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

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВІРТУАЛЬНОГО СИМУЛЯТОРА З ТЕМИ "ГРАФІЧНІ МЕТОДИ МІНІМІЗАЦІЇ БУЛЕВИХ ФУНКЦІЙ" НАВЧАЛЬНОГО КУРСУ "ДИСКРЕТНА МАТЕМАТИКА"

зі спеціальності 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

ступеня бакалавр

Виконавець роботи Шмельов Віталій Вячеславович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.ф.-м.н. Парфьонова Т. О.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 57 с., 18 рис., 1 таблиця, 1 додаток, 10 джерел.

BULOVA АЛГЕБРА, КАРТА КАРНО, МІНІМІЗАЦІЯ, КВАЙН-МАК-КЛАСКІ, БЛЕЙК-ПОРЕЦЬКИЙ, ДИСКРЕТНА МАТЕМАТИКА, REACT, TYPESCRIPT

Об'єктом розробки є програмне забезпечення — віртуальний симулятор для вивчення графічних методів мінімізації булевих функцій.

Предметом розробки є реалізація інтерактивного інтерфейсу, алгоритмів побудови карт Карно та методів мінімізації логічних виразів у контексті навчального курсу «Дискретна математика».

Метою роботи є створення повноцінного вебзастосунку, що дозволяє студентам візуально вивчати та практикувати мінімізацію булевих функцій з використанням діаграм Карно, методу Квайна–Мак-Класкі та методу Блейка–Порецького.

У процесі розробки реалізовано такі можливості:

- створення таблиці істинності з будь-якою кількістю змінних (2–4);
- автоматичне побудова карти Карно з розміщенням значень за кодом Грея;
- візуалізація процесу мінімізації з поясненням кожного кроку;
- інтерактивна система навчальних завдань з автоперевіркою;
- підтримка трьох методів мінімізації: Карно, Квайна–Мак-Класкі, Блейка–Порецького;
- покроковий туторіал для нових користувачів;
- можливість експорту та копіювання результатів;
- адаптивний інтерфейс з сучасним дизайном (TailwindCSS + Framer Motion).

Архітектура проєкту побудована на сучасному фронтенд-стеку: використано React + TypeScript для реалізації логіки, TailwindCSS для стилізації, зберігання стану — через хуки та контекст. Візуалізація карт Карно та підсвічування груп мінтермів реалізована з використанням CSS Grid та SVG-елементів.

Результатом роботи став стабільний, функціональний вебдодаток, що дозволяє студентам ефективно засвоювати методи мінімізації булевих функцій у

зручному графічному середовищі. Додаток протестовано на різних типах пристроїв і забезпечує зручну навігацію для користувача.

Практична цінність роботи полягає у створенні інструменту, який може бути інтегрований у дистанційні курси, лабораторні роботи та самостійну підготовку з дисципліни «Дискретна математика». Програма дозволяє поєднувати візуалізацію, теорію та практику в одному інтерфейсі.

Розроблене рішення має перспективу масштабування та подальшого розширення — зокрема, для підтримки більшої кількості змінних, створення схем логічних елементів, інтеграції з навчальними платформами та збереженням прогресу користувачів.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕННЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
Карта Карно	Таблиця для візуального спрощення булевих функцій
Мінтерм	Набір значень змінних, для якого булева функція дорівнює 1
ДНФ	Диз’юнктивна нормальна форма
КНФ	Кон’юнктивна нормальна форма
Don't care (байдужі умови)	Значення, які не впливають на результат функції і можуть бути довільно вибрані
Код Грея	Система двійкового кодування, у якій сусідні значення відрізняються одним бітом
Алгоритм Квайна–Мак-Класкі	Метод табличної мінімізації булевих функцій
Метод Блейка–Порецького	Алгебраїчний метод спрощення логічних виразів
Симулятор	Програмне забезпечення для моделювання логічних перетворень
React	JavaScript-бібліотека для створення інтерфейсу користувача
TypeScript	Надбудова над JavaScript з підтримкою типізації
TailwindCSS	CSS-фреймворк для швидкої розробки адаптивного інтерфейсу
Framer Motion	Бібліотека для створення анімацій у React
Карта Карно	Таблиця для візуального спрощення булевих функцій

ВСТУП

У сучасну цифрову епоху стрімко зростає потреба у доступному, інтуїтивно зрозумілому та візуально орієнтованому програмному забезпеченні для підтримки вивчення складних математичних дисциплін. Однією з таких дисциплін є дискретна математика, зокрема її ключовий розділ — булева алгебра та мінімізація логічних функцій, що становить основу цифрової схемотехніки, логічного програмування та комп'ютерних технологій загалом.

Викладання теми мінімізації булевих функцій традиційно базується на алгебраїчних перетвореннях, які є досить абстрактними для студентів молодших курсів. Саме тому надзвичайно важливо створити візуальні та інтерактивні засоби, що дозволяють побачити процес мінімізації "в дії" — із покроковими поясненнями, підсвічуванням мінтермів, побудовою карт Карно, тощо. Особливої актуальності ця задача набуває у контексті дистанційного навчання, коли студенти навчаються самостійно, без постійного супроводу викладача.

Одним із найбільш ефективних рішень у таких умовах є веб-симулятори, які дозволяють відтворити динамічний процес спрощення булевих функцій, при цьому залишаючись доступними у будь-якому браузері, з будь-якого пристрою.

Актуальність теми обумовлена потребою у створенні віртуального навчального інструменту, який дозволяє поєднувати теоретичні знання з практичними навичками через інтерактивне середовище, що візуалізує процеси мінімізації булевих функцій графічними методами (особливо з використанням карт Карно).

Мета роботи — алгоритмізація та створення програмного забезпечення у вигляді віртуального симулятора, що реалізує процеси графічної мінімізації булевих функцій з акцентом на навчальні потреби студентів у курсі «Дискретна математика».

Завдання роботи:

- Проаналізувати навчальні та функціональні вимоги до системи;
- Ознайомитися з графічними методами мінімізації булевих функцій (карта Карно, код Грея, метод Квайна-Мак-Класкі);

- Алгоритмізувати процеси побудови таблиці істинності, карти Карно та групування мінтермів;
- Реалізувати інтерактивний веб-застосунок з використанням сучасних технологій (React, TypeScript, Tailwind CSS);
- Здійснити візуалізацію кроків мінімізації та формування логічного виразу;
- Додати пояснення до кожного етапу обробки функції;
- Реалізувати режим практичних завдань і навчальний режим;
- Розробити інструкцію користувача для ефективного використання симулятора.

Об’єкт дослідження — процес графічної мінімізації булевих функцій у навчальному процесі з дискретної математики. Предмет дослідження — методи алгоритмізації та реалізації програмного забезпечення візуального симулятора з теми мінімізації логічних функцій.

Практичне значення роботи полягає у створенні повноцінного навчального інструменту, який може бути використаний у закладах вищої освіти для самостійної чи аудиторної роботи студентів при вивченні логіки, цифрових схем, основ штучного інтелекту та комп’ютерної інженерії. Застосунок забезпечує гнучкий, інтерактивний і доступний підхід до засвоєння складного матеріалу за допомогою графічних методів.

ПОСТАНОВКА ЗАДАЧІ

Вивчення логічних основ обчислювальних процесів є важливим елементом підготовки фахівців у галузі комп’ютерних наук. Однією з ключових тем у курсі «Дискретна математика» є **булеві функції** та їх **мінімізація**. Оптимізація логічних виразів дозволяє зменшити складність цифрових схем, підвищити ефективність програмних і апаратних засобів, а також формує в студентів алгоритмічне мислення.

Проте традиційне вивчення цієї теми часто обмежується абстрактною теорією, без наочності та можливості самостійно моделювати та спрощувати логічні функції. Це ускладнює процес засвоєння матеріалу, особливо в умовах дистанційного або змішаного навчання.

Основною метою даного проєкту є **створення інтерактивного віртуального симулятора**, який дозволяє візуалізувати процес мінімізації булевих функцій з використанням **графічних методів** — зокрема карт Карно — у контексті навчального курсу «Дискретна математика».

Для досягнення цієї мети необхідно реалізувати наступні завдання:

1. **Проаналізувати навчальні потреби** студентів при вивченні булевих функцій та методів їх мінімізації;
2. **Вивчити та формалізувати** графічні методи мінімізації логічних функцій (карти Карно, код Грея, метод Квайна–Мак–Класкі);
3. **Розробити алгоритми** побудови таблиці істинності, генерації карт Карно та виділення груп мінтермів;
4. **Створити програмне забезпечення** у вигляді вебзастосунку з використанням сучасних технологій фронтенду (React, TypeScript, Tailwind CSS);
5. **Забезпечити інтерактивність** інтерфейсу: підсвічування клітинок, пояснення кроків мінімізації, підсумкове формування логічного виразу;
6. **Реалізувати навчальний режим**, що містить теоретичні пояснення та практичні завдання;
7. **Розробити інструкцію користувача** та протестувати зручність використання

застосунку для студентів;

8. **Перевірити ефективність системи** через приклади використання та оцінку якості зворотного зв’язку.

Виконання цих завдань дозволить створити навчальний інструмент нового покоління, який не лише полегшить засвоєння складного теоретичного матеріалу, але й сприятиме розвитку навичок аналітичного мислення, самостійної роботи та алгоритмізації в студентів.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Сутність та цілі дистанційного навчання

Дистанційне навчання є однією з ключових форм сучасної освіти, що ґрунтується на використанні інформаційно-комунікаційних технологій для організації навчального процесу за відсутності безпосереднього фізичного контакту між викладачем і студентом. Такий формат навчання виник як відповідь на потребу суспільства у доступній, гнучкій, масштабованій та ефективній освітній моделі, яка дозволяє студентам здобувати знання незалежно від місця їх перебування, часу доби чи життєвих обставин.[1]

У XXI столітті розвиток дистанційного навчання стрімко прискорився завдяки поширенню інтернету, появі інтерактивних вебтехнологій, мобільних пристроїв та зростанню попиту на безперервну освіту. Цей формат став особливо актуальним в умовах пандемії COVID-19, яка змусила освітні установи всього світу переходити на віддалений режим роботи. Проте навіть після стабілізації ситуації дистанційне навчання не втратило своєї актуальності — навпаки, воно трансформувалося у важливу складову змішаного навчання, відкривши нові можливості для індивідуалізації освітнього процесу.

Сутність дистанційного навчання полягає в тому, що студент стає більш самостійним учасником освітнього процесу, а роль викладача трансформується з носія знань у наставника, фасилітатора, що координує процес пізнання. Важливими ознаками дистанційного навчання є гнучкість, інтерактивність, адаптивність до потреб і рівня знань студента. За допомогою спеціального програмного забезпечення, навчальних платформ, мультимедійних засобів та симуляторів студенти можуть не лише читати теоретичні матеріали, а й взаємодіяти з ними — будувати, моделювати, аналізувати процеси, що суттєво підвищує ефективність засвоєння знань.

Особливої ваги дистанційне навчання набуває у викладанні точних дисциплін, зокрема — дискретної математики. Цей курс традиційно вважається складним для сприйняття через високий рівень абстракції, значну кількість

логічних операцій і формалізм подання. Саме тому надзвичайно важливо впроваджувати у дистанційне навчання засоби візуалізації, інтерактивні модулі та програмні симулятори, які здатні зробити абстрактні поняття більш наочними та зрозумілими.

Таким чином, створення інтерактивного застосунку для вивчення теми «Графічні методи мінімізації булевих функцій» у дистанційному форматі повністю відповідає сучасним тенденціям у сфері освіти. Такий інструмент не тільки забезпечує доступність знань, але й сприяє розвитку алгоритмічного мислення, навичок логічного аналізу, що є невід'ємною частиною підготовки фахівців з комп'ютерних наук.

1.2. Булеві функції та їх мінімізація

Булеві функції — це математичні функції, що працюють з двійковими значеннями, зазвичай представленими як 0 і 1, або логічно — як «хибність» і «істинність». Вони є основою побудови цифрових логічних схем, що застосовуються у комп'ютерних технологіях, електроніці, телекомунікаціях та багатьох інших галузях. Формально булева функція від n змінних — це відображення $f:\{0,1\}n\rightarrow\{0,1\}$, яке призначає кожному набору вхідних значень певне логічне значення.

У практичному сенсі булеві функції дозволяють описувати поведінку логічних елементів, цифрових пристроїв, умов у програмному забезпеченні. Наприклад, за допомогою булевих функцій можна змоделювати умову доступу, реакцію на подію або логіку перемикання схеми. [2]

Мінімізація булевих функцій — це процес спрощення логічного виразу без зміни його функціональності. Основною метою мінімізації є зменшення кількості логічних елементів, необхідних для реалізації функції, що має велике значення в контексті цифрової схемотехніки. Простіші схеми займають менше місця на мікросхемах, споживають менше енергії та працюють швидше. Крім того, для навчального процесу мінімізація функцій є ключовою вправою, що допомагає

студенту глибше зрозуміти структуру логічних виразів, знайти в них закономірності та опанувати основи логічного мислення.

