Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Алгоритми та структури даних
Частина 2

Методичні рекомендації
до виконання лабораторних робіт студентами
денної та заочної форми навчання спеціальностей
122 "Комп’ютерні науки",
123 "Комп’ютерна інженерія" та
125 "Кібербезпека та захист інформації"

Кропивницький 2025

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Мелешко Є. В., Лисенко І.А.

Алгоритми та структури даних
Частина 2

Методичні рекомендації
до виконання лабораторних робіт студентами
денної та заочної форми навчання спеціальностей
122 "Комп’ютерні науки",
123 "Комп’ютерна інженерія" та
125 "Кібербезпека та захист інформації"

Затверджено
на засіданні кафедри
кібербезпеки та
програмного
забезпечення
Протокол №_1_
від 25.08.2025

Кропивницький 2025

Алгоритми та структури даних. Частина 2. Методичні рекомендації до виконання лабораторних робіт студентами денної та заочної форми навчання спеціальностей 122 "Комп’ютерні науки", 123 "Комп’ютерна інженерія" та 125 "Кібербезпека та захист інформації". – Кропивницький: ЦНТУ, 2025. – 62 с.

Дані методичні рекомендації адресовані майбутнім розроблювачам програмного забезпечення, що навчаються за спеціальностями 122 "Комп’ютерні науки", 123 "Комп’ютерна інженерія" та 125 "Кібербезпека та захист інформації". Вони охоплюють базові алгоритми та структури даних, зокрема, алгоритми роботи з графами, алгоритми генерації та тестуванню псевдовипадкових чисел, алгоритми колаборативної фільтрації для рекомендаційних систем, базові моделі штучних нейронних мереж, а також алгоритми оптимізації функцій із застосуванням генетичних алгоритмів. Ці методичні рекомендації можуть використовуватися також студентами інших спеціальностей. Вони містять основні теоретичні положення та практичні завдання, необхідні для засвоєння учбового матеріалу.

Укладачі: Мелешко Єлизавета Владиславівна,
доктор технічних наук, професор
Лисенко Ірина Анатоліївна,
кандидат технічних наук

Рецензенти: Дрєєв Олександр Миколайович,
кандидат технічних наук, доцент
Минайленко Роман Миколайович,
кандидат технічних наук, доцент

ЗМІСТ

Вступ

Методичні рекомендації до лабораторних робіт з дисципліни «Алгоритми та структури даних» для студентів спеціальностей 122 "Комп’ютерні науки", 123 "Комп’ютерна інженерія" та 125 "Кібербезпека та захист інформації" містять в собі теоретичний матеріал, завдання та інструкції виконання до лабораторних робіт і призначені для студентів, що вивчають програмування, а також можуть бути корисними для школярів, вчителів та просто особистостей, що цікавляться інформаційними технологіями.

Також у цих методичних рекомендаціях міститься приклад оформлення звіту з лабораторної роботи, приклад оформлення лістингу програми, шкала оцінювання успішності студентів та список рекомендованої літератури.

Дані методичні рекомендації починають знайомити з базовими алгоритмами та стурктурами даних у програмуванні, зокрема, з алгоритми роботи з графами, алгоритми генерації та тестуванню псевдовипадкових чисел, алгоритми колаборативної фільтрації для рекомендаційних систем, базові моделі штучних нейронних мереж, а також алгоритми оптимізації функцій із застосуванням генетичних алгоритмів. Приводиться багато практичних прикладів та ілюстрацій, які допомагають краще засвоїти викладений матеріал.

Лабораторна робота № 1

Тема: Алгоритми роботи з графами. Алгоритм Прима-Крускала

Мета: Навчитися реалізовувати алгоритм Прима-Крускала для побудови каркасного дерева графу

Граф – (в програмуванні) це динамічна структура даних, що складається з сукупності об’єктів та зв’язків між ними. Об’єкти називаються вузлами або вершинами, а зв’язки між ними – ребрами або дугами.

Способи представлення графів у комп'ютерних програмах

Існує досить велике число різноманітних способів представлення графів. Розглянемо найбільш зручні з погляду програмування.

Матриця суміжності

Матриця суміжності Sm - це квадратна матриця розміром NxN (N - кількість вершин у графі), заповнена одиницями й нулями за наступним правилом:

Якщо в графі є ребро e, що з'єднує вершини u і v, то Sm[u,v] = 1, у противному випадку Sm[u,v] = 0.

Помітимо, що дане визначення підходить як для орієнтованих, так і для неорієнтованих графів: матриця суміжності для неорієнтованого графа буде симетричною щодо своєї головної діагоналі, а для орграфа - несиметричною.

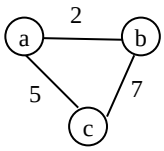
Задати зважений граф за допомогою матриці суміжності теж можливо. Необхідно лише внести невелику зміну у визначення:

Якщо в графі є ребро e, що з'єднує вершини u і v, то Sm[u,v] = vaga(e), у противному випадку Sm[u,v] = 0.

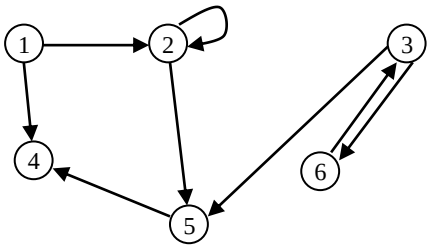
Невелике ускладнення виникне в тому випадку, якщо в графі дозволяються ребра з вагою 0. Тоді треба зберігати два масиви: один з нулями й одиницями, які служать показником наявності ребер, а другий - з вагами цих ребер.

Зручність матриці суміжності полягає в наочності й прозорості алгоритмів, заснованих на її використанні. А незручність - у дещо завищеній вимозі до пам'яті: якщо граф далекий від повного, то в масиві, що зберігає матрицю суміжності, виникає багато "порожніх місць" (нулів). Крім того, для "спілкування" з користувачем цей спосіб подання графів не занадто зручний: його краще застосовувати тільки для внутрішнього представлення даних.

На рисунку 1 наведено два приклади матриць суміжності для неорієнтованого та орієнтованого графу.



	a	**b**	**c**
a	0	2	5
b	2	0	7
c	5	7	0



	1	**2**	**3**	**4**	**5**	**6**
1	0	1	0	1	0	0
2	0	1	0	0	1	0
3	0	0	0	0	1	1
4	0	0	0	0	0	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0

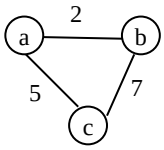
Рисунок 1 – Приклади матриць суміжності зваженого неорієнтованого графа (зліва) та незваженого орієнтованого графа (справа)

Список ребер

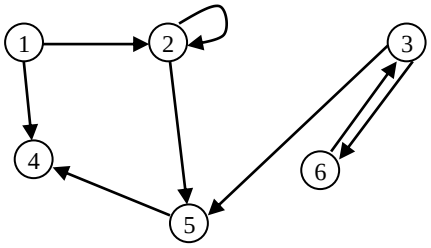
Цей спосіб задання графів найбільш зручний для зовнішнього представлення вхідних даних. Нехай кожний рядок вхідного файлу містить інформацію про одне ребро:

<номер_початкової_вершини> <номер_кінцевої_вершини> [<вага_ребра>]

Приклади списків ребер зображені на рисунку 2.



a, b	2
a, c	5
b, c	7



1, 2
1, 4
2, 2
2, 5
3, 5
3, 6
5, 4
6, 3

Рисунок 2 – Приклади списків ребер зваженого неорієнтованого графа (зліва) та незваженого орієнтованого графа (справа)

Якщо задається орієнтований граф, то номери вершин розуміються як упорядкована пара, а якщо граф неорієнтований - як неупорядкована.

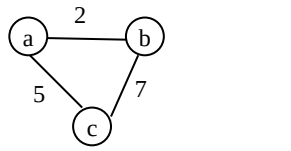
Списки суміжності

Цей спосіб задання графів полягає у тому, що для кожної вершини вказується список всіх суміжних з нею вершин (для орграфа - список вершин, що є кінцями вихідних ребер). Конкретний формат вхідного файлу, що містить списки суміжності, необхідно зазначати окремо. Наприклад, у нашому прикладі початкова вершина відділена від списку суміжності двокрапкою, а ваги ребер (у зваженому графі) вказані в круглих дужках:

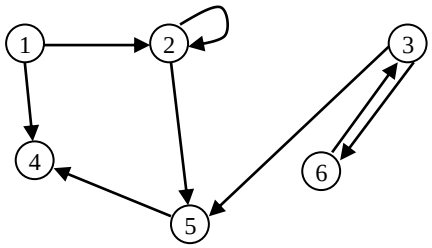
<номер_початкової_вершини>: <номер_суміжної_вершини 1> [(<вага ребра>)],
<номер_суміжної_вершини 2> [(<вага ребра>)], ...

Найбільш природно застосовувати цей спосіб для задання орграфів, однак і для інших варіантів він теж підходить.

Приклади списків суміжності зображені на рисунку 3.



a: b(2), c(5)
b: a(2), c(7)
c: a(5), b(7)



1: 2, 4
2: 2, 5
3: 5, 6
5: 4
6: 3

Рисунок 3 – Приклади списків суміжності зваженого неорієнтованого графа (зліва) та незваженого орієнтованого графа (справа)

Ієрархічний список

Цей спосіб представлення графів є всього лише внутрішньою реалізацією списку суміжності: в одному лінійному списку містяться номери початкових вершин, а в інших - номери суміжних вершин або покажчики на ці вершини.

Мінімальне каркасне дерево графа

Каркасне дерево зв'язаного неорієнтованого графа – ациклічний зв'язний підграф цього графа, який містить всі його вершини. Неформально кажучи, каркасне дерево складається з деякої підмножини ребер графа, таких, що рухаючись цими ребрами можна з будь-якої вершини графа потрапити до будь-якої іншої.

Мінімальне каркасне дерево – це каркас графа, що має мінімальну можливу вагу, де під вагою дерева розуміється сума ваг ребер, що входять до нього.

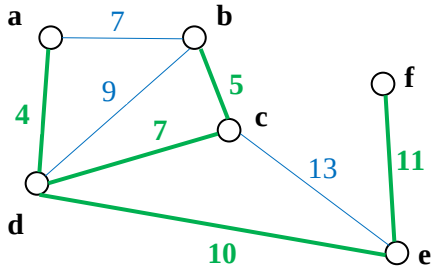


Рисунок 4 – Приклад графа з мінімальним каркасним деревом. Сумарна вага зображеного мінімального каркасного дерева рівна 37. Це мінімальне каркасне дерево не унікальне: видаленням ребра (c,d) та додаванням ребра (a,b) одержується нове мінімальне каркасне дерево

Алгоритм Прима-Крускала – алгоритм побудови мінімального каркасного дерева зваженого зв'язного неорієнтованого графа.

Приклад 1. Розглянемо прикладну задачу, для рішення якої необхідно побудувати мінімальне каркасне дерево.

В Кіровоградській області потрібно з’єднати N сіл оптоволоконними лініями зв’язку для розширення мережі Інтернету. Всі відстані між селами відомі. Знайти найменшу можливу довжину лінії, яка з’єднає всі N сіл.

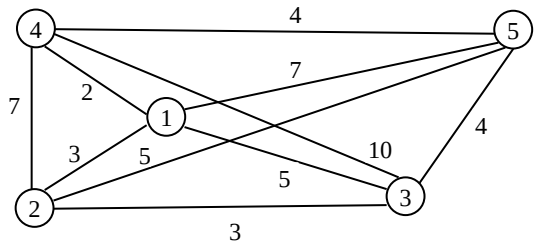


Рисунок 5 – Граф вузли якого – села, а ребра – відстані між ними

	1	**2**	**3**	**4**	**5**
1	0	3	5	2	7
2	3	0	3	7	5
3	5	3	0	10	4
4	2	7	10	0	4
5	7	5	4	4	0

Рисунок 6 – Матриця суміжності графа з рисунку 5

Розфарбовуємо всі вузли графа у різний колір. Для простоти позначимо кольори довільними числами (рис. 7).

Вершина	1	2	3	4	5
Колір	1	2	3	4	5

Рисунок 7 – Матриця кольорів вершин графу

В матриці суміжності даного графа будемо розглядати лише її частину виділену на рисунку 8.

	1	**2**	**3**	**4**	**5**
1	0	3	5	2	7
2	3	0	3	7	5
3	5	3	0	10	4
4	2	7	10	0	4
5	7	5	4	4	0

Рисунок 8 – Елементи матриці суміжності графа, з якими буде працювати алгоритм Прима-Крускала

Далі у циклі поки всі вершини не будуть розфарбовані в один колір повторюємо зазначені нижче дії.

1 Ітерація циклу. Знаходимо у виділеній частині ребро з мінімальною вагою.

	1	**2**	**3**	**4**	**5**
1	0	3	5	2	7
2	3	0	3	7	5
3	5	3	0	10	4
4	2	7	10	0	4
5	7	5	4	4	0

Рисунок 9

Це ребро (1, 4), вага якого дорівнює 2. Зберігаємо назву ребра.

Позначаємо вузли 1 та 4 однаковим кольором – вони тепер частина майбутнього мінімального каркасного дерева, яке ми будуємо.

Вершина	1	2	3	4	5
Колір	1	2	3	1	5

Рисунок 10

2 Ітерація циклу. Знаходимо у виділеній частині ребро з мінімальною вагою серед ребер з різнокольровими вершинами.

	1	**2**	**3**	**4**	**5**
1	0	3	5	2	7
2	3	0	3	7	5
3	5	3	0	10	4
4	2	7	10	0	4
5	7	5	4	4	0

Рисунок 11

Це ребро (1, 2), вага якого дорівнює 2. Зберігаємо назву ребра.

Позначаємо вузли 1 та 2 однаковим кольором.

Вершина	1	2	3	4	5
Колір	1	1	3	1	5

Рисунок 12

(!) Якщо у результаті розфарбовування графу отримаємо два різні ланцюжки з двох різних кольорів, при їх з'єднанні ребром слід один ланцюжок перефарбувати повністю у колір іншого.

3 Ітерація циклу. Знаходимо у виділеній частині ребро з мінімальною вагою серед ребер з різнокольровими вершинами.

	1	**2**	**3**	**4**	**5**
1	0	3	5	2	7
2	3	0	3	7	5
3	5	3	0	10	4
4	2	7	10	0	4
5	7	5	4	4	0

Рисунок 13

Це ребро (2, 3), вага якого дорівнює 3. Зберігаємо назву ребра.

Позначаємо вузли 1 та 2 однаковим кольором.

Вершина	1	2	3	4	5
Колір	1	1	1	1	5

Рисунок 14

4 Ітерація циклу. Знаходимо у виділеній частині ребро з мінімальною вагою серед ребер з різнокольровими вершинами.

