Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

ФІОТ

ІСТ

Лабораторна робота №2

по курсу *“*Теорія алгоритмів”

на тему “Методи розробки алгоритмів. Частина 1.”

Варіант 3

Виконали ст. групи ІС-31:

Коваль Богдан

Михайлова Софія

Шмигельський Ілля

Перевірив: ст. вик. Дорошенко К.С.

Киів 2024

**Комп’ютерний практикум 1**

**Тема:** Методи розробки алгоритмів. Частина 1.

**Мета роботи:** порівняння алгоритмів розв’язку задачі, побудованих різними методами.

**Завдання**

1. Для свого варіанту зробити наступні дії:

* 1) Сформулювати постановку задачі
* 2) Обрати відповідні 2 алгоритми з теоретичної частини практикуму або на свій розсуд
* 3) Накреслити блок-схеми, на яких виконано аналіз складності алгоритмів
* 4) Написати програмний код
* 5) Провести дослідження продуктивності роботи алгоритмів, зробити результати
* дослідження у вигляді графіків або діаграм;
* 6) Зробити висновки про доцільність використання кожного з алгоритмів для типових
* вхідних даних та про відповідність результатів експериментального дослідження
* аналітичним оцінкам складності.
* 7) Скласти таблицю тестування.
* 8) Навести скрин-шоти роботи програми. для заданої задачі;ю
* 9) Дати відповіді на контрольні питання

**Постановка задачі**

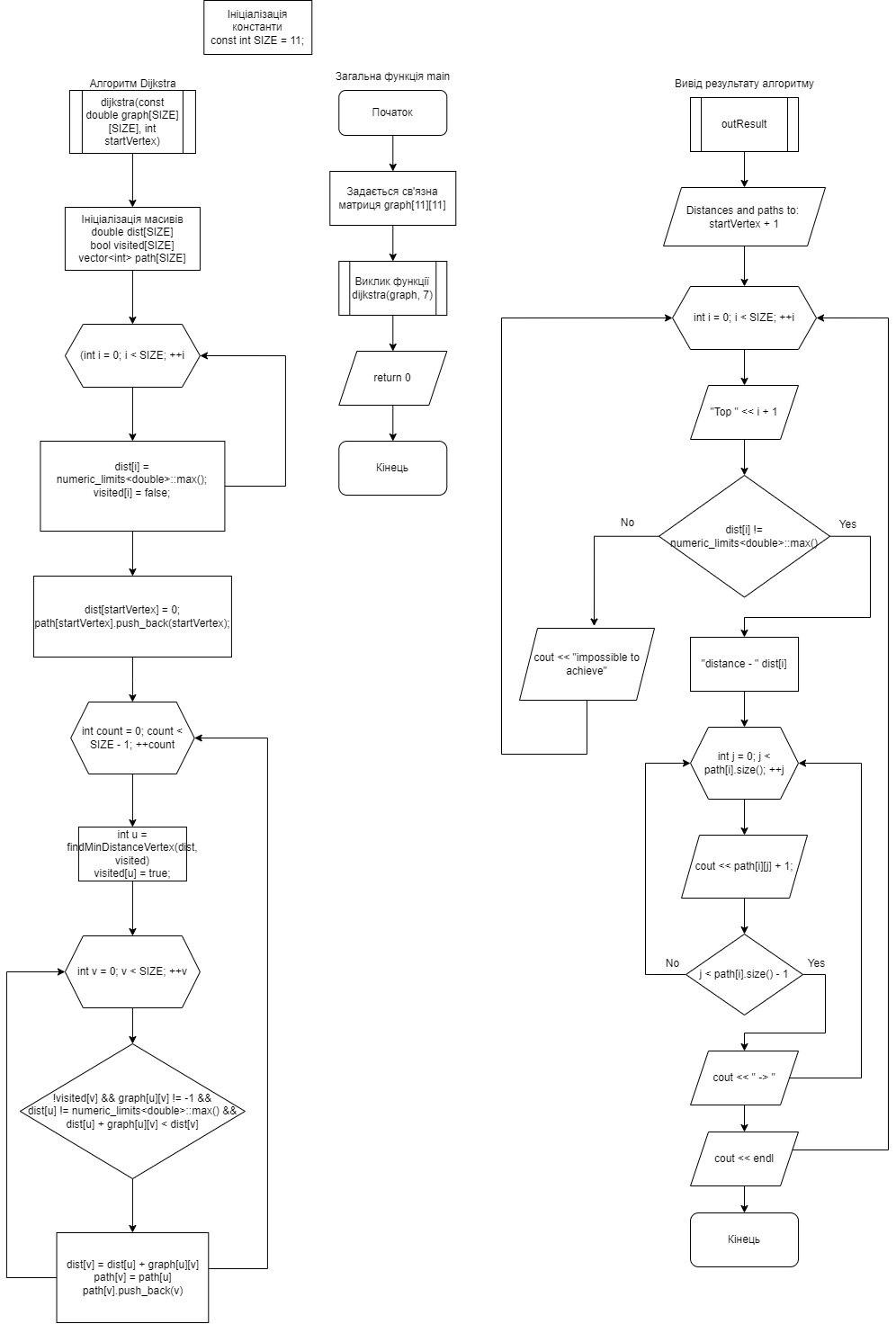
Не всі студенти НТУУ «КПІ ім. Ігоря Сікорського» полюбляють гуляти пішки, тому компанія хлопців і дівчат з університету вирішила прокласти транспортні маршрути від свого навчального закладу до інших вищеперерахованих місць за умови найменшої вартості такого проєкту. Допоможіть їм прокласти такі маршрути.

