ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему:
РЕАЛІЗАЦІЯ НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОБУДОВИ КАНОНІЧНИХ ФОРМ БУЛЕВИХ ФУНКЦІЙ

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Волторніст Артем Володимирович
_____ «__»_____2025 р.
(підпис)
Науковий керівник к. ф.-м. н., доцент, Парфьонова Тетяна Олександрівна
_____
_____ «__»_____2025 р.
(підпис)
Рецензент _____

Полтава-2025

АНОТАЦІЯ

Записка: 58 с., 47 рис., 0 таблиць, 16 джерел.

КАНОНІЧНІ ФОРМИ БУЛЕВИХ ФУНКЦІЙ, БУЛЕВА АЛГЕБРА, ЛОГІКА, PYTHON, ПРОГРАМНА РЕАЛІЗАЦІЯ.

Мета роботи – проектування та програмна реалізація навчального програмного забезпечення.

Предмет розробки – розробка та програмна реалізація програмного забезпечення, створеного на мові програмування Python у середовищі PyCharm, за допомогою якого можна виконувати побудову канонічних форм булевих функцій.

Об'єкт розробки – підвищення ефективності вивчення булевої алгебри та математичної логіки за допомогою комп'ютера.

Методи дослідження та інформаційне забезпечення – використання матеріалів з теми «Булева алгебра», інтегроване програмне середовище PyCharm, об'єктно-орієнтована мова програмування Python.

Елементи програми розроблено на основі віконної програми з використанням бібліотек PyCharm.

Програмне забезпечення являє собою калькулятор, що будує задану формулу та містить покрокову демонстрацію розв'язку з можливістю побудови кон'юнктивних й диз'юнктивних нормальних форм та їх досконалі варіанти. Перед початком виконання завдань користувач зможе переглянути інструкцію з використання програми.

Отриманий результат. Зроблено огляд схожого програмного забезпечення для обрахунку булевої алгебри, виявлено позитивні та негативні сторони. Розроблено алгоритм для більш ефективного процесу розрахунку. Здійснено програмну реалізацію та, відповідно, протестовано. У результаті тестування виявлено, що програму можна використовувати для швидкого опрацювання матеріалу самостійно.

Наукова апробація. За матеріалами роботи написано тези (Міжнародна наукова студентська конференція «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті»).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	**Пояснення умовних позначень, скорочень, символів**
ДДНФ	Досконала диз’юнктивна нормальна форма
ДНФ	Диз’юнктивна нормальна форма
ДКНФ	Досконала кон’юнктивна нормальна форма
КНФ	Кон’юнктивна нормальна форма
ПЗ	Програмне забезпечення

ВСТУП

Інтенсивний розвиток цифрових технологій зумовлює зростання вимог до якості підготовки фахівців у галузі комп’ютерних наук, інформаційних систем, електроніки та автоматизації. Одним із ключових напрямів у цій підготовці є вивчення дискретної математики та логіки, а саме – опанування основ булевої алгебри, яка є фундаментом для побудови логічних схем, комп'ютерних архітектур, алгоритмів керування та цифрових пристроїв. Знання і навички роботи з булевими функціями, зокрема вміння перетворювати їх у канонічні форми, є невід’ємною частиною підготовки майбутнього інженера-програміста або системного аналітика.

Особливе місце серед методів аналізу булевих функцій займають канонічні нормальні форми: досконала диз’юнктивна нормальна форма (ДДНФ) та досконала кон’юнктивна нормальна форма (ДКНФ). Вони використовуються для формального подання логічних виразів, їхньої подальшої мінімізації та реалізації в електронних схемах. Побудова ДКНФ і ДДНФ дозволяє однозначно інтерпретувати логічну функцію, формально її описати, а також здійснювати перетворення у зручній для реалізації формі.

У процесі навчання виникає необхідність не лише пояснити теоретичні основи побудови канонічних форм, але й продемонструвати сам процес у покроковому форматі, що особливо важливо для візуального засвоєння матеріалу студентами. Для цього ефективним засобом є використання навчального програмного забезпечення, яке здатне автоматично виконувати побудову ДДНФ і ДКНФ, при цьому відображаючи хід розв’язання та надаючи користувачу змогу самостійно експериментувати з вхідними даними.

На сьогодні існує низка програмних засобів, що працюють з логічними функціями, однак більшість із них розраховані на інженерів або досвідчених користувачів, мають складний інтерфейс або не підтримують українську мову. Більше того, вони не завжди орієнтовані на освітній процес – тобто не містять пояснень, інтуїтивного управління та покрокової побудови результатів. Це створює передумови для розробки нового інструменту, який буде враховувати специфіку

навчання, мати україномовний інтерфейс, бути простим у використанні та дозволяти студентам ефективно засвоювати матеріал через практику.

Мета роботи – розробка навчального програмного забезпечення для побудови канонічних форм булевих функцій.

Об’єктом розробки є логічні основи опрацювання даних у програмному забезпеченні.

Предметом розробки є програмне забезпечення, створене на мові програмування Python у середовищі PyCharm, за допомогою якого здійснюється побудова канонічних нормальних форм.

Методи розробки – структурний аналіз логічних виразів, методи побудови таблиць істинності, алгоритми формування канонічних форм, об'єктно-орієнтована мова програмування Python, а також принципи розробки графічного інтерфейсу.

Кваліфікаційна робота складається з чотирьох розділів, а саме: постановка задачі, інформаційний огляд, теоретична частина, практична частина.

Результатом виконання роботи є створення програмного забезпечення у вигляді програми-калькулятора, що формує булеві функції, на мові програмування Python у середовищі PyCharm.

Обсяг пояснювальної записки: 69 стор., в т.ч. основна частина 58 стор., додатки – 1 стор., джерел - 16 назв.

1. ПОСТАНОВКА ЗАДАЧІ

У межах даного кваліфікаційної роботи необхідно розробити навчальне програмне забезпечення, яке дозволяє здійснювати побудову канонічних форм булевих функцій: диз'юнктивної нормальної форми (ДНФ), кон'юнктивної нормальної форми (КНФ), досконалої диз'юнктивної нормальної форми (ДДНФ) та досконалої кон'юнктивної нормальної форми (ДКНФ).

Завданням розробки є створення інструменту, який автоматизує побудову зазначених форм, має зручний та інтуїтивно зрозумілий графічний інтерфейс, україномовну локалізацію та включає навчальні елементи (пояснення кроків побудови). Програмне забезпечення повинно бути орієнтоване на студентів, які вивчають основи цифрової логіки, та сприяти глибшому розумінню теоретичного матеріалу через практичну взаємодію.

Для досягнення поставленої мети передбачається виконати такі етапи:

1. Аналіз предметної області:
    - Дослідити теоретичні основи булевих функцій та методів побудови канонічних форм.
    - Розглянути особливості досконалих форм (ДДНФ і ДКНФ) та їх роль у логічному синтезі.
2. Огляд існуючого програмного забезпечення:
    - Вивчити наявні інструменти для побудови булевих форм.
    - Визначити їхні переваги та недоліки.
3. Формування вимог до програмного забезпечення:
    - Визначити функціональні та нефункціональні вимоги до системи.
    - Спроєктувати структуру інтерфейсу та логіку обробки булевих виразів.
4. Розробка програмного забезпечення:
    - Реалізувати механізми введення функцій у вигляді логічного виразу.
    - Здійснити алгоритмічну побудову ДНФ, КНФ, ДДНФ, ДКНФ.
5. Тестування та перевірка коректності:
    - Перевірити роботу ПЗ на прикладах з різною кількістю змінних.

- Провести оцінювання точності, стабільності та зручності використання.

6. Оформлення пояснювальної записки та інструкції користувача:
    - Документально зафіксувати результати розробки, опис функцій програми та процес використання.

Завдання полягає у побудові канонічних форм булевої функції на основі її опису у вигляді логічного виразу або таблиці істинності. Задача формалізується наступним чином:

Вхідні дані:

- набір булевих змінних $x_1, x_2,\dots x_n$,;
- логічний вираз, сформований із використанням базових логічних операторів (AND, OR, NOT і тп.), який задає булеву функцію.

Очікуваний результат:

- побудова таблиці істинності;
- побудова відповідних канонічних форм:
    - диз'юнктивна нормальна форма (ДНФ);
    - кон’юнктивна нормальна форма (КНФ);
    - досконала диз'юнктивна нормальна форма (ДДНФ);
    - досконала кон’юнктивна нормальна форма (ДКНФ);
- відображення проміжних етапів;
- представлення отриманих форм у зручному для перегляду вигляді.

Характеристики моделі:

- Детермінованість: побудова ДКНФ та ДДНФ відбувається за чітким і однозначним алгоритмом, що дозволяє отримувати стабільні результати;
- Гнучкість: програмний інструмент дозволяє працювати з виразами різної складності;

- Освітній фокус: модель реалізується у вигляді програмного інструменту з україномовним інтерфейсом, що дозволяє легко зрозуміти структуру функції та хід її побудови;
- Інтерактивність: користувач вводить логічний вираз вручну, після чого відбувається автоматична побудова канонічних форм із поясненнями.

Обмеження задачі:

- Вхідні вирази повинні містити лише латинські літери;
- Дотримання коректності введення дужок;
- Підтримуються лише задані логічні оператори.

Розроблене програмне забезпечення реалізує описану модель задачі з акцентом на навчальне застосування, допомагаючи користувачам зрозуміти механізм формування ДКНФ і ДДНФ для довільних булевих виразів.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Огляд робіт в яких розглянуті питання, схожі з темою роботи

Одним із важливих етапів підготовки до розробки програмного забезпечення є аналіз та порівняння з уже існуючими рішеннями, які реалізують схожі функції або вирішують подібні задачі. Такий огляд дозволяє точніше окреслити актуальність теми, виявити недоліки в наявних реалізаціях і визначити напрями, в яких новий проєкт може запропонувати вдосконалення. Крім того, він створює підґрунтя для подальших досліджень і розвитку обраної тематики.

На просторах Інтернету було обрано декілька варіантів подібного програмного забезпечення. А саме WolframAlpha, Karnaugh Map Tools, eMathHelp та Logic Calculator

При проведені інформаційного аналізу, першим було проаналізовано базу алгоритмів WolframAlpha (рис. 2.1).

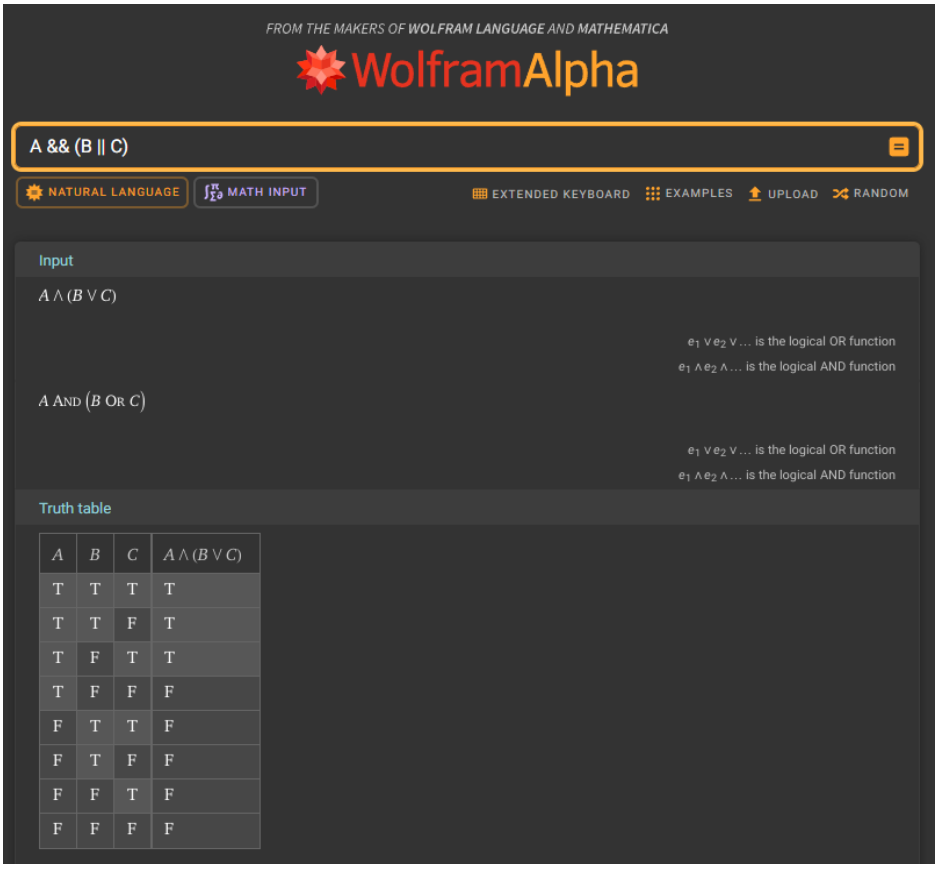

Рисунок 2.1 – Онлайн калькулятор WolframAlpha

Для аналізу було взято також інструмент для побудови карт Карно - Karnaugh Map Tools (рис. 2.2).