Існує кілька методів мінімізації булевих функцій. Найпростішими є методи ручного перетворення виразів за допомогою аксіом булевої алгебри. Однак у разі складніших функцій, які залежать від великої кількості змінних, такі методи стають неефективними. Тому були розроблені спеціальні алгоритми, які дозволяють формалізовано і систематично зменшувати функції.

До найбільш відомих і широко використовуваних методів належать графічні — як-от карти Карно (діаграми Карно) — та табличні, зокрема алгоритм Квайна-Мак-Класкі. Графічні методи ефективні для функцій з невеликою кількістю змінних (до чотирьох), оскільки дозволяють наочно об’єднувати групи мінтермів і отримувати скорочений вираз. Алгоритмічні методи, навпаки, дозволяють автоматизувати процес і працювати з більшими функціями, хоча можуть втрачати в наочності.

Особливої актуальності мінімізація набуває в освітньому контексті. Вивчення цього процесу допомагає не лише опанувати формальні методи, але й розвиває навички аналізу, логіки, структурного мислення. Тому включення задач з мінімізації булевих функцій до курсу "Дискретна математика" є цілком обґрунтованим та необхідним.

У рамках цієї роботи саме процес мінімізації було обрано як центральний об’єкт моделювання в навчальному застосунку. Реалізація цього процесу у вигляді віртуального симулятора дозволяє поєднати алгоритмічний підхід, теоретичну базу та візуальний компонент, що значно підвищує ефективність навчання

1.3. Графічні методи мінімізації (карти Карно, діаграми Вейча)

Графічні методи мінімізації булевих функцій дозволяють здійснити візуальне спрощення логічних виразів, що є особливо важливим у навчальному процесі. Найбільш поширеними інструментами такого підходу є **діаграми Карно** (або карти Карно) та **карти Вейча**.

Карта Вейча є одним із перших способів подання булевої функції у вигляді візуальної структури. Вона базується на використанні прямокутної таблиці, де кожна область відповідає певному поєднанню значень змінних. У центрі карти знаходяться області, що належать одночасно кільком множинам, що дозволяє легко виявити мінтерми, які можна об'єднати. Наприклад, на рис. 1.1 представлено класичну карту Вейча для чотирьох змінних A, B, C, D. (див. рис. 1.1)

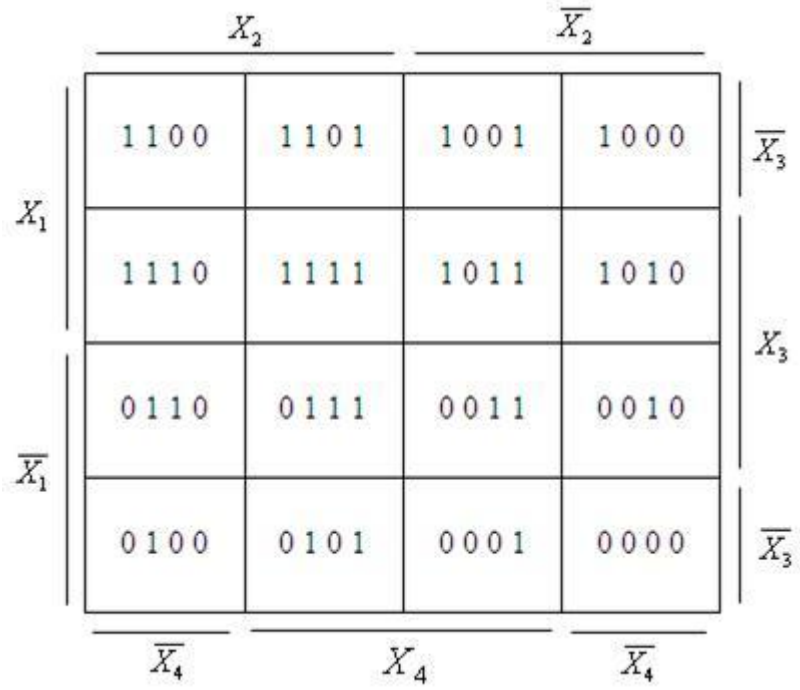

Рисунок 1.1 – Діграма Вейча для чотирьох змінних

Попри свою наочність, карта Вейча має певні обмеження: вона менш зручна для аналізу суміжності між комірками, оскільки не використовує код Грея — метод кодування, при якому два сусідні значення відрізняються лише одним бітом. Саме тому більш зручною та поширеною у практиці вважається **карта Карно**. [3]

Карта Карно, на відміну від Вейча, будується на основі коду Грея та дозволяє впорядкувати клітинки так, щоб сусідні мінтерми мали найменшу можливу різницю. Це спрощує процес виявлення груп суміжних одиниць (мінтермів), які можуть бути об'єднані для спрощення логічного виразу. На рис. 1.2 наведено приклад класичної карти Карно для чотирьох змінних X1,X2,X3,X4. Комірки заповнені двійковими кодами, що відповідають кожному мінтерму

функції.

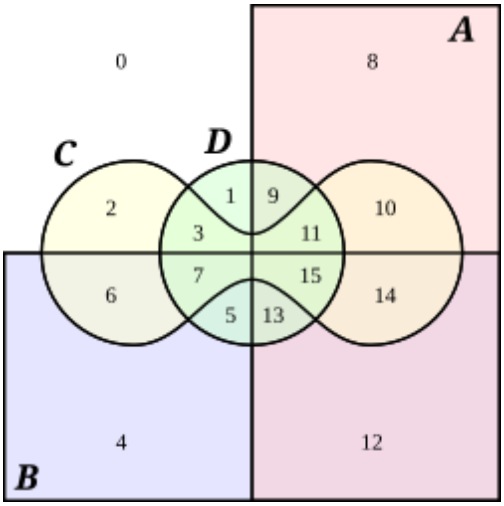

Рисунок 1.2 – Карта Карно для 4 змінних

Карта Карно дозволяє об’єднувати групи з 2, 4, 8 або більше клітинок, у яких значення функції дорівнює 1. Це об’єднання реалізує логічне поглинання змінних, що, у свою чергу, приводить до скорочення булевого виразу. Наприклад, якщо мінтерми 1100 та 1101 мають значення 1, то вони можуть бути об’єднані у вираз, де змінна X4 буде поглинута.

В навчальному процесі карта Карно є одним з найефективніших інструментів: вона дозволяє студентам краще зрозуміти структуру логічної функції, візуально побачити можливості для скорочення, а також сформувати уявлення про логіку функціонування цифрових схем. Саме тому в межах цієї роботи карта Карно реалізована як ключовий візуальний елемент у навчальному симуляторі з інтерактивним підсвічуванням груп, побудовою таблиці істинності та поясненням кожного етапу мінімізації.

1.4. Існуючі програмні рішення для навчання дискретної математики

Для кращого розуміння підходів до навчання теми мінімізації булевих функцій та побудови інтерактивного програмного інструменту було проведено аналіз низки існуючих рішень, які активно використовуються в навчальному

процесі з дискретної математики та цифрової логіки. Основна мета такого аналізу — виявити переваги, недоліки, ефективні візуальні компоненти та алгоритмічні реалізації, які можуть бути адаптовані або покращені в межах нашого програмного продукту.

Logic Friday — безкоштовна програма для аналізу й мінімізації булевих функцій, що підтримує як алгебраїчні, так і графічні методи. Її інтерфейс орієнтований на практикуючих інженерів, тому є доволі технічним і складним для першокурсників. Програма дозволяє будувати таблиці істинності, здійснювати аналіз у СДНФ/СКНФ, а також відображати функції у вигляді логічної схеми. Однак відсутність інтерактивної карти Карно та пояснень до кроків обмежує її навчальний потенціал (див. рис. 1.3).

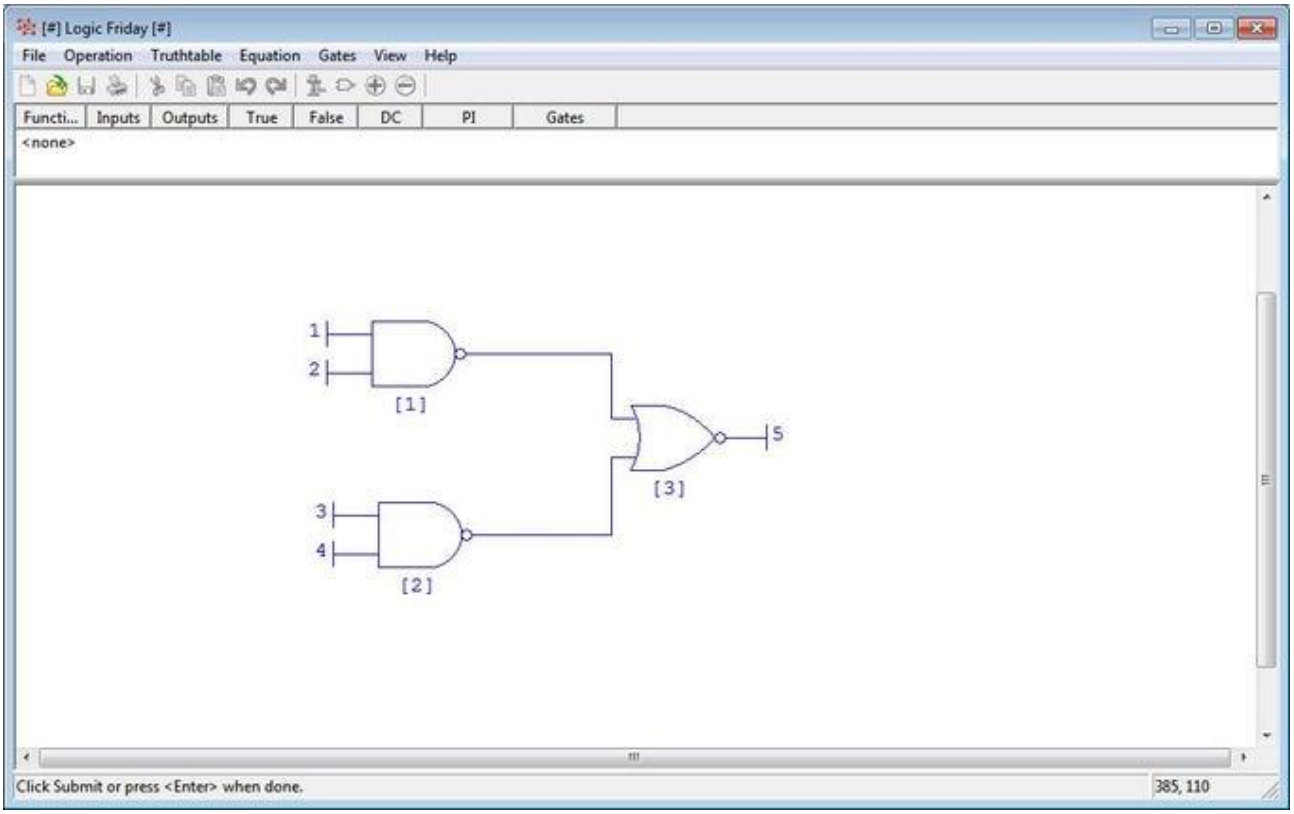

Рисунок 1.3– Інтерфейс програми Logic Friday

Wolfram Alpha (Pro) — потужний онлайн-сервіс, що підтримує введення логічних виразів і виконує їх автоматичну мінімізацію. Незважаючи на вражаючу обчислювальну потужність, сервіс не показує проміжних кроків і не має жодної

візуальної компоненти, що значно знижує його ефективність для освітніх цілей. Він більше підходить для перевірки готових результатів, ніж для навчання(див. рис 1.4). [4]