	1	**2**	**3**	**4**	**5**
1	0	3	5	2	7
2	3	0	3	7	5
3	5	3	0	10	4
4	2	7	10	0	4
5	7	5	4	4	0

Рисунок 15

Це ребро (3, 5), вага якого дорівнює 4. Зберігаємо назву ребра.

Позначаємо вузли 1 та 2 однаковим кольором.

Вершина	1	2	3	4	5
Колір	1	1	1	1	1

Рисунок 16

Тепер всі вузли графа розфарбовані в однаковий колір, тобто виконана умова виходу з циклу.

Після виконання циклу ми одержали наступний список ребер: (1, 4), (1, 2), (2, 3), (3, 5), на основі нього будуємо каркасне дерево зображене на рисунку 17.

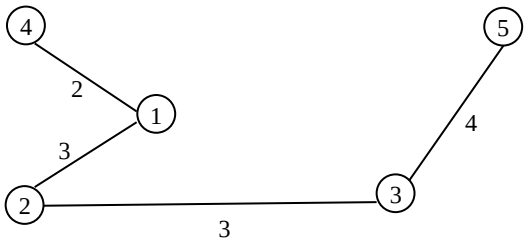


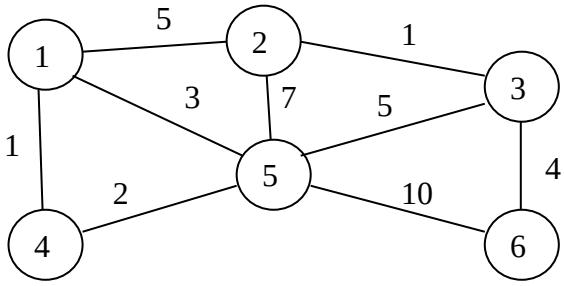
Рисунок 17 – Мінімальне каркасне дерево для графу з рисунку 2, його сумарна вага дорівнює 12

Результат: Оптоволоконну лінію потрібно провести між наступними парами сіл (1, 4), (1, 2), (2, 3), (3, 5). Необхідна довжина дроту рівна 12 умовним одиницям відстані.

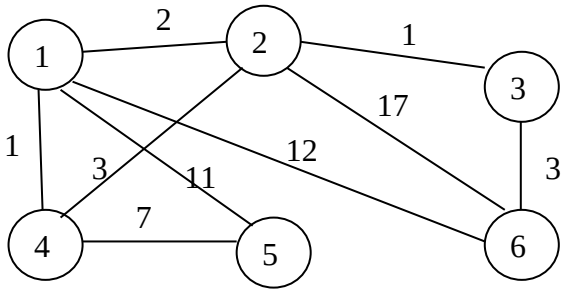
Завдання до лабораторної роботи:

Реалізувати програмно абстрактний тип даних неорієнтований граф. Створити його екземпляр, який заповнити даними, вказаними у Вашому варіанті. Розробити програму, що повинна побудувати для даного графу мінімальне каркасне дерево, використовуючи алгоритм Прима-Крускала, визначити сумарну вагу побудованого дерева, вивести одержані дані на екран.

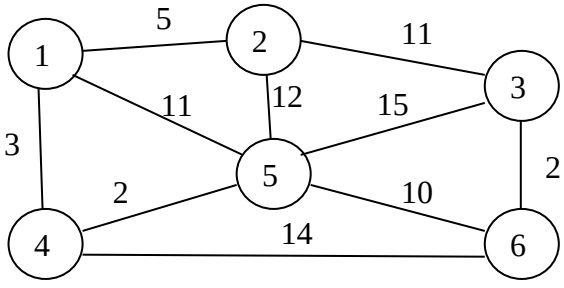
Варіант 1



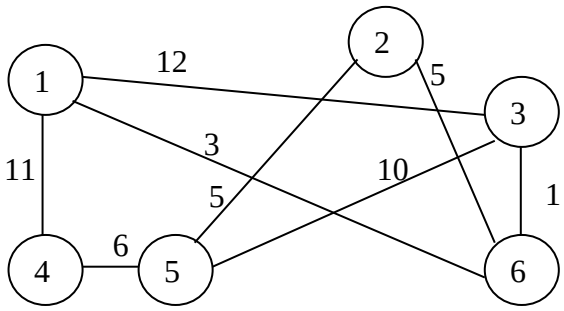
Варіант 2



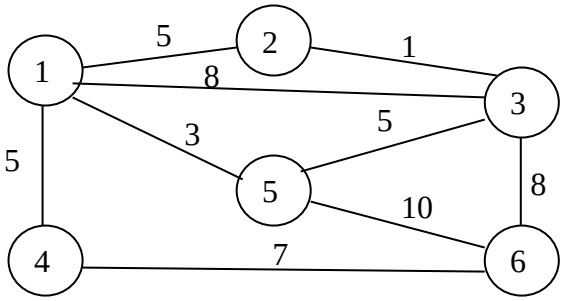
Варіант 3



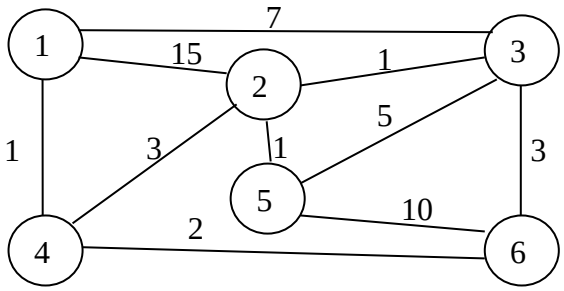
Варіант 4



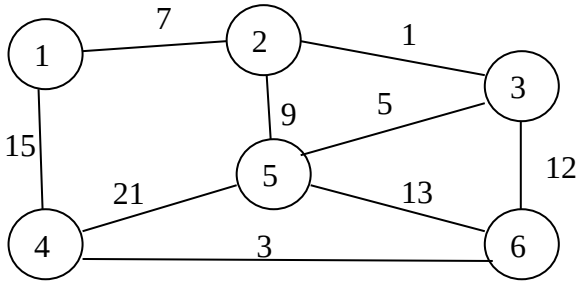
Варіант 5



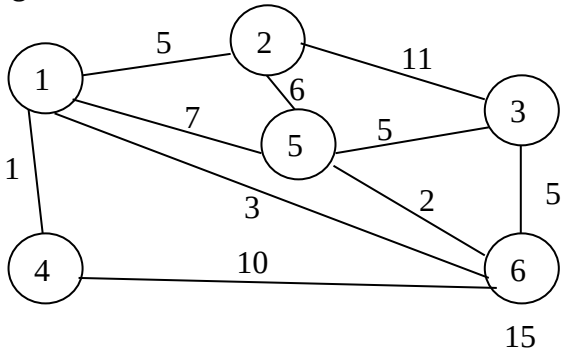
Варіант 6



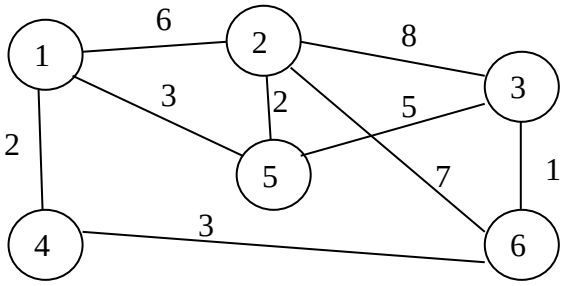
Варіант 7



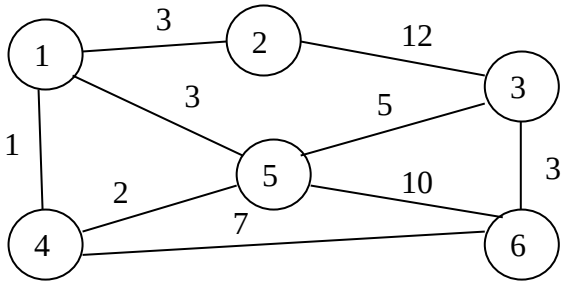
Варіант 8



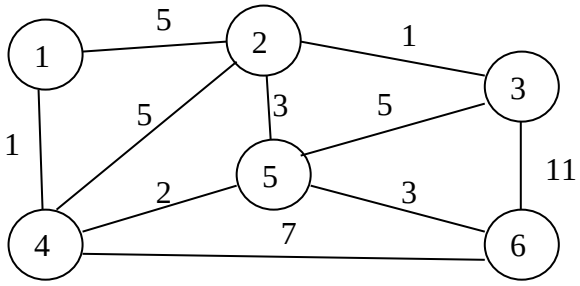
Варіант 9



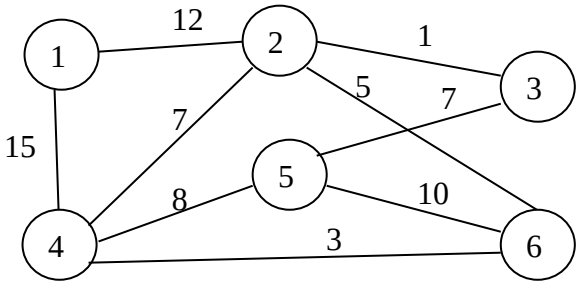
Варіант 10



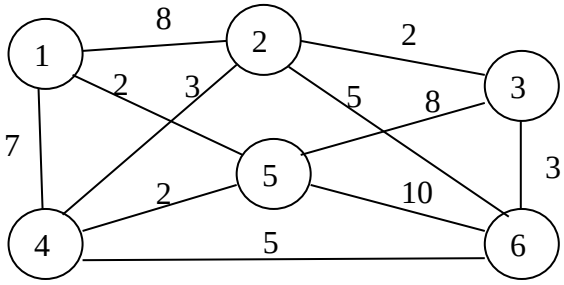
Варіант 11



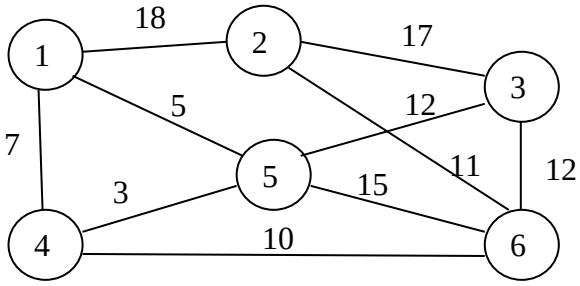
Варіант 12



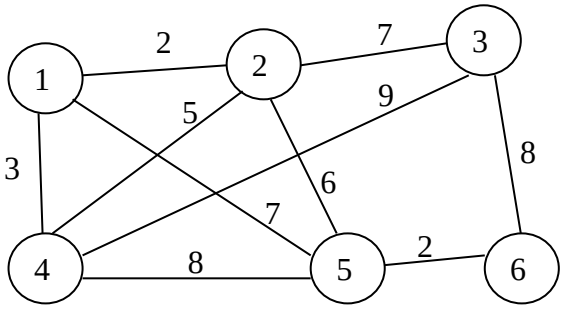
Варіант 13



Варіант 14



Варіант 15



Контрольні питання:

1. Дайте означення графа, вершини, ребра. Чим **зважений** граф відрізняється від **незваженого**?
2. Назвіть і коротко опишіть **три основні способи подання графів** у програмі: матриця суміжності, список ребер, списки суміжності. У чому перевага матриці суміжності?
3. Що таке **каркасне дерево** та **мінімальне каркасне дерево** графа? Яка величина мінімізується в алгоритмі Прима-Крускала?
4. У чому полягає основна ідея алгоритму **Прима-Крускала** при побудові мінімального каркасного дерева? Як використовується «розфарбування» вершин (множини/компоненти)?
5. Чому алгоритм Прима-Крускала застосовують тільки до **зв'язних неорієнтованих зважених графів**? Що станеться, якщо граф не є зв'язним?

Лабораторна робота № 2

Тема: Алгоритми роботи з графами. Алгоритм Дейкстри

Мета: Навчитися реалізовувати алгоритм Дейкстри для пошуку найкоротшого шляху

Алгоритм Дейкстри

В теорії графів, **задача про найкоротший шлях** полягає в знаходженні такого шляху між двома вершинами (або вузлами) графу, що сума ваг ребер, з яких він складається, мінімальна. Прикладом може бути знаходження найкоротшого шляху між двома пунктами на дорожній мапі; в цьому випадку, вершинами є пункти, а ребрами - відрізки дороги із вагами, рівними часу, необхідному для подолання цього відрізку. Інший приклад знаходження оптимального шляху проходження IP-пакету в мережі Інтернет. В такому випадку вузлами будуть маршрутизатори, ребрами – з'єднання між ними, а вагами ребер – пропускна здатність з'єднань.

Базові алгоритми знаходження найкоротших шляхів у зважених графах: алгоритм Дейкстри, алгоритм Флойда-Уоршелла, алгоритм Форда-Беллмана.

Принцип роботи алгоритму Дейкстри

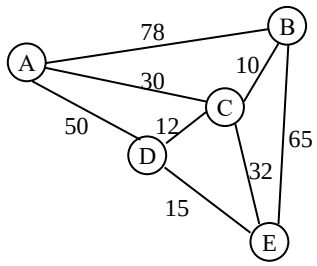
Алгоритм Дейкстри знаходить найкоротшу відстань від однієї вершини до всіх інших. Всі ваги ребер повинні бути додатними числами.

Кожній вершині графа ставиться у відповідність мітка mark, що містить мінімальну відстань від першої вершини до поточної вершини, мінімальний шлях від першої до поточної вершини зберігається в масиві L. На початку всі мітки дорівнюють ∞. Алгоритм полягає у тому, що на кожному кроці перебираються всі можливі варіанти просування по графу. При знаходженні більш короткого шляху від вершини V_1 до вершини V_2, значення мітки вершини V_2 замінюється на довжину цього шляху, і так доти, поки не буде пройдено весь граф. Вкінці роботи програми у масиві L будуть міститися найкоротші шляхи від першої вершини до всіх інших вершин.

Приклад 2. Розглянемо прикладну задачу, для рішення якої необхідно знайти найкоротшу відстань у графі між двома вершинами.

Логістичний відділ служби доставки X хоче визначити, яким шляхом швидше доставити посилки з міста A в місто E. Проаналізувавши карту доріг країни, спеціалісти логістичного відділу побудували граф, зображений на рис. 1.

а)

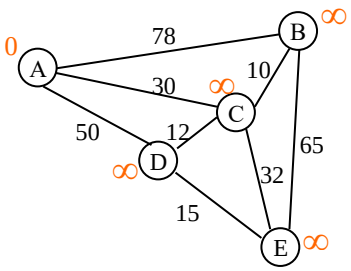


в)

	A	**B**	**C**	**D**	**E**
A	0	78	30	50	0
B	78	0	10	0	65
C	30	10	0	12	32
D	50	0	12	0	15
E	0	65	32	15	0

Рисунок 1 – а) граф, вершини якого міста, ребра – дороги між ними, а ваги ребер – довжини доріг; в) матриця суміжності наведеного графу

1 Крок. Привласнюємо кожній вершині мітку. Першій – 0, іншим – ∞. Створюємо масив L, в який будемо записувати шляхи від A до інших вершин та масив Mark для значення міток і інформації про закреслені мітки.