Вихідні дані: масив з заданими маршрутами.

**Побудова моделі**

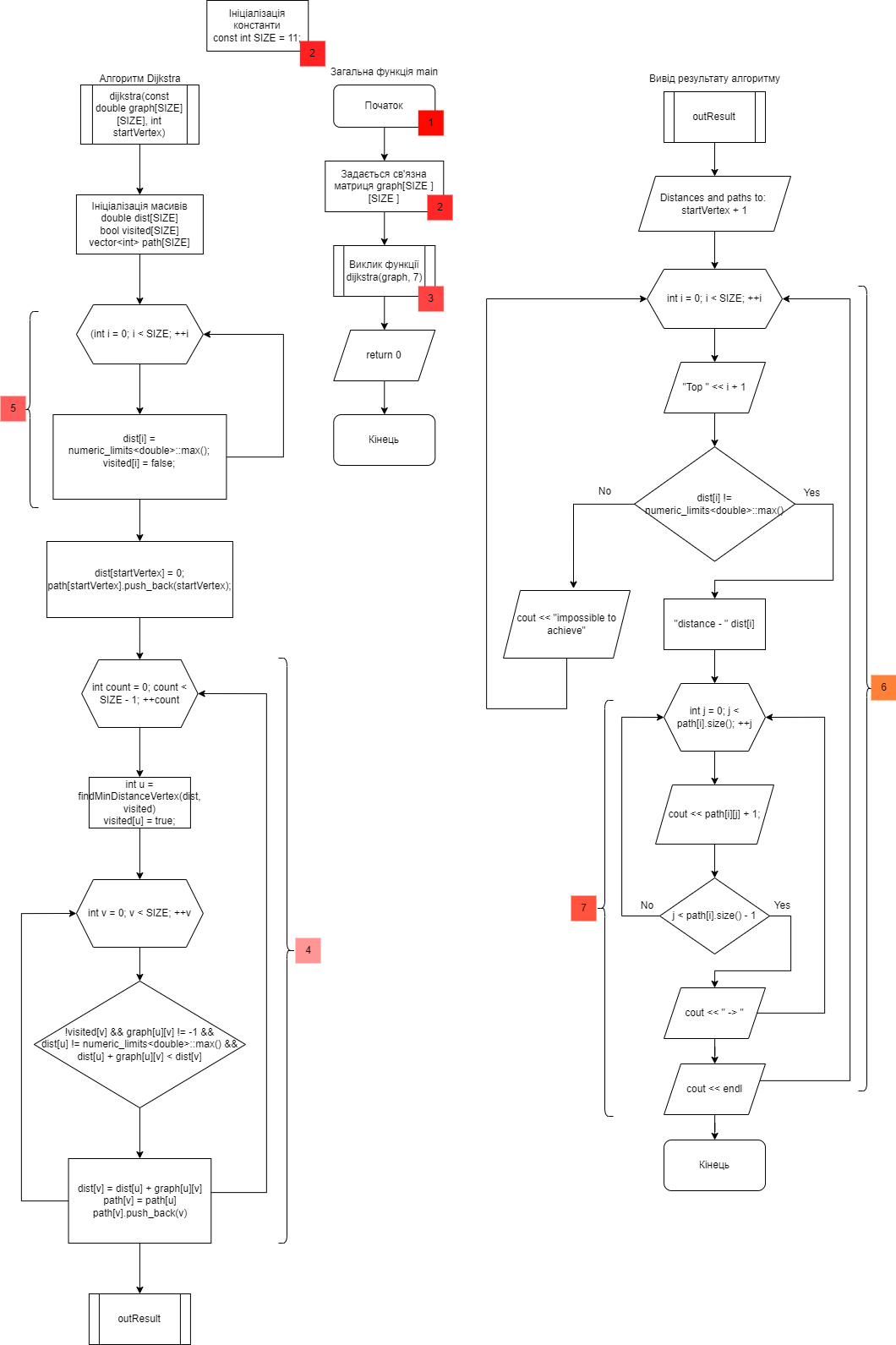
**Інтерфейс користувача**

**Розробка алгоритму**

****

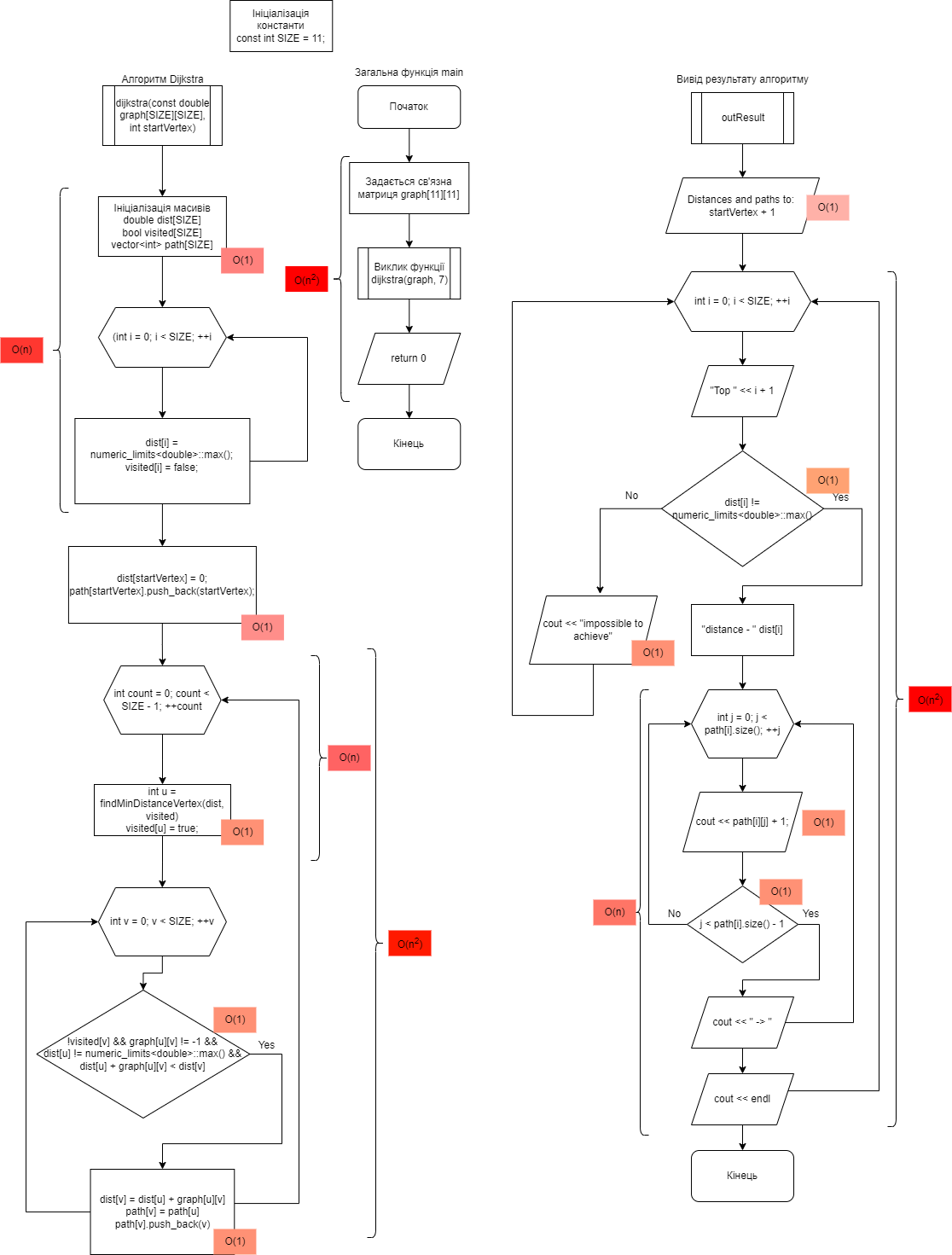
**Правильність алгоритму**

1. Обґрунтування правомірності кожного кроку: блок ***1*** - початок програми, блок ***2*** відповідає за введення початкових даних – без них алгоритм не працюватиме; блок ***3*** відповідає за виклик головної функції, який відповідає за пошук за алгоритмом Дейкстрі.
2. Доведення кінцевості алгоритму: кінцевість алгоритму залежить від блоків 4***, 5***. У наданому коді кількість елементів у матриці визначається за допомогою константної змінної SIZE, яка задається прямо в програмі Отже, кількість вершин у матриці є скінченною, оскільки вона обмежена значенням, заданою в програмі. Таким чином, матриця завжди буде містити кінцеву кількість вершин. **Використання властивостей вхідних даних**: Якщо вхідні дані обмежені за певними параметрами, можна аналізувати ці властивості для підтвердження кінцевості алгоритму.
3. Для виведення отриманих даних алгоритмом використані блоки ***6, 7.***