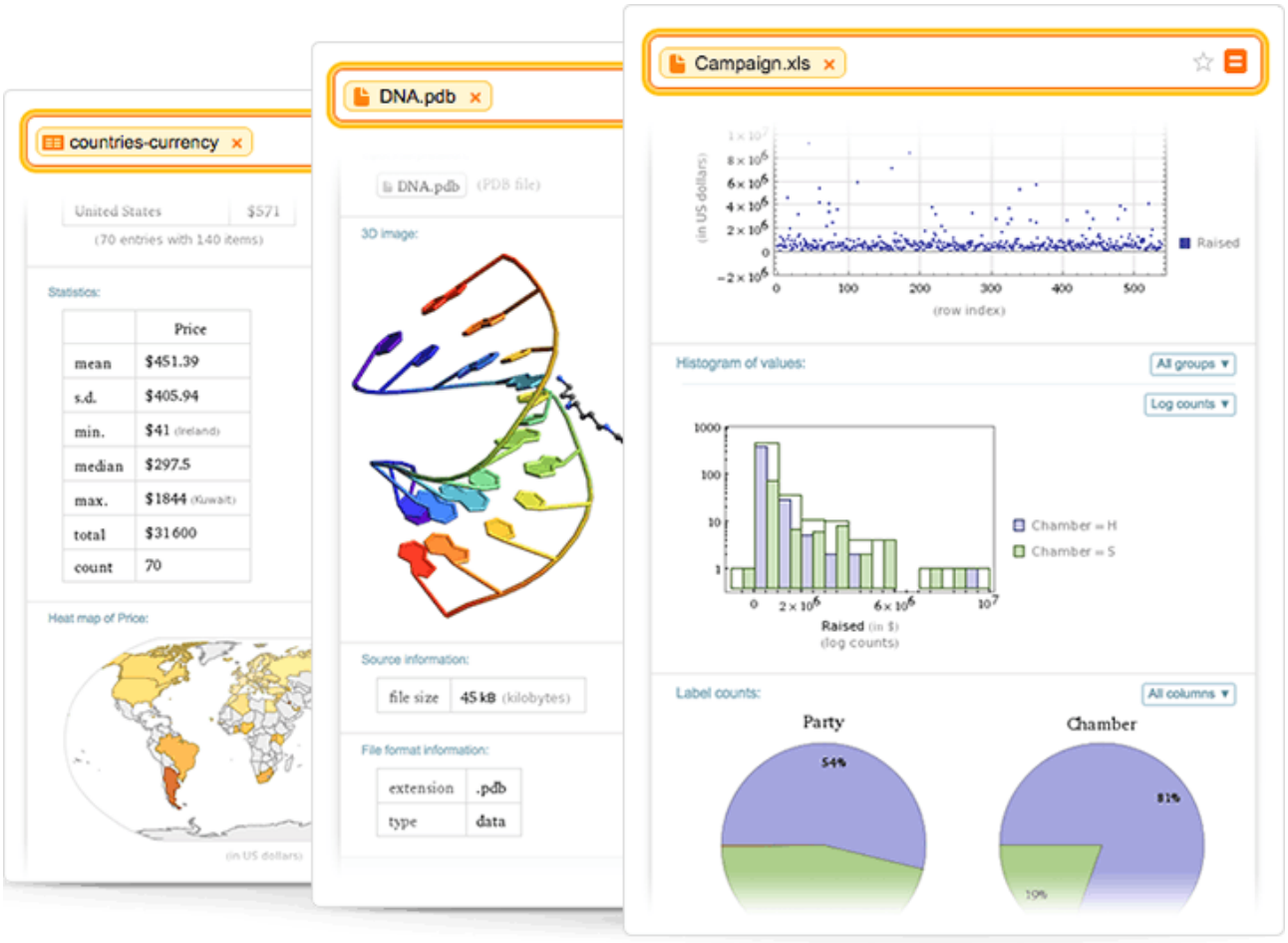

Рисунок 1.4 – Wolfram Alpha

Digital Works — симулятор цифрових схем, який дозволяє будувати та моделювати роботу логічних елементів. Інтерфейс програми наближений до електротехнічного середовища, має складну структуру і потребує ґрунтовної підготовки. Карти Карно відсутні, а навчальні функції реалізовані опосередковано, через побудову схем вручну (див. рис. 1.5).

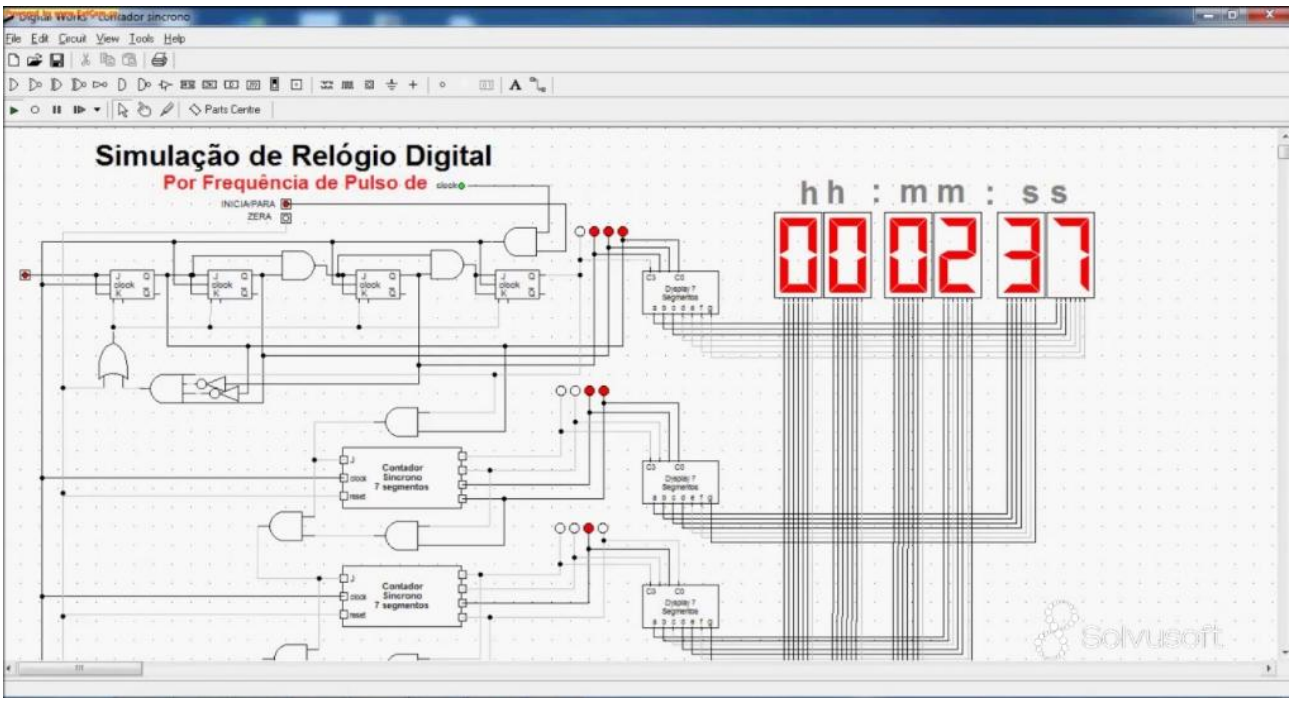

Рисунок 1.5 – Digital works

Logicly — інтерактивне середовище з яскравим візуальним інтерфейсом, яке дозволяє створювати логічні схеми «перетягуванням» елементів. Орієнтоване на учнів середньої школи. Серед недоліків — відсутність роботи з логічними формулами у вигляді виразів та невелика кількість алгоритмічних інструментів. Карти Карно або пояснення мінімізації функцій не передбачено (див. рис. 1.6).

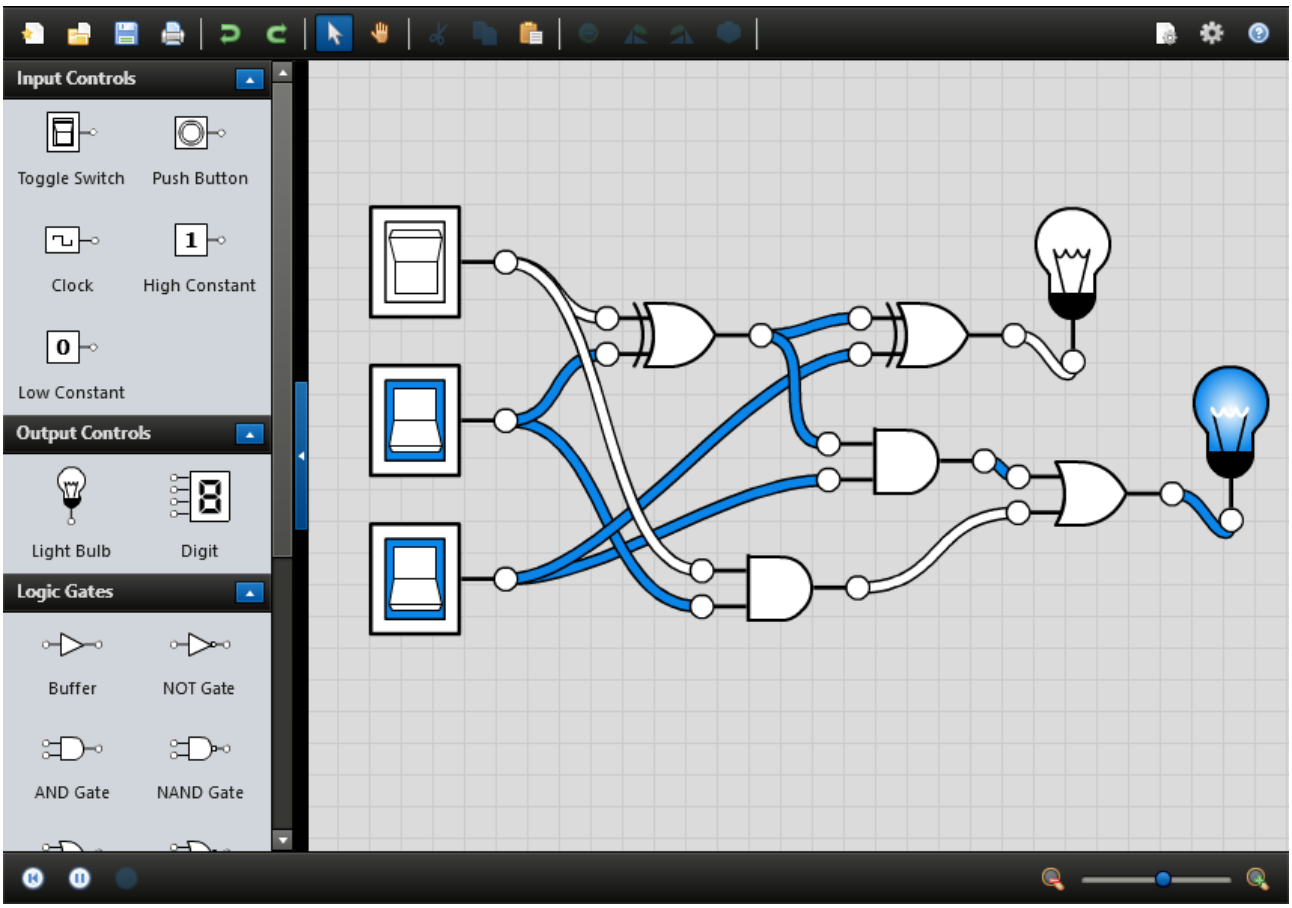

Рисунок 1.6 – Візуальне середовище Logicly

Також варто згадати про проєкти типу Boolean Calculator, KarnaughMap.net, Truthtable.info — це прості онлайн-сервіси для побудови таблиць істинності та мінімізації функцій. Вони часто не мають збереження прогресу, візуалізації групування або покрокового пояснення процесу. Їхній функціонал базується переважно на одній функції (наприклад, побудова карти Карно без пояснення), тому такі сервіси не можна вважати повноцінними навчальними інструментами.

Аналіз показав, що хоча на ринку існують як десктопні, так і веб-інструменти для роботи з булевими функціями, жоден із них не поєднує в собі всі компоненти, важливі для навчального застосунку: пояснення кроків, інтерактивність, адаптивність до рівня користувача, підтримка графічних методів і збереження прогресу. [5]

Саме ці недоліки враховано при розробці нашого веб-застосунку — інтерактивного симулятора для вивчення графічних методів мінімізації булевих функцій. На відміну від перелічених рішень, наша система реалізує поетапну

мінімізацію з детальними поясненнями, інтерактивною картою Карно, підтримкою як навчального, так і практичного режиму, що робить її більш придатною для дистанційного навчання у рамках дисципліни «Дискретна математика».

2. ТЕОРЕТИЧНА ЧАСТИНА

2.1. Основи булевої алгебри

Булева алгебра є фундаментальною математичною дисципліною, яка лежить в основі багатьох галузей комп’ютерних наук, цифрової схемотехніки, програмування та логіки. Вперше вона була формалізована англійським математиком Джорджем Булем у середині XIX століття. На відміну від класичної алгебри, булева алгебра оперує не числовими значеннями, а логічними — «істина» (1) і «хибність» (0), що відповідає двійковій системі числення.

У булевій алгебрі змінні можуть приймати лише два значення: 0 або 1. Основними логічними операціями, які визначають структуру булевих виразів, є:

- **Кон’юнкція (логічне І)** — операція добутку (позначається як ∧ або множення):

  $A \cdot B = 1$ лише тоді, коли $A = 1$ і $B = 1$;
- **Диз’юнкція (логічне АБО)** — операція суми (позначається як ∨ або +): $A + B = 1$, якщо принаймні одна з величин дорівнює 1;
- **Заперечення (логічне НЕ)** — унарна операція інверсії (позначається як ¬A або $\bar{A}$):$\neg A = 1$, якщо $A = 0$, і навпаки.