L =

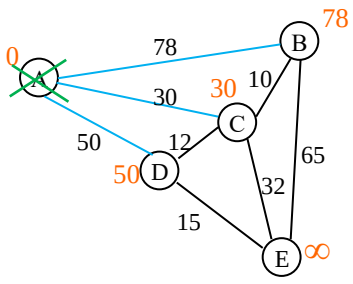
A	**B**	**C**	**D**	**E**
A				

Mark =

A	**B**	**C**	**D**	**E**
0	∞	∞	∞	∞
1	1	1	1	1

Рисунок 2

2 Крок. Присвоїмо міткам вершин, що знаходяться на відстані одного переходу від вершини A, довжини цих переходів, якщо вони менші за поточні значення мітки. Вкажемо нові маршрути та закреслимо вершину A.



L =

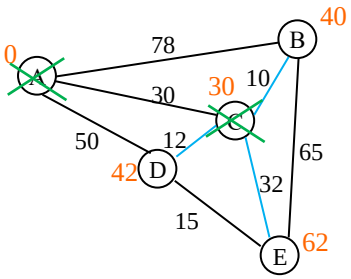
A	**B**	**C**	**D**	**E**
A	A	A	A	

Mark =

A	**B**	**C**	**D**	**E**
0	78	30	50	∞
0	1	1	1	1

Рисунок 3

3 Крок. Виберемо незакреслену вершину з найменшою міткою. Зараз це вершина C. Присвоїмо міткам вершин, що знаходяться на відстані одного переходу від вершини C, довжини цих переходів + значення мітки C, якщо ці значення менші за поточні значення відповідних міток. Вкажемо нові маршрути та закреслимо вершину C.



L =

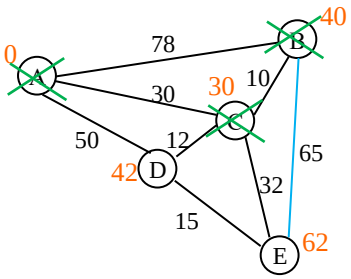
A	B	C	D	E
A	C	A	C	C

Mark =

A	B	C	D	E
0	40	30	42	62
0	1	0	1	1

Рисунок 4

4 Крок. Виберемо незакреслену вершину з найменшою міткою. Зараз це вершина B. Присвоїмо міткам вершин, що знаходяться на відстані одного переходу від вершини B, довжини цих переходів + значення мітки B, якщо ці значення менші за поточні значення відповідних міток. Нових маршрутів немає. Закреслимо вершину B.



L =

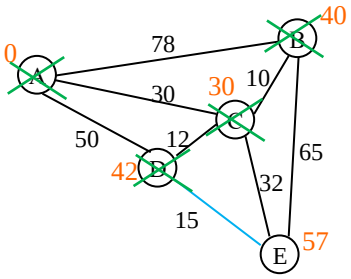
A	B	C	D	E
A	C	A	C	C

Mark =

A	B	C	D	E
0	40	30	42	62
0	0	0	1	1

Рисунок 5

5 Крок. Виберемо незакреслену вершину з найменшою міткою. Зараз це вершина D. Присвоїмо міткам вершин, що знаходяться на відстані одного переходу від вершини D, довжини цих переходів + значення мітки D, якщо ці значення менші за поточні значення відповідних міток. Вкажемо нові маршрути та закреслимо вершину D.



L =

A	B	C	D	E
A	C	A	C	D

Mark =

A	B	C	D	E
0	40	30	42	57
0	0	0	0	1

Рисунок 6

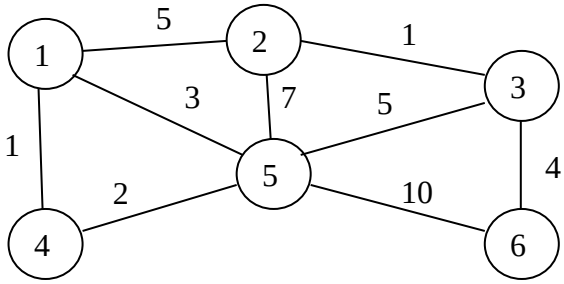
Результат: Матриця L вказує в яке місто через яке їхати. З неї можна у зворотному порядку прослідкувати найкоротший маршрут з будь-якого міста

до міста A. Найкоротший шлях від міста E до міста A лежить через міста D та C. Отже, значення необхідного маршруту: $A \to C \to D \to E$.

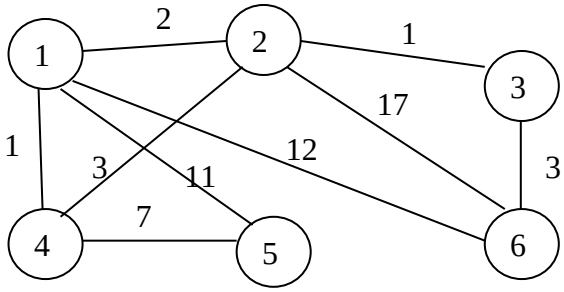
Завдання до лабораторної роботи:

Реалізувати програмно абстрактний тип даних неорієнтований граф. Створити його екземпляр, який заповнити даними, вказаними у Вашому варіанті. Розробити програму, що повинна визначити та вивести на екран найкоротший шлях від вершини 1 до вершини 6 за допомогою алгоритму Дейкстри.

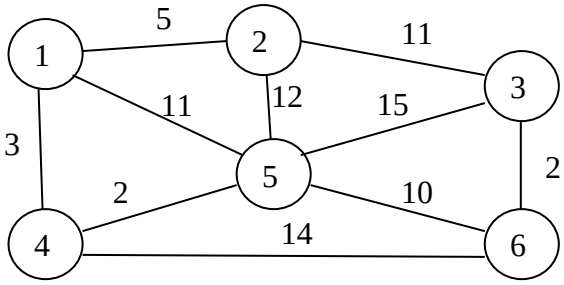
Варіант 1



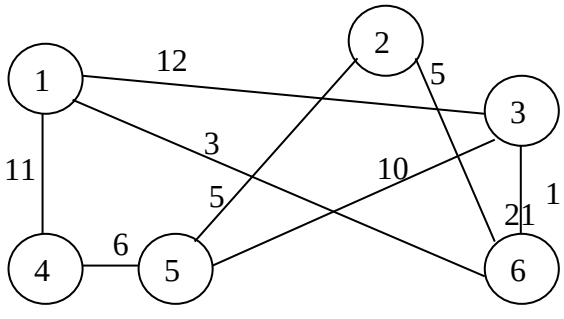
Варіант 2



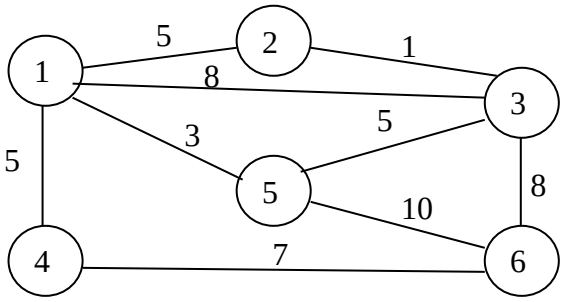
Варіант 3



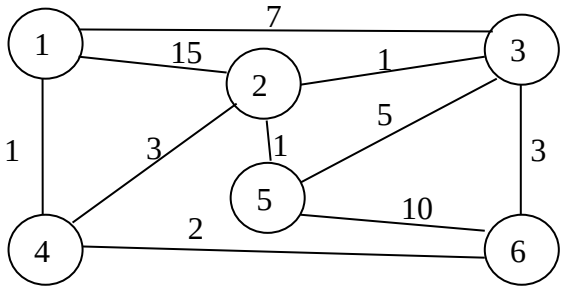
Варіант 4



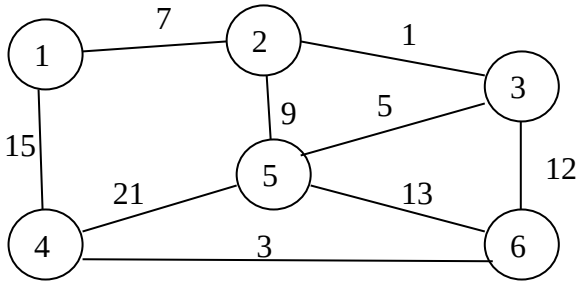
Варіант 5



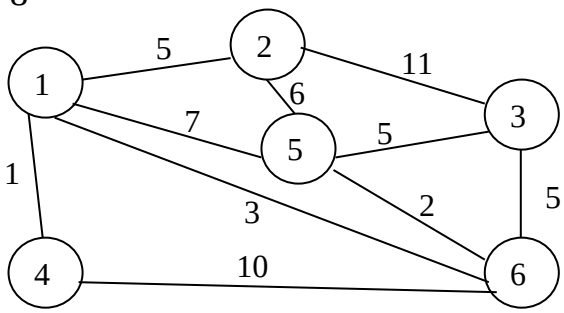
Варіант 6



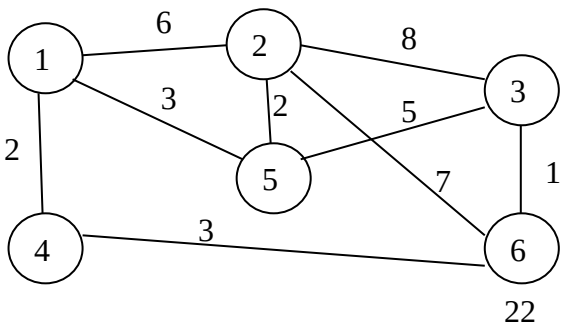
Варіант 7



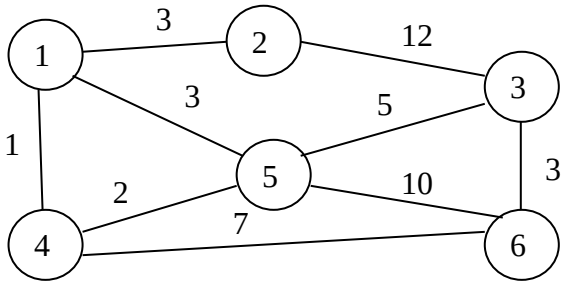
Варіант 8



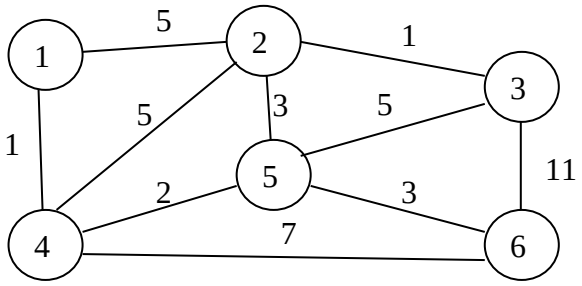
Варіант 9



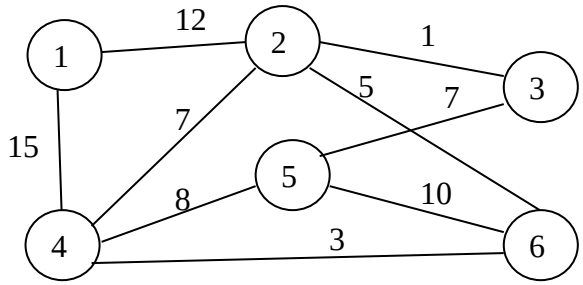
Варіант 10



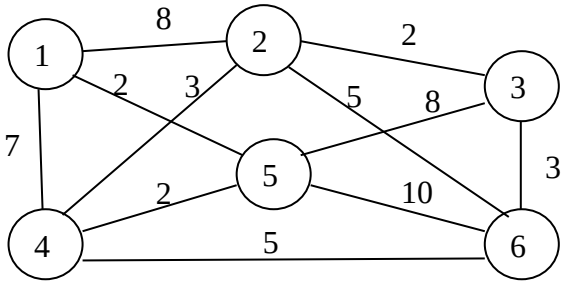
Варіант 11



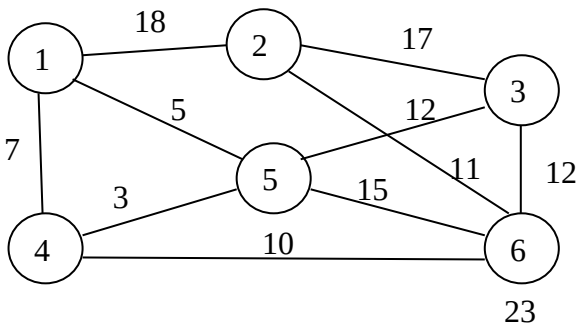
Варіант 12



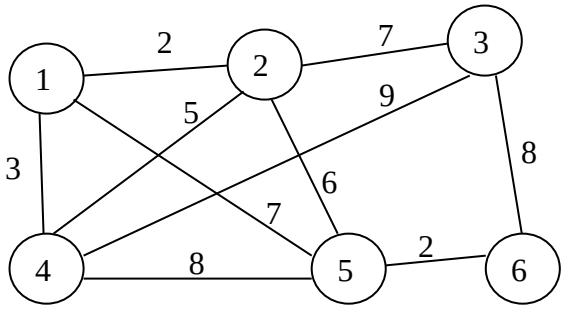
Варіант 13



Варіант 14



Варіант 15



Контрольні питання:

1. Сформулюйте задачу пошуку **найкоротшого шляху** в зваженому графі. Які обмеження накладаються на ваги ребер для алгоритму Дейкстри?
2. Що зберігають масиви **mark** та **L** в алгоритмі Дейкстри? Як ініціалізуються їхні значення на початку роботи?
3. Який вузол обирається на кожному кроці алгоритму Дейкстри і що з ним робиться після обробки всіх його суміжних вершин?
4. Як оновлюється мітка вершини v при переході з вершини u? Запишіть словесно формулу оновлення.
5. Як відновити сам **маршрут** (послідовність вершин) від початкової вершини до цільової, маючи заповнений масив L?

Лабораторна робота № 3

Тема: Генератори псевдовипадкових чисел

Мета: Навчитися реалізовувати алгоритми генерації псевдовипадкових чисел

Псевдовипадкові числа – числа, які використовуються замість випадкових чисел. Псевдовипадкові числа отримують в ЕОМ програмним способом використовуючи деяке рекурентне співвідношення.