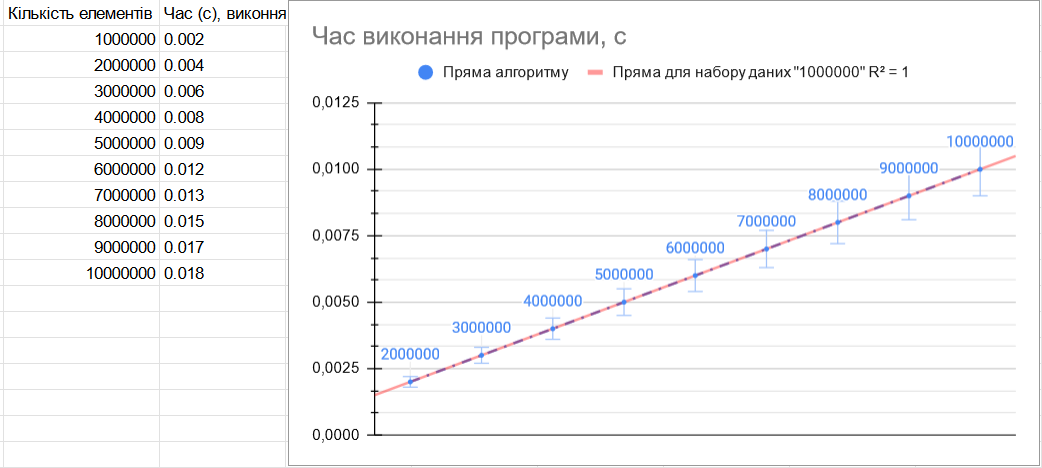
**Аналіз алгоритму та його складності**

Загальна складність алгоритму ***O(n^2). n - В даному випадку статична і дорівнює розміру матриці***



**Реалізація алгоритму**

**Скласти таблицю тестування**

****

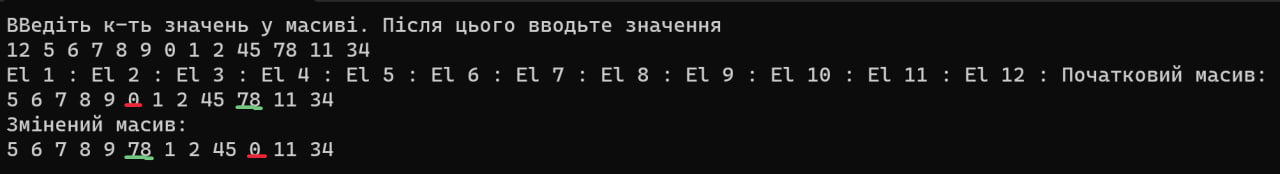
****

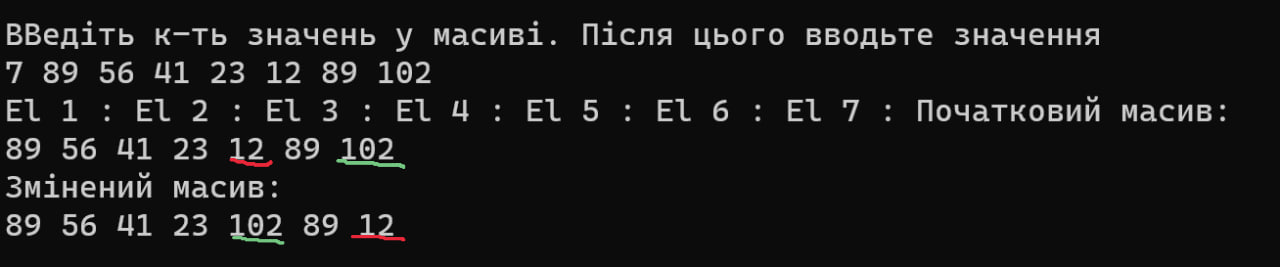
| **Вхідні дані** | | | **Результат** | | Призначення тесту |
| --- | --- | --- | --- | --- | --- |
| choice  (Вибір варіанту введення масиву) *1-Console*  *2-Генерація*  *3-Зчитування з файлу* | array\_size  (Задання розміру масиву) | inp\_array  (задання масиву) | Console  (вивід в консоль) | output.txt  (вивід у файл) | **Опис помилки** |
| 1 | 5 | 41 67 34 0 69 | 41 67 34 69 0 | 41 67 34 69 0 | Загальна перевірка працездатності алгоритму та коректності результату. |
| 2 | 5 | випадкове | 3 4 2 8 3 5 | 3 4 2 8 3 5 | Загальна перевірка працездатності алгоритму та коректності результату. |
| 2 | q | випадкове | Помилка: введено некоректний символ. | - | Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки) |
| 2 | 3.5 | випадкове | Помилка: введено некоректний символ (довжина масива має бути цілим числом). | - | Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки) |
| 1 | 2.5 | - | Помилка: введено некоректний символ. | - | Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки) |
|  | \* | - | Помилка: введено некоректний символ. | - | Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки) |
| 1 | -1 | - | Помилка: введено некоректні дані (від'ємне значення довжини масиву) | - | Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки) |
| 1 | 3 | 2 q | Помилка: введено некоректний символ. | - | Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки) |