На основі цих трьох базових операцій можна побудувати будь-який булевий вираз. Важливо, що в булевій алгебрі діють особливі закони, які дозволяють спрощувати вирази:

- **Закон ідемпотентності**: $A + A = A$, $A \cdot A = A$;
- **Закон поглинання**: $A + A \cdot B = A$;
- **Закон де Моргана**:

  $\neg(A \cdot B) = \neg A + \neg B$,

  $\neg(A + B) = \neg A \cdot \neg B$;
- **Дистрибутивність**:

  $A \cdot (B + C) = A \cdot B + A \cdot C$,

  $A + B \cdot C = (A + B) \cdot (A + C)$.

У навчальному контексті булева алгебра використовується не лише як інструмент логічного моделювання, а й як методика розвитку логічного мислення, аналітичних навичок та розуміння принципів побудови цифрових систем. Знання булевої алгебри є необхідним для подальшого вивчення електроніки, мікропроцесорної техніки, комп'ютерної логіки та формальних мов.

Окреме місце у булевій алгебрі займає поняття **булевої функції** — це функція, яка на вхід приймає кілька булевих змінних і повертає логічне значення. Кожну булеву функцію можна подати у вигляді:

- **таблиці істинності**, де перелічено всі можливі комбінації вхідних значень та відповідні результати;
- **логічного виразу**, записаного за допомогою операторів І, АБО, НЕ;
- **нормальної форми** — повної диз'юнктивної (СДНФ) або кон'юнктивної (СКНФ) форми;
- **графічного подання**, наприклад, за допомогою карти Карно.

Булева алгебра є не лише базою для побудови логічних схем, а й основою алгоритмів мінімізації логічних виразів. Саме ця властивість дозволяє скорочувати складність обчислень, оптимізувати програмне забезпечення та цифрові пристрої. [6]

У рамках нашої розробки основи булевої алгебри реалізовано через створення функцій для побудови таблиці істинності, генерацію всіх можливих комбінацій змінних, а також формування логічних виразів на основі карт Карно. Таким чином, теоретична основа булевої алгебри була покладена в основу функціональності навчального застосунку.

2.2. Алгоритми побудови карт Карно та діаграм

Візуальне представлення логічних функцій відіграє важливу роль у їх аналізі та спрощенні. Саме тому в булевій алгебрі та цифровій логіці широко використовуються **карти Карно** та **діаграми Вейна**, які дозволяють відображати логічні залежності між змінними та наочно визначати можливості для мінімізації.

Діаграма Вейна

Діаграма Вейна — це спосіб подання множин і логічних співвідношень між ними у вигляді перекривних кіл або областей на площині. Вона надає можливість відобразити всі можливі комбінації логічних змінних за допомогою площ, що відповідають істинним значенням. У випадку чотирьох змінних (A, B, C, D) кожна з них представлена окремим кругом, а їх поєднання формує 16 унікальних областей, які відповідають мінтермам булевої функції (див. рис. 1.1).

Хоча діаграми Вейна є надзвичайно зручними для пояснення основ логіки та візуалізації взаємозв'язків між множинами, у цифровій логіці вони мають обмежене застосування через складність побудови при великій кількості змінних. Їх основною освітньою перевагою є інтуїтивність, тому вони часто використовуються на початкових етапах вивчення булевої алгебри.

Карта Карно

Карта Карно (англ. Karnaugh Map) є вдосконаленим методом, що дозволяє більш ефективно представляти булеву функцію для подальшої мінімізації. Вона має вигляд прямокутної таблиці, де кожна клітинка відповідає одній комбінації значень змінних, а значення в клітинці — результат булевої функції для цієї комбінації (0 або 1). Кількість клітинок завжди дорівнює 2n, де n — кількість змінних.

Основним елементом побудови карти Карно є **код Грея** — особливий порядок упорядкування бінарних чисел, у якому сусідні значення відрізняються лише одним бітом. Це дозволяє легко визначити, які мінтерми можна об'єднати, що є основою процесу спрощення логічного виразу. Наприклад, мінтерми з номерами 0100 (4) та 0101 (5) можуть бути об'єднані, оскільки вони відрізняються лише в одному розряді.

При побудові карти Карно для чотирьох змінних (наприклад, X1,X2,X3,X4) використовується сітка 4×4, де дві змінні визначають рядки, а дві — стовпці. Написи зверху і збоку формуються за допомогою коду Грея: 00, 01, 11, 10, що забезпечує правильне розташування мінтермів у таблиці (див. рис. 1.2).

Алгоритм побудови карти Карно включає такі основні кроки:

1. Вибір кількості змінних і формування всіх можливих комбінацій 2n.

2. Формування сітки таблиці з використанням коду Грея для рядків і стовпців.
3. Заповнення клітинок значеннями булевої функції на основі таблиці істинності.
4. Групування сусідніх одиниць у прямокутники розміром 2^n (2, 4, 8…), які відповідають спрощеним логічним термам.

На відміну від діаграм Вейна, карта Карно не лише ілюструє функцію, а й дозволяє її **автоматично мінімізувати**. Саме тому вона широко використовується як у теоретичних курсах, так і у практичних симуляторах, зокрема — у нашому навчальному застосунку.

У нашому проєкті побудова карти Карно реалізована алгоритмічно. Змінні автоматично кодуються у форматі Грея, створюється відповідна сітка, значення функції заносяться у клітинки, після чого активується алгоритм виділення груп для мінімізації. Уся побудова відбувається в інтерактивному режимі з візуальним підсвічуванням груп та покроковими поясненнями. Це дозволяє студенту не лише побачити результат, але й зрозуміти логіку процесу побудови та скорочення. [7]

Таким чином, поєднання діаграм Вейна та карт Карно в навчальному середовищі забезпечує глибоке розуміння логіки функцій та принципів їх оптимізації.

2.3. Використання графічного підходу для навчання

Сучасна освітня практика все більше орієнтується на використання візуальних та інтерактивних інструментів, які сприяють кращому засвоєнню складного матеріалу. Це особливо актуально в галузі математичної освіти, де значна частина понять має абстрактну природу і є важкою для сприйняття без відповідної візуалізації. Одним з ефективних засобів підвищення якості засвоєння знань у цій сфері є **графічний підхід**, який передбачає подання теоретичних і практичних понять через діаграми, схеми, карти, таблиці та інші візуальні форми.

У контексті вивчення булевих функцій і методів їх мінімізації графічний підхід набуває особливої цінності. Оскільки більшість студентів на початкових

етапах мають труднощі з формальним оперуванням логічними виразами, графічні засоби дозволяють візуалізувати структуру логічної функції, продемонструвати взаємозв’язки між змінними та пояснити логіку процесу спрощення.

Одним із найпоширеніших способів графічного представлення булевих функцій є **карта Карно**. Вона дозволяє перетворити таблицю істинності у двовимірну сітку, де кожна комірка відповідає певному мінтерму функції. Завдяки використанню коду Грея у впорядкуванні змінних, карта Карно дозволяє легко знаходити та групувати сусідні одиниці — логічні комбінації, що відрізняються лише одним значенням. Візуальне групування таких комірок спрощує розуміння принципу мінімізації і дає змогу швидко сформувати скорочений логічний вираз. [8]

Інтеграція графічного підходу в навчальний процес має низку переваг:

- Залучення візуальної пам’яті — студенти краще запам’ятовують матеріал, представлений графічно;
- Інтуїтивне розуміння логіки функції — за допомогою візуального групування, підсвітки, кольорових маркерів;
- Стимулювання самостійного мислення — студенти самі шукають спрощення, виділяючи групи на карті;
- Зменшення когнітивного навантаження — складні алгебраїчні вирази замінюються на більш доступну візуальну форму.

У нашому програмному застосунку графічний підхід реалізований на декількох рівнях. По-перше, карта Карно побудована у вигляді інтерактивної сітки, де користувач може бачити значення функції у кожній комірці та як ці значення групуються для формування спрощеного виразу. По-друге, для кожного етапу мінімізації надається пояснення: які мінтерми об’єднуються, чому саме вони, та який вигляд має новий логічний терм. Цей підхід забезпечує не лише виведення результату, а й формування **розуміння механізму спрощення**.

Крім того, інтерфейс програми адаптований до навчального процесу: кольорове підсвічування груп, крокова анімація об’єднань, підказки та легенда забезпечують зручну навігацію для студентів, які тільки починають вивчення цієї

теми. Таким чином, графічний підхід стає не просто допоміжним інструментом, а **центральною дидактичною складовою**, яка визначає ефективність усього навчального середовища.

З огляду на сучасні тенденції цифрової освіти, використання графічного підходу у викладанні дискретної математики не лише доцільне, але й необхідне. Воно забезпечує доступність, наочність, інтерактивність і адаптивність навчального матеріалу, що значною мірою сприяє формуванню якісних знань у студентів.

3. ПРАКТИЧНА ЧАСТИНА

3.1. Архітектура та структура застосунку

Розроблений вебзастосунок є інтерактивним навчальним симулятором для графічної мінімізації булевих функцій, основним завданням якого є візуалізація та пояснення процесу мінімізації з використанням карт Карно. Архітектура застосунку побудована таким чином, щоб забезпечити максимальну модульність, зручність для користувача та розширюваність функціоналу в майбутньому.

Застосунок реалізовано за допомогою односторінкової архітектури (SPA), що дозволяє уникнути перезавантажень сторінок та забезпечує швидкий, динамічний відгук інтерфейсу. Усі компоненти реалізовано з використанням React — сучасної JavaScript-бібліотеки для побудови користувацьких інтерфейсів. Для стилізації інтерфейсу застосовується Tailwind CSS, а для типізації та структурованості коду — TypeScript.

Фізична структура проєкту

```
minimization-simulator/
├── src/
│   ├── components/          # Основні UI-компоненти
│   │   ├── TruthTableInput.tsx      # Введення таблиці істинності
│   │   ├── KarnaughMap.tsx          # Візуалізація карти Карно
│   │   ├── MinimizationSteps.tsx    # Кроки мінімізації з поясненнями
│   │   ├── Practice.tsx             # Практичні завдання
│   │   └── Theory.tsx               # Теоретичні матеріали
│   ├── utils/               # Алгоритми та обробка даних
│   │   ├── minimize.ts        # Мінімізація функцій (Quine-McCluskey)
│   │   └── grayCode.ts        # Генерація коду Грея
│   ├── types/               # TypeScript типи даних
│   │   └── Term.ts            # Типи для логічних термів і кроків
│   ├── pages/               # Роутинг між сторінками
```

```
│   └── App.tsx            # Головна точка входу
```

Логічна структура та взаємодія модулів

1. Модуль введення даних (TruthTableInput): Дозволяє користувачу вибрати кількість змінних (від 2 до 4) та задати значення булевої функції для кожної комбінації вхідних змінних.
2. Модуль побудови карти Карно (KarnaughMap): Автоматично будує карту Карно відповідно до заданої таблиці істинності. Відображає клітинки згідно з кодом Грея, підсвічує групи мінтермів, що об’єднуються.
3. Модуль алгоритмічного обчислення (minimize.ts): Реалізує спрощення логічної функції методом Квайна–Мак–Класкі з поясненням кожного кроку. Для кожної групи формується булевий вираз і коментар, чому відбулося об’єднання.
4. Модуль виводу кроків мінімізації (MinimizationSteps): Відображає покроковий процес об'єднання термів, пояснює логіку дій, формує підсумковий логічний вираз та підсвічує відповідні клітинки карти.
5. Навчальний модуль (Theory.tsx): Містить інтерактивну довідкову інформацію, що пояснює поняття булевої алгебри, карти Карно, методів мінімізації та приклади.
6. Практичний модуль (Practice.tsx): Генерує завдання різної складності, які користувач повинен розв’язати за допомогою симулятора. Додає елемент самоперевірки.
7. Система переходів (React Router): Реалізує навігацію між розділами: "Теорія", "Симулятор", "Практика", "Домашня сторінка".

Використані технології

- React – компонентна модель для створення динамічного інтерфейсу;
- TypeScript – типобезпечна надбудова над JavaScript;

- Tailwind CSS – гнучка утилітарна система стилів;
- Framer Motion – для анімацій переходів та елементів;
- Vite – швидкий інструмент для збірки та розробки;
- LocalStorage API – для збереження прогресу користувача.

Переваги архітектури

- Інтерактивність: користувач бачить зміни одразу без перезавантаження;
- Модульність: кожен компонент можна змінювати чи розширювати незалежно;
- Навчальна логіка: покрокове пояснення мінімізації сприяє глибокому розумінню;
- Адаптивність: інтерфейс коректно працює на різних пристроях.