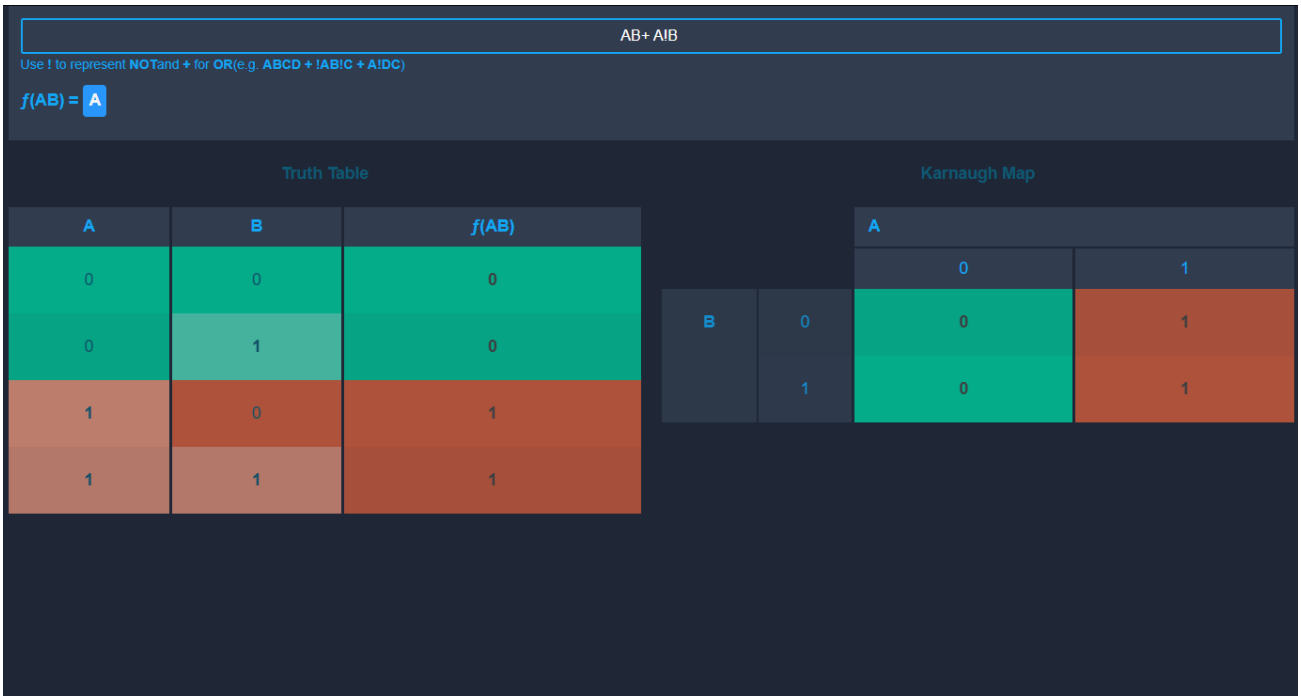

Рисунок 2.2 – Онлайн ПЗ для побудови карт Карно Karnaugh Map Tools

Наступним було розглянуто онлайн калькулятор математичної логіки Logic Calculator (рис. 2.3).

Рисунок 2.3 – Онлайн калькулятор математичної логіки Logic Calculator

Наступним було розглянуто онлайн калькулятор булевої алгебри eMathHelp (рис. 2.4).

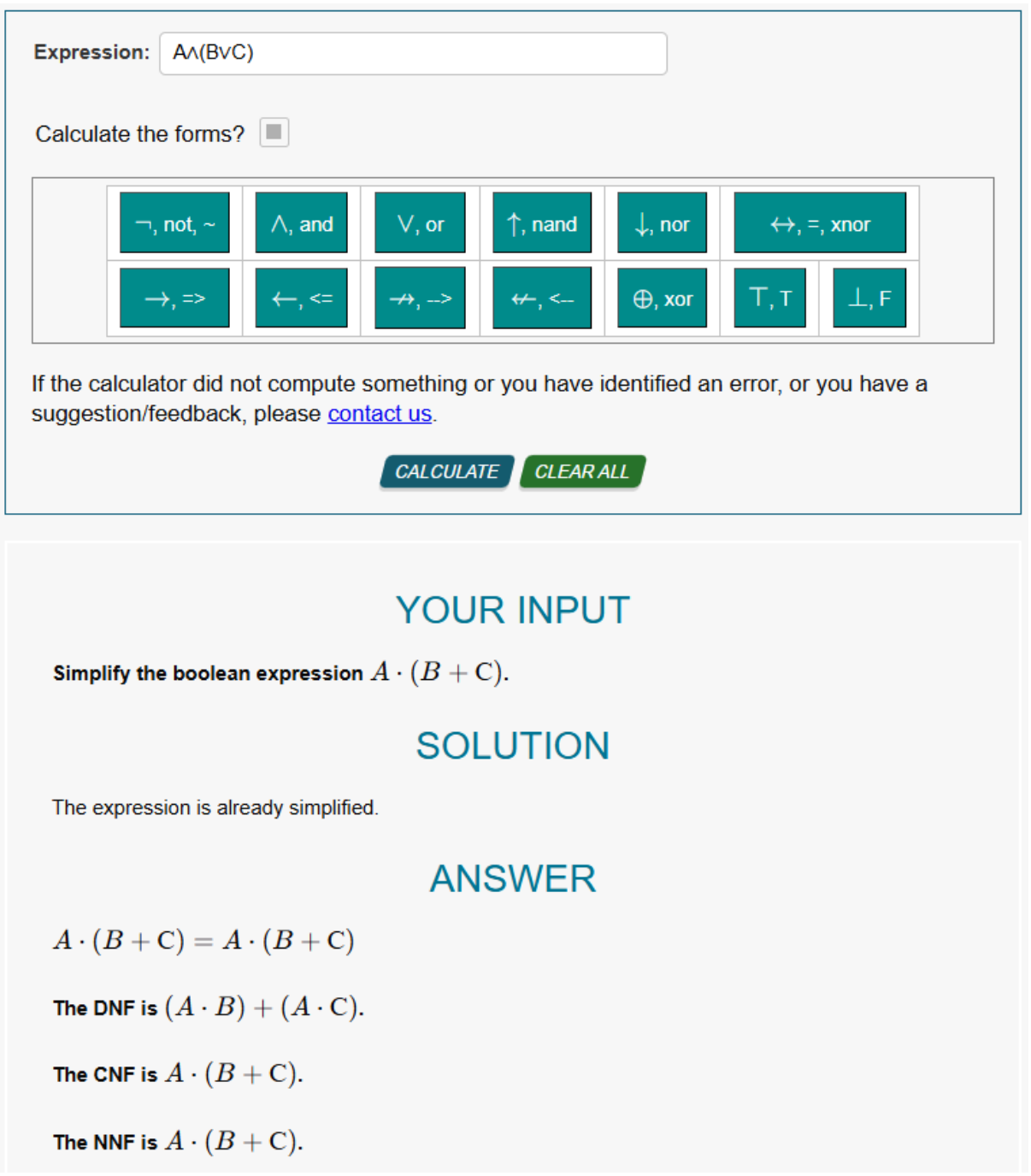

Рисунок 2.4 – Онлайн калькулятор булевої алгебри eMathHelp

2.2. Позитивні та негативні аспекти розглянутих робіт

Для повноцінного аналізу наявного програмного забезпечення доцільно оцінити їхні переваги та недоліки з точки зору ефективності. Особливо важливо враховувати такі критерії, як зручність інтерфейсу, наявність навчального супроводу, реалізація побудови канонічних форм, а також доступність для початківців.

База алгоритмів WolframAlpha має наступні переваги:

1. Потужний обчислювальний сервіс із підтримкою логічних виразів;
2. Можливість побудови таблиці істинності;
3. Працює зі складними багаторівневими логічними виразами;
4. Зручна інтеграція з іншими математичними інструментами.

Також було виявлено наступний ряд недоліків:

1. Відсутня автоматична побудова ДДНФ та ДКНФ.
2. Результат подається без покрокових пояснень.
3. Інтерфейс англомовний або німецькомовний.
4. Орієнтований більше на результат, ніж на навчання.

Позитивні сторони Karnaugh Map Tools:

1. Дає змогу візуально побудувати функцію на основі карти Карно.
2. Зручний інструмент для візуального спрощення логічних виразів.
3. Дає можливість вручну змінювати значення в таблиці.

Має наступні недоліки:

1. Не реалізує автоматичне формування ДКНФ/ДДНФ.
2. Вимагає ручного вводу значень – не працює без підготовки.
3. Відсутність пояснень до результату.
4. Орієнтований більше на інженерні задачі, ніж на навчальні.

При аналізі калькулятора математичної логіки Logic Calculator було виявлені наступні переваги:

1. Проста структура та інтерфейс, доступний для новачків.
2. Підтримка введення логічних виразів у текстовій формі.
3. Формує таблицю істинності.

Також було виявлено наступний ряд недоліків:

1. Неможливість перегляду логіки обробки виразу.
2. Тільки англомовна версія.
3. Відповідь відображається на новій сторінці

Для онлайн калькулятора булевої алгебри eMathHelp було виявлено наступні переваги:

1. Зручний синтаксис для введення логічних виразів.
2. Автоматичне формування таблиці істинності.
3. Працює швидко навіть з виразами середньої складності.

Програмне забезпечення містить наступні недоліки:

1. Відсутня побудова саме досконалих форм (ДДНФ/ДКНФ).
2. Побудова нормальних булевих форм та таблиці істинності знаходяться в різних розділах.
3. Інтерфейс тільки англійською мовою.

2.3. Обґрунтування актуальності теми роботи

Актуальність теми даної роботи зумовлена потребою у створенні зручного, доступного та навчально орієнтованого програмного інструменту для побудови канонічних форм булевих функцій. Знання та навички, пов’язані з аналізом логічних виразів, є фундаментальними для студентів технічних спеціальностей, зокрема в галузях комп’ютерних наук, електроніки, автоматизації та інформаційних технологій. Булева алгебра лежить в основі цифрової логіки, схемотехніки, проєктування мікропроцесорних пристроїв та програмного моделювання систем.

Опанування канонічних форм, таких як ДДНФ та ДКНФ, є необхідною складовою курсу дискретної математики, цифрової логіки та дисциплін з розробки

логічних схем. Проте на практиці побудова таких форм вручну є досить складною, особливо при збільшенні кількості змінних. Помилки на цьому етапі часто зумовлюють нерозуміння суті перетворень, що негативно позначається на якості засвоєння навчального матеріалу.

Аналіз існуючих програмних засобів показав, що більшість із них не реалізує повноцінної побудови ДДНФ і ДКНФ, не пояснює етапів формування виразів, має перевантажений інтерфейс, не підтримує українську мову. Більше того, наявні інструменти часто розроблені для інженерних чи виробничих потреб, а не для використання в навчальному процесі. Це обмежує їхню цінність як допоміжних засобів у процесі викладання та самостійного опрацювання логічних операцій.

У контексті переходу до цифрових форматів навчання, особливо в умовах дистанційного чи змішаного навчання, потреба у створенні інтуїтивного україномовного програмного забезпечення навчального призначення стає особливо актуальною. Таке ПЗ повинно не лише виконувати обчислення, а й пояснювати суть перетворень та дозволяти студенту аналізувати результат своєї роботи.

Розробка програмного засобу для побудови канонічних форм булевих функцій з простим інтерфейсом, можливістю введення логічного виразу та автоматичним виводом ДДНФ/ДКНФ сприятиме покращенню якості навчання, підвищенню рівня розуміння логічних основ цифрових систем та формуванню стійких практичних навичок у студентів.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Алгоритмізація задачі за завданням роботи

Побудова таблиці істинності - це перший і необхідний крок у процесі перетворення булевої функції у її канонічну форму. У логіці цей етап забезпечує повний опис логічної поведінки функції, а в програмному середовищі – формує основу для автоматичного синтезу нормальних форм.

Алгоритм побудови таблиці істинності реалізується згідно з такою логічною послідовністю:

1. Виділення змінних з виразу
   - Після введення логічного виразу користувачем система здійснює аналіз виразу з метою виявлення всіх унікальних змінних.
   - Для цього використовується синтаксичний розбір символів у рядку та відбір лише літер, що не є логічними операторами.
   - У результаті формується впорядкований список змінних, наприклад: ['A', 'B', 'C'].
2. Генерація наборів значень змінних
   - Для булевої функції з n змінними можливі 2^n комбінацій значень.
   - Алгоритм формує усі набори із нулів і одиниць – бінарні вектори довжини n, які представляють усі можливі стани системи.
   - Цей процес відповідає повному перебору області визначення функції:

$$\{0,1\}^n = \left\{\left(x_1, x_{2,\dots,} x_n\right) | x_i \in \{0,1\}\right\}$$

3. Обчислення значень функції
   - Кожен згенерований набір значень змінних підставляється у вихідний логічний вираз.
   - Операторна частина виразу перетворюється у форму, яку система може обчислити (наприклад, $\neg \rightarrow not,\ \wedge \rightarrow and,\ \vee \rightarrow\ or$).

- Для кожної комбінації обчислюється значення функції $f(x_1, \dots x_n)$ та додається до таблиці.

4. Формування таблиці

- Таблиця істинності формується як сукупність рядків, кожен з яких містить:
  - значення змінних (наприклад: A = 0, B = 1, C = 1);
  - значення функції для цього набору.
- У підсумку утворюється структура з 2^n рядків, яка повністю описує поведінку функції.

Алгоритм (словесний опис):

1. Прочитати логічний вираз, введений користувачем.
2. Знайти всі змінні у виразі.
3. Визначити кількість змінних n.
4. Згенерувати всі 2^n комбінацій значень змінних.
5. Для кожної комбінації:
   - підставити значення у вираз;
   - обчислити логічний результат;
   - зберегти набір і результат у таблиці.
6. Після обробки всіх комбінацій –вивести повну таблицю істинності.

З позиції математичної логіки, таблиця істинності виконує роль відображення всіх можливих логічних станів системи. Вона дозволяє не лише перевірити правильність логічного виразу, але й провести аналіз, синтез або перетворення функції у її канонічні подання – ДДНФ або ДКНФ. Таблиця також виступає як проміжний етап для перевірки еквівалентності логічних виразів, виявлення суперечностей, чи побудови логічних схем.

Кон’юнктивна нормальна форма (КНФ) є одним з основних алгебраїчних способів представлення булевих функцій. Вона відіграє важливу роль у логічному аналізі, побудові логічних схем і синтезі цифрових пристроїв. У загальному вигляді КНФ –це кон’юнкція диз’юнктів, де кожен диз’юнкт є елементарним виразом, що складається з кількох змінних, об’єднаних операцією диз’юнкції (логічне "або").

Формально:

$$f(x_1, \dots x_n) = D_1 \wedge D_2 \wedge \cdots \wedge D_m$$

де кожен $D_i = (l_1 \vee l_2 \vee \cdots \vee l_k)$,l_j – змінна або її заперечення.

