АЛГОРИТМИ ТА ОБЧИСЛЮВАЛЬНА СКЛАДНІСТЬ

1. Основи структури даних і алгоритми

1.1. Поняття алгоритму. Визначення його часової та просторової (за обсягом пам'яті) складності.

Переглянути ролик для ознайомлення або повтору теми: [YouTube](https://www.youtube.com/watch?v=cXCuXNwzdfY)

Ознайомитися з наступними підрозділами (1.3, 4.5, 5.3, 5.5, 6.4, 7.6, 8.6, 9.6) за посиланням: [PDF](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjlw6Snst-FAxVXgv0HHVNXC38QFnoECDYQAQ&url=https%3A%2F%2Fela.kpi.ua%2Fbitstream%2F123456789%2F48282%2F1%2FOsnovy_alhorytmizatsii.pdf&usg=AOvVaw286ufqsNrEqsrP-NFHbXpo&opi=89978449)

1.2. Поняття абстрактного типу даних. Абстрактні типи даних: стеки, списки, вектори, словники, множини, мультимножини, черги, черги з пріоритетами.

Плейліст де коротко розповідають основні структури даних: [YouTube](https://youtube.com/playlist?list=PLQOaTSbfxUtAIipl4136nwb4ISyFk8oI4&si=BXlMlEnfCMIUGSgB)

Книга в якій теорія та практика, приклади коду мовою Python: [PDF](https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf)

1.3. Кортежі, множини, словники, одно- та двобічнозв'язні списки. Реалізація абстрактних типів даних з оцінюванням складності операцій.

Книга в якій теорія та практика, приклади коду мовою Python: [PDF](https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf)

1.4. Базові алгоритми та їх складність: пошук, сортування (прості сортування вибором, вставками, обмінами та удосконалені сортування деревом, сортування Шелла, швидке сортування).

Переглянути ролик для ознайомлення або повтору теми (Сортування): [YouTube](https://www.youtube.com/watch?v=PF7AqefS4MU)

1.5. Алгоритми на графах та їх складність:

* пошук в ширину і глибину;
* пошук зв'язних компонентів;
* побудова кістякового дерева;
* побудова найкоротших шляхів з виділеної вершини;
* побудова найкоротших шляхів між двома вершинами;

Переглянути плейліст для ознайомлення або повтору теми : [YouTube](https://www.youtube.com/watch?v=VehB3eglQMQ)

Для детального вивчення теми переглянути плейлист: [YouTube](https://youtube.com/playlist?list=PLVZ5b_sJYXlE3_-1xJM7V0GGSCvaHXmg9&si=tjcvdWvnpqcZ5Iru)

Ознайомитися з матеріалом за посиланням: [PDF](https://repository.kpi.kharkov.ua/server/api/core/bitstreams/402b2361-3833-40c2-845c-90bea2c44830/content)

2. Стратегії розроблення алгоритмів.

2.1. Стратегія «розділяй та володарюй» та приклади застосування.

Ця стратегія передбачає поділ складної задачі на менші підзадачі, розв'язок яких є більш простим, і потім комбінує рішення підзадач для отримання загального розв'язку.

Приклад застосування: Сортування злиттям (Merge Sort)

У сортуванні злиттям масив рекурсивно розбивається на дві половини, кожна з яких сортується окремо, а потім об'єднується відсортований масив. Цей процес повторюється, доки не залишиться лише один елемент у кожному підмасиві.

def merge\_sort(arr):

    if len(arr) > 1:

        mid = len(arr) // 2

        L = arr[:mid]

        R = arr[mid:]

        merge\_sort(L)

        merge\_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):

            if L[i] < R[j]:

                arr[k] = L[i]

                i += 1

            else:

                arr[k] = R[j]

                j += 1

            k += 1

        while i < len(L):

            arr[k] = L[i]

            i += 1

            k += 1

        while j < len(R):

            arr[k] = R[j]

            j += 1

            k += 1

arr = [38, 27, 43, 3, 9, 82, 10]

merge\_sort(arr)

print("Відсортований масив:", arr)

2.2. Стратегія балансування та приклади застосування.

Ця стратегія використовується для забезпечення рівномірного розподілу ресурсів або обробки даних з метою підтримки ефективності та оптимальності роботи алгоритмів.

Приклад застосування: Черги з пріоритетами (Priority Queues)

У чергах з пріоритетами кожен елемент має асоційований пріоритет, і елементи видаються в порядку їхнього пріоритету, а не в порядку додавання.

import heapq

class PriorityQueue:

    def \_\_init\_\_(self):

        self.\_queue = []

        self.\_index = 0

    def push(self, item, priority):

        heapq.heappush(self.\_queue, (-priority, self.\_index, item))

        self.\_index += 1

    def pop(self):

        return heapq.heappop(self.\_queue)[-1]

q = PriorityQueue()

q.push('task1', 3)

q.push('task2', 1)

q.push('task3', 2)

print(q.pop())  # task1

print(q.pop())  # task3

print(q.pop())  # task2

2.3. Динамічне програмування та приклади застосування.

Ця стратегія використовується для розв'язання складних задач, розбиваючи їх на простіші підзадачі та зберігаючи проміжні результати для майбутнього використання.

Приклад застосування: Знаходження найбільшої зростаючої підпослідовності (Longest Increasing Subsequence)

У цьому завданні необхідно знайти найдовшу послідовність чисел, які знаходяться у зростаючому порядку.

def lis(arr):

    n = len(arr)

    lis = [1] \* n

    for i in range(1, n):

        for j in range(0, i):

            if arr[i] > arr[j] and lis[i] < lis[j] + 1:

                lis[i] = lis[j] + 1

    return max(lis)

arr = [10, 22, 9, 33, 21, 50, 41, 60]

print("Довжина найбільшої зростаючої підпослідовності:", lis(arr))

Книга в матеріал для більш делального ознайомлення з темою: [PDF](https://e-tk.lntu.edu.ua/pluginfile.php/20063/mod_resource/content/0/Тема%2010.%20Алгоритмічні%20стратегії.pdf)

3. Моделі обчислень.

3.1. Імперативний та декларативний підходи до праграмування.

<https://foxminded.ua/imperatyvne-ta-deklaratyvne-prohramuvannia/>

<https://dou.ua/forums/topic/37548/#:~:text=Імперативний%20і%20декларативний%20підходи%20у,виконання%20команд%20(імперативний%20підхід)>.

3.2. Розв'язні, напіврозв'язні та нерозв'язні проблеми. Проблема зупинки.

Дуже складний матеріал: <https://studfile.net/preview/7328750/>