Таким чином, архітектура застосунку є оптимальною для реалізації навчального симулятора з високою інтерактивністю, адаптивністю та орієнтацією на користувача. Обрана структура дозволяє легко масштабувати функціонал, додавати нові алгоритми та інтегрувати режим контролю знань.

3.2. Реалізація основних функцій симулятора

У процесі реалізації віртуального симулятора з теми «Графічні методи мінімізації булевих функцій» було створено низку функціональних модулів, які забезпечують не лише побудову таблиці істинності та карти Карно, а й автоматичну мінімізацію логічних функцій, покрокові пояснення дій та зручну взаємодію з користувачем. Основна логіка реалізована у вигляді окремих компонентів з використанням React та TypeScript, що забезпечує гнучкість і масштабованість програми.

1. Алгоритм мінімізації (Quine–McCluskey)

Ключовим обчислювальним ядром симулятора є реалізація алгоритму Квайна–Мак-Класкі, який здійснює спрощення логічної функції шляхом групування

мінтермів у прості імпліканти. Алгоритм представлено у модулі quineMcCluskey.ts, який виконує:

- перетворення мінтермів у бінарний вигляд;
- групування за кількістю одиниць;
- ітеративне об'єднання термів;
- побудову таблиці покриття;
- вибір суттєвих імплікант.

export function quineMcCluskey(minterms: number[], dontCares: number[], variables: number) { ... }

Цей модуль також формує структуру даних для подальшого виводу кроків мінімізації з поясненнями.

2. Побудова та візуалізація карти Карно

Важливою складовою інтерфейсу є візуалізація карти Карно, яка динамічно будується на основі введених значень у таблиці істинності. Компонент KarnaughMap.tsx генерує сітку відповідно до кількості змінних і кодів Грея, що дозволяє візуально відобразити сусідство мінтермів.

```
<div className="grid">
 {truthTable.map((value, index) => (
   <div key={index} className={`cell ${value ? 'active' : ''}`}>
    {value ? '1' : '0'}
   </div>
 ))}
</div>
```

Крім того, реалізовано підсвічування груп об’єднаних клітинок, що допомагає користувачу зрозуміти, які саме мінтерми були об'єднані у процесі мінімізації.

3. Таблиця істинності

Інтерактивна таблиця істинності (TruthTable.tsx) дозволяє користувачу самостійно задавати значення булевої функції. Вона автоматично генерується залежно від кількості змінних, і кожна клітинка є активним елементом, що змінює значення fff при натисканні.

```
<td>
 <button
  onClick={() => handleToggle(index)}
  className={value ? 'active' : ''}
 >
  {value ? '1' : '0'}
 </button>
</td>
```

Це дозволяє миттєво побачити зміни на карті Карно та в результаті мінімізації.

4. Компонент введення формул (екранна клавіатура)

Для зручного введення логічних виразів передбачено екранну клавіатуру (FormulaKeyboard.tsx), яка дозволяє додавати змінні та логічні оператори без необхідності вводити їх з клавіатури. Це особливо корисно для навчальних цілей та мобільних пристроїв.

```
<button onClick={() => onInput("¬")}>¬</button>
<button onClick={() => onInput("∧")}>∧</button>
```

Також реалізовано динамічну генерацію кнопок для всіх доступних змінних.

5. Система покрокового пояснення мінімізації

Унікальною особливістю симулятора є модуль MinimizationSteps.tsx, який відображає кожен крок мінімізації логічної функції:

- тип кроку (групування, об’єднання, суттєві терми, фінальний результат);
- текстове пояснення дії;
- список термів, що беруть участь у кроці.

```
<MinimizationStep
 type="combining"
 explanation="Комбінації 010 та 011 об'єднуються → x1¬x2"
 terms={[...]}
>
```

Це дозволяє не просто отримати кінцевий вираз, а зрозуміти логіку мінімізації, що значно покращує засвоєння матеріалу студентами.

6. Взаємодія компонентів і навігація

Усі компоненти об’єднано у головному компоненті App.tsx. Використання React Router дозволяє реалізувати перемикання між сторінками: Теорія, Симулятор, Практика. Програма зберігає стан користувача, що дозволяє не втрачати дані при оновленні сторінки.

Результат

Завдяки цим реалізованим функціям симулятор дозволяє:

- створити і редагувати таблицю істинності;
- автоматично побудувати карту Карно;
- виконати алгоритмічну мінімізацію з поясненням кожного кроку;
- побачити підсумковий вираз у СДНФ або скороченій формі;
- взаємодіяти з інтерфейсом швидко, зручно та інтуїтивно.

Таким чином, реалізований функціонал симулятора не лише відображає логіку процесів мінімізації, а й перетворює складну абстрактну тему на візуально зрозумілий навчальний процес.

3.3. Приклади роботи програми та результати мінімізації

У цьому розділі наведено приклади використання реалізованого симулятора для побудови карт Карно та виконання мінімізації булевих функцій. Кожен приклад ілюструє ключові можливості застосунку: введення таблиці істинності, побудову карти Карно, підсвічування груп мінтермів, покрокову мінімізацію та отримання остаточного логічного виразу.

Приклад 1. Функція від двох змінних

Введено таблицю істинності функції:

x1	x2	f
0	0	0
0	1	1
1	0	1
1	1	1

Після побудови карти Карно система автоматично підсвічує групи мінтермів та виконує мінімізацію:

Результат:

$f(x1,x2)=x1+\neg x1 \cdot x2$

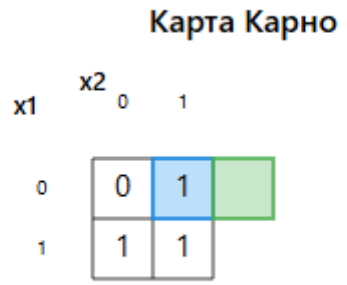

Рисунок 3.1 – Приклад мінімізації для функції двох змінних

Приклад 2. Функція від трьох змінних

Значення функції дорівнює 1 для мінтермів з номерами 1, 2, 4, 5, 6, 7:

x1	x2	x3	f
0	0	1	1
0	1	0	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

На карті Карно чітко видно можливість групування чотирьох мінтермів у прямокутник 2×2, а також додаткові пари для оптимізації.

Результат:

f(x1,x2,x3)=x1+x2·x3+¬x2·¬x3

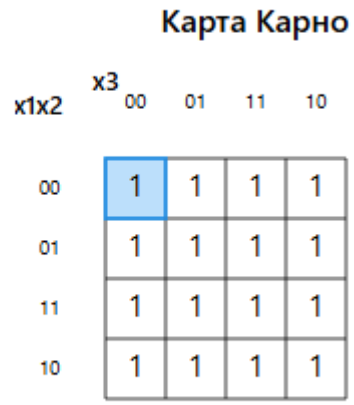

Рисунок 3.2– Мінімізація функції трьох змінних у симуляторі

Приклад 3. Практичне завдання

У режимі «Практичні завдання» студенту пропонується вирішити завдання з теми графічної побудови логічної функції. Інтерфейс автоматично генерує завдання, наприклад:

Завдання 1: Представити функцію

y=(x1→¬x2)(x1∨x2∨x3)

на карті Карно.

Користувач повинен:

1. Ввести логічний вираз у текстове поле.
2. Використовувати інтерактивну клавіатуру для додавання змінних і операцій.
3. Дотримуватися синтаксису (¬ для заперечення, ∧ — кон’юнкція, ∨ — диз’юнкція, → — імплікація).
4. За потреби — скористатися підказкою, яка пояснює правила побудови виразів.

5. Перевірити правильність результату та перейти до наступного завдання.

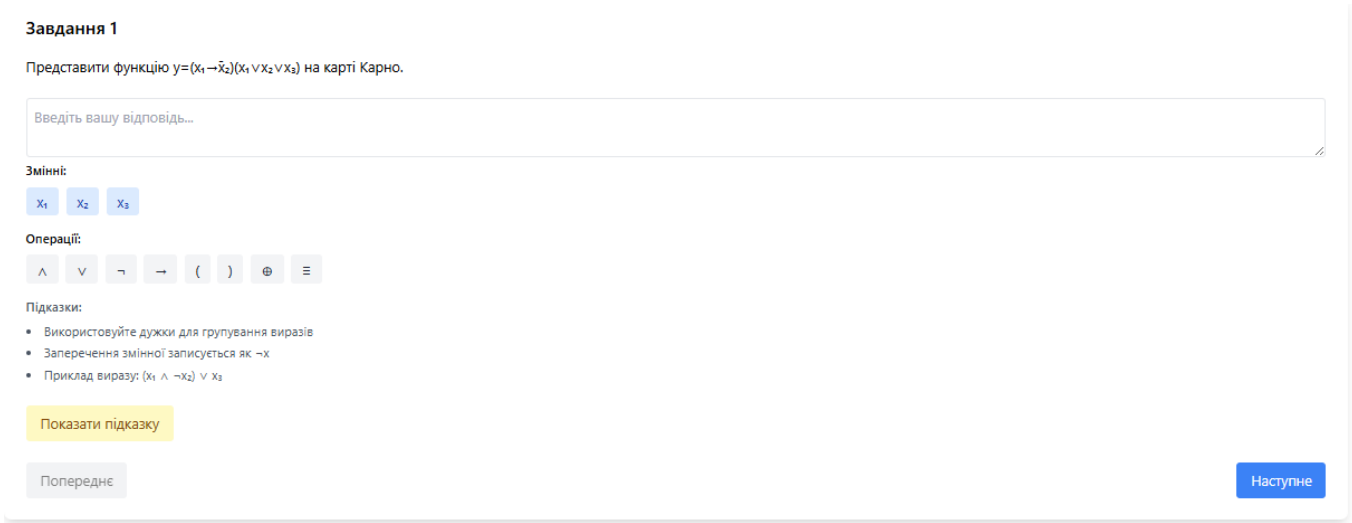

Рисунок 3.3 – Режим практичного завдання: побудова виразу для карти Карно

Коментарі до реалізації

- Усі приклади були протестовані на функціях з 2–4 змінними.
- Мінімізація виконується коректно для СДНФ з будь-якими поєднаннями мінтермів.
- Інтерфейс адаптивний: працює як на десктопних, так і на мобільних пристроях.
- Покрокове пояснення суттєво підвищує розуміння логіки об’єднання мінтермів.

3.4. Інструкція для користувача

1. Початок роботи із симулятором (див. рис. 3.4)

• Після запуску застосунку користувач бачить стартовий екран із параметром **«Кількість змінних»**, де можна обрати значення від 2 до 4.

• Для початку потрібно вказати бажану кількість змінних, яка буде використана для побудови таблиці істинності.

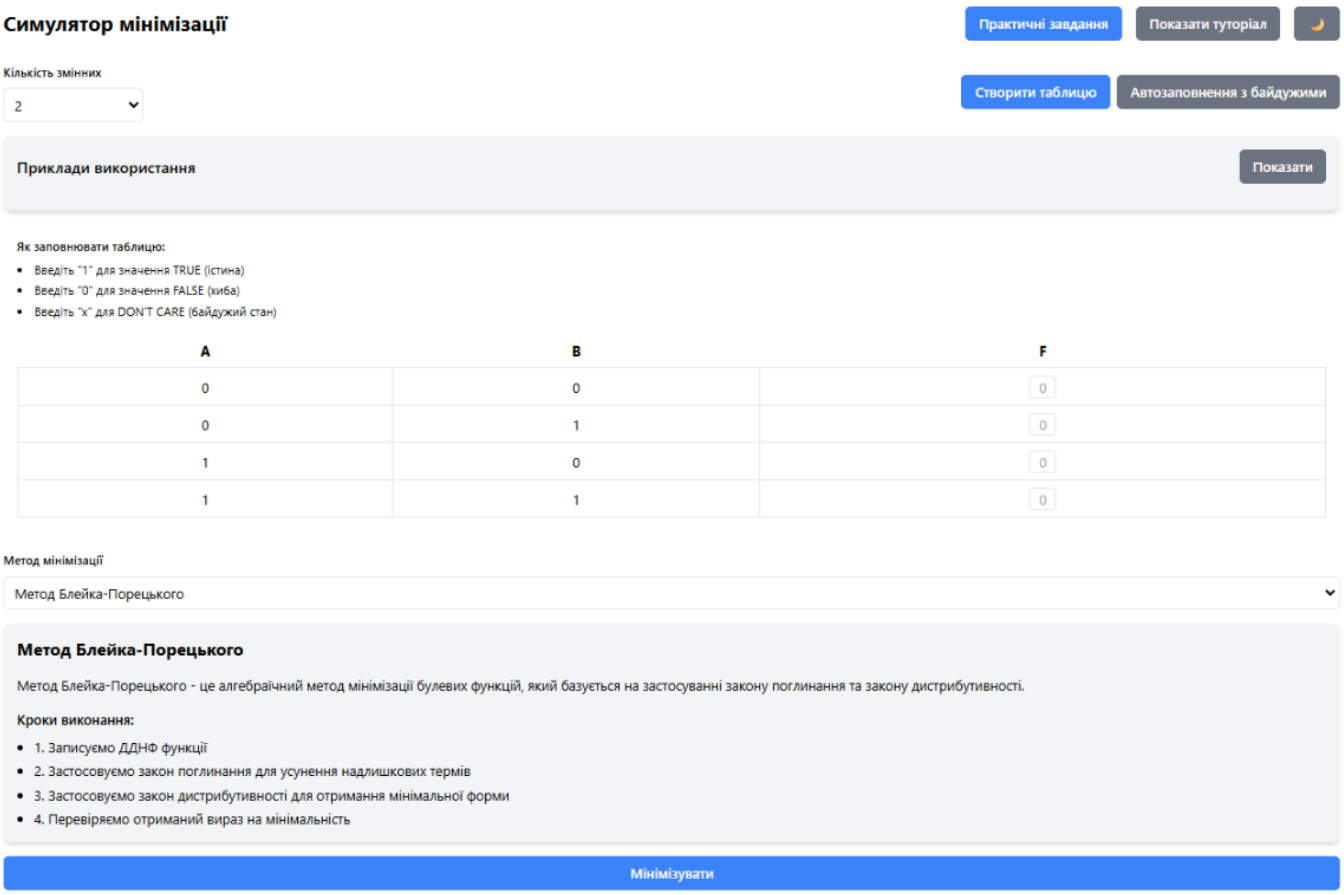

Рисунок 3.4 – Головна сторінка

2. Введення значень у таблицю істинності (див. рис. 3.5)

• Після вибору кількості змінних автоматично генерується **таблиця істинності**.