Процес побудови КНФ ґрунтується на аналізі таблиці істинності функції. Для КНФ нас цікавлять усі ті набори значень змінних, при яких функція має значення 0 (тобто результат $False$). Саме вони визначають, які диз'юнкти увійдуть до виразу. Виконується наступна послідовність дій:

1. Створення таблиці істинності функції.
    - Побудована раніше таблиця істинності містить усі можливі набори значень змінних та відповідні значення функції.
2. Виділення рядків з результатом 0.
    - Для кожного рядка, в якому функція набуває значення 0, формується елементарний диз'юнкт, який виключає цю комбінацію.
3. Побудова диз'юнктів:
    - Кожен диз'юнкт утворюється так:
        - Якщо змінна у комбінації має значення 1 → у диз'юнкті записується її заперечення (¬x);
        - Якщо 0 → записується сама змінна (x);
    - Наприклад, для рядка (A = 0, B = 1, C = 0): диз'юнкт буде:

$$A \vee \neg B \vee C.$$

    - На відміну від ДКНФ, тут не обов'язково включати всі змінні. Якщо певні змінні не впливають на результат їх можна пропустити.
4. Поєднання всіх диз'юнктів за допомогою кон'юнкції (∧).
    - Усі диз'юнкти, отримані з кожного "нульового" рядка, об'єднуються логічною операцією "і".

КНФ повністю еквівалентна оригінальній булевій функції, але представлена у вигляді, зручному для логічного аналізу, перевірки виконуваності (наприклад, у задачах SAT) та реалізації за допомогою елементів цифрових схем. Кожен диз'юнкт

у КНФ гарантує "відсіювання" невірних (незадовільних) наборів значень. Таким чином, КНФ описує логіку, яка істинна скрізь, окрім точок, явно заборонених у диз'юнктах.

Диз'юнктивна нормальна форма (ДНФ) є однією з основних форм подання булевої функції. Вона широко застосовується в логічному аналізі, автоматизованому доведенні тверджень, синтезі логічних схем і в курсах дискретної математики як базова форма для розуміння структури логічних виразів.

У загальному вигляді ДНФ – це диз'юнкція (логічне "або") кон'юнкцій (логічне "і") змінних та/або їх заперечень.

Формально:

$$f(x_1, \dots x_n) = K_1 \vee K_2 \vee \cdots \vee K_m$$

де кожен $K_i = (l_1 \wedge l_2 \wedge \cdots \wedge l_k)$, l_j – змінна або її заперечення.

Побудова ДНФ базується на аналізі таблиці істинності функції. Для цієї форми важливими є рядки таблиці, в яких функція набуває значення 1 (тобто $True$). Кожен такий рядок визначає одну елементарну кон'юнкцію, яка "включає" відповідну комбінацію змінних у підсумкову формулу.

Послідовність дій:

1. Формування таблиці істинності.
    - Таблиця містить усі можливі комбінації значень змінних та результат функції.
2. Вибір "одиничних" рядків.
    - Відбираються лише ті рядки, де значення функції дорівнює 1.
3. Побудова кон'юнктів:
    - Для кожного обраного рядка створюється кон'юнкція змінних:
        - Якщо змінна має значення 1 – до кон'юнкції входить сама змінна.
        - Якщо 0 — її заперечення ($\neg x$).
    - Наприклад, для набору (A=0, B=1, C=1) відповідний кон'юнкт:

$$\neg A \wedge \mathrm{B} \wedge \mathrm{C}.$$

4. Об’єднання всіх кон’юнктів диз’юнкцією:
    - Отримані кон’юнкції поєднуються логічною операцією "або" (∨) в одну формулу – ДНФ.

ДНФ є повним логічним описом функції в межах області істинності. Кожна кон’юнкція у формулі точно відповідає одному набору значень змінних, за якого функція істинна. Таким чином, ДНФ – це опис умов, за яких функція приймає значення 1.

Ця форма особливо корисна для аналізу систем із заданими дозволеними станами, при оптимізації логічних виразів та для реалізації логічних схем у вигляді "сум логічних добутків".

Досконала кон’юнктивна нормальна форма (ДКНФ) – це спеціальний випадок звичайної КНФ, у якій кожен диз’юнкт повністю відповідає одному конкретному рядку таблиці істинності, де функція набуває значення 0. На відміну від скороченої КНФ, у досконалій формі кожен диз’юнкт містить усі змінні функції (у прямому або запереченому вигляді), тому така форма є унікальною для кожної булевої функції.

ДКНФ забезпечує повний опис усіх умов, за яких функція хибна, і є одним із найточніших логічних подань.

ДКНФ має вигляд:

$$f(x_1, \dots x_n) = D_1 \wedge D_2 \wedge \dots \wedge D_m$$

де кожен диз’юнкт D_i – це диз’юнкція з усіх змінних функції, і побудований так, щоб функція була рівною нулю лише тоді, коли всі диз’юнкти істинні.

Алгоритм побудови ДКНФ

1. Побудова таблиці істинності функції.
    - Повна таблиця всіх можливих комбінацій значень змінних та відповідного значення функції.
2. Вибір рядків, де функція дорівнює 0.
    - Лише ці рядки формують основу для побудови ДКНФ.

3. Формування диз’юнктів (макстермів):
    - Кожен диз’юнкт будується на основі конкретного рядка:
        - Якщо змінна у цьому рядку дорівнює 1 → у диз’юнкті записується її заперечення ($\neg x$);
        - Якщо змінна дорівнює 0 → записується сама змінна (x);
    - Таким чином, диз’юнкт виключає саме цю комбінацію.
    - Наприклад, для набору (A=0, B=1, C=0), диз’юнкт буде:

$$A \vee \neg B \vee C$$

4. Об’єднання всіх диз’юнктів операцією "і" ($\wedge$):
    - Усі побудовані диз’юнкти поєднуються у єдину формулу – ДКНФ.
5. Якщо деякий член α КНФ не містить змінної x, вона вводиться на основі закону логічного протиріччя $x\neg x = 0$ та властивості диз’юнкції $\alpha \vee 0 = \alpha$.
6. Робиться спрощення одержаних формул за ідемпотентністю: $\alpha \vee \alpha = \alpha$; $\alpha\alpha = \alpha$.

Особливості ДКНФ:

- Повна форма – кожен диз’юнкт містить усі змінні, незалежно від того, чи впливають вони на значення функції.
- Унікальність – для кожної булевої функції існує єдина досконала КНФ.
- Пряма залежність від таблиці істинності – кожен диз’юнкт у ДКНФ відповідає конкретному набору, де функція хибна.

Приклад

Для булевої функції з трьома змінними:

$$f(A, B, C)$$

якщо в таблиці істинності функція дорівнює 0 при наборах:

- (A=0, B=0, C=1),
- (A=1, B=0, C=0),

то ДКНФ буде:

$$(A \vee B \vee \neg C) \wedge (\neg A \vee B \vee C)$$

Досконала диз’юнктивна нормальна форма (ДДНФ) – це особливий випадок диз’юнктивної нормальної форми, в якій кожен кон’юнкт відповідає точному набору змінних, за якого булева функція набуває значення 1. На відміну від звичайної (скороченої) ДНФ, у досконалій формі кожна змінна обов’язково входить до кожного кон’юнкта – або в прямому вигляді, або в запереченні.

ДДНФ є унікальною для кожної булевої функції та є найдеталізованішим логічним поданням області істинності функції.

ДДНФ має вигляд:

$$f(x_1, \dots x_n) = K_1 \vee K_2 \vee \dots \vee K_m$$

де кожен $K_i = (l_1 \wedge l_2 \wedge \dots \wedge l_k)$, причому кожен літерал l_j – це або змінна x_j, або її заперечення $-x_j$, відповідно до конкретної комбінації значень.

Алгоритм побудови ДДНФ

1. Побудова таблиці істинності функції.
    - Таблиця відображає всі можливі набори значень змінних та відповідне значення функції.
2. Вибір "одиничних" рядків.
    - Аналізуються лише ті рядки, в яких функція набуває значення 1 (тобто результат = $True$).
3. Формування кон’юнктів (мінтермів):
    - Для кожного такого рядка створюється повна кон’юнкція з усіх змінних:
        - Якщо значення змінної = 1 → записується сама змінна;
        - Якщо значення = 0 → записується її заперечення (¬x).
    - Наприклад, для набору (A=1, B=0, C=1), відповідний мінтерм:

$$A \wedge \neg \text{B} \wedge \text{C}.$$

4. Об’єднання всіх мінтермів через диз’юнкцію (логічне "або"):
    - Отримані кон’юнкції поєднуються у вираз за допомогою ∨.

5. Якщо деякий член α ДНФ не містить змінної x, вона вводиться на основі закону виключенного третього і властивості кон'юнкції $x \vee \neg x = 1$, $\alpha 1 = \alpha$.
6. Робиться спрощення одержаних формул за ідемпотентністю: $\alpha \vee \alpha = \alpha$; $\alpha\alpha = \alpha$.

Особливості ДДНФ:

- Повна форма: кожен терм включає всі змінні функції.
- Унікальність: для кожної булевої функції існує лише одна ДДНФ.
- Повне охоплення: ДДНФ охоплює всю область істинності функції, описуючи її детально.

Приклад

Для функції трьох змінних:

$$f(A, B, C)$$

яка дорівнює 1 при наборах:

- (A=1, B=0, C=1),
- (A=0, B=1, C=1),

відповідна ДДНФ буде:

$$(A \wedge B\neg \wedge C) \vee (\neg A \wedge B \wedge C)$$

3.2. Розробка блок-схеми, яка підлягає програмуванню

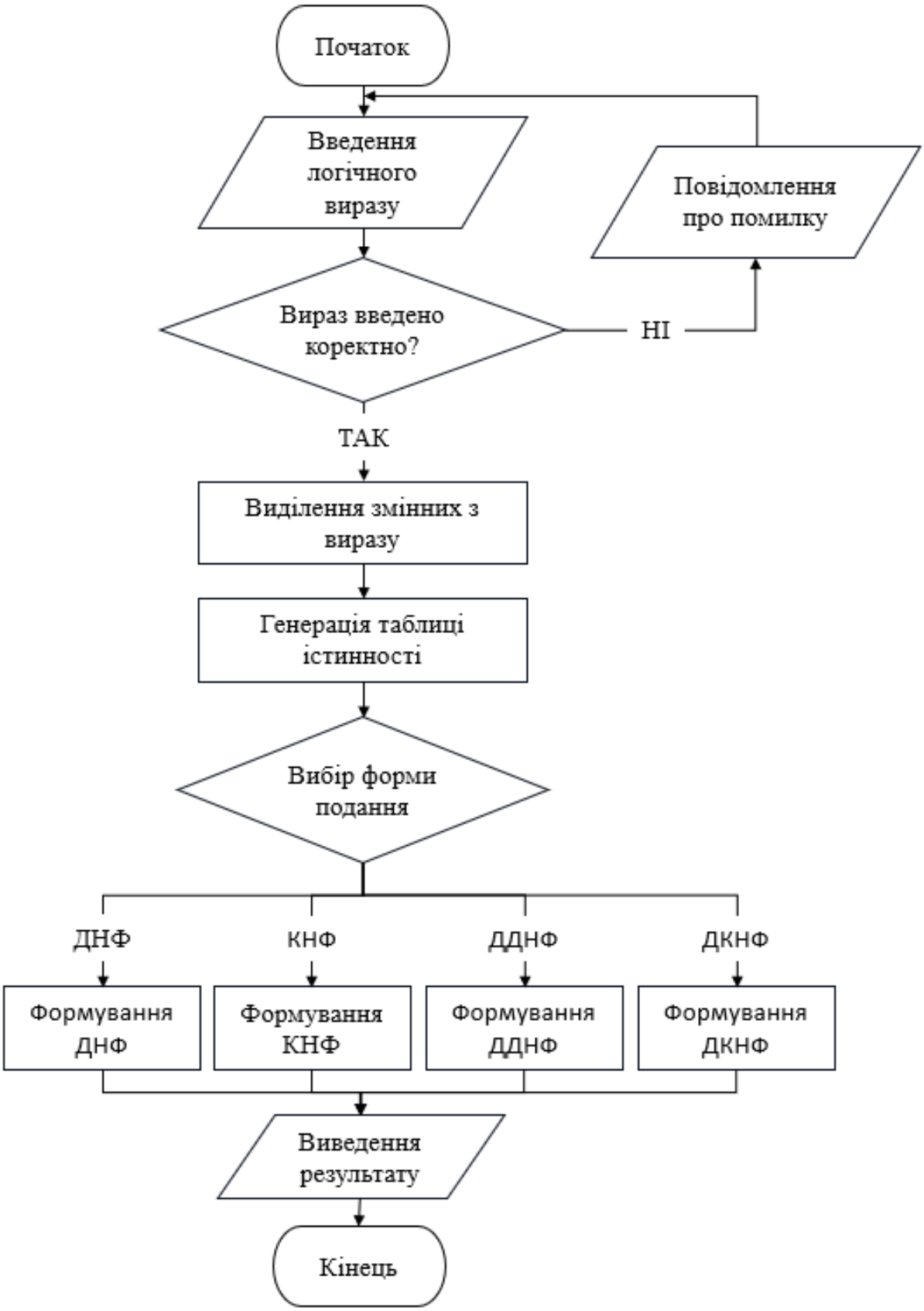

Рисунок 3.1 – Блок-схема роботи алгоритму

3.3.Обгрунтування вибору програмних інструментів для реалізації теми роботи

Мова програмування Python була створена голландським програмістом Гвідо ван Россумом наприкінці 1980-х років, а її перший публічний реліз відбувся у 1991 році. Розробка цієї мови мала на меті створення інструменту, що поєднує простоту у вивченні, зручність у читанні коду та ефективність у вирішенні як навчальних, так і практичних задач.