Це значить, що кожне наступне число α_{k+1} утворюють із попереднього α_k (або групи попередніх чисел), використовуючи деякий алгоритм, який використовує арифметичні та логічні операції.

Часто для створення унікальної послідовності псевдовипадкових чисел початковий її член ініціалізують, наприклад, остачею від ділення поточного часу в мілісекундах на певне число.

Генератори псевдовипадкових чисел широко використовуються в техніці і науці, а також в багатьох криптографічних застосуваннях, наприклад при генерації ключів та загальносистемних параметрів. На сьогодні розроблена достатньо велика кількість різних типів генераторів псевдовипадкових послідовностей.

Мінімальний генератор Парка-Міллера

Найпростіша послідовність, яку можна запропонувати для реалізації генератора рівномірного розподілу:

$$I(j+1)=a*I(j)(\text{mod } m)$$

при відповідному виборі констант. Константи, які були запропоновані Парком і Міллером:

$$a=7^5=16807,\ m=2^{31}-1=2147483647$$

Пряма реалізація цього методу можлива на мовах асемблера, але мови високого рівня можуть при цьому зафіксувати переповнення. Для обходу цього Scharge запропонував метод часткової факторизації модуля. Модуль розкладається у вираз:

$$m=a*q+r$$

Якщо **r<q** і **0<z< m-1**, то при цьому величини **a*(z mod q)** і **r*[z/q]** завжди лежать в інтервалі 0,..., m-1. Для множення **(a*z)(mod m)** при цьому використовується алгоритм:

- **t = a(z mod q)-r[z/q].**
- **якщо t<0, то t += m.**
- **(a*z)(mod m)=t.**

У випадку констант Парку-Міллера можна використовувати q=12773 і r=2836.

Алгоритм Л'Екюера, що комбінує дві послідовності

Якщо потрібне число викликів, що перевищує по порядку 10^8, то для цього випадку Л'Екюер рекомендує комбінувати виведення двох послідовностей із близькими, але відрізняючимися константами. У його дослідженнях гарний результат був отриманий для значень:

m1=2147483563, a1=40014, q1=53668, r1=12211; m2=2147483399, a2=40692, q2=52774, r2=3791.

Лінійний конгруентний метод

Даний алгоритм був запропонований Д. Х. Лемером в 1948 році. Рекурентна формула виглядає в такий спосіб:

$x_n = (ax_{n-1} + c) \bmod m.$

Період породжуваної послідовності не перевищує m. Природно, що від вибору параметрів a, c, m і значення першого члена x_0 істотно залежать основні властивості породжуваної послідовності. Довжина періоду буде максимальною (рівною m) тільки в тому випадку, коли:

- НСД(c, m) = 1 (тобто c і m взаємно прості);
- $a - 1$ кратно всім простим дільникам m;
- якщо m кратно 4, то й $(a - 1) \bmod 4 = 0$.

Як приклад можна взяти наступні значення параметрів:

- $m = 2^{16}$ або 2^{32} (для простоти реалізації),
- a = 1 664 525,
- c = 1 013 904 223.

Алгоритм Блюма-Блюма-Шуба

В 1986 році троє авторів Ленор Блюм, Мануель Блюм і Майкл Шуб запропонували алгоритм генерації псевдовипадкової послідовності, стійкий до зворотних перетворень. Основна рекурентна формула алгоритму:

$x_n = (x_{n-1})^2 \bmod pq,$

$z_n = parity\ (x_n),$

де p і q — два великих простих числа. Для підвищення якості одержуваної послідовності на черговому кроці вибираються не всі біти x_n, а тільки молодші, або навіть тільки біт парності. З отриманих «випадкових бітів» формуються двійкові псевдовипадкові числа довільної розрядності. Однією з особливостей обчислювальної формули є наскрізна можливість обчислити x_n без генерації всіх попередніх членів послідовності.

$x_n = (x_0)^{2n \bmod (p-1)(q-1)} \bmod pq,$

Даний алгоритм більш вимогливий до обчислювальних ресурсів, але, з іншого боку, має гарні статистичні характеристики.

Завдання: Реалізувати розглянуті генератори псевдовипадкових чисел.

Контрольні питання:

1. Що таке **псевдовипадкові числа**? Чим вони відрізняються від “справжніх” випадкових чисел і яку роль відіграє початкове значення (зерно, seed)?

2. Запишіть загальну рекурентну формулу **лінійного конгруентного генератора**. Від яких параметрів залежить період послідовності і які умови забезпечують максимальний період?

3. Як виглядає рекурентна формула **мінімального генератора Парка–Міллера**? Для чого використовується прийом Шарга (Scharage) при його реалізації?

4. У чому ідея комбінування **двох послідовностей Л’Екюера**? Навіщо взагалі об’єднувати два генератори з різними параметрами?

5. Запишіть основну формулу генератора **Блюма–Блюма–Шуба**. Чому цей генератор вважається обчислювально складнішим, але більш придатним для криптографічних застосувань?

Лабораторна робота № 4

Тема: Тестування генераторів псевдовипадкових чисел

Мета: Навчитися реалізовувати алгоритми тестування генераторів псевдовипадкових чисел

Тестування

В основі всіх тестів лежать наступні принципи, необхідно генерувати велику кількість випадкових чисел з діапазону від 0.0 (включно) до 1.0 (виключно). Одержувані в результаті роботи генераторів значення будуть розбиватися на декілька категорій, буде підраховуватися кількість значень у кожній категорії, а потім ймовірність влучення значення в кожну категорію. На основі результатів обчислень буде визначатися значення функції хі-квадрат, на основі якого буде прогонятися тест за критерієм хі-квадрат. При цьому кількість ступенів свободи буде на одиницю менше, ніж кількість категорій значень.

Тест на однорідність

Найпростіший тест - перевірка на однорідність. Фактично випадкові числа будуть перевірятися на рівномірність розподілу по діапазону від 0.0 до 1.0. Розіб'ємо весь діапазон на 100 піддіапазонів, сформуємо набір з 1000000 випадкових чисел і обчислимо кількість значень, що потрапили в кожний піддіапазон. У піддіапазоні 0 будуть перебувати значення від 0.00 до 0.01, у піддіапазоні 1 - значення від 0.01 до 0.02 і т.д. Ймовірність влучення випадкового числа в будь-який піддіапазон становить 0.01. Для отриманого розподілу обчислимо значення параметра хі-квадрат і порівняємо його з даними для стандартного розподілу хі-квадрат, що перебувають у рядку, для 99 ступенів свободи.

Примітка. Критерій хі-квадрат (або критерій узгодженості Пірсона) використовується для перевірки гіпотези про закон розподілу. Значення хі-квадрат обчислюється за формулою:

$$\rho(X) = \sum_{j=1} k \frac{(\nu_j - np_j)^2}{np_j}$$

де, k – повний об’єм вибірки, p_j – теоретичне значення ймовірності потрапляння в і-тий інтервал, v_j - відповідне емперичне значення, n –кількість дослідів.

Таблиця 6.2. Процентні точки розподілу хі-квадрат

	1%	5%	95%	99%
V = 1	0.000157	0.00393	3.84	6.63
V = 2	0.0201	0.103	5.99	9.21
V = 3	0.115	0.352	7.81	11.3
V = 4	0.297	0.711	9.49	13.3
V = 5	0.554	1.15	11.1	15.1
V = 6	0.872	1.64	12.6	16.8

Вид таблиці злегка лякає, але зрозуміти її зовсім не складно. Значення, наведені в таблиці, являють собою значення розподілу хі-квадрат для v

ступенів свободи (грецька буква v - це стандартний символ для позначення ступенів свободи). У вільній інтерпретації можна сказати, що значення ступенів свободи на одиницю менше кількості можливих типів подій. Наприклад, при киданні двох монет можливі три типи подій: "дві решки", "орел і решка" і "два орли". Отже, для цього експерименту кількість ступенів свободи буде дорівнює 2. Рядок для v = 2 містить чотири значення - по одному значенню в кожному із чотирьох стовпців. Значення в стовпці 1% (0.0201) можна інтерпретувати в такий спосіб: "Значення критерію Хі повинне бути менше 0.0201 тільки 1% часу". Інакше кажучи, при повторенні експерименту 100 разів тільки приблизно в одному з них буде отримане значення Хі, менше 0.0201. Якщо буде виявлено, що в багатьох експериментах буде отримане значення менше 0.0201, можна буде з досить високим ступенем упевненості сказати, що кидання монет не є випадковою подією, тобто монети мають зміщений центр ваги. Те ж саме можна сказати й для стовпця 5%. Про стовпець 95% можна сказати, що значення параметра Хі повинне бути менше 5.99 приблизно 95% часу або, що еквівалентно, значення параметра Хі повинне бути більше 5.99 приблизно 5% часу. Аналогічні міркування справедливі й для стовпця 99%.

Приклад тесту на однорідність:

```
procedure UniformityTest(RandGen : TtdBasePRNG;
                                    var ChiSquare : double;
                                    var DegsFreedo : integer);
var
   BucketNumber,
   i : integer;
   Expected, ChiSqVal : double;
   Bucket : array [0..pred(UniformityIntervals)] of integer;
begin

//обчислити кількість цифр в кожному піддіапазоні
   FillChar(Bucket, sizeof(Bucket), 0);
   for i:=0 to pred(UniformityCount) do begin
   BucketNumber := trunc(RandGen.AsDouble* UniformityIntervals);
   inc(Bucket[BucketNumber]);
   end;

//обчислити значення параметра хі-квадрат
Expected := UniformityCount / UniformityIntervals;
ChiSqVal := 0.0;
for i:=0 to pred(UniformityIntervals) do
   ChiSqVal := ChiSqVal + (Sqr(Expected – Bucket[i]) / Expected);

//повернути значення
ChiSquare := ChiSqVal;
```

```
    DegsFreedom := pred(UniformityIntervals)
end;
```

Завдання: Протестувати отримані за допомогою розроблених у попередній лабораторній роботі генераторів випадкових чисел послідовності на однорідність.

Контрольні питання:

1. Для чого взагалі проводять тест на однорідність для генераторів псевдовипадкових чисел? Що саме він перевіряє?
2. Чому інтервал [0.0; 1.0) ділять на рівні піддіапазони (інтервали)? Що таке Bucket і що в нього записується під час тесту?
3. Що означає очікувана кількість попадань в кожний піддіапазон і як вона обчислюється (через кількість чисел та кількість інтервалів)?
4. Запишіть словесно, як обчислюється значення статистики χ^2 (хі-квадрат) за результатами експерименту. Від яких величин вона залежить?
5. Що таке ступені свободи в критерії χ^2 та чому для тесту з k інтервалами вони дорівнюють k – 1? Як інтерпретуються граничні значення з таблиці розподілу χ^2 (1%, 5%, 95%, 99%)?

Лабораторна робота № 5

Тема: Колаборативна фільтрація для реалізації рекомендаційних систем
Мета: Навчитися реалізовувати алгоритми колаборативної фільтрації

Методи колаборативної фільтрації (collaborative filtering) – застосовують інформацію про рейтинги, які користувачі виставляють об’єктам, та засновані на визначенні схожості користувачів чи об’єктів.

Методи колаборативної фільтрації, що використовують моделі сусідства, здійснюють прогнозування вподобань користувачів на основі ступеню сусідства між елементами рекомендаційної системи. Ступінь сусідства визначається за допомогою коефіцієнтів подоби, обчислення яких здійснюється на основі різних параметрів та метрик.

Тож першим кроком методів, заснованих на колаборативній фільтрації, є обчислення *коефіцієнтів подоби* користувачів та/або об’єктів системи, обчислення яких здійснюються найчастіше на основі оцінок, які користувачі виставляють об’єктам. Крім оцінок можуть використовуватися дані про здійснені перегляди, поставлені теги, написані коментарі тощо.

У найбільш простому випадку збираються дані про поставлені користувачами оцінки та записуються у матрицю рейтингів (рис. 1).

	Об’єкт 1	**Об’єкт 2**	**…**	**Об’єкт n**
Користувач 1	5	3	…	3
Користувач 2	4	–	…	2
…	…	…	…	…
Користувач m	–	4	…	4

Рис. 1. Матриця рейтингів

В матриці рейтингів значення є оцінками конкретного користувача для конкретного об’єкту. Відсутні значення в таблиці рейтингів є невідомими, тобто, для певного об’єкту певний користувач не виставив оцінку.

Розглянемо метрики, що можуть використовуватися для визначення коефіцієнтів подоби у колаборативній фільтрації.

Якщо об’єкт (або користувач), що описується m ознаками, представити точкою у k-мірному просторі, то подібність об'єктів один з одним буде визначатися як відстань в даному метричному просторі. У випадку з матрицею рейтингів таке представлення можливе – ознаками будуть в такому разі оцінки об’єктам. Найбільш поширені метрики подоби, що використовуються в такому випадку: евклідова відстань (1), кореляція Пірсона (2) або косинусна подоба (3):

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{m} (x_{1i} - x_{2i})^2}, \quad (1)$$

$$d(x_1, x_2) = \frac{\sum_{i=1}^{m}(x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)}{\sqrt{\sum_{i=1}^{m}(x_{1i} - \bar{x}_1)^2}\sqrt{\sum_{i=1}^{m}(x_{2i} - \bar{x}_2)^2}}, \quad (2)$$

$$d(x_1, x_2) = \frac{\bar{X}_1 \cdot \bar{X}_2}{\|\bar{X}_1\| \cdot \|\bar{X}_2\|} = \frac{\sum_{i=1}^{m} x_{1i} \cdot x_{2i}}{\sqrt{\sum_{i=1}^{m}(x_{1i})^2}\sqrt{\sum_{i=1}^{m}(x_{2i})^2}}, \quad (3)$$

де $d(x_1, x_2)$ – відстань між об'єктами x_1 та x_2; x_{1i}, x_{2i} – значення i-ї ознаки відповідно у 1-го та 2-го об'єкту; w_i – вага, що привласнюється i-ій змінній; $\sum^{-1}$ – матриця зворотна коваріаційній матриці, розрахованій по всій вибірці; $\bar{X}_1$, $\bar{X}_2$ – вектори значень ознак у 1-го та 2-го об'єкту; $\bar{x}_1$, $\bar{x}_2$ – середні значення ознак відповідно у 1-го та 2-го об'єкту; m – кількість ознак.

Методи колаборативної фільтрації на основі моделі сусідства поділяється на два види:

1. Засновані на схожості користувачів (User/User, User-based).

2. Засновані на схожості об’єктів (Item/Item, Item-based).

Колаборативна фільтрація, заснована на схожості користувачів (user-based). Розглянемо найпростіший спосіб реалізації колаборативної фільтрації, заснованої на схожості користувачів. Всі інші способи є ускладненнями даного.

