рекурсивними структурами.

Проте кожен рекурсивний виклик потребує витрат пам’яті стеку для збереження параметрів і локальних змінних, що може призвести до швидкого заповнення стеку для глибоких викликів [6]. Наприклад, неефективне витрачання оперативної пам’яті є однією з головних проблем рекурсивних алгоритмів: кожен виклик створює новий контекст функції [7]. У багатьох випадках рекурсивні алгоритми повільніші за ітеративні. Під час роботи з рекурсивним алгоритмом може виникнути проблема переповнення стеку. Отже, під час використання рекурсії потрібно впевнено встановлювати базовий випадок і обмежувати глибину викликів.

Отже, рекурсія є методом алгоритмізації, який дає змогу розбивати задачу на підзадачі однаковим алгоритмом за дотримання вимог стислості коду та простоти моделювання складних ітеративних процесів.

## Список використаних джерел

1. Recursion Algorithms. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/dsa/recursion-algorithms/ (дата звернення: 28.11.2025).

2. C#. Рекурсія. Приклади рекурсивних методів (функцій) в C#. *BestProg.net*. URL: https://www.bestprog.net/uk/2019/01/21/recursion-examples-of-recursive-methods-functions-in-c-ua/ (дата звернення: 28.11.2025).

3. Recursion in Algorithm Design: A Deep Dive. URL: https://www.numberanalytics.com/blog/recursion-in-algorithm-design-deep-dive (дата звернення: 28.11.2025).

4. Рекурсивні функції та алгоритми. URL: https://sancho.iasa.kpi.ua/ASD_HTM/Recurs01.html (дата звернення: 28.11.2025).

5. Programm.top (C#/.NET). URL: https://programm.top/uk/c-sharp/tutorial/recursion/ (дата звернення: 28.11.2025).

**УДК 004.65**

*Бородій І. І., аспірант,*
*Осухівська Г. М., канд. техн. наук,*
*доцент, завідувач кафедри*
*комп'ютерних систем та мереж*

## ФОРМУВАННЯ ВХІДНИХ ДАНИХ ДЛЯ АНАЛІТИЧНИХ СИСТЕМ НА ОСНОВІ АРХІТЕКТУР DATA LAKEHOUSE

*Тернопільський національний технічний університет імені Івана Пулюя, м. Тернопіль*

Для побудови аналітичних систем даних провідною технологією є концепція Data Lakehouse, яка поєднує переваги Data Lake та Data Warehouse. Вона забезпечує гнучке зберігання структурованих і напівструктурованих типів даних, ефективне управління метаданими та оптимізоване виконання аналітичних запитів. У контексті реалізації найпоширенішими систем Lakehouse є табличні формати Apache Hudi, Apache Iceberg та Delta Lake.

Apache Iceberg – це продукт компанії Netflix, що є табличним форматом з відкритим кодом, призначений для роботи з великими обсягами аналітичних даних у сховищах типу Data Lakehouse. Apache Hudi є системою збереження даних, орієнтованого на роботу з великими обсягами інформації, що постійно змінюється. Delta Lake є технологією для забезпечення транзакційної цілісності та контролю версій у хмарних системах зберігання даних.

У сучасних роботах розглядають можливості Data Lakehouse архітектур, зокрема у межах порівняння їх продуктивності та особливостей обробки даних [1]. Ефективність використання Lakehouse-підходів у масштабованих аналітичних системах підтверджена в роботі [2]. Авторами дослідження [3] було показано, що вибір способу реалізації ETL-процесів суттєво впливає на продуктивність обробки даних у середовищі Apache Spark.

Метою дослідження є формування вхідних даних для аналітичних систем на основі архітектур Data Lakehouse, щоб здійснити вибір найоптимальнішої її реалізації за критеріями швидкості завантаження даних та ефективності збереження у дисковому просторі.

Формування вхідних наборів даних для аналітичних систем на основі архітектур Data Lakehouse здійснювалося з декількох джерел. Зокрема, вебресурс open-meteo надає погодинні показники температури атмосферного повітря за 2022–2024 рр. з прив'язкою до географічних координат. Цей структурований масив даних було завантажено у файл формату CSV обсягом 7 ГБ. Додатково у дослідженні використано CSV-файл із вебресурсу simplemaps.com, який містить перелік міст та їх координати, розміром приблизно 5 МБ. Напівструктуровані дані у форматі JSON отримано з вебресурсу iqair.com, що містить інформацію про показники