**Перевірка працездатності алгоритму**

| **Inp\_1** | 4 | 7 | 2 | 9 | 3 | 8 | 4 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Out\_1** | 4 | 7 | 9 | 2 | 3 | 8 | 4 |
| **Inp\_2** | 2 | 8 | 1 | 3 | 8 | 4 | 2 |
| **Out\_2** | 2 | 1 | 8 | 3 | 8 | 4 | 2 |
| **Inp\_3** | 3 | 7 | 1 | 8 | 5 | 3 | 2 |
| **Out\_3** | 3 | 7 | 8 | 1 | 5 | 3 | 2 |

**Скріншоти роботи програми**

****

****

**Контрольні запитання**

1. Перерахуйте відомі вам методи розробки алгоритмів. Докладніше розкажіть про один з них.

### **Метод частинних цілей (**“Розділяй та володарюй”**)**

Цей метод полягає у тому, що глобальна велика задача ділиться (якщо це можливо) на окремі задачі.

### **Динамічне програмування**

Динамічним програмуванням (в найбільш загальній формі) називають процес покрокового розв'язку задачі.

### **Метод сходження**

Даний метод полягає у тому, щоби протягом пошуку найкращого розв'язку алгоритм відшукував все кращі та кращі варіанти розв'язку.

### **Програмування з поверненнями назад**

Іноді доводиться мати справу з задачами пошуку оптимального розв'язку, коли неможливо застосувати жоден з відомих алгоритмів, які здатні допомогти відшукати оптимальний варіант розв'язку, і залишається застосувати останній засіб — повний перебір.

### **Метод спроб та помилок**

Багато задач не допускають аналітичного розв'язку, а тому їх доводиться вирішувати [методом спроб та помилок](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BF%D1%80%D0%BE%D0%B1_%D1%96_%D0%BF%D0%BE%D0%BC%D0%B8%D0%BB%D0%BE%D0%BA), тобто перебираючи усі можливі варіанти та відкидаючи їх у випадку невдачі.

2. Перерахуйте переваги та недоліки наступних методів розробки алгоритмів: методу часткових (проміжних) цілей, методу підйому (локального пошуку), методу відпрацювання назад.

**Метод часткових (проміжних) цілей:**

* Переваги:
  + Дозволяє розбити складну задачу на кілька менших підзадач, що полегшує розробку.
  + Забезпечує можливість поетапного вирішення завдання, підвищуючи зрозуміння процесу розробки.
  + Сприяє структуризації коду та підвищує його читабельність.
* Недоліки:
  + Може призвести до виникнення додаткової складності, якщо підзадачі потребують великої кількості взаємодії або синхронізації.
  + Не завжди ефективний для деяких типів задач, які не легко розбити на підзадачі.

**Метод підйому (локального пошуку):**

* Переваги:
  + Дозволяє знаходити локальні максимуми/мінімуми функції шляхом поступового зміщення в напрямку оптимального розв'язку.
  + Ефективний для задач, де можливо використання ітеративного покращення результату.
  + Зазвичай простий у реалізації та швидкий у виконанні.
* Недоліки:
  + Може застрягати в локальних максимумах/мінімумах, не знаходячи глобально оптимального розв'язку.
  + Не гарантує знаходження оптимального розв'язку для всіх випадків.

**Метод відпрацювання назад:**

* Переваги:
  + Ефективний для задач з оптимальною підструктурою, таких як задачі розбиття на частини або найкоротший шлях.
  + Забезпечує гарантований знаходження оптимального розв'язку, якщо проблема має оптимальну підструктуру.
* Недоліки:
  + Може бути дуже витратним з точки зору пам'яті та обчислень для деяких складних задач.
  + Не ефективний для задач з великою кількістю підзадач або залежностей між ними.