• Користувач повинен ввести:

- 1 — якщо функція істинна;
- 0 — якщо хибна;
- x — якщо байдуже (DON’T CARE).

• Також доступна кнопка **«Автозаповнення з байдужими»**, яка дозволяє швидко створити приклад із мінтермами та байдужими умовами.

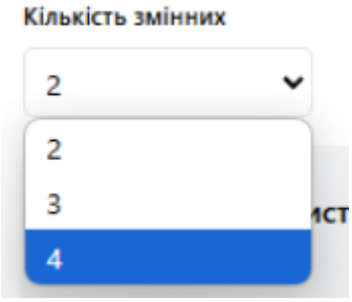

Рисунок 3.5 – Таблиця істинності та автозаповнення

3. Вибір методу мінімізації (див. рис. 3.6)

• Після заповнення таблиці користувач обирає бажаний **метод мінімізації** із випадаючого списку:

- **Метод Блейка-Порецького**
- **Метод Квайна-МакКласкі**
- **Карти Карно**

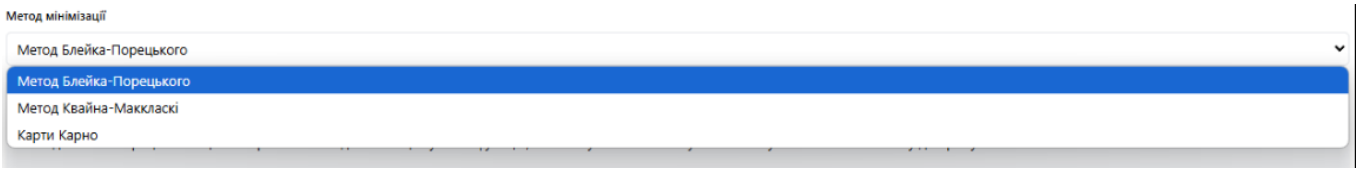

Рисунок 3.6 – Список методів мінімізації

4. Вивчення прикладів використання (див. рис. 3.7)

• Перед початком роботи можна натиснути кнопку **«Показати»** у блоці **«Приклади використання»**, щоб ознайомитися з поясненнями щодо:

- поняття **мінтермів**;
- способу використання **байдужих умов**.

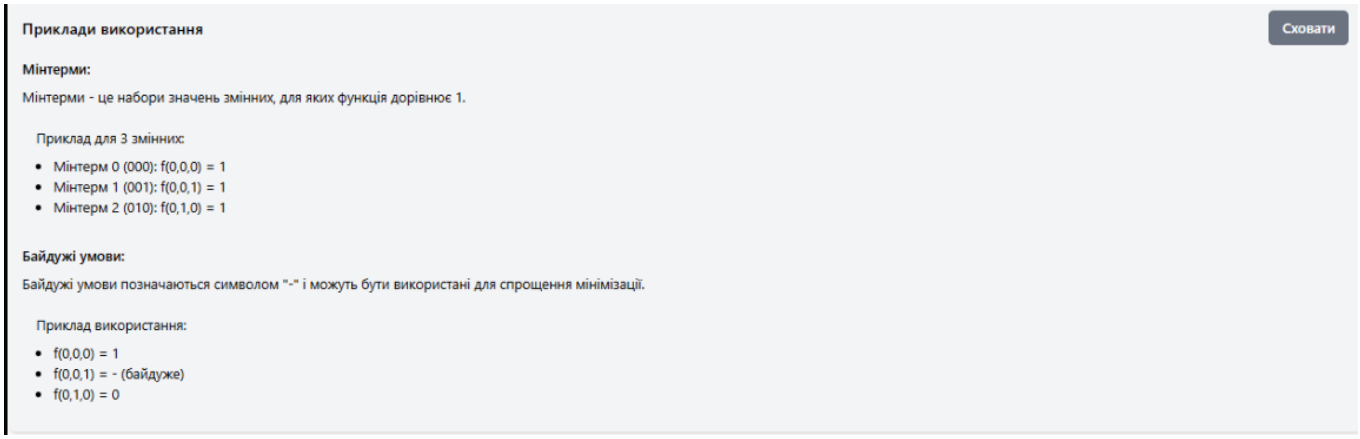

Рисунок 3.7 – Приклади використання та пояснення мінтермів

5. Використання туторіала (див. рис. 3.8)

• Кнопка **«Показати туторіал»** запускає покрокову інструкцію, яка допомагає новим користувачам швидко ознайомитися з функціоналом симулятора.

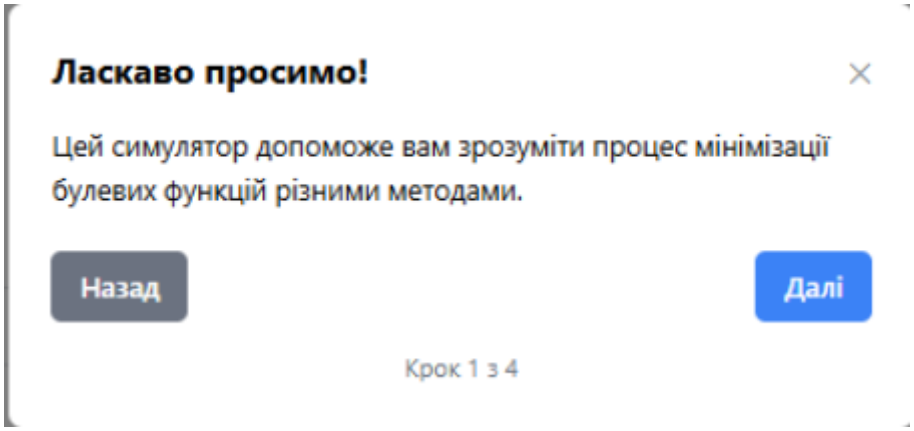

Рисунок 3.8 – Стартовий крок інструкції (туторіалу)

6. Мінімізація методом Блейка-Порецького (див. рис. 3.9)

• Після вибору цього методу, програма автоматично згенерує покрокове пояснення:

- показує мінтерми;
- формує ДНФ;
- застосовує закони булевої алгебри;
- виводить мінімізований вираз.

• Доступні додаткові опції: **зберегти кроки**, **експортувати** результат або

копіювати мінімізований вираз.

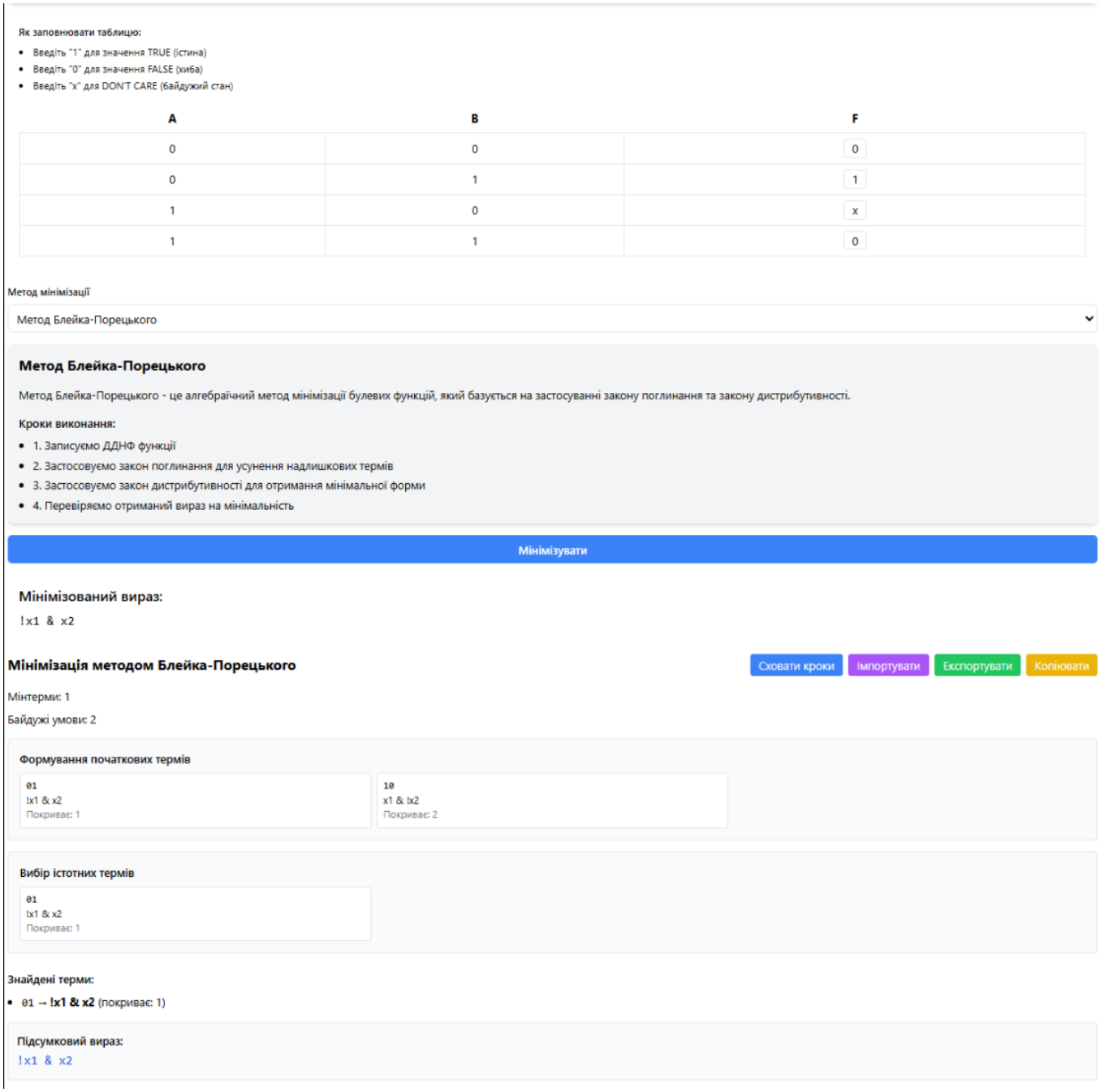

Рисунок 3.9 – Приклад мінімізації методом Блейка-Порецького

7. Практичні завдання (див. рис. 3.10 – 3.12)

• Користувач може перейти в режим **«Практичні завдання»**, де система автоматично формує завдання за темою:

– Завдання 1 (рис. 3.10): Побудова функції за формулою

• Потрібно записати логічний вираз і перетворити його у карту Карно.