Python є мовою високого рівня з інтерпретованим виконанням та автоматичним керуванням пам’яттю. Вона підтримує кілька парадигм програмування, зокрема процедурне, об’єктно-орієнтоване та функціональне, що робить її універсальним інструментом для різноманітних сфер застосування – від аналізу даних і машинного навчання до веброзробки, автоматизації систем і створення настільного програмного забезпечення.

Однією з головних переваг Python є її лаконічний і зрозумілий синтаксис, що значно полегшує процес навчання та супроводу програм. Завдяки своїй простоті та великій спільноті розробників Python став однією з найпопулярніших мов програмування у світі. У сфері освіти вона посіла особливе місце як базова мова для навчання програмуванню, оскільки дозволяє швидко перейти від теоретичних понять до реалізації прикладних алгоритмів.

Основні особливості Python

Мова програмування Python вирізняється низкою особливостей, що зробили її однією з найпопулярніших та найзручніших мов як для початківців, так і для професійних розробників. Її ключова відмінність – це орієнтація на простоту, читабельність та зрозумілість коду, що безпосередньо впливає на швидкість розробки програм та легкість їх подальшої підтримки.

Однією з основних особливостей Python є високий рівень абстракції. Програми на Python мають мінімум «технічного шуму» – у коді відсутні зайві

конструкції, властиві багатьом іншим мовам (наприклад, типізація змінних у Java чи C++). Це дозволяє зосередитись безпосередньо на логіці задачі.

Python є інтерпретованою мовою, тобто код виконується без компіляції. Це дозволяє швидко тестувати фрагменти програми, що є особливо зручним на етапі навчання або прототипування. Крім того, мова підтримує автоматичне керування пам’яттю, що позбавляє програміста необхідності вручну слідкувати за ресурсами.

Ще однією важливою особливістю є підтримка декількох парадигм програмування: процедурного, об’єктно-орієнтованого, функціонального, а також елементів модульного та подійного підходів. Така гнучкість дає змогу реалізовувати прості скрипти і великі програмні системи в межах однієї мови.

Python має велику стандартну бібліотеку, яка містить інструменти для роботи з текстом, файлами, мережами, математикою, логікою, базами даних тощо. Крім цього, доступна величезна кількість зовнішніх бібліотек для розширення функціоналу – від обробки зображень до машинного навчання. Ця екосистема робить Python універсальним інструментом у сучасному програмуванні.

І, нарешті, Python – це кросплатформенна мова, яка працює на всіх основних операційних системах: Windows, Linux, macOS. Це дозволяє розробляти програмне забезпечення, яке легко адаптується до будь-якого середовища користувача без необхідності переписування коду.

Завдяки поєднанню цих особливостей Python визнано ідеальним інструментом для створення як навчального, так і прикладного програмного забезпечення – зокрема, для задач обробки булевих функцій та побудови їх канонічних форм, як у межах цієї курсової роботи.

Python має низку суттєвих переваг, що роблять його популярним серед розробників та особливо придатним для навчальних цілей. До основних переваг належать:

- Простий та читабельний синтаксис, що наближений до природної мови й полегшує навчання.
- Висока швидкість розробки завдяки зручним засобам роботи з даними та відсутності необхідності в оголошенні типів.

- Велика стандартна бібліотека та розвинена екосистема сторонніх модулів.
- Кросплатформеність – код Python виконується на різних операційних системах без змін.
- Підтримка кількох парадигм програмування – процедурної, об’єктно-орієнтованої та функціональної.
- Автоматичне управління пам’яттю, що зменшує ризик помилок.
- Широке використання в освіті, науці, автоматизації, аналізі даних і веброзробці.
- Завдяки цим перевагам Python став доцільним вибором для реалізації навчального програмного засобу в межах даної курсової роботи.

Завдяки цим перевагам Python став доцільним вибором для реалізації навчального програмного засобу в межах даної курсової роботи.

Середовище програмування Python є гнучким, зручним і добре пристосованим для розробки додатків різної складності – від простих скриптів до повноцінних настільних або вебпрограм. Python не вимагає окремого компілятора: достатньо інтерпретатора Python та текстового редактора або інтегрованого середовища розробки (IDE), що значно спрощує процес створення і тестування програм.

Розглянемо середовища які найчастіше використовуються для роботи з Python.

1. IDLE

Integrated Development and Learning Environment – стандартне середовище, яке постачається разом із Python. Має простий інтерфейс, підтримує виконання коду, автодоповнення та налагодження. Ідеальне для початківців

2. PyCharm

PyCharm – потужне професійне середовище розробки, розроблене компанією JetBrains. Має вбудовану підтримку роботи з проєктами, інтелектуальне автозаповнення, аналіз коду, відлагоджувач і підтримку версійного контролю (Git). Підходить для створення складніших додатків, у тому числі графічних або вебзастосунків.

3. Visual Studio Code

VS Code – легке, але дуже гнучке середовище з розширенням для Python. Дає змогу швидко створювати проєкти, запускати і відлагоджувати код, працювати з віртуальними середовищами.

4. Jupyter Notebook

Jupyter Notebook – використовується здебільшого для наукових і освітніх задач, дозволяє поєднувати код, текст, формули та графіку в одному документі. У певних випадках може бути зручним і для демонстрації логіки булевих функцій.

У межах реалізації програмного забезпечення для побудови канонічних форм булевих функцій було використано середовище розробки PyCharm. Завдяки підтримці модульної структури це середовище дозволяє легко керувати кількома файлами, що містять окремі функціональні блоки програми (інтерфейс, обчислення, побудова форм). Використання PyCharm забезпечило комфортну розробку, зручне налагодження логіки програми й підвищило загальну якість реалізації програмного коду.

4. ПРАКТИЧНА ЧАСТИНА

4.2. Опис процесу реалізації програми

Розроблене програмне забезпечення побудовано за принципом модульності, що означає поділ усього функціоналу на незалежні логічні компоненти. Кожен з них відповідає за конкретну задачу в загальному процесі обчислення булевих функцій та побудови їх канонічних форм. Такий підхід забезпечує зручність у розробці, тестуванні та подальшому розширенні програми. Розглянемо по черзі основні модулі програми.

1.Модуль boolean_calculator_ui.py

Модуль *boolean_calculator_ui* відповідає за створення графічного вікна програми, через яке користувач може:

- ввести логічний вираз;
- вибрати тип нормальної форми (ДНФ, КНФ, ДДНФ, ДКНФ);
- запустити обчислення;
- переглянути результат у зручному форматі.

Інтерфейс реалізований із використанням бібліотеки tkinter.

Спочатку створюється клас графічного інтерфейсу, що містить усі методи та елементи GUI. Весь інтерфейс створюється всередині цього класу (Рисунок 4.1).

```
class BooleanCalculatorUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Калькулятор булевих функцій")
        self.root.geometry("550x400")

        self.evaluator = BooleanExpressionEvaluator()
        self.table_generator = TruthTableGenerator(self.evaluator)
        self.normal_form_builder = NormalFormBuilder(self.table_generator)

        self.variables = set()
        self.function_entry = None
        self.output_text = None
        self.notebook = None

        self._setup_welcome_screen()
```

Рисунок 4.1. – Реалізація стартового вікна

При запуску програми створюється стартове інформаційне вікно, яке веде до основного функціоналу програми. Воно містить інформацію про тему роботи, виконавця роботи та назву вищого навчального закладу, в якому ця робота виконувалася. Вікно містить кнопку «Продовжити» (Рисунок 4.2).

```
def _setup_welcome_screen(self):  1 usage
    self.welcome_frame = ttk.Frame(self.root)
    self.welcome_frame.pack(fill='both', expand=True)

    welcome_label = ttk.Label(
        self.welcome_frame,
        text="""
        Кваліфікаційна робота з теми:

        "Реалізація навчального програмного забезпечення
        для побудови канонічних форм булевих функцій"

        Виконав студент групи КН 6-41
        Волторніст Артем

        ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
        """,
        font=('Helvetica', 12, 'bold')
    )
    welcome_label.pack(pady=50)

    continue_btn = ttk.Button(
        self.welcome_frame,
        text="Продовжити",
        command=self._setup_main_interface
    )
    continue_btn.pack(pady=20)
```

Рисунок 4.2. – Реалізація стартового вікна

Після натискання на кнопку «Продовжити» викликається метод *setup_main_interface*, стартове вікно видаляється та заміняється на вікно основного інтерфейсу. Змінюється розмір головного вікна з 550x400 на 800x600 для зручного розміщення елементів управління. Створюється дві основні вкладки: *self.calculator_tab* – вкладка з функціональною складовою, *self.instructions_tab* – вкладка з інформацією для користувача. (Рисунок 4.3)

```
def _setup_main_interface(self):  1 usage
    self.welcome_frame.destroy()

    self.root.geometry("800x600")

    self.notebook = ttk.Notebook(self.root)
    self.notebook.pack(fill='both', expand=True, padx=10, pady=10)

    self.calculator_tab = ttk.Frame(self.notebook)
    self.notebook.add(self.calculator_tab, text='Калькулятор')
    self._setup_calculator_tab()

    self.instructions_tab = ttk.Frame(self.notebook)
    self.notebook.add(self.instructions_tab, text='Інструкція')
    self._setup_instructions_tab()
```

Рисунок 4.3. – Реалізація основного інтерфейсу

Створюються панелі для введення логічного виразу *self._create_input_frame()*, панель з кнопками логічних операцій *self._create_buttons_frame()*, панель з кнопками запуску потрібних обчислень *self._create_compute_buttons_frame()* та область куди виводиться результат *self._create_output_frame()* (Рисунок 4.4)

```
def _setup_calculator_tab(self):  1 usage
    self._create_input_frame()
    self._create_buttons_frame()
    self._create_compute_buttons_frame()
    self._create_output_frame()
```

Рисунок 4.4. – Реалізація панелей основного інтерфейсу

```
def _setup_instructions_tab(self):  1 usage
    instructions_text = tk.Text(self.instructions_tab, wrap="word", padx=10, pady=10)
    instructions_text.pack(fill="both", expand=True)

    instructions = """
    Інструкція з використання калькулятора булевих функцій
    """

    instructions_text.insert( index: "1.0", instructions)
    instructions_text.configure(state="disabled")
```

Рисунок 4.5. – Реалізація інструкції користувача

В панель *self.instructions_tab,* за допомогою віджета для відображення тексту *tk.Text* з заданими параметрами, додається текстова інструкція. (Рисунок 4.5.)

Далі за допомогою елементу *Entry* реалізується поле для введення булевої функції з заданою шириною та збільшеним розміром шрифту, для зручнішого читання виразу. Додаються текстові позначення для цього поля. (Рисунок 4.6.)

```
def _create_input_frame(self):  1 usage
    input_frame = ttk.LabelFrame(self.calculator_tab, text="Введіть булеву функцію")
    input_frame.pack(pady=10, padx=10, fill="x")

    ttk.Label(input_frame, text="Функція:").grid(row=0, column=0, padx=5, pady=5)

    self.function_entry = ttk.Entry(
        input_frame,
        width=80,
        font=('TkDefaultFont', 15)
    )
    self.function_entry.grid(
        row=0, column=1,
        padx=5, pady=5,
        columnspan=2,
        sticky="ew"
    )

    input_frame.columnconfigure( index: 1, weight=1)
```

Рисунок 4.6. – Реалізація поля введення виразу

Слідуючим етапом була реалізація панелі кнопок логічних виразів. Для кожного оператора створена окрема кнопка (*op*) та за допомогою методу *_insert_operator()* заданий відповідний символ. Кнопки розташовані у один рядок та кожна в окремій колонці з заданим відступом для відокремлення (Рисунок 4.7.).

```
def _create_buttons_frame(self):  1 usage
    buttons_frame = ttk.Frame(self.calculator_tab)
    buttons_frame.pack(pady=10)

    operators = ["¬", "∧", "∨", "→", "⊕", "≡", "(", ")"]
    for i, op in enumerate(operators):
        ttk.Button(buttons_frame, text=op,
                   command=lambda o=op: self._insert_operator(o)) \
            .grid(row=0, column=i, padx=5)
```

Рисунок 4.7. – Реалізація кнопок логічних виразів

Наступним кроком була реалізація панелі для запуску основних обчислень. Вона відбувалась схожою з попереднім етапом схемою. Створюється фрейм у вкладці калькулятора із заданим відступом від інших елементів, визначається список кнопок, які представлені як кортеж із текстом на кнопці та відповідною функцією, що виконується при натисканні. Кнопка «Таблиця істинності» викликає метод *_show_truth_table*, який будує таблицю істинності. «ДНФ», «КНФ», «ДДНФ», «ДКНФ» - викликають метод *_build_normal_form* із відповідним параметром для побудови потрібної нормальної форми. Кнопки розташовані в один рядок, кожна у своїй колонці (Рисунок 4.8.).

```
def _create_compute_buttons_frame(self):  1 usage
    compute_frame = ttk.Frame(self.calculator_tab)
    compute_frame.pack(pady=10)

    buttons = [
        ("Таблиця істинності", self._show_truth_table),
        ("ДНФ", lambda: self._build_normal_form("ДНФ")),
        ("КНФ", lambda: self._build_normal_form("КНФ")),
        ("ДДНФ", lambda: self._build_normal_form("ДДНФ")),
        ("ДКНФ", lambda: self._build_normal_form("ДКНФ"))
    ]

    for i, (text, command) in enumerate(buttons):
        ttk.Button(compute_frame, text=text, command=command) \
            .grid(row=0, column=i, padx=5)
```

Рисунок 4.8. – Реалізація кнопок для запуску основних обчислень

Далі стояло завдання реалізувати область виводу результатів. Було створено багаторядкове текстове поле із заданою висотою в 15 рядків та можливістю вікну розтягуватись у межах рамки. Це основна зона де з’являється результат після натискання кнопок. За допомогою віджету *Scrollbar*, праворуч, було додано смугу прокрутки, яка займає всю вертикальну висоту рамки (Рисунок 4.9.). Також було додано метод *_insert_operator* для вставки логічного символу у полу введення функції коли користувач натискає одну із відповідних кнопок (Рисунок 4.10.).

```
def _create_output_frame(self):  1 usage
    output_frame = ttk.LabelFrame(self.calculator_tab, text="Результат")
    output_frame.pack(pady=10, padx=10, fill="both", expand=True)

    self.output_text = tk.Text(output_frame, wrap="word", height=15)
    self.output_text.pack(pady=5, padx=5, fill="both", expand=True)

    scrollbar = ttk.Scrollbar(output_frame, orient="vertical",
                              command=self.output_text.yview)
    scrollbar.pack(side="right", fill="y")
    self.output_text.configure(yscrollcommand=scrollbar.set)
```