Після того, як для рекомендаційної системи зібрані дані та побудована матриця користувачів-об’єктів, для формування рекомендацій певному користувачу необхідно здійснити наступну послідовність дій:

1. Обчислити множину коефіцієнтів подоби даного користувача з усіма іншими користувачами (4):

$$K_i = \{k_{i,1}, k_{i,2}, \ldots, k_{i,j}\}, \quad (4)$$

де $k_{i,j}$ – коефіцієнт подоби між i-тим та j-тим користувачами.

2. Ранжувати користувачів відносно i-того користувача за ступенем подоби на нього. Обрати TopN найбільш схожих на нього користувачів або усіх користувачів, для яких вдалося обчислити коефіцієнт подоби.

3. Обчислити для об’єктів системи, які ще не переглядав (не оцінював) i-тий користувач, зважену суму оцінок інших користувачів (5):

$$S_{i,q}=\sum_{j=1}^{n} r_{q,j} \cdot k_{i,j} \ , \tag{5}$$

де $r_{q,j}$ – оцінка q-того об'єкту, поставлена j-тим користувачем; n – довжина списку TopN схожих на i-того користувача користувачів.

4. Розділити кожну зважену суму на суму коефіцієнтів подоби всіх користувачів, що ставили оцінки відповідному об'єкту (6), щоб об'єкти, що отримали більше оцінок не одержали перевагу:

$$\tilde{r}_{i,q}=\frac{S_{i,q}}{\sum_{j=1}^{n} k_{i,j}} . \tag{6}$$

Рівняння (6) дає прогноз оцінки i-того користувача q-тому об'єкту.

5. Відсортувати множину об'єктів, одержану після третього кроку на основі значень прогнозованих оцінок та рекомендувати користувачу перші N об'єктів у списку.

В реальних системах для кожної рекомендації неможливо використовувати в обчисленнях дані всіх сотень тисяч чи навіть мільйонів користувачів системи, тому у формулах використовуються K найближчих сусідів – користувачів максимально схожих на користувача, якому обчислюються рекомендації (згаданий вище TopN).

Один з варіантів цієї групи методів, що має назву алгоритм GroupLens, замість кроків 3 та 4 розглянутих вище, для прогнозування оцінки використовує наступну формулу:

$$\tilde{r}_{i,q}=\bar{r}_i+\frac{\sum_{j=1}^{n}\left(r_{j,q}-\bar{r}_j\right)\cdot k_{i,j}}{\sum_{j=1}^{n}\left|k_{i,j}\right|}, \tag{7}$$

де $\bar{r}_i$ – середня оцінка i-того користувача; $\bar{r}_j$ – середня оцінка j-того користувача.

Методи колаборативної фільтрації засновані на схожості користувачів потребують постійних перерахунків коефіцієнтів подоби, так як дані про дії користувачів постійно оновлюються і їх вподобання з часом змінюються.

Колаборативна фільтрація, заснована на схожості об'єктів (item-based). Основна ідея даних методів полягає у тому, щоб для кожного об'єкту заздалегідь визначити множину схожих на нього об'єктів. Тоді для формування рекомендацій певному користувачу достатньо буде знайти ті об'єкти, яким він

поставив найбільші оцінки, та створити зважений список N об'єктів, максимально схожих на них. Результати порівняння об'єктів змінюються не так часто, як результати порівняння користувачів. Тож на першому кроці необхідно дослідити всі наявні дані, а подальші перерахунки можна робити рідко, вибираючи моменти часу, коли навантаження на веб-сайт мінімальне.

Методи колаборативної фільтрації засновані на схожості об'єктів працюють швидше, ніж методи засновані на схожості користувачів, так як багато обчислень можна здійснити заздалегідь. Тож їх можна з успіхом використовувати на великих об'ємах даних.

Для прогнозування оцінки на основі даного підходу треба знайти зважене середнє для оцінених користувачем об'єктів:

$$\tilde{r}_{i,q}=\bar{r}_i+\frac{\sum_{p=1}^{n}\left(r_{j,p}-\bar{r}_i\right)\cdot k_{q,p}}{\sum_{p=1}^{n}\left|k_{q,p}\right|}, \qquad (8)$$

де $k_{q,p}$ – коефіцієнт кореляції між q-тим об'єктом та p-тим об'єктом; n – довжина списку схожих на q-тий об'єкт об'єктів.

Використовуючи методи колаборативної фільтрації засновані на схожості об'єктів можна рідше перераховувати коефіцієнти подоби, ніж в методах заснованих на схожості користувачів, адже коефіцієнти подоби для об'єктів змінюються рідше, ніж для користувачів. Також Item-based методи мають більшу стійкість до інформаційних атак, ніж User-based методи.

Методи колаборативної фільтрації на основі моделі сусідства не потребують використання складних ресурсозатратних алгоритмів, показують високу точність прогнозування вподобань для користувачів, які уже виставили певну, мінімально необхідну, кількість оцінок об'єктам системи. Чим більше оцінок користувач виставить об'єктам системи, тим точніше ці методи сформують для нього списки рекомендацій. Недоліками цих методів є вразливість до проблеми холодного старту та інформаційних атак.

Завдання: Реалізувати алгоритми колаборативної фільтрації на основі подоби користувачів та на основі подоби предметів. Як вхідні дані для алгоритмів використати відкритий набір даних MovieLens Datasets.

Контрольні питання:

1. Що таке **колаборативна фільтрація** і яку інформацію вона використовує для побудови рекомендацій?

2. У чому принципова різниця між підходами **User-based** та **Item-based** колаборативної фільтрації?

3. Які метрики подібності (міри схожості) можна використовувати для обчислення коефіцієнтів подоби між користувачами/об’єктами? Назвіть щонайменше дві.

4. Як за формулами колаборативної фільтрації обчислюється **прогнозована оцінка** користувача для об’єкта (яку ідею використовують формули (5)/(6) або (7)/(8))? Сформулюйте словами.

5. Назвіть дві основні **переваги** та два **недоліки** методів колаборативної фільтрації на основі моделі сусідства (user/item-based).

Лабораторна робота № 6

Тема: Штучні нейронні мережі. Моделювання формальних логічних функцій. Прогнозування часових рядів

Мета: Отримати початкові навички по створенню штучних нейронних мереж, що здатні виконувати прості логічні функції, та нейронних мереж, що здатні прогнозувати часові ряди.

Теоретичні відомості

Штучний нейрон – вузол штучної нейронної мережі, що є спрощеною моделлю природного нейрона.

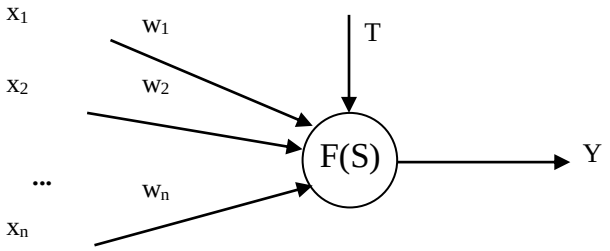


Рисунок 1 – Схема штучного нейрона

x_1- x_n – входи нейрона (синапси);
w_1- w_n – вагові коефіцієнти входів;
S – зважена сума входів нейрона;
F(S) – функція активації нейрона;
T – порогове значення (значення, після якого нейрон переходить у стан збудження), є не у всіх типів штучних нейронів;
Y – вихід нейрона (аксон).

Зважена сума S обчислюється за наступною формулою:

$$S = x_1 \cdot w_1 + x_2 \cdot w_2 + ... + x_n \cdot w_n. \quad (1)$$

Функція активації F(S) – визначає залежність сигналу на виході нейрона від зваженої суми сигналів на його входах. Використання різних функцій активації дозволяє вносити нелінійність в роботу нейрона і в цілому нейронної мережі.

Приклади функцій активації

Лінійна передавальна функція:

$$F(S) = \begin{cases} 0 \ if \ S \le 0 \\ 1 \ if \ S \ge 1 \\ S \ else \end{cases}, \quad (2)$$

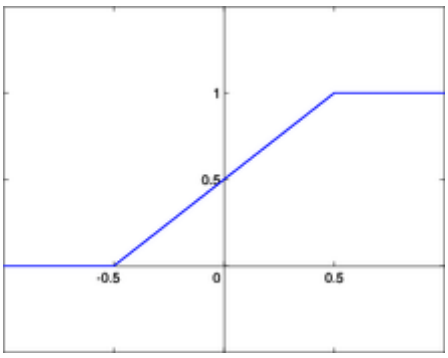


Рисунок 2 – Лінійна передавальна функція

Порогова передавальна функція:

$$F(S) = \begin{cases} 1 \ if \ \ S \geq 0 \\ 0 \ \ else \end{cases}, \tag{3}$$

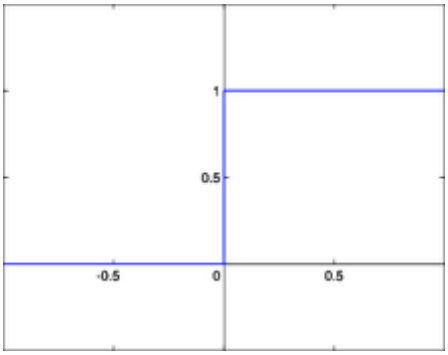


Рисунок 3 – Порогова передавальна функція

Сигмоїдальна передавальна функція:

$$F(S) = \frac{1}{(1+\exp(-S))}, \tag{4}$$

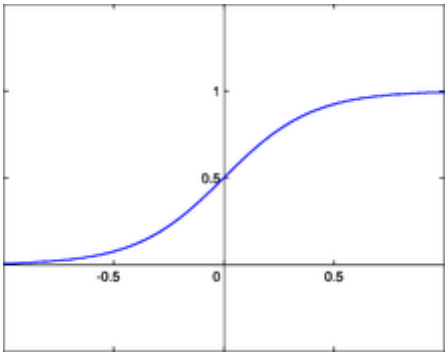


Рисунок 4 – Сигмоїдальна передавальна функція

Моделювання формальних логічних функцій за допомогою нейронів та нейронних мереж
Моделювання логічної функції "I" (AND)

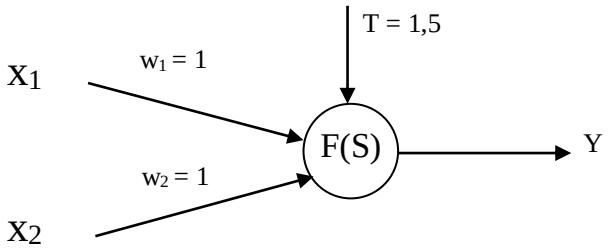


Рисунок 5 – Схема штучного нейрону, налаштованого на моделювання логічної функції "I"

Функція активації даного нейрона:

$$F(S) = \begin{cases} 0 \ if \ S < 1{,}5 \\ 1 \ if \ S >= 1{,}5 \end{cases} \quad (5)$$

Таблиця 1 – Таблиця істинності логічної функції "I" (AND)

x_1	x_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Розглянемо як обчислюється вихідний сигнал даного нейрона при різних вхідних даних:

$x_1 = 0$; $x_2 = 0$
S = 0·1+0·1 = 0
Y = F(S) = 0 (тому що S < 1,5)

$x_1 = 0$; $x_2 = 1$
S = 0·1+1·1 = 1
Y = F(S) = 0 (тому що S < 1,5)

$x_1 = 1$; $x_2 = 0$
S = 1·1+0·1 = 1
Y = F(S) = 0 (тому що S < 1,5)

$x_1 = 1$; $x_2 = 1$
S = 1·1+1·1 = 2
Y = F(S) = 1 (тому що S > 1,5)

Моделювання логічної функції "АБО" (OR)

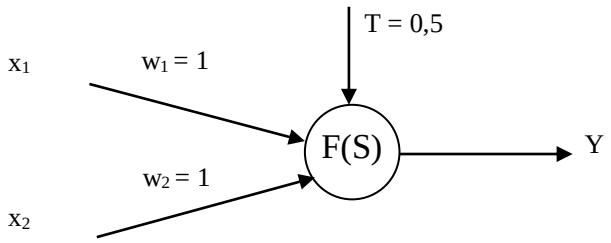


Рисунок 6 – Схема штучного нейрону, налаштованого на моделювання логічної функції "АБО"

Функція активації даного нейрона:

$$F(S) = \begin{cases} 0 \ \ f \ \ S < 0{,}5 \\ 1 \ \ f \ \ S >= 0{,}5 \end{cases} \tag{6}$$

Таблиця 2 – Таблиця істинності логічної функції "АБО" (OR)

x_1	x_2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Розглянемо як обчислюється вихідний сигнал даного нейрона при різних вхідних даних:

$x_1 = 0$; $x_2 = 0$
$S = 0 \cdot 1 + 0 \cdot 1 = 0$
$Y = F(S) = 0$ (тому що $S < 0{,}5$)

$x_1 = 0$; $x_2 = 1$
$S = 0 \cdot 1 + 1 \cdot 1 = 1$
$Y = F(S) = 1$ (тому що $S > 0{,}5$)

$x_1 = 1$; $x_2 = 0$
$S = 1 \cdot 1 + 0 \cdot 1 = 1$
$Y = F(S) = 1$ (тому що $S > 0{,}5$)

$x_1 = 1$; $x_2 = 1$
$S = 1 \cdot 1 + 1 \cdot 1 = 2$
$Y = F(S) = 1$ (тому що $S > 0{,}5$)

Моделювання логічної функції "НІ" (NOT)

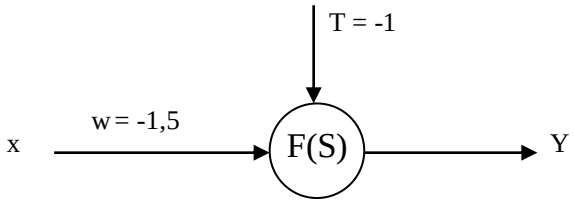


Рисунок 7 – Схема штучного нейрону, налаштованого на моделювання логічної функції "НІ"

Функція активації даного нейрона:

$$F(S) = \begin{cases} 0 \ if \ S < -1 \\ 1 \ if \ S >= -1 \end{cases} \quad (7)$$

Таблиця 3 – Таблиця істинності логічної функції "НІ" (NOT)

x	Y
0	1
1	0

Розглянемо як обчислюється вихідний сигнал даного нейрона при різних вхідних даних:

x = 0
S = 0·(-1,5) = 0
Y = F(S) = 1 (тому що S > -1)

x = 1
S = 1·(-1,5) = -1,5
Y = F(S) = 0 (тому що S < -1)

Моделювання логічної функції "Виключне АБО" (XOR)