3. Який тип алгоритмів називають «жадібними» і чому?

"Жадібні" алгоритми вирішують проблеми, обираючи на кожному кроці локально найкращий варіант, не дбаючи про майбутні наслідки. Це може бути доцільно в тих випадках, коли вибір на кожному кроці не впливає на глобальний оптимум. Однак жадібні алгоритми не завжди забезпечують найкращий глобальний результат, оскільки вони не розглядають всі можливі варіанти.

4. Дайте характеристику евристичним алгоритмам. В яких випадках доцільно

використовувати цей тип алгоритмів? Опишіть загальний підхід до побудови

евристичних алгоритмів.

**Евристичні алгоритми** - це підхід який базується на експертному інтуїції або простих правилах. Евристичні алгоритми зазвичай не гарантують знаходження оптимального розв'язку, але часто є швидкими та ефективними у випадках складних задач.

Евристичні алгоритми доречні в таких випадках:

1. Коли вимагається **швидка відповідь** або наближене розв'язання, і **точний розв'язок** вимагає значних обчислювальних ресурсів.
2. У випадках, коли не існує ефективного або практичного алгоритму для точного розв'язку проблеми.
3. Коли можливі різні варіанти розв'язку, і не завжди зрозуміло, який є оптимальним.

Загальний підхід до побудови евристичних алгоритмів включає наступні етапи:

1. **Формулювання проблеми**: Ретельне визначення постановки задачі та критеріїв оцінки розв'язку.
2. **Вибір евристик**: Визначення простих правил або стратегій, які можуть допомогти при прийнятті рішення.
3. **Розробка алгоритму**: Реалізація алгоритму на основі вибраних евристик та стратегій.
4. **Тестування та оцінка**: Проведення експериментів для оцінки ефективності та точності роботи алгоритму, а також для визначення його робочих меж.
5. **Покращення та оптимізація**: Вдосконалення алгоритму шляхом виправлення недоліків та впровадження нових ідей або евристик.

5. Проаналізуйте, що спільного мають та чим відрізняються алгоритми, що

використовують пошук з поверненням, та алгоритми, що використовують метод гілок

та границь.

6. Поясніть, для чого можна використовувати метод альфа-бета відсікання.

Метод альфа-бета відсікання є ефективним алгоритмом для оптимізації пошуку в грі на кшталт шахи або шашки. Основна мета цього методу - зменшити кількість вузлів, які потрібно перевірити в дереві гри, шляхом відсікання гілок дерева, які точно не приведуть до оптимального розв'язку.

7. Поясніть термін «структурне програмування». Для чого воно застосовується?

Структурне програмування - це методологія програмування, основна ідея якої полягає в тому, щоб програма складалася зі структурованих блоків (наприклад, послідовностей, виборів, циклів), які можна легко розуміти та підтримувати.

**Висновки**

У цій частині роботи ми дослідили різні методи розробки алгоритмів і порівняли їх ефективність у розв'язанні задачі. Ми розглянули такі методи, як декомпозиція, жадібні алгоритми, динамічне програмування та евристичні методи.

Декомпозиція дозволяє розбити складну задачу на менші підзадачі, що спрощує її розв'язання та робить код більш структурованим. Жадібні алгоритми приймають локально оптимальні рішення на кожному кроці, що може призвести до знаходження підходящого розв'язку, але не завжди оптимального. Динамічне програмування забезпечує ефективний пошук оптимальних рішень шляхом зберігання результатів підзадач. Евристичні методи дозволяють знаходити наближені розв'язки у випадках, коли точний розв'язок є недосяжним через обмеженість ресурсів або складність задачі.

Вибір методу залежить від конкретної задачі, її складності та обмежень. Для найкращого результату може знадобитися поєднання декількох методів або адаптація їх до конкретної ситуації.