Рисунок 4.9. – Реалізація поля виводу результату обчислень

```
def _insert_operator(self, operator):  1 usage
    self.function_entry.insert(tk.END, operator)
```

Рисунок 4.10. – Реалізація поля виводу результату обчислень

Наступним кроком за допомогою методу get() реалізовувалась можливість витягування змінних із введеного логічного виразу, перевірка його коректності та підготовка до подальших обчислень. Якщо вираз порожній *messagebox.showerror* виводить повідомлення про помилку. Метод *extract_variables* аналізує текст виразу та витягує всі унікальні змінні (наприклад ‘A’, ‘B’, ‘C’). Якщо змінної не знайдено, *messagebox.showerror* також виводить повідомлення про помилку. Список змінних зберігається для подальшого використання (Рисунок 4.11.).

```
def _get_variables(self):  3 usages
    expr = self.function_entry.get()

    if not expr:
        messagebox.showerror( title: "Помилка",  message: "Введіть булеву функцію")
        return None

    variables = self.evaluator.extract_variables(expr)

    if not variables:
        messagebox.showerror( title: "Помилка",  message: "Не знайдено змінних у виразі")
        return None

    self.variables = set(variables)
    return variables
```

Рисунок 4.11. – Реалізація методу витягування змінних та їх перевірка

Далі зчитується логічний вираз введений користувачем. За допомогою модуля *self.table_generator* генерується таблиця істинності – перелік усіх можливих комбінацій значень змінних і відповідних результатів виразу. Обчислюється значення виразу, якщо всі змінні дорівнюють *True*.Метод *self.output_text.delete* Очищає текстове поле виводу, а *self.output_text.insert* виводить результат обчислень. Далі виводиться повна таблиця істинності на основі раніше отриманих даних. У разі помилки користувач отримує повідомлення з поясненням (Рисунок 4.12.).

```
def _evaluate_function(self):
    variables = self._get_variables()
    if not variables:
        return

    expr = self.function_entry.get()
    try:
        truth_table = self.table_generator.generate(expr, variables)
        result = self.evaluator.calculate_expression(expr, {v: True for v in variables})

        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, f"Результат обчислення: {result}\n\n")
        self._display_truth_table(truth_table, variables, expr)

    except Exception as e:
        messagebox.showerror( title: "Помилка",  message: f"Невірний вираз: {str(e)}")
```

Рисунок 4.12. – Реалізація побудови таблиці істинності

Наступним кроком була розробка методу для відображення таблиці істинності булевої функції введеної користувачем. Спочатку *_get_variables* витягує змінні з логічного виразу, потім *function_entry.get()* зчитує булевий вираз введений у відповідне полу (*Entry*). Після чого викликається метод *generate()* з модуля *table_generator* який створює всі можливі комбінації значень змінних, обчислює значення функції для кожної комбінації та формує повну таблицю істинності. Очищається текстове поле, де буде виводитися таблиця. Викликається метод *_display_truth_table()*, який оформлює таблицю в текстовий вигляд, додає заголовки колон та виводить таблицю в *output_text*. Також передбачено, в разі необхідності, повідомлення про помилку. (Рисунок 4.13.).

```
def _show_truth_table(self):  1 usage
    variables = self._get_variables()
    if not variables:
        return

    expr = self.function_entry.get()
    try:
        truth_table = self.table_generator.generate(expr, variables)
        self.output_text.delete(1.0, tk.END)
        self._display_truth_table(truth_table, variables, expr)

    except Exception as e:
        messagebox.showerror( title: "Помилка",  message: f"Невірний вираз: {str(e)}")
```

Рисунок 4.13. – Реалізація генерації та виводу таблиці істинності

За реалізацію відображення таблиці істинності у текстовому полі відповідає метод *def _display_truth_table()*. Він отримує вже підготовлену таблицю від генератора та виводить її у зручному вигляді для користувача. Додається заголовок «Таблиця істинності», рядок заголовків таблиці та виводиться в текстове поле. Цикл *for row in truth_table:* проходить по кожному рядку, перетворює значення змінних, усі змінні об’єднуються в рядок, після чого виводяться в поле *output_text* (Рисунок 4.14.).

```
def _display_truth_table(self, truth_table, variables, expr):  2 usages
    self.output_text.insert(tk.END, "Таблиця істинності:\n")
    headers = variables + [expr]
    self.output_text.insert(tk.END, "\t".join(headers) + "\n")

    for row in truth_table:
        values = [str(int(row[v])) for v in variables] + [str(int(row['result']))]
        self.output_text.insert(tk.END, "\t".join(values) + "\n")
```

Рисунок 4.14. – Реалізація відображення таблиці істинності

Далі формується метод що відповідає за побудову однієї з чотирьох канонічних форм булевої функції на основі введеного виразу. Спочатку отримується список змінних із виразу, зчитується введений булевий вираз та генерується таблиця істинності. Потім створюється словник відповідностей між назвою форми і функцією, яка її будує. *normal_form* викликає відповідну функцію побудови і отримує фінальний вираз у заданій формі. Метод *get_steps()* отримує етапи побудови. Далі виводяться етапи побудови, табличні або рядкові, та їх опис. Після чого виводиться фінальний вираз у потрібній формі. Якщо під час побудови сталася помилка – виводиться повідомлення для користувача з поясненням (Рисунок 4.15.).

```
def _build_normal_form(self, form_type):  4 usages
    variables = self._get_variables()
    if not variables:
        return

    expr = self.function_entry.get()
    try:
        truth_table = self.table_generator.generate(expr, variables)

        form_builders = {
            "ДНФ": self.normal_form_builder.build_dnf,
            "КНФ": self.normal_form_builder.build_cnf,
            "ДДНФ": self.normal_form_builder.build_perfect_dnf,
            "ДКНФ": self.normal_form_builder.build_perfect_cnf
        }

        normal_form = form_builders[form_type](truth_table, variables)
        steps = self.normal_form_builder.get_steps()

        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, f"{form_type} для функції {expr}:\n\n")

        self.output_text.insert(tk.END, "Етапи побудови:\n")
        for i, (desc, data) in enumerate(steps, 1):
            self.output_text.insert(tk.END, f"{i}. {desc}\n")
            if data:
                if isinstance(data, list):
                    headers = list(data[0].keys()) if data else []
                    self.output_text.insert(tk.END, "\t" + "\t".join(headers) + "\n")
                    for row in data:
                        values = [str(int(row.get(k, ''))) for k in headers]
                        self.output_text.insert(tk.END, "\t" + "\t".join(values) + "\n")
                else:
                    self.output_text.insert(tk.END, f"\t{data}\n")
            self.output_text.insert(tk.END, "\n")

        self.output_text.insert(tk.END, f"\nФінальна {form_type}:\n{normal_form}")

    except Exception as e:
        messagebox.showerror( title: "Помилка",  message: f"Помилка при побудові форми: {str(e)}")
```

Рисунок 4.15. – Реалізація побудови канонічної форми

2.Модуль boolean_expression_evaluator.py

Модуль *boolean_expression_evaluator* відповідає за аналіз і обчислення булевих виразів. Його основна задача – інтерпретувати логічний вираз, введений користувачем і правильно його обчислити.

Першим кроком у побудові цього модуля була підстановка значень змінних у вираз за допомогою метеду *replace()*. Він заміняє символ змінної на відповідне значення (*True* або *False*). Далі логічні символи перетворюються у оператори. Функція *eval()* виконує отриманий вираз як Python-код. Після обчислень значення приводяться до логічного типу *bool()*, щоб у результаті завжди поверталось *True* або *False*. Якщо у процесі підстановки або виконання *eval()* виникає помилка генерується повідомлення «Невірний вираз». Наступний модуль перебирає кожен символ у виразі *expr*, перевіряє чи є він буквою та повертає відсортований список змінних до *variables* (Рисунок 4.16.).

```
class BooleanExpressionEvaluator:
    @staticmethod  2 usages (2 dynamic)
    def calculate_expression(expr, values):
        replaced_expr = expr
        for var, val in values.items():
            replaced_expr = replaced_expr.replace(var, str(val))

        replaced_expr = replaced_expr.replace("¬", " not ")
        replaced_expr = replaced_expr.replace("∧", " and ")
        replaced_expr = replaced_expr.replace("∨", " or ")
        replaced_expr = replaced_expr.replace("→", " <= ")
        replaced_expr = replaced_expr.replace("⊕", " != ")
        replaced_expr = replaced_expr.replace("≡", " == ")

        try:
            return bool(eval(replaced_expr))
        except:
            raise ValueError("Невірний вираз")

    @staticmethod  1 usage (1 dynamic)
    def extract_variables(expr):
        variables = set()
        for char in expr:
            if char.isalpha() and char not in ['∧', '∨', '¬', '→', '⊕', '≡']:
                variables.add(char)
        return sorted(variables)
```

Рисунок 4.16. – Реалізація обчислення значень булевих виразів

3.Модуль truth_table_generator.py

Модуль *truth_table_generator* відповідає за генерацію таблиці істинності для булевого виразу.

Клас приймає у конструктор об'єкт *evaluator*, який має метод *calculate_expression()* – він використовується для обчислення логічних виразів. Спочатку будується порожня таблиця істинності та створюються всі можливі комбінації значень змінних. Від кількості змінних залежить кількість комбінацій. Створюється словник *dict()*, який зіставляє кожну змінну з її значенням. Далі обчислюється результат логічного виразу *expr* при коректних значеннях змінних. За допомогою операції *append()* рядок додається до загальної таблиці. Готова таблиця істинності повертається у вигляді списку словників (Рисунок 4.17.).

```
from itertools import product


class TruthTableGenerator:
    def __init__(self, evaluator):
        self.evaluator = evaluator

    def generate(self, expr, variables):  3 usages (3 dynamic)
        truth_table = []
        for values in product([False, True], repeat=len(variables)):
            row = dict(zip(variables, values))
            try:
                row['result'] = self.evaluator.calculate_expression(expr, row)
            except:
                raise ValueError("Невірний вираз")
            truth_table.append(row)
        return truth_table
```

Рисунок 4.17. – Реалізація відображення таблиці істинності

4.Модуль normal_form_builder.py

Модуль *normal_form_builder* відповідає за побудову канонічних форм булевих функцій на основі таблиці істинності. Зокрема, він реалізує алгоритми для формування: ДНФ, КНФ, ДДНФ, ДКНФ.

Спочатку ініціалізується порожній список *steps* в якому зберігатимуться етапи побудови форми, які потім виводяться користувачу. Операція *append()* додає новий крок у список *step*.

Метод *build_dnf* відповідає за побудову ДНФ. Викликається внутрішній метод *_build_terms()*, вибирає всі рядки з *result = 1*, формує кон'юнкції змінних та з'єднує їх через диз'юнкцію. Якщо термів немає повертається *False* (Рисунок 4.18.)

```
class NormalFormBuilder:
    def __init__(self, table_generator):
        self.table_generator = table_generator
        self.steps = []

    def _reset_steps(self):  4 usages
        self.steps = []

    def _add_step(self, description, data=None):  16 usages
        self.steps.append((description, data))

    def get_steps(self):  1 usage (1 dynamic)
        return self.steps

    def build_dnf(self, truth_table, variables):  1 usage (1 dynamic)
        self._reset_steps()
        self._add_step("Початок побудови ДНФ")
        terms = self._build_terms(truth_table, variables, target_result: True, inner_op: " ∧ ", outer_op: " ∨ ")
        result = terms if terms else "False"
        self._add_step( description: "ДНФ побудовано", result)
        return result
```

Рисунок 4.18. – Реалізація побудови ДНФ

Метод *build_cnf* відповідає за побудову КНФ. Першим кроком фільтруються всі рядки де результат булевої функції *False*. Для кожного рядка створюється диз'юнкція змінних. Після чого усі побудовані диз'юнкції з'єднуються між собою через кон'юнкцію, утворюючи КНФ. Етапи розв'язку та таблиці істинності зберігаються за допомогою *_add_step* для подальшого виводу (Рисунок 4.19.)