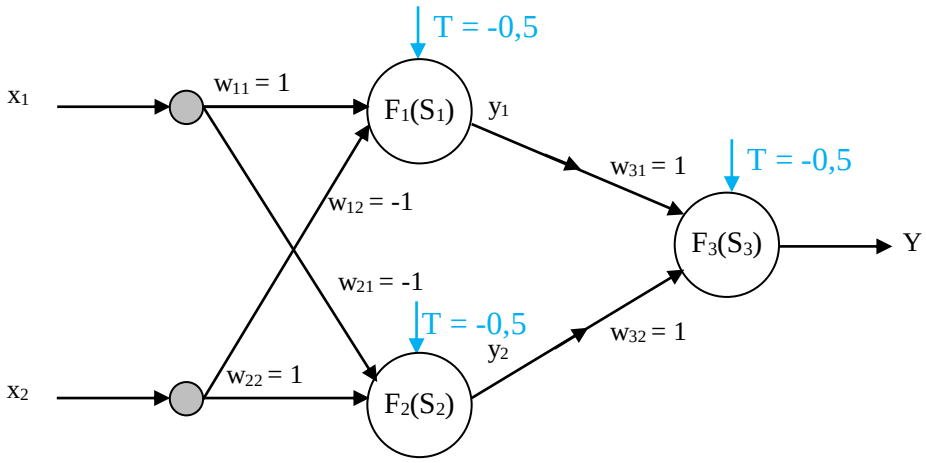


Рисунок 8 – Схема штучної нейронної мережі, налаштованої на моделювання логічної функції "Виключне АБО" (XOR)

Таблиця 4 – Таблиця істинності логічної функції "Виключне АБО" (XOR)

x_1	x_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Функція активації даного нейрона:

$$F(S) = \begin{cases} 0 \ if \ S < 0{,}5) \\ 1 \ if \ S >= 0{,}5 \end{cases} \quad (8)$$

Розглянемо як обчислюється вихідний сигнал даної мережі при різних вхідних даних:

$x_1 = 1$; $x_2 = 1$
$S_1 = 1 \cdot 1 + 1 \cdot (-1) = 0$
$Y_1 = F(S) = 0$ (тому що $S < 0{,}5$)
$S_2 = 1 \cdot (-1) + 1 \cdot 1 = 0$
$Y_2 = F(S) = 0$ (тому що $S < 0{,}5$)
$S_3 = 0 \cdot 1 + 0 \cdot 1 = 0$
$Y_3 = F(S) = 0$ (тому що $S < 0{,}5$)

$x_1 = 0$; $x_2 = 1$
$S_1 = 0 \cdot 1 + 1 \cdot (-1) = -1$
$Y_1 = F(S) = 0$ (тому що $S < 0{,}5$)
$S_2 = 0 \cdot (-1) + 1 \cdot 1 = 1$
$Y_2 = F(S) = 1$ (тому що $S > 0{,}5$)
$S_3 = 0 \cdot 1 + 1 \cdot 1 = 1$
$Y_3 = F(S) = 1$ (тому що $S > 0{,}5$)

Прогнозування часових рядів за допомогою нейрону з сигмоїдальною функцією активації

Задача прогнозування часових рядів, в яких є певні закономірності, може бути вирішена за допомогою нейромережі, яка може навчатися. Відомо, що людський мозок здатний до самонавчання, причому досягає успіхів найчастіше, не знаючи природи процесів, що лежать в основі виконуваних дій.
Наприклад, щоб потрапити м'ячем у баскетбольне кільце, робот-баскетболіст повинен виміряти відстань до кільця й напрямок, розрахувати параболічну траєкторію, і зробити кидок з урахуванням маси м'яча й опору повітря. Людина ж обходиться без цього тільки через тренування. Багаторазово здійснюючи

кидки й спостерігаючи результати, вона коректує свої дії, поступово вдосконалюючи свою техніку. При цьому в її мозку формуються відповідні структури нейронів, відповідальні за техніку кидків.

Рисунок 9 – Алгоритм навчання нейронних мереж

1. Вибір структури нейромережі, це складна задача, яку ми будемо розглядати в наступних лабораторних роботах. В даній лабораторній роботі візьмемо мережу, що складається з одного нейрону, зображеного на рисунку 1.

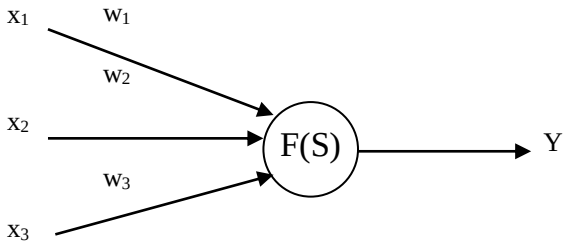


Рисунок 10 – Штучний нейрон для прогнозування значень часового ряду

Зважена сума та функція активації даного нейрона:

$$S_i = x_{i-3} \cdot w_1 + x_{i-2} \cdot w_2 + x_{i-1} \cdot w_3, \quad (9)$$

$$Y_i = 1/(1+\exp(-S_i))*10, \quad (10)$$

де w_1, w_2, w_3 – синаптичні ваги;
x_{i-3}, x_{i-2}, x_{i-1} – вхідні сигнали – відомі попередні значення часового ряду (*i-й* набор вхідних даних);
S_i – зважена сума *i-го* набору вхідних даних;
Y_i – прогнозоване значення *i-го* члена часового ряду x_i;
10 – масштабний множник.

2. Навчання полягає в тому, що на вхід мережі подаються спеціальні тренувальні дані, тобто такі вхідні дані, вихідний результат для яких відомий. На виході формуються результуючі дані, результати порівнюються з очікуваними, і обчислюється значення помилки. Після цього в певній послідовності виконується корекція параметрів нейронної мережі із метою мінімізації функції помилки. Якщо задовільної точності досягти не вдається, варто змінити структуру мережі й повторити навчання на множині тренувальних даних.

Таблиця 5 – Приклад тренувальних даних для нейронної мережі, що здійснює прогнозування значень часового ряду

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
1,59	5,73	0,48	5,28	1,35	5,91	0,77	5,25	1,37	4,42	0,26	4,21	1,90	4,08	1,40

Перші 13 чисел будемо використовувати для навчання мережі як тренувальний набір даних. Останні два члени ряду в навчанні не будуть брати участь, а служитимуть для тестування мережі.

Прогнозування полягає в тому, щоб на основі x_i, x_{i+1}, x_{i+2} обчислити x_{i+3}. Іншими словами, нейронна мережа «ковзає» уздовж часового ряду, «обмацуючи» синапсами по три сусідніх числа, та намагається прогнозувати значення наступні за ними. Таким чином, для наведеного вище прикладу вхідними й вихідними величинами будуть наступні (див. табл. 6).

Таблиця 6 – Очікувані значення часового ряду на кожному кроці навчання

і-тий набір даних	**Вхід нейрона**	**Вихід (очікуваний результат)**
1	1,59 5,73 0,48	5,28
2	5,73 0,48 5,28	1,35
3	0,48 5,28 1,35	5,91
4	5,28 1,35 5,91	0,77
	і т.д.	

Навчання нейронної мережі полягає в знаходженні таких значень ваг w, при яких нейромережа буде здатна видавати на основі вхідних даних вірні вихідні дані з певною наперед заданою точністю.

Дана задача задовільно вирішується за допомогою **алгоритму зворотного поширення (*back propagation*)**, що полягає в наступному:

1) Спочатку всі вагові коефіцієнти нейронної мережі встановлюються довільно. Можна скористатися функцією random, або просто присвоїти всім ваговим коефіцієнтам 1.

2) Через мережу пропускаються тренувальні дані (перший набір вхідних даних), і обчислюється сумарна функція помилки (сума квадратів помилки):

$$E = \sum_{i=1}^{N} (Y_i - y_i)^2 , \qquad (11)$$

де Y_i – обчислене значення виходу нейрона;

y_i – правильне значення наступного члену часового ряду.

3) Обчислюється значення похідної функції помилки E'_t для кожного вагового коефіцієнта:

$$E'_i = (Y_i - y_i) \cdot (exp(-s_i) / (1+exp(-s_i))^2) \cdot x_i \quad (12)$$

4) На основі E'_{i1}, E'_{i2}, та E'_{i3}, здійснюється розрахунок виправлень Δw_{i1}, Δw_{i2}, та Δw_{i3} до відповідних вагових коефіцієнтів за наступною формулою:

$$\Delta w_i = - \nu \cdot E'_i, \quad (13)$$

де ν – коефіцієнт швидкості навчання. Виправлення необхідно знайти для кожного *i-го* набору вихідних даних, і обчислити середні значення $\Delta w_{середнє1}$, $\Delta w_{середнє2}$, та $\Delta w_{середнє3}$ для всього набору:

$$\Delta w_{середнє} = \frac{1}{N} \sum_{i=1}^{N} \Delta w_i , \quad (14)$$

де N – кількість наборів вхідних даних для навчання.

5) Вагові коефіцієнти коректуються на величину обчислених виправлень:

$$w = w + \Delta w_{середнє}, \quad (15)$$

6) Поточне значення сумарної функції помилки E зберігається в іншій змінній:

$$E_0 = E. \quad (16)$$

Кроки 2–5 алгоритму зворотного поширення повторюються для кожного *i-того* набору вхідних даних (назвемо це *цикли навчання*), поки функція помилки не знизиться до заданого рівня, наприклад:

$$|E - E_0| < 0{,}0001 \quad (17)$$

Кількість ітерацій у процесі навчання мережі може досягати сотень і навіть тисяч. Тому доречно зробити додаткову умову виходу із циклу, на випадок якщо навчання з заданим рівнем точності буде тривати непримустимо довго, або відбудеться зациклення. Додатковою умовою виходу може бути натиснення користувачем кнопки "Стоп", або досягнення певної кількості циклів навчання, наприклад: i > 1000000.

3. Тестування, тобто контроль точності на спеціальних тестових даних, виконується після того, як нейронна мережа навчена. Це означає, що всі дані варто розбити на дві підмножини: на першій з них виконується навчання мережі, а на другій - тестування. За аналогією з навчанням людини тестування можна вподібнити іспиту. В нашому випадку для тестових даних ми залишили визначення нейронною мережею чисел x_{14} та x_{15} нашого часового ряду.

Завдання:
Написати програму для реалізації штучних нейронів та нейронних мереж для:
- моделювання логічної функції І,
- моделювання логічної функції АБО,
- моделювання логічної функції НІ,
- моделювання логічної функції Виключне АБО,
- прогнозування часового ряду (приклади часових рядів див. в додатку до лабораторної роботи А).

Додаткове завдання:
Написати програму для реалізації штучної нейронної мережі, що моделює логічну функцію, таблиця істинності якої наводиться в таблиці 7.

Таблиця 7 – Таблиця істинності логічної функції

x_1	x_2	x_2	y
0	0	0	1
0	1	0	1
1	0	0	0
1	1	1	1

Додаток А

Таблиця А.1 - Варіанти часових рядів

№ варіанту	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
1	2,56	4,20	1,60	4,29	1,17	4,40	0,88	4,14	0,07	4,77	1,95	4,18	0,04	5,05	1,40
2	0,20	5,14	0,47	4,37	1,22	4,29	1,89	4,51	0,32	5,80	1,37	5,77	0,88	4,86	1,94
3	1,92	4,01	1,48	5,45	1,56	5,42	1,28	4,34	1,51	5,49	1,32	4,00	0,49	4,19	1,53
4	0,13	5,97	0,57	4,02	0,31	5,55	0,15	4,54	0,65	4,34	1,54	4,70	0,58	5,83	0,03
5	2,16	3,19	1,85	4,84	0,55	4,20	1,68	4,74	0,14	5,68	0,48	5,03	0,18	5,99	0,09
6	2,54	5,28	0,78	5,72	0,58	4,65	0,91	5,80	1,76	5,67	1,73	5,70	1,03	5,00	1,79
7	1,69	3,38	1,40	5,56	1,86	5,62	0,46	5,51	0,26	5,13	1,18	5,98	1,36	5,09	1,29
8	1,19	5,61	0,89	6,00	1,04	5,98	0,03	6,00	1,83	4,23	0,60	4,15	0,13	5,01	1,87
9	0,87	4,12	0,93	4,62	1,51	5,76	0,50	5,48	0,95	4,03	0,92	5,15	1,66	5,01	0,40
10	2,82	3,48	0,60	4,76	1,51	5,51	1,48	5,19	0,48	5,22	0,21	4,19	0,07	4,63	0,49
11	2,64	4,66	1,87	4,05	1,73	5,31	1,67	5,96	0,13	5,64	1,52	4,07	0,22	4,79	0,73
12	2,65	5,60	1,21	5,48	0,73	4,08	1,88	5,31	0,78	4,36	1,71	5,62	0,43	4,21	1,21
13	2,37	4,85	1,97	4,17	1,39	4,66	1,26	4,40	0,46	5,54	1,34	5,80	1,61	5,97	1,95
14	1,88	4,52	1,91	5,66	1,23	5,50	1,14	5,29	1,60	4,31	0,06	5,33	0,07	4,62	0,69
15	0,78	4,95	1,19	4,08	0,80	4,25	0,22	4,63	1,48	4,97	0,53	5,50	1,28	5,79	0,44
16	0,58	3,38	0,91	5,80	0,91	5,01	1,17	4,67	0,60	4,81	0,53	4,75	1,01	5,04	1,07
17	0,51	4,82	0,43	4,71	1,92	5,86	1,24	4,69	0,72	5,26	0,90	4,55	1,46	5,21	1,50
18	0,07	3,58	0,44	5,33	0,56	5,24	1,99	4,38	0,89	4,53	1,82	4,13	1,88	5,97	1,18

№ варіанту	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
19	1,44	4,60	1,22	5,90	1,34	4,31	1,02	4,35	0,82	4,18	1,60	4,86	1,45	4,97	1,00
20	2,57	4,35	1,27	5,46	1,30	4,92	1,31	4,14	1,97	5,67	0,92	4,76	1,72	4,44	1,49
21	0,79	3,84	0,92	4,50	0,96	5,51	1,14	5,32	0,39	4,99	1,36	5,81	1,90	4,79	1,41
22	0,99	4,72	1,59	5,29	1,53	5,58	0,84	5,79	0,21	5,94	0,42	5,98	1,18	5,55	0,11
23	2,92	3,56	0,15	5,11	1,38	4,44	1,61	4,11	1,97	4,50	1,37	5,08	1,76	5,19	1,58
24	0,48	4,30	0,91	4,85	0,53	4,51	1,95	5,88	0,63	5,79	0,92	5,18	1,88	4,84	0,22
25	1,88	4,98	0,06	5,26	1,16	5,06	0,58	5,28	1,41	5,57	1,19	5,36	1,40	4,30	0,09
26	2,57	5,77	0,38	4,73	0,10	5,93	1,35	4,70	1,62	5,51	1,78	5,66	1,47	5,52	1,88
27	0,11	4,87	1,52	4,47	0,34	5,44	1,20	5,21	1,48	5,93	0,62	5,48	1,34	4,25	0,65
28	1,07	3,17	1,08	5,99	1,28	4,11	0,25	5,82	0,96	4,83	1,10	4,31	0,81	5,49	1,92
29	1,59	5,74	0,48	5,28	1,34	5,91	0,77	5,25	1,37	4,42	0,26	4,21	1,90	4,08	1,40
30	0,68	5,78	0,25	5,58	1,31	4,28	1,57	5,75	0,41	5,55	0,90	5,86	0,03	5,57	0,30

Контрольні питання:

1. Що таке штучний нейрон і які ролі відіграють його входи, ваги, зважена сума та функція активації?