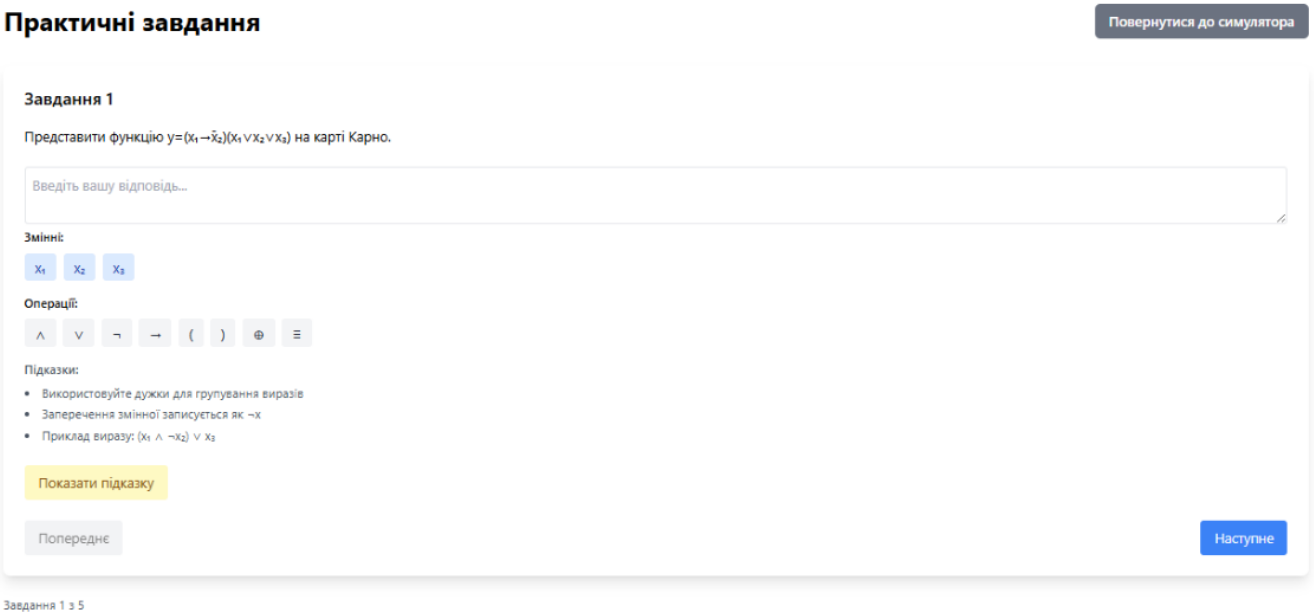

Рисунок 3.10 – Побудова функції за заданою формулою

– Завдання 2 (рис. 3.11): Аналіз заданої карти Карно

• Користувач має знайти усі можливі подання функції та вибрати мінімальне.

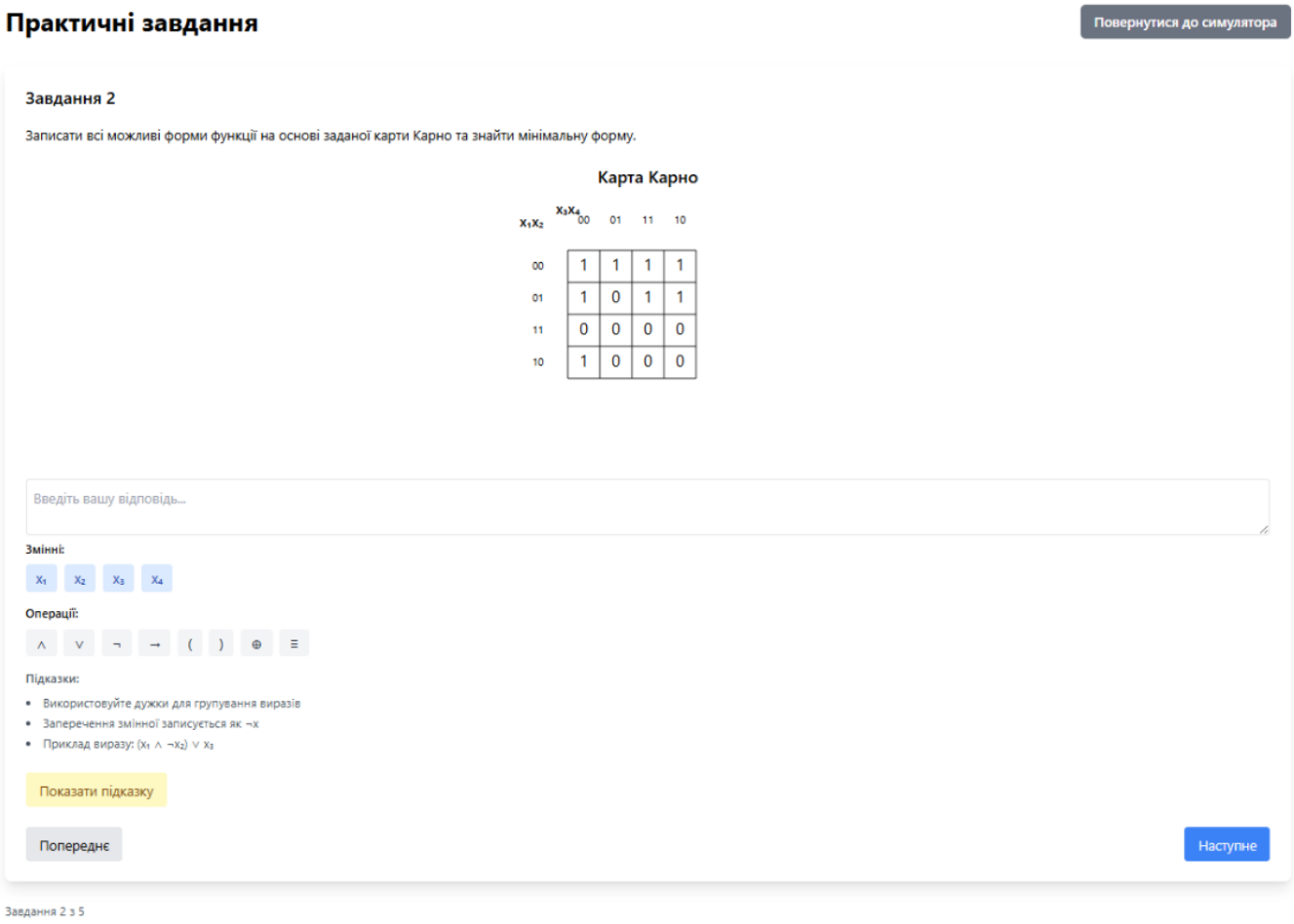

Рисунок 3.11 – Задача на мінімізацію за картою Карно

– Завдання 4 (рис. 3.12): Побудова функції за контактною схемою

• На основі візуальної контактної схеми потрібно сформулювати відповідну булеву функцію.

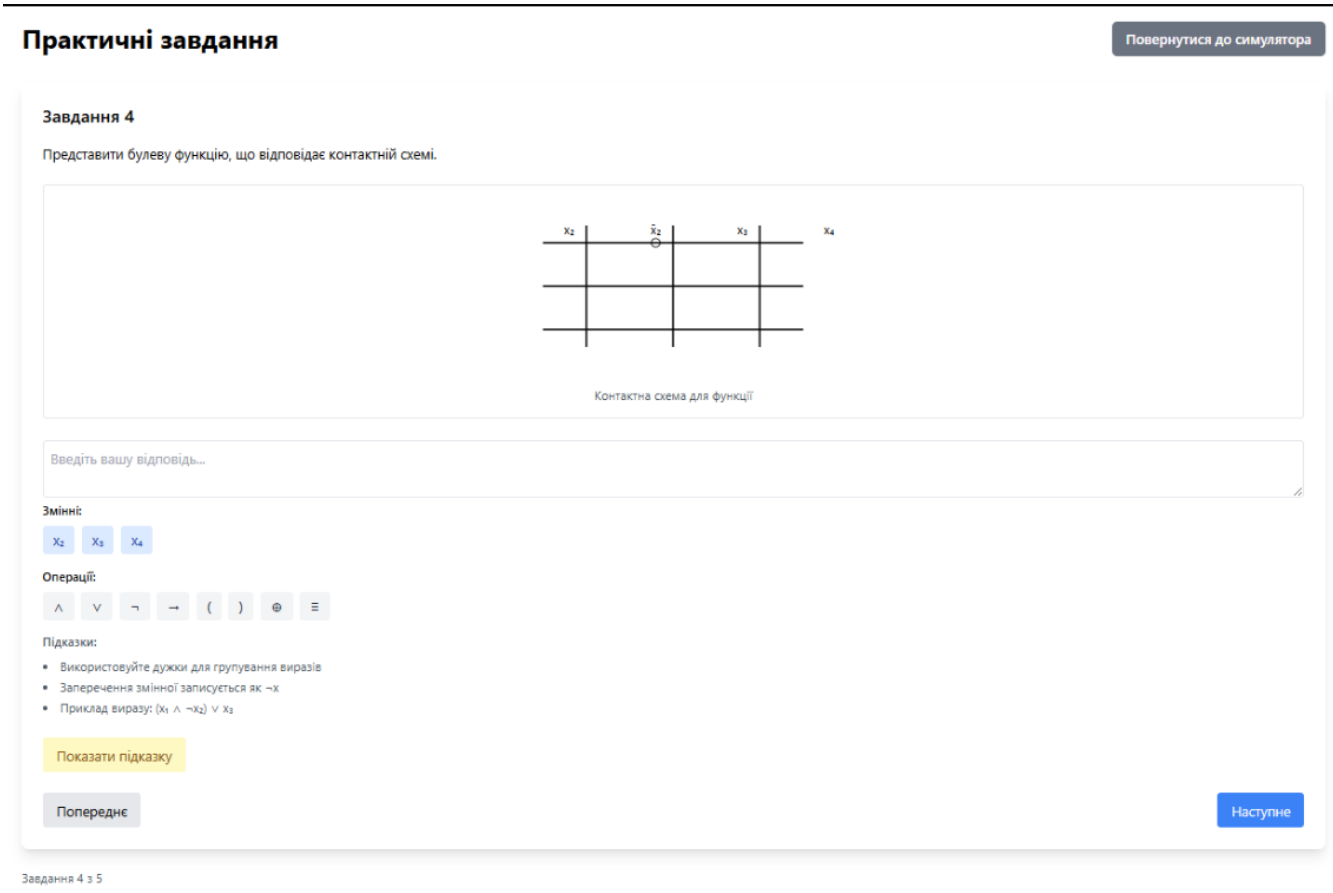

Рисунок 3.12 – Контактна схема у практичному завданні

ВИСНОВКИ

У дипломній роботі було розроблено та реалізовано віртуальний симулятор, призначений для вивчення графічних методів мінімізації булевих функцій у межах навчального курсу «Дискретна математика». Розроблений програмний продукт поєднує інструменти автоматичної побудови таблиць істинності, візуалізації карт Карно, мінімізації логічних функцій за допомогою методів Блейка–Порецького, Квайна–Мак-Класкі, а також систему практичних завдань.

У процесі виконання роботи були досягнуті наступні результати:

- Проведено огляд та аналіз теоретичних основ булевої алгебри, графічних методів мінімізації та сучасних підходів до навчання у сфері дискретної математики.
- Сформульовано функціональні та нефункціональні вимоги до програмного забезпечення, обґрунтовано вибір стеку технологій для реалізації інтерфейсу, алгоритмів та візуалізації.
- Реалізовано інтерактивний веб-додаток на основі React та TypeScript із використанням TailwindCSS для оформлення інтерфейсу та бібліотек фреймворку Framer Motion для анімацій.
- Забезпечено підтримку трьох методів мінімізації булевих функцій, включно з графічною побудовою карт Карно, що дозволяє візуально зрозуміти процес логічного спрощення.
- Впроваджено навчальний режим із теоретичним супроводом і покроковими підказками, що сприяє закріпленню знань.
- Розроблено систему практичних завдань з автоперевіркою, що робить застосунок придатним до використання в освітніх закладах.
- Створено детальну інструкцію користувача, яка полегшує навігацію в інтерфейсі та роботу з симулятором навіть без попередньої підготовки.

Практичне значення розробленого симулятора полягає в його здатності наочно демонструвати складні теоретичні концепції булевої алгебри та логічного моделювання. Це робить його корисним як для студентів, що вивчають дискретну

математику, так і для викладачів як візуальний інструмент на лекціях чи лабораторних роботах.

Подальший розвиток проєкту передбачає:

- реалізацію збереження прогресу та авторизації користувачів;
- підтримку побудови схем на основі мінімізованих функцій;
- розширення практичних завдань і введення рівнів складності;
- впровадження адаптивної версії для мобільних пристроїв.

Таким чином, поставлена мета дипломної роботи була досягнута, а розроблений симулятор повністю відповідає вимогам до сучасного навчального програмного забезпечення.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ольховська, О. В., Черненко, О. О. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня бакалавра. Полтава: ПУЕТ, 2022. – 67 с. – 1 електрон. опт. диск (CD-ROM).
2. Документація React [Електронний ресурс]. – Режим доступу: https://reactjs.org/docs/getting-started.html
3. Документація TypeScript [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs
4. Документація Tailwind CSS [Електронний ресурс]. – Режим доступу: https://tailwindcss.com/docs
5. Документація Vite – інструмент для фронтенд-розробки [Електронний ресурс]. – Режим доступу: https://vitejs.dev/
6. Метод Квайна-Мак-Класкі. Пояснення та приклади [Електронний ресурс]. – Режим доступу: https://www.electronics-tutorials.ws/
7. Інтерактивний симулятор булевих функцій Logic.ly [Електронний ресурс]. – Режим доступу: https://logic.ly/demo
8. Документація Framer Motion [Електронний ресурс]. – Режим доступу: https://www.framer.com/motion/
9. Boolean Function Minimizer (Wolfram Alpha) [Електронний ресурс]. – Режим доступу: https://www.wolframalpha.com/
10. Документація React Router [Електронний ресурс]. – Режим доступу: https://reactrouter.com/en/main/start/tutorial

ДОДАТОК А.