Метод *build_perfect_dnf* відповідає за побудову ДДНФ. Викликається допоміжний метод *_build_perfect_terms* який обирає рядки з таблиці, де *result = 1*, для кожного з них створює повну кон'юнкцію змінних та об'єднує всі мінтерми через диз'юнкцію. Потім додається фінальний крок з результатом (Рисунок 4.20.).

```python
def build_cnf(self, truth_table, variables):  1 usage (1 dynamic)
    self._reset_steps()
    self._add_step("Початок побудови КНФ")

    self._add_step( description: "Таблиця істинності", truth_table)

    false_rows = [row for row in truth_table if not row['result']]
    self._add_step( description: "Рядки з результатом False", false_rows)

    if not false_rows:
        self._add_step("Немає рядків з False, КНФ = True")
        return "True"

    terms = []
    for row in false_rows:
        term_parts = []
        for var in variables:
            if row[var]:
                term_parts.append(f"¬{var}")
            else:
                term_parts.append(var)
        term = " ∨ ".join(term_parts)
        terms.append(f"({term})")
        self._add_step(f"Додані диз'юнкції для рядка {row}: {term}")

    result = " ∧ ".join(terms)
    self._add_step( description: "Фінальна КНФ", result)
    return result
```

Рисунок 4.19. – Реалізація побудови КНФ

```python
def build_perfect_dnf(self, truth_table, variables):  1 usage (1 dynamic)
    self._reset_steps()
    self._add_step("Початок побудови ДДНФ")
    terms = self._build_perfect_terms(truth_table, variables, target_result: True, inner_op: " ∧ ", outer_op: " ∨ ")
    result = terms if terms else "False"
    self._add_step( description: "ДДНФ побудовано", result)
    return result
```

Рисунок 4.20. – Реалізація побудови ДДНФ

Метод *build_perfect_cnf* відповідає за побудову ДКНФ. Спочатку вибираються всі рядки де результат функції *False*. Для кожного хибного рядка формується повна диз'юнкція. Доданий диз'юнкт обгортається в дужки й зберігається в список *terms*. Всі диз'юнкти обєднуються через кон'юнкцію. Етапи розв'язку зберігаються за допомогою *_add_step* для подальшого виводу (Рисунок 4.21.)

```
def build_perfect_cnf(self, truth_table, variables):  1 usage (1 dynamic)
    self._reset_steps()
    self._add_step("Початок побудови ДКНФ")

    self._add_step( description: "Таблиця істинності", truth_table)

    false_rows = [row for row in truth_table if not row['result']]
    self._add_step( description: "Рядки з результатом False", false_rows)

    if not false_rows:
        self._add_step("Немає рядків з False, ДКНФ = True")
        return "True"

    terms = []
    for row in false_rows:
        term_parts = []
        for var in variables:
            term_parts.append(var if not row[var] else f"¬{var}")
        term = " ∨ ".join(term_parts)
        terms.append(f"({term})")
        self._add_step(f"Додані диз'юнкції для рядка {row}: {term}")

    result = " ∧ ".join(terms)
    self._add_step( description: "Фінальна ДКНФ", result)
    return result
```

Рисунок 4.21. – Реалізація побудови ДКНФ

Метод *_build_terms* є допоміжною функцією, яка використовується для побудови ДНФ та КНФ на основі таблиці істинності. Спочатку створюється проміжний список термів (*terms*). Далі перебираються всі рядки таблиці і відбираються лише ті, в яких *result* дорівнює заданому *target_result*. Для кожного рядка створюється терм. За допомогою функції *join()* з'єднуються частини терму,

після чого з’єднуються вже всі терми, формуючи фінальний логічний вираз (Рисунок 4.22.)

```
def _build_terms(self, truth_table, variables, target_result, inner_op, outer_op):
    terms = []
    for row in truth_table:
        if row['result'] == target_result:
            term_parts = []
            for var in variables:
                term_parts.append(var if row[var] else f"¬{var}")
            terms.append(inner_op.join(term_parts))
    return outer_op.join(terms)
```

Рисунок 4.22. – Реалізація побудови термів

Метод *_build_perfect_terms* призначений для побудови досконалих нормальних форм – ДДНФ та ДКНФ. Як і в попередньому методі, створюється список, в який будуть додаватися побудовані терми (*terms*). Потім перебираються рядки і обробляються лише ті, де *result* дорівнює заданому *target_result*. Для кожної змінної = 1 вставляється змінна в прямому вигляді. Для змінної 0 – вставляється заперечення. Отримується повний терм, що вказує всі змінні. Після чого, частини термів з’єднуються та обгортаються в дужки. Потім усі побудовані терми об’єднуються (Рисунок 4.23.).

```
def _build_perfect_terms(self, truth_table, variables, target_result, inner_op, outer_op):
    terms = []
    for row in truth_table:
        if row['result'] == target_result:
            term_parts = []
            for var in variables:
                term_parts.append(var if row[var] else f"¬{var}")
            terms.append(f"({inner_op.join(term_parts)})")
    return outer_op.join(terms)
```

Рисунок 4.23. – Реалізація побудови повних термів

5.Модуль main.py

Модуль *main* виконує роль точки входу в програму та відповідає за запуск графічного інтерфейсу користувача. Він імпортує бібліотеку tkinter та головний клас інтерфейсу BooleanCalculatorUI. Ініціалізує головне вікно та працює в режимі очікування дій користувача (*mainloop()*) (Рисунок 4.24.).

```
import tkinter as tk

from app.boolean_calculator_ui import BooleanCalculatorUI

if __name__ == "__main__":
    root = tk.Tk()
    app = BooleanCalculatorUI(root)
    root.mainloop()
```

Рисунок 4.24. – Реалізація побудови ініціатора графічного інтерфейсу

4.2. Опис програми

Розроблене програмне забезпечення призначене для вирішення завдань, пов’язаних із побудовою та аналізом булевих функцій у контексті вивчення цифрової логіки. Програма створена як навчальний інструмент, що дозволяє студентам і викладачам легко працювати з логічними виразами, будувати таблиці істинності, а також формувати різні канонічні представлення функцій. Особливістю програми є її методична орієнтованість – вона не просто виконує обчислення, а й надає покрокове пояснення кожного етапу побудови, що сприяє кращому розумінню логіки формування виразів. Програма дозволяє в інтерактивному режимі досліджувати властивості логічних функцій, що є корисним як під час самостійного вивчення, так і у процесі викладання.

Як вже було зазначено, програма реалізована на мові Python та побудована за модульним принципом. Основна структура додатка включає наступні компоненти:

- Обробник логічних виразів (*boolean_expression_evaluator.py*) – відповідає за розпізнавання змінних, підстановку значень та обчислення результатів логічних виразів.
- Генератор таблиці істинності (*truth_table_generator.py*) – створює повну таблицю значень логічної функції на основі всіх можливих комбінацій змінних.
- Побудовник канонічних форм (*normal_form_builder.py*) – реалізує алгоритми формування ДНФ, КНФ, ДДНФ, ДКНФ з урахуванням навчальної деталізації.
- Інтерфейс користувача (*boolean_calculator_ui.py*) – графічна оболонка програми, побудована за допомогою бібліотеки Tkinter. Вона забезпечує взаємодію користувача з усіма функціональними можливостями додатка.

Запуск програми здійснюється через головний модуль (*main.py*), який ініціалізує усі компоненти, об’єднує їх у єдину систему та відкриває вікно користувача. Такий підхід забезпечує розділення логіки та інтерфейсу, що спрощує розробку, тестування та супровід програми.

Інтерфейс користувача реалізований за допомогою бібліотеки Tkinter, яка входить до стандартної поставки Python і дозволяє створювати графічні інтерфейси без використання сторонніх інструментів. Зовнішній вигляд програми оформлений мінімалістично та просто, що сприяє зосередженню користувача на змісті задачі, а не на освоєнні програми.

Основні елементи інтерфейсу:

1. Поле введення логічного виразу – дозволяє вводити булеву функцію у зручному для користувача форматі (наприклад, A ∧ B ∨ ¬C) (Рисунок 4.25.)

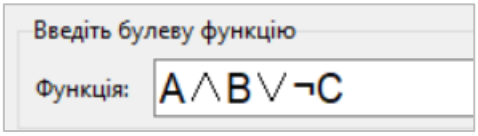

Рисунок 4.25. – Зовнішній вигляд поля введення виразу

2. Кнопки логічних операторів – забезпечують швидке вставлення операторів (∧, ∨, ¬, →, ⊕, ≡, дужки), що особливо зручно при роботі без клавіатури з символами.

Рисунок 4.26. – Зовнішній вигляд кнопок-операторів

3. Кнопки дій – для побудови таблиці істинності та кожної з чотирьох канонічних форм (ДНФ, КНФ, ДДНФ, ДКНФ).

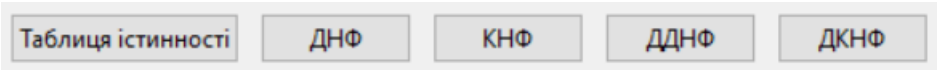

Рисунок 4.27. – Зовнішній вигляд кнопок дій

4. Поле виводу результату – багаторядкове текстове поле, де виводиться повний результат обчислення: етапи побудови, проміжні таблиці, сформовані логічні вирази.

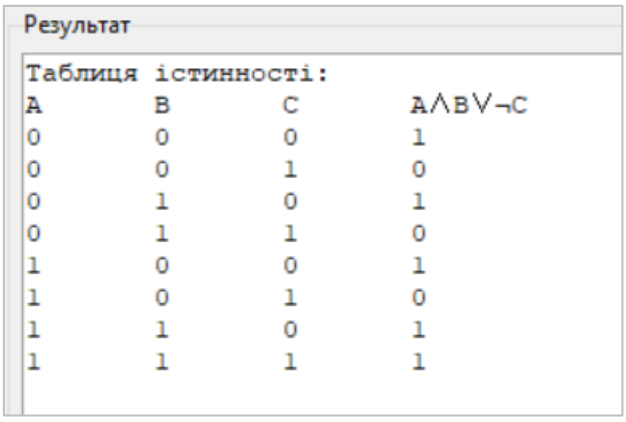

Рисунок 4.27. – Зовнішній вигляд поля виводу результату обчислень

5. Вкладка з інструкціями – допомагає новим користувачам швидко ознайомитися з можливостями програми.

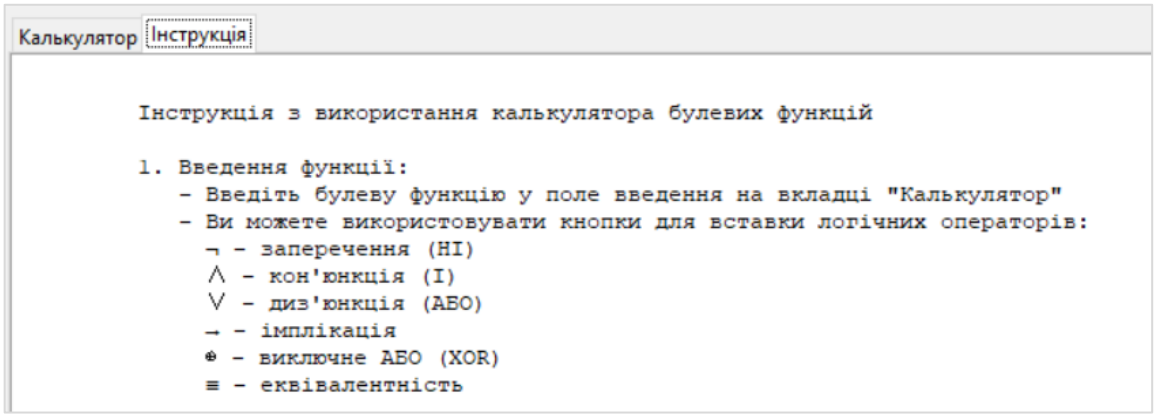

Рисунок 4.28. – Зовнішній вигляд інструкції користувача

Користувач може легко вводити вираз, натискати на потрібну операцію – і миттєво бачити результат у вигляді таблиць і логічних форм. Програма орієнтована як на початківців, так і на студентів, які вже мають базові знання в галузі дискретної математики.

Принцип роботи програми полягає у послідовному виконанні кількох логічних кроків:

1. Зчитування логічного виразу, який вводить користувач у полі вводу.
2. Аналіз виразу: визначення списку змінних, перевірка синтаксису, заміна символів на ті, що розуміє Python (наприклад, ¬ → *not*).
3. Генерація таблиці істинності: перебираються всі можливі комбінації значень змінних, обчислюється результат виразу для кожної з них.
4. Побудова канонічної форми (залежно від вибору користувача):
    - для ДНФ і ДДНФ – обробляються рядки таблиці, де результат = 1;
    - для КНФ і ДКНФ – обробляються рядки, де результат = 0;
5. Формування результату:
    - побудовані вирази виводяться у зручному форматі;
    - додатково відображаються етапи побудови;

6. Обробка помилок: у разі некоректного введення користувач отримує повідомлення з поясненням (Рисунок 4.29.).

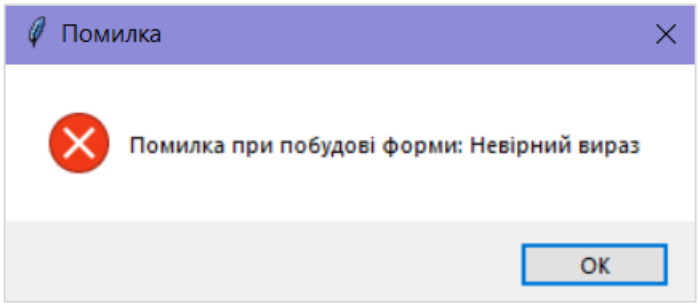

Рисунок 4.29. – Приклад помилки при некоректно введеному виразі

Програма працює в інтерактивному режимі – результат оновлюється після кожного натискання відповідної кнопки. Такий підхід дозволяє користувачеві експериментувати з різними логічними виразами та спостерігати, як змінюються канонічні форми в залежності від побудови таблиці істинності.

4.3. Перевірка валідності. Аналіз можливостей програмної реалізації

Для впевненості у правильності роботи реалізованого програмного забезпечення було проведено перевірку валідності результатів обчислення логічних виразів. Зокрема, протестовано побудову таблиці істинності та формування таких представлень булевих функцій, як ДНФ, КНФ, ДДНФ і ДКНФ.

Було виконано обчислення для типових логічних виразів, наприклад:

- $A \wedge B$;
- $A \oplus C$;
- $A \wedge (\neg B \vee C)$;
- $A \rightarrow (B \wedge \neg C) \equiv D$.

Результати програми порівнювалися з результатами, отриманими вручну, а також за допомогою онлайн-сервісів, таких як і Logic Calculator і eMathHelp. У всіх випадках результати збігались, а порядок побудови форм – відповідав теоретичним алгоритмам. Порівняємо розрахунки на прикладі наступних виразів:

1. $A \wedge B$

Переглянемо побудову таблиці істинності застосунку Logic Calculator та реалізованого програмного забезпечення (Рисунок 4.30-31.)

Таблиця істинності:

A	B	A∧B
0	0	0
0	1	0
1	0	0
1	1	1

Рисунок 4.30. – Побудови таблиці істинності за допомогою реалізованого ПЗ

A B	A ∧ B
1 1	1
1 0	0
0 1	0
0 0	0

Рисунок 4.31. – Побудови таблиці істинності за допомогою Logic Calculator

Наступним етапом йде порівняння побудови ДКНФ за допомогою Logic Calculator та розробленого програмного забезпечення (Рисунок 4.32-33.)