2. Чому для моделювання логічних функцій (І, АБО, НІ) достатньо одного нейрона з пороговою активацією, а для функції XOR потрібна багатошарова нейромережа?

3. У чому полягає ідея навчання нейромережі методом зворотного поширення помилки при прогнозуванні часових рядів (що робимо з помилкою і вагами)?

4. Які саме значення використовуються як вхідні дані та як «очікуваний вихід» при навчанні нейрона прогнозувати часовий ряд у цій лабораторній роботі?

5. За якими критеріями (умовами зупинки) визначається, що навчання нейромережі можна завершити?

Лабораторна робота № 7

Тема: Оптимізація функції із застосуванням генетичних алгоритмів

Мета: Написати програму оптимізації функції з використанням генетичного алгоритму

Теоретичні відомості

На практиці часом складно, а часом і неможливо, зафіксувати властивості функціональної залежності вихідних параметрів від вхідних величин, ще складніше провести аналітичний опис такої залежності. Ця обставина значно ускладнює застосування класичних методів оптимізації, оскільки більшість із них ґрунтуються на використанні апріорної інформації про характер поводження цільової функції, а задача визначення приналежності функції до того або іншого класу порівняна за складністю з початковою.

У зв'язку із цим виникає необхідність побудови таких методів оптимізації, які були б здатні відшукувати рішення при практично повній відсутності припущень про характер досліджуваної функції при рішенні цілочисельних або комбінаторних оптимізаційних задач. Цим вимогам у значній мірі задовольняють еволюційні обчислення, які являють собою алгоритми пошуку, оптимізації або навчання, заснованих на деяких формалізованих принципах природного еволюційного процесу розвитку живих організмів.

Парадигму **генетичних алгоритмів** запропонував Джон Холланд, що опублікував на початку 60-х років її основні положення. А загальне визнання вона одержала після виходу у світ в 1975 році його класичної праці «Адаптація в природних і штучних системах». Генетичний алгоритм був отриманий у процесі узагальнення й імітації в штучних системах таких властивостей живої природи, як природний відбір, пристосовність до мінливих умов середовища, спадкування нащадками життєво важливих властивостей від батьків і т.д.

Тому що алгоритми в процесі пошуку використовують деяке кодування множини параметрів замість самих параметрів, то він ефективно застосовуються для рішення дискретних і комбінаторних задач оптимізації, визначених як на числових множинах, так і на кінцевих множинах довільної природи. Оскільки для роботи алгоритму в якості інформації про оптимізуєму функцію використовуються лише її значення в розглянутих точках простору пошуку, і не потрібно обчислень ні похідних, ні яких-небудь інших характеристик, то даний алгоритм застосуємий до широкого класу функцій, зокрема, не маючих аналітичного опису.

Використання набору початкових точок дозволяє застосовувати для їхнього формування різні способи, що залежать від специфіки розв'язуваної оптимізаційної задачі, у тому числі можливе задання такого набору безпосередньо людиною. Таким чином, основна відмінність генетичних алгоритмів полягає в представленні будь-якої альтернативи рішення у вигляді бітового рядка фіксованої довжини, маніпуляції з якою здійснюються під час відсутності всякого зв'язку з її смисловою інтерпретацією. Тобто в цьому випадку застосовується єдине універсальне представлення будь-якої задачі.

Ідея генетичного алгоритму

Уявимо собі штучний світ, населений множиною істот (особин), причому кожна істота - це деяке рішення нашої задачі. Будемо вважати особину тим більше пристосованою, чим краще відповідне рішення (чим більше значення цільової функції воно дає). Тоді задача максимізації цільової функції зводиться до пошуку найбільш пристосованої істоти. Звичайно, ми не можемо оселити в наш віртуальний світ всі істоти відразу, тому що їхнє число визначається початковою множиною альтернатив. Замість цього ми будемо розглядати багато поколінь, що змінюють один одного.

Тепер, якщо ми зуміємо ввести в дію природний відбір і генетичне спадкування, то отриманий світ буде підкорятися законам еволюції. Помітимо, що, відповідно до нашого визначення пристосованості, метою цієї штучної еволюції буде саме створення найкращих рішень.

Очевидно, еволюція - нескінченний процес, у ході якого пристосованість особин поступово підвищується. Примусово зупинивши цей процес через досить довгий час після його початку й вибравши найбільш пристосовану особину в поточному поколінні, одержимо якщо не абсолютно точну, то близьку до оптимальної відповідь. В цьому й полягає основна ідея генетичного алгоритму. Перейдемо тепер до точних визначень і опишемо роботу генетичного алгоритму більш детально.

Основний зміст генетичного пошуку

Генетичні алгоритми, будучи однією з парадигм еволюційних обчислень, являють собою алгоритми пошуку, побудовані на принципах, подібних із принципами природного відбору й генетики. Якщо говорити узагальнено, вони поєднують у собі принцип виживання найбільш перспективних особин-рішень і структурований випадково-детермінований обмін інформацією, у якому є присутнім елемент випадковості, що моделює природні процеси спадкування й мутації. Додатковою властивістю цих алгоритмів є невтручання людини в розвиваючийся процес пошуку. Людина може впливати на нього лише опосередковано, задаючи певні параметри.

Чим обумовлена популярність генетичних алгоритмів? Як було вже відзначено, ГА дозволяють знайти більш гарні або «раціональні» рішення NP-повних практичних оптимізаційних задач за менший час, ніж інші методи, звичайно застосовувані в цих випадках. Звичайно, термін «гарні» або «раціональні» не строгий у математичному змісті.

Під «раціональними» розуміються рішення, які задовольняють дослідника. Адже в більшості реальних задач немає необхідності знаходити саме глобальний оптимум. Частіше всього метою пошуку є рішення, що задовольняють певним обмеженням. Наприклад, час випробування устаткування не повинний перевищувати певної заданої величини. У цьому сенсі досить знайти саме «раціональне», тобто розумне рішення. Друга важлива причина росту популярності ГА полягає в стрімкому зростанні продуктивності сучасних комп'ютерів.

Переваги **генетичних алгоритмів** стають більш очевидними, якщо розглянути основні їхні відмінності від традиційних методів. Основних відмінностей п'ять:

1. Генетичні алгоритми працюють із кодами, у яких представлений набір параметрів, що прямо залежать від аргументів цільової функції.
2. Для пошуку генетичний алгоритм використовує декілька точок пошукового простору одночасно (розпаралелення), а не переходить від точки до точки, як це робиться в традиційних методах. Тобто ГА оперують одночасно з усією сукупністю припустимих рішень.
3. Генетичні алгоритми в процесі роботи не використовують ніякої додаткової інформації, що підвищує швидкість його роботи.
4. Генетичний алгоритм використовує як ймовірнісні правила для породження нових точок пошуку, так і детерміновані правила для переходу від одних точок до інших.
5. Генетичні алгоритми здійснюють пошук оптимального рішення за однією й тою ж стратегією, як для унімодальних, так і для багатоекстремальних функції.

Генетичний алгоритм працює з **кодовими послідовностями** (КП) — кодами безвідносно їхньої значеннєвої інтерпретації. Тому сама КП і її структура описуються поняттям **генотип**, а його інтерпретація, з погляду розв'язуваної задачі, поняттям **фенотип**. Кожна КП представляє, по суті, точку простору пошуку. Екземпляр кодової послідовності називають хромосомою, особиною або індивідуумом.

У принципі, ГА не обмежені бінарним або цілочисельним поданням. Відомі й інші реалізації, побудовані винятково на векторах речовинних числах. Незважаючи на те, що для багатьох реальних завдань більше підходять рядки змінної довжини, у даний час структури фіксованої довжини найпоширеніші й вивчені.

На кожному кроці роботи генетичний алгоритм використовує декілька точок пошуку одночасно. Сукупність цих точок є набором кодових послідовностей (особин), які утворюють початкову множину рішень — **К** (популяцію). Кількість особин у популяції називають **розміром популяції**. На кожному кроці роботи генетичний алгоритм обновляє початкову множину **К** шляхом створення нових КП і знищення «безперспективних», не задовольняючих критерію цільової функції. Кожне відновлення інтерпретується як зміна поколінь і звичайно ідентифікується за заданим розміром.

У процесі роботи алгоритму генерація нових особин відбувається на основі моделювання процесу розмноження. При цьому, природно, що породжуючі особини називаються батьками, а породжені — нащадками. Батьківська пара, як правило, породжує пари нащадків. Безпосередня генерація нових рядків із двох обраних відбувається за рахунок роботи **оператора схрещування** (випадково-детермінованого обміну), що у процесі роботи алгоритму може застосовуватися не до всіх пар батьків.

Частина цих пар може переходити в популяцію наступного покоління безпосередньо. Наскільки часто буде виникати така ситуація, залежить від ймовірності застосування оператора схрещування, що є одним з параметрів генетичного алгоритму.

Моделювання процесу генерації нових точок пошуку здійснюється за рахунок роботи **оператора мутації**, що задається певною ймовірністю. Оскільки еволюційний процес біологічних видів супроводжується загибеллю останніх, то породження нащадків повинне супроводжуватися знищенням інших безперспективних особин. Вибір пар батьків з популяції для породження нащадків робить **оператор відбору**, а вибір особин для знищення - **оператор редукції**.

Характеристики генетичного алгоритму вибираються таким чином, щоб забезпечити малий час роботи, з одного боку, і пошук якомога кращого рішення, з іншого.

Загальний вид генетичного алгоритму

Незважаючи на те, що теорія генетичних алгоритмів в інформатиці з'явилася порівняно недавно, цей напрямок було швидко підхоплено вченими. І до теперішнього часу існує досить велика кількість їхніх різновидів еволюційних алгоритмів. Проте їхню основу становить базова модель, представлена на рис. 1.

Рисунок 1 – Базовий генетичний алгоритм

Різниця полягає лише в генетичних операціях, представлених усередині зазначеного алгоритму. Виключення становлять гібридні алгоритми, у яких до зазначених блоків можуть додаватися елементи класичних алгоритмів оптимізації.

Далі розглянемо детально методологію побудови генетичного алгоритму.

Створення хромосом

Хромосома представляє собою набір зістикованих значень змінних, представлений у двійковому коді. Розглянемо метод кодування:

Нехай X описується діапазоном значень [Xmin...Xmax], з точністю E.

Спочатку обчислюється кількість значень N, яких необхідно закодувати, за формулою:

$$N = \frac{Xmax - Xmin}{E} \tag{1}$$

Потім підбирається таке значення кількості бітів NB, що задовольняє умові:

$$2^{NB} \geq N \tag{2}$$

Далі обчислюється крок дискретизації заданого діапазону:

$$D = \frac{Xmax - Xmin}{NB} \tag{3}$$

Знаючи крок дискретизації ми можемо закодувати значення X натуральним числом Nx за формулою:

$$Nx = \frac{X}{D} \tag{4}$$

Після чого Nx кодується у двійковій системі числення.

У випадку мінімізації функції багатьох змінних двійкові коди кожної змінної зістиковуються в певній послідовності й зберігається інформація про кількість бітів, якою кодується кожна змінна.

Функція пристосованості

Під функцією пристосованості розуміється деяка цільова функція, екстремум якої й потрібно відшукати. Однак, у деяких літературних джерелах вказується, що ця функція є зворотною від цільової функції у випадках коли потрібно відшукати мінімум. Але це вносить певні незручності в перевірці рішення. Тому під функцією пристосованості краще розуміти саме цільову функцію. Якщо доводиться використовувати генетичний алгоритм для навчання нейронних, нейронечітких мереж або ж добірки коефіцієнтів ПІД-регуляторів за зразком (навчальною вибіркою), то як функція пристосованості звичайно виступає класична функція суми квадратів помилки F:

$$F = \sum_i \left(Y_i - Y'_i\right)^2 \tag{5}$$

де i - номер елемента навчальної вибірки;

Y - вектор значень вихідний змінної;

Y' - вектор відповідних значень мережі (регулятора).

Ініціалізація початкової популяції

На цьому етапі визначається кількість хромосом (особин) у популяції K. У різних джерелах літератури вказується, що воно залежить від кількості вхідних змінних і звичайно вибирається за правилом:

$$K \geq 2 \cdot M \tag{6}$$

де M - кількість вхідних змінних.

Далі задаються початкові значення хромосомам. Якщо відомі деякі наближення до мінімумів, то деяким хромосомам привласнюються ці значення. Всім іншим значення задаються випадково. Потім обчислюються відповідні їм значення функції пристосованості.

Селекція

Суть операції селекції полягає у відборі пар хромосом для рекомбінації. І може здійснюватися трьома способами:

1. Ранжирування.

2. Випадковий вибір.

3. Виважено випадковий вибір.

При ранжируванні всі хромосоми впорядковуються за убуванням їхньої функції пристосованості й пари розбиваються послідовно за принципом «краща із кращої».

При випадковому виборі пари утворюються випадковим чином.

При зважено випадковому виборі кожній хромосомі привласнюється певний бал залежний від функції пристосованості. Потім всі хромосоми розташовуються послідовно на шкалі суми балів з діапазоном свого бального значення. Пари формуються шляхом випадкового вибору числа на шкалі. Отже чим більше бал у хромосоми, тим більше в неї шансів. Даний спосіб іноді ще називають «принципом рулетки».

Рекомбінація (кросовер)

На даному етапі відбувається обмін генетичною інформацією. Рекомбінація є найбільш важливим генетичним оператором. Вона генерує нові хромосоми, поєднуючи генетичний матеріал двох батьківських. Існує декілька варіантів рекомбінації. Найбільш простим є одноточковий. У цьому варіанті просто беруться дві хромосоми, і розрізаються у випадково обраній точці. Результуючі хромосоми виходять із початку однієї й кінця іншої батьківських хромосом.

0011001011100 10|11000	-->	0011001011100 10***11100***
11010110110100 0|11100		***11010110110100 0***11000

Мутація

Мутація являє собою випадкову зміну хромосоми (звичайно простою зміною стану одного з бітів на протилежний). Даний оператор дозволяє більш

швидко знаходити локальні екстремуми з одного боку, і дозволяє "перескочити" на інший локальний екстремум з іншого.