==========

ДОДАТОК А. ОСНОВНИЙ КОД ПРОГРАМИ

==========

А.1. Алгоритм Квайна-Мак-Класкі (quineMcCluskey.ts)

```typescript
import { Term, MinimizationStep } from '../types';

export function quineMcCluskey(
  minterms: number[],
  dontCares: number[],
  variables: number
): { terms: Term[]; steps: MinimizationStep[] } {
  // Конвертація в бінарний формат
  const terms = [...minterms, ...dontCares].map(m => ({
    binary: m.toString(2).padStart(variables, '0'),
    minterms: [m],
    isPrime: true,
    isEssential: false
  }));

  // Групування за кількістю одиниць
  const groups = new Map<number, Term[]>();
  terms.forEach(term => {
    const ones = term.binary.split('1').length - 1;
    if (!groups.has(ones)) groups.set(ones, []);
    groups.get(ones)!.push(term);
  });

  // Пошук простих імплікант
  const primeImplicants = findPrimeImplicants(groups);

  // Створення таблиці покриття
  const coverageTable = createCoverageTable(primeImplicants, minterms);
```

```
  // Визначення суттєвих простих імплікант
  const essentialImplicants = findEssentialImplicants(coverageTable);

  return {
    terms: essentialImplicants,
    steps: generateSteps(groups, primeImplicants, essentialImplicants)
  };
}

function findPrimeImplicants(groups: Map<number, Term[]>): Term[] {
  let primeImplicants: Term[] = [];
  let currentGroups = groups;

  while (true) {
    const nextGroups = new Map<number, Term[]>();
    let foundNew = false;

    // Спроба об'єднати терми з сусідніх груп
    for (let i = 0; i < currentGroups.size - 1; i++) {
      const group1 = currentGroups.get(i) || [];
      const group2 = currentGroups.get(i + 1) || [];

      for (const term1 of group1) {
        for (const term2 of group2) {
          const combined = combinePair(term1, term2);
          if (combined) {
            foundNew = true;
            const ones = combined.binary.split('1').length - 1;
            if (!nextGroups.has(ones)) {
              nextGroups.set(ones, []);
            }
            nextGroups.get(ones)!.push(combined);
          }
        }
      }
    }

    // Додавання непоєднаних термів до простих імплікант
```

```
    for (const group of currentGroups.values()) {
      for (const term of group) {
        if (term.isPrime) {
          primeImplicants.push(term);
        }
      }
    }

    if (!foundNew) break;
    currentGroups = nextGroups;
  }

  return primeImplicants;
}

function combinePair(term1: Term, term2: Term): Term | null {
  let diffCount = 0;
  let diffPos = -1;

  for (let i = 0; i < term1.binary.length; i++) {
    if (term1.binary[i] !== term2.binary[i]) {
      diffCount++;
      diffPos = i;
    }
  }

  if (diffCount !== 1) return null;

  const newBinary = term1.binary.split('');
  newBinary[diffPos] = '-';

  return {
    binary: newBinary.join(''),
    minterms: [...term1.minterms, ...term2.minterms],
    isPrime: true,
    isEssential: false
  };
}
```

```

A.2. Компонент карти Карно (KarnaughMap.tsx)
---
```typescript
import React from 'react';
import { generateGrayCode } from '../utils/grayCode';

interface KarnaughMapProps {
  truthTable: boolean[];
  variables: string[];
  highlighted?: {
    startX: number;
    startY: number;
    width: number;
    height: number;
    color: string;
  }[];
}

export const KarnaughMap: React.FC<KarnaughMapProps> = ({
  truthTable,
  variables,
  highlighted = []
}) => {
  const size = Math.sqrt(truthTable.length);
  const grayCode = generateGrayCode(Math.log2(size));
  const cellSize = 40;

  const renderCell = (value: boolean, x: number, y: number) => {
    return (
      <div
        key={`${x}-${y}`}
        className={`cell ${value ? 'active' : ''}`}
        style={{
          width: cellSize,
          height: cellSize,
          border: '1px solid #ccc',
```
```

```
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'center',
        backgroundColor: value ? '#e3f2fd' : 'white'
      }}
    >
      {value ? '1' : '0'}
    </div>
  );
};

const renderLabels = () => {
  const rowLabels = grayCode.map(code =>
    variables
      .slice(0, Math.log2(size))
      .map((v, i) => (code[i] ? v : `¬${v}`))
      .join('')
  );

  const colLabels = grayCode.map(code =>
    variables
      .slice(Math.log2(size))
      .map((v, i) => (code[i] ? v : `¬${v}`))
      .join('')
  );

  return { rowLabels, colLabels };
};

const { rowLabels, colLabels } = renderLabels();

return (
  <div className="karnaugh-map" style={{ padding: '20px' }}>
    <div className="grid" style={{
      display: 'grid',
      gridTemplateColumns: `auto repeat(${size}, ${cellSize}px)`,
      gap: '2px'
    }}>
```

```
 {/* Верхні мітки */}
 <div style={{ width: cellSize, height: cellSize }}></div>
 {colLabels.map((label, i) => (
  <div key={`col-${i}`} className="label" style={{
   width: cellSize,
   height: cellSize,
   display: 'flex',
   alignItems: 'center',
   justifyContent: 'center',
   fontWeight: 'bold'
  }}>
   {label}
  </div>
 ))}

 {/* Рядки з мітками та значеннями */}
 {rowLabels.map((label, i) => (
  <React.Fragment key={`row-${i}`}>
   <div className="label" style={{
    width: cellSize,
    height: cellSize,
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'center',
    fontWeight: 'bold'
   }}>
    {label}
   </div>
   {Array(size).fill(null).map((_, j) => (
    renderCell(truthTable[i * size + j], j, i)
   ))}
  </React.Fragment>
 ))}
</div>

{/* Підсвічені області */}
{highlighted.map((area, index) => (
 <div
```

```
          key={`highlight-${index}`}
          style={{
            position: 'absolute',
            left: `${area.startX * cellSize}px`,
            top: `${area.startY * cellSize}px`,
            width: `${area.width * cellSize}px`,
            height: `${area.height * cellSize}px`,
            backgroundColor: `${area.color}33`,
            border: `2px solid ${area.color}`,
            pointerEvents: 'none'
          }}
        />
      ))}
    </div>
  );
};
```

А.3. Компонент введення формул (FormulaKeyboard.tsx)

```typescript
import React from 'react';

interface FormulaKeyboardProps {
  onInput: (value: string) => void;
  variables: string[];
}

export const FormulaKeyboard: React.FC<FormulaKeyboardProps> = ({
  onInput,
  variables
}) => {
  const symbols = [
    { label: "∧", value: "∧", description: "Кон'юнкція (І)" },
    { label: "∨", value: "∨", description: "Диз'юнкція (АБО)" },
    { label: "¬", value: "¬", description: "Заперечення (НЕ)" },
    { label: "→", value: "→", description: "Імплікація" },
    { label: "(", value: "(", description: "Дужка відкриваюча" },
```

```
  { label: ")", value: ")", description: "Дужка закриваюча" }
];

return (
  <div className="formula-keyboard">
    <div className="mb-4">
      <h3 className="text-sm font-medium mb-2">Змінні:</h3>
      <div className="flex flex-wrap gap-2">
        {variables.map((variable) => (
          <button
            key={variable}
            onClick={() => onInput(variable)}
            className="px-3 py-1 bg-blue-100 text-blue-800 rounded hover:bg-blue-200"
            title={`Змінна ${variable}`}
          >
            {variable}
          </button>
        ))}
      </div>
    </div>

    <div className="mb-4">
      <h3 className="text-sm font-medium mb-2">Операції:</h3>
      <div className="flex flex-wrap gap-2">
        {symbols.map((symbol) => (
          <button
            key={symbol.value}
            onClick={() => onInput(symbol.value)}
            className="px-3 py-1 bg-gray-100 text-gray-800 rounded hover:bg-gray-200"
            title={symbol.description}
          >
            {symbol.label}
          </button>
        ))}
      </div>
    </div>

    <div className="text-sm text-gray-600">
```

```
        <h3 className="font-medium mb-2">Підказки:</h3>
        <ul className="list-disc list-inside space-y-1">
          <li>Використовуйте дужки для групування виразів</li>
          <li>Заперечення змінної записується як ¬x</li>
          <li>Приклад виразу: (x₁ ∧ ¬x₂ ) ∨ x₃ </li>
        </ul>
      </div>
    </div>
  );
};
```

А.4. Компонент таблиці істинності (TruthTable.tsx)

```typescript
import React, { useState, useEffect } from 'react';

interface TruthTableProps {
  variables: string[];
  onChange: (values: boolean[]) => void;
  initialValues?: boolean[];
}

export const TruthTable: React.FC<TruthTableProps> = ({
  variables,
  onChange,
  initialValues
}) => {
  const [values, setValues] = useState<boolean[]>(
    initialValues || Array(Math.pow(2, variables.length)).fill(false)
  );

  useEffect(() => {
    if (initialValues) {
      setValues(initialValues);
    }
  }, [initialValues]);
```

```
const handleToggle = (index: number) => {
  const newValues = [...values];
  newValues[index] = !newValues[index];
  setValues(newValues);
  onChange(newValues);
};

const getBinaryRepresentation = (index: number): string[] => {
  const binary = index.toString(2).padStart(variables.length, '0');
  return binary.split('');
};

return (
  <div className="overflow-x-auto">
    <table className="min-w-full table-auto">
      <thead>
        <tr>
          {variables.map((variable) => (
            <th
              key={variable}
              className="px-4 py-2 text-center border-b"
            >
              {variable}
            </th>
          ))}
          <th className="px-4 py-2 text-center border-b">F</th>
        </tr>
      </thead>
      <tbody>
        {values.map((value, index) => (
          <tr key={index}>
            {getBinaryRepresentation(index).map((bit, bitIndex) => (
              <td
                key={bitIndex}
                className="px-4 py-2 text-center border-b"
              >
                {bit}
              </td>
```

```
            ))}
            <td className="px-4 py-2 text-center border-b">
              <button
                onClick={() => handleToggle(index)}
                className={`w-8 h-8 rounded ${
                  value
                    ? 'bg-blue-500 text-white'
                    : 'bg-gray-200 text-gray-700'
                }`}
              >
                {value ? '1' : '0'}
              </button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
 );
};
```

A.5. Типи даних (types.ts)

```typescript
export interface Term {
  binary: string;
  minterms: number[];
  isPrime: boolean;
  isEssential: boolean;
}

export interface MinimizationStep {
  type: 'grouping' | 'combining' | 'essential' | 'final';
  terms: Term[];
  explanation: string;
  groups?: Term[][];
}
```

```
export interface Task {
  id: number;
  title: string;
  description: string;
  type: "expression" | "map" | "circuit" | "minimize";
  variables: string[];
  correctAnswer: string;
  hints: string[];
  mapData?: {
    values: boolean[];
    highlighted?: {
      startX: number;
      startY: number;
      width: number;
      height: number;
      color: string;
    }[];
  };
}

export interface Progress {
  completedTasks: number[];
  currentLevel: number;
  score: number;
  hintsUsed: Map<number, number>;
  attempts: Map<number, number>;
}
```

А.6. Утиліти для генерації кодів Грея (grayCode.ts)

```typescript
export function generateGrayCode(bits: number): string[] {
  if (bits <= 0) return [''];

  // Генерація кодів Грея для bits-1
  const prevCodes = generateGrayCode(bits - 1);
```

```
    // Додавання 0 спереду до першої половини
    const result: string[] = prevCodes.map(code => '0' + code);

    // Додавання 1 спереду до другої половини (в зворотному порядку)
    for (let i = prevCodes.length - 1; i >= 0; i--) {
      result.push('1' + prevCodes[i]);
    }

    return result;
  }

  export function binaryToGray(binary: string): string {
    let gray = binary[0];
    for (let i = 1; i < binary.length; i++) {
      gray += (parseInt(binary[i-1]) ^ parseInt(binary[i])).toString();
    }
    return gray;
  }

  export function grayToBinary(gray: string): string {
    let binary = gray[0];
    for (let i = 1; i < gray.length; i++) {
      binary += (parseInt(binary[i-1]) ^ parseInt(gray[i])).toString();
    }
    return binary;
  }
```