Фінальна ДКНФ:
(A ∨ B) ∧ (A ∨ ¬B) ∧ (¬A ∨ B)

Рисунок 4.32. – Побудови ДКНФ за допомогою реалізованого ПЗ

$$(\neg A \vee B) \wedge (A \vee \neg B) \wedge (A \vee B)$$

Рисунок 4.33. – Побудови ДКНФ за допомогою Logic Calculator

2. $A \to (B \wedge \neg C) \equiv D$.

Тепер порівняємо побудову таблиці істинності в калькуляторі eMathHelp та реалізованого програмного забезпечення (Рисунок 4.34-35.)

a	b	c	d	$(a \to (b \wedge \neg c)) \leftrightarrow d$
True	True	True	True	False
True	True	True	False	True
True	True	False	True	True
True	True	False	False	False
True	False	True	True	False
True	False	True	False	True
True	False	False	True	False
True	False	False	False	True
False	True	True	True	True
False	True	True	False	False
False	True	False	True	True
False	True	False	False	False
False	False	True	True	True
False	False	True	False	False
False	False	False	True	True
False	False	False	False	False

Рисунок 4.34. – Побудови таблиці істинності за допомогою eMathHelp

Таблиця істинності:

A	B	C	D	(A→(B∧¬C))≡D
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Рисунок 4.35. – Побудови таблиці істинності за допомогою реалізованого ПЗ

При збільшенні кількості змінних програма продовжує функціонувати, але обсяг обчислень (таблиця істинності) суттєво зростає, що є характерним для всіх логічних систем.

Подальшим кроком буде порівняння побудови нормальних форм в калькуляторі eMathHelp та реалізованому програмному забезпеченні (Рисунок 4.36-37.)

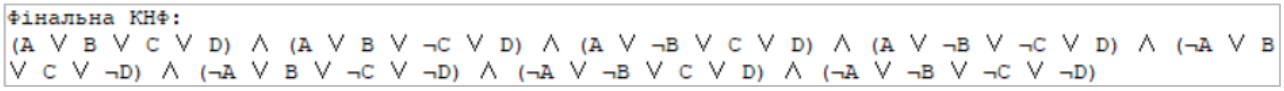

Рисунок 4.36. – Побудови КНФ за допомогою реалізованого ПЗ

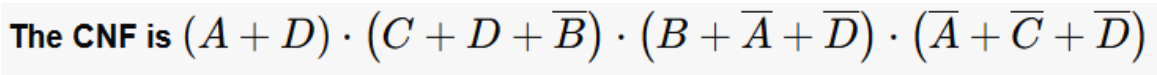

Рисунок 4.37. – Побудови КНФ за допомогою eMathHelp

Результати перевірки підтвердили коректність роботи програми у всіх протестованих випадках. Програма демонструє стабільну роботу за умови введення коректних булевих виразів. Таблиця істинності формується відповідно до логіки булевих функцій, а побудова канонічних форм (ДНФ, КНФ, ДДНФ, ДКНФ) виконується згідно з теоретичними принципами. Наявність повідомлень про помилки при некоректному введенні свідчить про достатню стійкість програми до типових помилок користувача. Це дозволяє зробити висновок, що програмне забезпечення є надійним у контексті навчального використання.

4.4. Необхідна користувачу програми інструкція

Для ефективного користування розробленим програмним забезпеченням необхідно дотримуватись певної послідовності дій та враховувати формат введення даних. Нижче подано покрокову інструкцію, яка дозволить користувачеві швидко

ознайомитися з функціональністю програми та уникнути типових помилок при роботі.

1. Інтерфейс користувача

У вікні програми відображаються такі основні елементи:

- Поле введення булевої функції – сюди користувач вводить логічний вираз;
- Кнопки логічних операторів – для швидкого вставлення символів: ¬, ∧, ∨, →, ⊕, ≡, дужок;
- Кнопки обчислень – викликають побудову:
  - Таблиці істинності;
  - ДНФ;
  - КНФ;
  - ДДНФ;
  - ДКНФ;
- Поле результату – нижче розташоване багаторядкове вікно, в якому відображаються результати обчислення та кроки побудови форм;
- Вкладка "Інструкція" – містить приклади виразів, підказки та коротку довідку з користування програмою.

2. Введення логічного виразу

Користувач повинен вводити логічний вираз у полі введення, дотримуючись наступних правил:

- Змінні позначаються великими латинськими літерами (наприклад: A, B, X);
- Підтримуються наступні оператори:
  - ¬ – логічне заперечення;
  - ∧ – кон’юнкція (логічне "І");
  - ∨ – диз’юнкція (логічне "АБО");
  - → – імплікація;
  - ⊕ – сума по модулю 2 (виключне АБО (XOR));
  - ≡ – еквівалентність;
- Дужки використовуються для групування частин виразу.

Приклад правильного введеного виразу:

Рисунок 4.38. – Приклад введення виразу

3. Виконання обчислень

Після введення виразу користувач може натиснути одну з функціональних кнопок:

- "Таблиця істинності" – відображає всі комбінації значень змінних та результат функції;
- "ДНФ" / "КНФ" – будує скорочені форми виразу;
- "ДДНФ" / "ДКНФ" – формує досконалі канонічні форми;
- У вікні результату відображається не лише фінальний вираз, а й покрокові етапи побудови, включаючи таблиці та пояснення.

4. Обробка помилок

Програма вміє розпізнавати поширені помилки, зокрема:

- Порожнє поле введення;
- Відсутність змінних у виразі;
- Некоректне використання символів або дужок.

У таких випадках з’являється повідомлення про помилку, яке допомагає користувачеві усунути проблему (Рисунок 4.39-41).

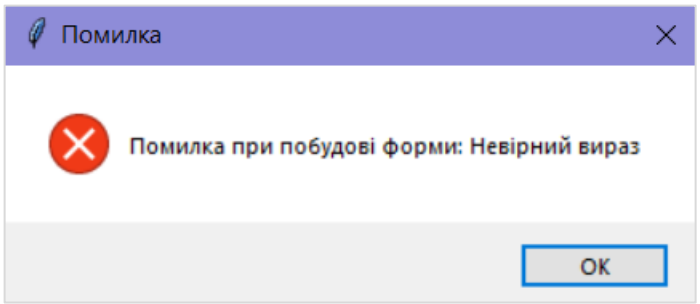

Рисунок 4.39. – Помилка при спробі побудови форм з некоректним виразом

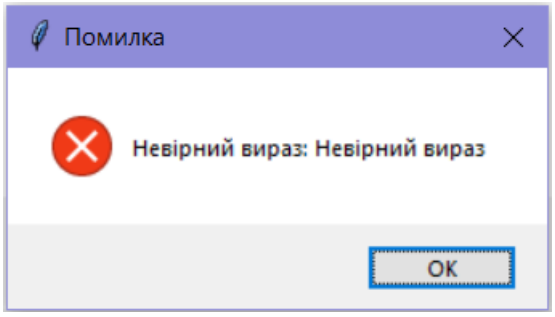

Рисунок 4.40. – Помилка при невірно вказаних змінних

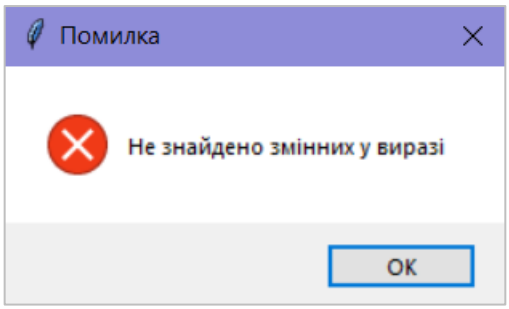

Рисунок 4.41. – Помилка при відсутності змінних у виразі

5. Рекомендації

- Перед натисканням кнопок переконайтесь, що введення завершене та синтаксично коректне;
- Вкладка "Інструкція" містить приклади правильних виразів та вимоги до введення – користуйтесь нею для ознайомлення (Рисунок 4.42.);
- Не перевищуйте 6–7 змінних у виразі – це призведе до великої таблиці істинності.

Інтерфейс програми є інтуїтивно зрозумілим, а її функціональність забезпечує зручну та ефективну роботу навіть для користувачів без поглиблених знань у програмуванні. Чітке дотримання інструкції дозволить безпомилково виконати всі обчислення та логічні побудови.

```
Інструкція з використання калькулятора булевих функцій

1. Введення функції:
   - Введіть булеву функцію у поле введення на вкладці "Калькулятор"
   - Ви можете використовувати кнопки для вставки логічних операторів:
     ¬ - заперечення (НІ)
     ∧ - кон'юнкція (І)
     ∨ - диз'юнкція (АБО)
     → - імплікація
     ⊕ - виключне АБО (XOR)
     ≡ - еквівалентність

2. Доступні операції:
   - Таблиця істинності: генерує повну таблицю істинності для функції
   - ДНФ: будує диз'юнктивну нормальну форму
   - КНФ: будує кон'юнктивну нормальну форму
   - ДДНФ: будує досконалу диз'юнктивну нормальну форму
   - ДКНФ: будує досконалу кон'юнктивну нормальну форму

3. Приклади введення:
   - A ∧ B → C
   - ¬(A ∨ B) ≡ C
   - A ⊕ B ⊕ C
   - (A → B) ∧ (B → A)

4. Вимоги до введення:
   - Використовуйте тільки латинські літери для змінних
   - Дотримуйтесь правильних дужок
   - Оператори повинні бути між змінними/виразами

5. Інтерпретація результатів:
   - У таблиці істинності 1 = True, 0 = False
   - Нормальні форми показуються з детальним поясненням кроків побудови
```

Рисунок 4.42. – Вкладка «Інструкція»

ВИСНОВКИ

В ході виконання кваліфікаційної роботи виконано головну задачу, а саме розроблено навчальне програмне забезпечення для побудови канонічних форм булевих функцій.

При виконанні було виконанні основні завдання:

1. Зроблено огляді та проаналізовано наявні роботи з аналогічним завданням та реалізацією. Виокремлено переваги та труднощі у використані в оглянутих програмних продуктах.
2. Розглянуто теоретичний матеріал з теми «Канонічні форми булевих функцій». Виокремлено основний матеріал.
3. Розроблено та описано структуру навчального калькулятора.
4. Розроблено алгоритм роботи програмного забезпечення з теми «Канонічні форми булевих функцій» відповідно до розробленої структури.
5. Розроблено блок-схему, яка відповідає алгоритму роботи програми.
6. Обґрунтовано вибір мови програмування, яка використовується для програмної реалізації
7. Описано процеси розробки програмного забезпечення.
8. Описано безпосередня робота навчального додатку.

У розробленому навчальному програмному засобі реалізовано п’ять основних функціональних блоків. Кожен із них орієнтований на опрацювання логічних виразів та побудову канонічних форм булевих функцій. Користувач має змогу ввести булеву функцію, після чого програма дозволяє автоматично згенерувати таблицю істинності, побудувати ДНФ, КНФ, а також досконалі форми — ДДНФ і ДКНФ.

Програма поєднує теоретичний та практичний аспекти, оскільки не лише формує, а й надає покрокове пояснення побудови кожної форми. Це дозволяє користувачеві простежити хід міркувань, навчитися формулювати логічні вирази правильно та зрозуміти принципи побудови канонічних форм.

Робота пройшла апробацію на міжнародній науковій студентській конференції «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті», яка пройшла 10 квітня 2025 року.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Черненко О. О. Методичні рекомендації щодо оформлення пояснювальних записок до курсового проекту для студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» ступеня бакалавра, магістра. Полтава, 2024. С 61.
2. Ємець О. О., Парфьонова Т. О. Дискретна математика. Навчальний посібник. Полтава, 2023. С. 284.
3. Tony R. Kuphaldt Булева алгебра. LibreTexts. 2022. URL: https://ukrayinska.libretexts.org/Робоча_сила/Технологія_електроніки/Книга%3A_Електричні_схеми_IV_-_цифрова_схемотехніка_(Kuphaldt)/07%3A_Булева_алгебра
4. Доля П. Г. Дискретна математика. Конспект лекцій. Харківський національний університет імені В. Н. Каразіна 2015. С. 190.
5. Марк Лутц Програмування на Python 4-е видання. PRINT2PRINT. 2018
6. Підручник з Python. Python Software Foundation. 2025. URL: https://docs.python.org/uk/3.13/tutorial/index.html
7. Путівник мовою програмування Python. О. Мізюк. 2024. URL: https://pythonguide.rozh2sch.org.ua
8. Tkinter – Python interface to Tcl/Tk. 2025 https://docs.python.org/3/library/tkinter.html
9. Дичка І. А., Легеза В. П., Онай М. В. Комп'ютерна логіка. Прикладна теорія цифрових автоматів: Комп'ютерний практикум. навч. посіб. для студ. спеціальності 121 «Інженеріяпрограмного забезпечення». Київ. : КПІ ім. Ігоря Сікорського, 2018. – 88с.
10. Уроки Python. 2025. URL: https://acode.com.ua/lessons-python/
11. Python статті. Spacelab. 2025. URL: https://spacelab.ua/materials/Python/
12. Безущак О., Ганюшкін О. Математична логіка : навч. посіб. – К. :ВПЦ «Київський університет». – 2023. – 143 с.
13. A Byte of Python. Swaroop Chitlur. 2023
14. Програмування на Пітон. Школа програмування. 2023.