00110010111001***0***11000	-->	00110010111001***1***11000

У деяких генетичних алгоритмах до мутації може додаватися операція інверсії.

Інверсія інвертує (змінює) порядок бітів у хромосомі шляхом циклічної перестановки (випадкова кількість разів).

00110010111001***11000***	-->	***11000***00110010111001**0**

Умова зупинки

Умова зупинки залежить від того, де ми застосовуємо генетичний алгоритм. Залежно від цього можна виділити наступні умови:

- Цільова функція досягла деякої заданої точності або значення. Цей випадок підходить для алгоритмів навчання із учителем нейронних або нейронечітких мереж, настроювання коефіцієнтів ПІД-регуляторів або інших математичних апаратів, пов'язаних з навчанням за зразком (навчальній вибірці).

- Досягнуто задане число поколінь. Цей випадок застосовується в тих випадках, коли система явно обмежена часом і обчислювальними ресурсами на пошук оптимального рішення. Звичайно це потрібно в завданнях адаптації систем у динамічному режимі.

- Протягом заданого числа поколінь оптимальне значення цільової функції не змінюється. Такий варіант можливий у випадках, коли алгоритм досяг якого-небудь глобального або сильно вираженого локального екстремума. Це умова необхідно, коли алгоритм використовується як для навчання із учителем (вихід із зациклення у випадку неможливості відшукати рішення із заданою точністю), так і при пошуку екстремума, значення неможливо визначити заздалегідь.

Умови, при виконанні яких задача вирішується ефективно генетичними алгоритмами:

– великий простір пошуку, ландшафт якого є негладким (містить трохи екстремумів);

– складність формалізації оцінки якості рішення функцією ступеня придатності;

– багатокритеріальність пошуку;

– пошук прийнятного рішення за заданими критеріями на відміну від пошуку єдиного оптимального.

Основні завдання, які можуть ефективно вирішувати генетичні алгоритми, можна звести до наступних класів:

– навчання нейронних і нейронечітких мереж із учителем;

– адаптація нейронних і нейронечітких мереж у динамічному режимі;

– настроювання коефіцієнтів ПІД-регуляторів;

– рішення класичних оптимізаційних завдань (завдання лінійного програмування, транспортні завдання й т.д.);

– рішення комбінаторних завдань.

Завдання:

Вибрати функцію й діапазон відповідно до варіанта. Побудувати графік функції. Написати програму знаходження максимуму й мінімуму функції на заданому діапазоні. Проаналізувати отримані результати.

Варіанти:

№	**Функція**
1.	Y(x)=x*sin(5*x), x=[-2...5]
2.	Y(x)=1/x*sin(5*x),x=[-5...5]
3.	Y(x)=2^x*sin(10x),x=[-3...3]
4.	Y(x)=x^(1/2)*sin(10*x),x=[0...5]
5.	Y(x)=15*sin(10*x)*cos(3*x),x=[-3...3]
6.	Y(x)=5*sin(10*x)*sin(3*x),x=[0...4]
7.	Y(x)=sin(10*x)*sin(3*x)/(x^2),x=[0...4]
8.	Y(x)=5*sin(10*x)*sin(3*x)/(x^(1/2)),x=[1...7]
9.	Y(x)=5*cos(10*x)*sin(3*x)/(x^(1/2)),x=[0...5]
10.	Y(x)=-5*cos(10*x)*sin(3*x)/(x^(1/2))x=[0...10]
11.	Y(x)=-5*cos(10*x)*sin(3*x)/(x^x),x=[0...5]
12.	Y(x)=5*sin(10*x)*sin(3*x)/(x^x),x=[0...8]

13. Y(x)=x^sin(10*x),x=[1...10]
14. Y(x)=-x^cos(5*x),x=[0...10]
15. Y(x)=x^cos(x^2),x=[0...10]
16. Y(x)=cos(x^2)/x,x=[0...5]
17. Y(x)=10*cos(x^2)/x^2,x=[0...4]
18. Y(x)=(1/x)*cos(x^2+1/x),x=[1...10]
19. Y(x)=sin(x)*(1/x)*cos(x^2+1/x),x=[-2...2]
20. Y(x)=5*sin(x)*cos(x^2+1/x)^2,x=[1...10]
21. Y(x)=5*sin(1/x)*cos(x^2+1/x)^2,x=[1...4]
22. Y(x)=5*sin(1/x)*cos(x^2)^3,x=[-4...4]
23. Y(x)=(x^3)*cos(x^2),x=[-2...2]
24. Y(x)=(x^3)+cos(15*x),x=[-2...2]
25. Y(x)=(3^x)+cos(15*x),x=[-1...2]

Контрольні питання:

1. У чому полягає основна ідея генетичного алгоритму та як інтерпретуються терміни «особина», «популяція», «генотип» і «фенотип» у контексті оптимізації функції?
2. Як здійснюється кодування змінної у хромосому: що таке діапазон [Xmin…Xmax], точність E, кількість бітів NB та крок дискретизації, і як із реального значення X отримати бітовий рядок?
3. Яка роль функції пристосованості у генетичному алгоритмі та чому зручно ототожнювати її безпосередньо з цільовою функцією (а не зворотною до неї)?
4. Які основні оператори ГА (селекція, кросовер, мутація) та яку роль кожен із них відіграє у процесі пошуку максимуму/мінімуму функції?
5. Які можливі умови зупинки генетичного алгоритму й чому на практиці часто обирають комбінацію кількох умов (фіксована кількість поколінь, сталий результат, досягнення заданої точності тощо)?

СПИСОК ЛІТЕРАТУРИ

1. Коваленко О. О., Ткаченко О. М., Чехместрук Р. Ю. Алгоритми та структури даних : навчальний посібник (електронне видання). Вінниця : ВНТУ, 2025. – 113 с. https://pdf.lib.vntu.edu.ua/books/2025/Kovalenko_2025_113.pdf
2. Креневич А. П. Алгоритми і структури даних : підручник. Київ : ВПЦ «Київський університет», 2021. – 200 с. https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf
3. Кублій Л. І. Алгоритми та структури даних. Основи алгоритмізації : підручник. Київ : КПІ ім. Ігоря Сікорського, 2022. – 528 с. https://ela.kpi.ua/handle/123456789/48282
4. Мелешко Є. В., Якименко М. С., Поліщук Л. І. Алгоритми та структури даних : навч. посіб. / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : Лисенко В.Ф., 2019. – 156 с. https://dspace.kntu.kr.ua/handle/123456789/8944
5. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.
6. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.
7. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.
8. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.
9. Lutz M. Learning Python, 5th Edition Fifth Edition. - O'Reilly Media, 2016. - 1643 p.
10. Lutz M. Python: Pocket Reference Fourth Edition. - O'Reilly Media, 2016. - 210 p.
11. McKinney W. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter 3rd Edition. - O'Reilly Media, 2022. - 579 p.
12. Алгоритми та структури даних (комп’ютерний практикум) : навч. посібник / уклад. Ю. Є. Грудзинський. Київ : КПІ ім. Ігоря Сікорського, 2022. – 100 с. https://ela.kpi.ua/server/api/core/bitstreams/0db974f9-16fa-459c-9f19-fab0021222ed/content
13. Бугаєва Л. М., Ковалюк Д. О. Алгоритми та структури даних. Комп’ютерний практикум : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2022. – 34 с. https://ela.kpi.ua/items/c9756f3a-e61f-4068-8d88-bd1e1dd950a9

14. Бульба С. С., Бречко В. О., Далека В. Д. Алгоритми та структури даних : навч.-метод. посібник. Харків : НТУ «ХПІ», 2021. – 141 с. https://repository.kpi.kharkov.ua/handle/KhPI-Press/54935
15. Савеленко О. К., Лисенко І. А., Іванченко О. О. CASE-технології у проектуванні інформаційних систем: навчальний посібник / Мін-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький: Видавець Лисенко В.Ф., 2018. – 240 с. https://dspace.kntu.kr.ua/handle/123456789/10278
16. https://www.codeproject.com/ – колективний блог з новинами та навчальними статтями про інформаційні технології та програмування.
17. http://stackoverflow.com/ – система питань і відповідей для професійних програмістів та новачків у програмуванні.
18. https://dou.ua/ – український веб-сайт з елементами колективного блогу, створений для розповсюдження новин, аналітичних статей та свіжої інформації пов'язаної із інформаційними технологіями.
19. http://www.algomation.com/ – це платформа для перегляду, обміну і створення візуалізацій алгоритмів.
20. https://prometheus.org.ua/ – українська платформа безкоштовних онлайн-курсів
21. http://moodle.kntu.kr.ua/ – Дистанційна освіта ЦНТУ.
22. http://www.tutorialspoint.com/python/ – Tutorialspoint / Python
23. https://docs.python.org/ – Python's documentation, tutorials, and guides are constantly evolving

Додаток 1
Приклад оформлення звіту з лабораторної роботи

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Лабораторна робота №_
з дисципліни «Алгоритми та структури даних»
на тему:
_____

Виконав:
студент групи _____
_____
(П.І.Б.)

Перевірив:
викладач
_____
(П.І.Б.)

Кропивницький 20__

Тема: _____
Мета: _____

Варіант №_

Завдання: _____
_____
_____
_____
_____

Хід роботи:

<Повинен містити теоретичні дані, скриншоти розробленого програмного забезпечення та лістинг>

Відповіді на контрольні питання:
1.
2.
3.
…

Додаток 2
Приклад оформлення лістингу програми

Лістинг програми

Файл Lab_2_1.cs

```
// (c) Іваненеко І.І., ЦНТУ, 2025
// дисципліна «Алгоритми та структури даних»
// лабораторна робота №2, варіант 12, завдання 1

using System;
using System.Text;

namespace ConsoleApplication9
{
class Program
    {
        //метод для введення цілих чисел із клавіатури
staticint ReadInt(string prompt)
        {
                Console.Write(prompt);
int x = int.Parse(Console.ReadLine());
return x;
        }
        //метод для виведення масиву на екран
staticvoid PrintArray(int[] array)
        {
for (int i = 0; i < array.Length; i++)
            {
                Console.Write("{0,5} ", array[i]);
            }
            Console.WriteLine();
        }

staticvoid Main(string[] args)
        {
int N = ReadInt("Введіть розмірність масиву: ");

int[] a = newint[N];

            //введемо елементи масиву з клавіатури
for (int i = 0; i < N; i++)
            {
                try
                {
                    a[i] = ReadInt("Введіть " + (i+1).ToString() +
                         "-й елемент масиву: ");
                }
catch(FormatException)
                {
                    Console.WriteLine("Невірний формат числа!");
                }
            }
```

…

Додаток 3
Шкала оцінювання: національна та ECTS

№	Сума балів за всі види навчальної діяльності	Оцінка ЄКТС	Оцінка за національною шкалою для екзамену
1	**90-100**	**A**	«відмінно»
2	**82-89**	**B**	«добре»
3	**74-81**	**C**	
4	**64-73**	**D**	«задовільно»
5	**60-63**	**E**	
6	**35-59**	**FX**	«незадовільно» з можливістю повторного складання
7	**1-34**	**F**	«незадовільно» з обов'язковим повторним вивченням дисципліни

Критерії оцінювання. Еквівалент оцінки в балах для кожної окремої теми може бути різний, загальну суму балів за тему визначено в навчально-методичній карті. Розподіл балів між видами занять (лекції, практичні заняття, самостійна робота) можливий шляхом спільного прийняття рішення викладача і студентів на першому занятті:

1) оцінку **«відмінно» (90-100 балів, A)** заслуговує студент, який:

- всебічно, систематично і глибоко володіє навчально-програмовим матеріалом;
- вміє самостійно виконувати завдання, передбачені програмою, використовує набуті знання і вміння у нестандартних ситуаціях;
- засвоїв основну і ознайомлений з додатковою літературою, яка рекомендована програмою;
- засвоїв взаємозв'язок основних понять дисципліни та усвідомлює їх значення для професії, яку він набуває;
- вільно висловлює власні думки, самостійно оцінює різноманітні життєві явища і факти, виявляючи особистісну позицію;
- самостійно визначає окремі цілі власної навчальної діяльності, виявив творчі здібності і використовує їх при вивченні навчальнопрограмового матеріалу, проявив нахил до наукової роботи.

2) оцінку **«добре» (82-89 балів, B)** – заслуговує студент, який:

- повністю опанував і вільно (самостійно) володіє навчально-програмовим матеріалом, в тому числі застосовує його на практиці, має системні знання достатньому обсязі відповідно до навчально-програмового матеріалу, аргументовано використовує їх у різних ситуаціях;
- має здатність до самостійного пошуку інформації, а також до аналізу, постановки і розв'язування проблем професійного спрямування;
- під час відповіді допустив деякі неточності, які самостійно виправляє, добирає переконливі аргументи на підтвердження вивченого матеріалу.

3) оцінку **«добре» (74-81 бал, C)** - заслуговує студент, який:

- в загальному роботу виконав, але відповідає на екзамені з певною кількістю помилок;

- вміє порівнювати, узагальнювати, систематизувати інформацію під керівництвом викладача, в цілому самостійно застосовувати на практиці, контролювати власну діяльність;

- опанував навчально-програмовий матеріал, успішно виконав завдання, передбачені програмою, засвоїв основну літературу, яка рекомендована програмою.

4) оцінку **«задовільно» (64-73 бали, D)** – заслуговує студент, який:

- знає основний навчально-програмовий матеріал в обсязі, необхідному для подальшого навчання і використання його у майбутній професії; - виконує завдання, але при рішенні допускає значну кількість помилок;

- ознайомлений з основною літературою, яка рекомендована програмою;

- допускає на заняттях чи екзамені помилки при виконанні завдань, але під керівництвом викладача знаходить шляхи їх усунення.

5) оцінку **«задовільно» (60-63 бали, E)** – заслуговує студент, який:

- володіє основним навчально-програмовим матеріалом в обсязі, необхідному для подальшого навчання і використання його у майбутній професії, а виконання завдань задовольняє мінімальні критерії. Знання мають репродуктивний характер.

6) оцінка **«незадовільно» (35-59 балів, FX)** – виставляється студенту, який:

- виявив суттєві прогалини в знаннях основного програмового матеріалу, допустив принципові помилки у виконанні передбачених програмою завдань.

7) оцінку **«незадовільно» (35 балів, F)** – виставляється студенту, який:

- володіє навчальним матеріалом тільки на рівні елементарного розпізнавання і відтворення окремих фактів або не володіє зовсім;

- допускає грубі помилки при виконанні завдань, передбачених програмою;

- не може продовжувати навчання і не готовий до професійної діяльності після закінчення університету без повторного вивчення даної дисципліни.