15. Довідники по Python. Онлайн школа itProger. URL: https://itproger.com/ua/spravka/python

16. Шепетяк О. Логіка. Підручник для студентів вищих навчальних закладів. – Київ: Фенікс, 2015. – 256 с

ДОДАТОК А
КОД ПРОГРАМИ

boolean_calculator_ui.py

```
from app.boolean_expression_evaluator import BooleanExpressionEvaluator
from app.normal_form_builder import NormalFormBuilder
from app.truth_table_generator import TruthTableGenerator
from tkinter import ttk, messagebox
import tkinter as tk


class BooleanCalculatorUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Калькулятор булевих функцій")
        self.root.geometry("550x400")

        self.evaluator = BooleanExpressionEvaluator()
        self.table_generator = TruthTableGenerator(self.evaluator)
        self.normal_form_builder = NormalFormBuilder(self.table_generator)

        self.variables = set()
        self.function_entry = None
        self.output_text = None
        self.notebook = None

        self._setup_welcome_screen()

    def _setup_welcome_screen(self):
        self.welcome_frame = ttk.Frame(self.root)
        self.welcome_frame.pack(fill='both', expand=True)

        welcome_label = ttk.Label(
            self.welcome_frame,
            text="""
            Кваліфікаційна робота з теми:
```

```
            "Реалізація навчального програмного забезпечення
            для побудови канонічних форм булевих функцій"

            Виконав студент групи КН б-41
            Волторніст Артем

            ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
            """,
            font=('Helvetica', 12, 'bold')
        )
        welcome_label.pack(pady=50)

        continue_btn = ttk.Button(
            self.welcome_frame,
            text="Продовжити",
            command=self._setup_main_interface
        )
        continue_btn.pack(pady=20)

    def _setup_main_interface(self):
        self.welcome_frame.destroy()

        self.root.geometry("800x600")

        self.notebook = ttk.Notebook(self.root)
        self.notebook.pack(fill='both', expand=True, padx=10, pady=10)

        self.calculator_tab = ttk.Frame(self.notebook)
        self.notebook.add(self.calculator_tab, text='Калькулятор')
        self._setup_calculator_tab()

        self.instructions_tab = ttk.Frame(self.notebook)
        self.notebook.add(self.instructions_tab, text='Інструкція')
        self._setup_instructions_tab()
```

```
def _setup_calculator_tab(self):
    self._create_input_frame()
    self._create_buttons_frame()
    self._create_compute_buttons_frame()
    self._create_output_frame()

def _setup_instructions_tab(self):
    instructions_text = tk.Text(self.instructions_tab, wrap="word", padx=10, pady=10)
    instructions_text.pack(fill="both", expand=True)

    instructions = """
    Інструкція з використання калькулятора булевих функцій

    1. Введення функції:
      - Введіть булеву функцію у поле введення на вкладці "Калькулятор"
      - Ви можете використовувати кнопки для вставки логічних операторів:
        ¬ - заперечення (НІ)
        ∧ - кон'юнкція (І)
        ∨ - диз'юнкція (АБО)
        → - імплікація
        ⊕ - виключне АБО (XOR)
        ≡ - еквівалентність

    2. Доступні операції:
      - Таблиця істинності: генерує повну таблицю істинності для функції
      - ДНФ: будує диз'юнктивну нормальну форму
      - КНФ: будує кон'юнктивну нормальну форму
      - ДДНФ: будує досконалу диз'юнктивну нормальну форму
      - ДКНФ: будує досконалу кон'юнктивну нормальну форму

    3. Приклади введення:
      - A ∧ B → C
      - ¬(A ∨ B) ≡ C
      - A ⊕ B ⊕ C
      - (A → B) ∧ (B → A)
```

```
    4. Вимоги до введення:
       - Використовуйте тільки латинські літери для змінних
       - Дотримуйтесь правильних дужок
       - Оператори повинні бути між змінними/виразами

    5. Інтерпретація результатів:
       - У таблиці істинності 1 = True, 0 = False
       - Нормальні форми показуються з детальним поясненням кроків побудови
    """

    instructions_text.insert("1.0", instructions)
    instructions_text.configure(state="disabled")

def _create_input_frame(self):
    input_frame = ttk.LabelFrame(self.calculator_tab, text="Введіть булеву функцію")
    input_frame.pack(pady=10, padx=10, fill="x")

    ttk.Label(input_frame, text="Функція:").grid(row=0, column=0, padx=5, pady=5)

    self.function_entry = ttk.Entry(
        input_frame,
        width=80,
        font=('TkDefaultFont', 15)
    )
    self.function_entry.grid(
        row=0, column=1,
        padx=5, pady=5,
        columnspan=2,
        sticky="ew"
    )

    input_frame.columnconfigure(1, weight=1)

def _create_buttons_frame(self):
    buttons_frame = ttk.Frame(self.calculator_tab)
    buttons_frame.pack(pady=10)
```

```
    operators = ["¬", "∧", "∨", "→", "⊕", "≡", "(", ")"]
    for i, op in enumerate(operators):
        ttk.Button(buttons_frame, text=op,
                   command=lambda o=op: self._insert_operator(o)) \
            .grid(row=0, column=i, padx=5)

def _create_compute_buttons_frame(self):
    compute_frame = ttk.Frame(self.calculator_tab)
    compute_frame.pack(pady=10)

    buttons = [
        ("Таблиця істинності", self._show_truth_table),
        ("ДНФ", lambda: self._build_normal_form("ДНФ")),
        ("КНФ", lambda: self._build_normal_form("КНФ")),
        ("ДДНФ", lambda: self._build_normal_form("ДДНФ")),
        ("ДКНФ", lambda: self._build_normal_form("ДКНФ"))
    ]

    for i, (text, command) in enumerate(buttons):
        ttk.Button(compute_frame, text=text, command=command) \
            .grid(row=0, column=i, padx=5)

def _create_output_frame(self):
    output_frame = ttk.LabelFrame(self.calculator_tab, text="Результат")
    output_frame.pack(pady=10, padx=10, fill="both", expand=True)

    self.output_text = tk.Text(output_frame, wrap="word", height=15)
    self.output_text.pack(pady=5, padx=5, fill="both", expand=True)

    scrollbar = ttk.Scrollbar(output_frame, orient="vertical",
                              command=self.output_text.yview)
    scrollbar.pack(side="right", fill="y")
    self.output_text.configure(yscrollcommand=scrollbar.set)

def _insert_operator(self, operator):
```

```
        self.function_entry.insert(tk.END, operator)

    def _get_variables(self):
        expr = self.function_entry.get()

        if not expr:
            messagebox.showerror("Помилка", "Введіть булеву функцію")
            return None

        variables = self.evaluator.extract_variables(expr)

        if not variables:
            messagebox.showerror("Помилка", "Не знайдено змінних у виразі")
            return None

        self.variables = set(variables)
        return variables

    def _evaluate_function(self):
        variables = self._get_variables()
        if not variables:
            return

        expr = self.function_entry.get()
        try:
            truth_table = self.table_generator.generate(expr, variables)
            result = self.evaluator.calculate_expression(expr, {v: True for v in variables})

            self.output_text.delete(1.0, tk.END)
            self.output_text.insert(tk.END, f"Результат обчислення: {result}\n\n")
            self._display_truth_table(truth_table, variables, expr)

        except Exception as e:
            messagebox.showerror("Помилка", f"Невірний вираз: {str(e)}")

    def _show_truth_table(self):
```

```
    variables = self._get_variables()
    if not variables:
        return

    expr = self.function_entry.get()
    try:
        truth_table = self.table_generator.generate(expr, variables)
        self.output_text.delete(1.0, tk.END)
        self._display_truth_table(truth_table, variables, expr)

    except Exception as e:
        messagebox.showerror("Помилка", f"Невірний вираз: {str(e)}")

def _display_truth_table(self, truth_table, variables, expr):
    self.output_text.insert(tk.END, "Таблиця істинності:\n")
    headers = variables + [expr]
    self.output_text.insert(tk.END, "\t".join(headers) + "\n")

    for row in truth_table:
        values = [str(int(row[v])) for v in variables] + [str(int(row['result']))]
        self.output_text.insert(tk.END, "\t".join(values) + "\n")

def _build_normal_form(self, form_type):
    variables = self._get_variables()
    if not variables:
        return

    expr = self.function_entry.get()
    try:
        truth_table = self.table_generator.generate(expr, variables)

        form_builders = {
            "ДНФ": self.normal_form_builder.build_dnf,
            "КНФ": self.normal_form_builder.build_cnf,
            "ДДНФ": self.normal_form_builder.build_perfect_dnf,
            "ДКНФ": self.normal_form_builder.build_perfect_cnf
```

```
        }

        normal_form = form_builders[form_type](truth_table, variables)
        steps = self.normal_form_builder.get_steps()

        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, f"{form_type} для функції {expr}:\n\n")

        self.output_text.insert(tk.END, "Етапи побудови:\n")
        for i, (desc, data) in enumerate(steps, 1):
            self.output_text.insert(tk.END, f"{i}. {desc}\n")
            if data:
                if isinstance(data, list):
                    headers = list(data[0].keys()) if data else []
                    self.output_text.insert(tk.END, "\t" + "\t".join(headers) + "\n")
                    for row in data:
                        values = [str(int(row.get(k, ''))) for k in headers]
                        self.output_text.insert(tk.END, "\t" + "\t".join(values) + "\n")
                else:
                    self.output_text.insert(tk.END, f"\t{data}\n")
            self.output_text.insert(tk.END, "\n")

        self.output_text.insert(tk.END, f"\nФінальна {form_type}:\n{normal_form}")

    except Exception as e:
        messagebox.showerror("Помилка", f"Помилка при побудові форми: {str(e)}")
```

boolean_expression_evaluator.py

```
class BooleanExpressionEvaluator:
    @staticmethod
    def calculate_expression(expr, values):
        replaced_expr = expr
        for var, val in values.items():
            replaced_expr = replaced_expr.replace(var, str(val))

        replaced_expr = replaced_expr.replace("¬", " not ")
```

```
        replaced_expr = replaced_expr.replace("∧", " and ")
        replaced_expr = replaced_expr.replace("∨", " or ")
        replaced_expr = replaced_expr.replace("→", " <= ")
        replaced_expr = replaced_expr.replace("⊕", " != ")
        replaced_expr = replaced_expr.replace("≡", " == ")

        try:
            return bool(eval(replaced_expr))
        except:
            raise ValueError("Невірний вираз")

    @staticmethod
    def extract_variables(expr):
        variables = set()
        for char in expr:
            if char.isalpha() and char not in ['∧', '∨', '¬', '→', '⊕', '≡']:
                variables.add(char)
        return sorted(variables)
```

normal_form_builder.py

```
class NormalFormBuilder:
    def __init__(self, table_generator):
        self.table_generator = table_generator
        self.steps = []

    def _reset_steps(self):
        self.steps = []

    def _add_step(self, description, data=None):
        self.steps.append((description, data))

    def get_steps(self):
        return self.steps

    def build_dnf(self, truth_table, variables):
        self._reset_steps()
```

```
    self._add_step("Початок побудови ДНФ")
    terms = self._build_terms(truth_table, variables, True, " ∧ ", " ∨ ")
    result = terms if terms else "False"
    self._add_step("ДНФ побудовано", result)
    return result

def build_cnf(self, truth_table, variables):
    self._reset_steps()
    self._add_step("Початок побудови КНФ")

    self._add_step("Таблиця істинності", truth_table)

    false_rows = [row for row in truth_table if not row['result']]
    self._add_step("Рядки з результатом False", false_rows)

    if not false_rows:
        self._add_step("Немає рядків з False, КНФ = True")
        return "True"

    terms = []
    for row in false_rows:
        term_parts = []
        for var in variables:
            if row[var]:
                term_parts.append(f"¬{var}")
            else:
                term_parts.append(var)
        term = " ∨ ".join(term_parts)
        terms.append(f"({term})")
        self._add_step(f"Додані диз'юнкції для рядка {row}: {term}")

    result = " ∧ ".join(terms)
    self._add_step("Фінальна КНФ", result)
    return result

def build_perfect_dnf(self, truth_table, variables):
```

```
    self._reset_steps()
    self._add_step("Початок побудови ДДНФ")
    terms = self._build_perfect_terms(truth_table, variables, True, " ∧ ", " ∨ ")
    result = terms if terms else "False"
    self._add_step("ДДНФ побудовано", result)
    return result

def build_perfect_cnf(self, truth_table, variables):
    self._reset_steps()
    self._add_step("Початок побудови ДКНФ")

    self._add_step("Таблиця істинності", truth_table)

    false_rows = [row for row in truth_table if not row['result']]
    self._add_step("Рядки з результатом False", false_rows)

    if not false_rows:
        self._add_step("Немає рядків з False, ДКНФ = True")
        return "True"

    terms = []
    for row in false_rows:
        term_parts = []
        for var in variables:
            term_parts.append(var if not row[var] else f"¬{var}")
        term = " ∨ ".join(term_parts)
        terms.append(f"({term})")
        self._add_step(f"Додані диз'юнкції для рядка {row}: {term}")

    result = " ∧ ".join(terms)
    self._add_step("Фінальна ДКНФ", result)
    return result

def _build_terms(self, truth_table, variables, target_result, inner_op, outer_op):
    terms = []
    for row in truth_table:
```

```
            if row['result'] == target_result:
                term_parts = []
                for var in variables:
                    term_parts.append(var if row[var] else f"¬{var}")
                terms.append(inner_op.join(term_parts))
        return outer_op.join(terms)

    def _build_perfect_terms(self, truth_table, variables, target_result, inner_op, outer_op):
        terms = []
        for row in truth_table:
            if row['result'] == target_result:
                term_parts = []
                for var in variables:
                    term_parts.append(var if row[var] else f"¬{var}")
                terms.append(f"({inner_op.join(term_parts)})")
        return outer_op.join(terms)
```

truth_table_generator.py

```
from itertools import product


class TruthTableGenerator:
    def __init__(self, evaluator):
        self.evaluator = evaluator

    def generate(self, expr, variables):
        truth_table = []
        for values in product([False, True], repeat=len(variables)):
            row = dict(zip(variables, values))
            try:
                row['result'] = self.evaluator.calculate_expression(expr, row)
            except:
                raise ValueError("Невірний вираз")
            truth_table.append(row)
        return truth_table
```

main.py

```
import tkinter as tk

from app.boolean_calculator_ui import BooleanCalculatorUI

if __name__ == "__main__":
    root = tk.Tk()
    app = BooleanCalculatorUI(root)
    root.mainloop()
```