

Метод карт Карно



Цель

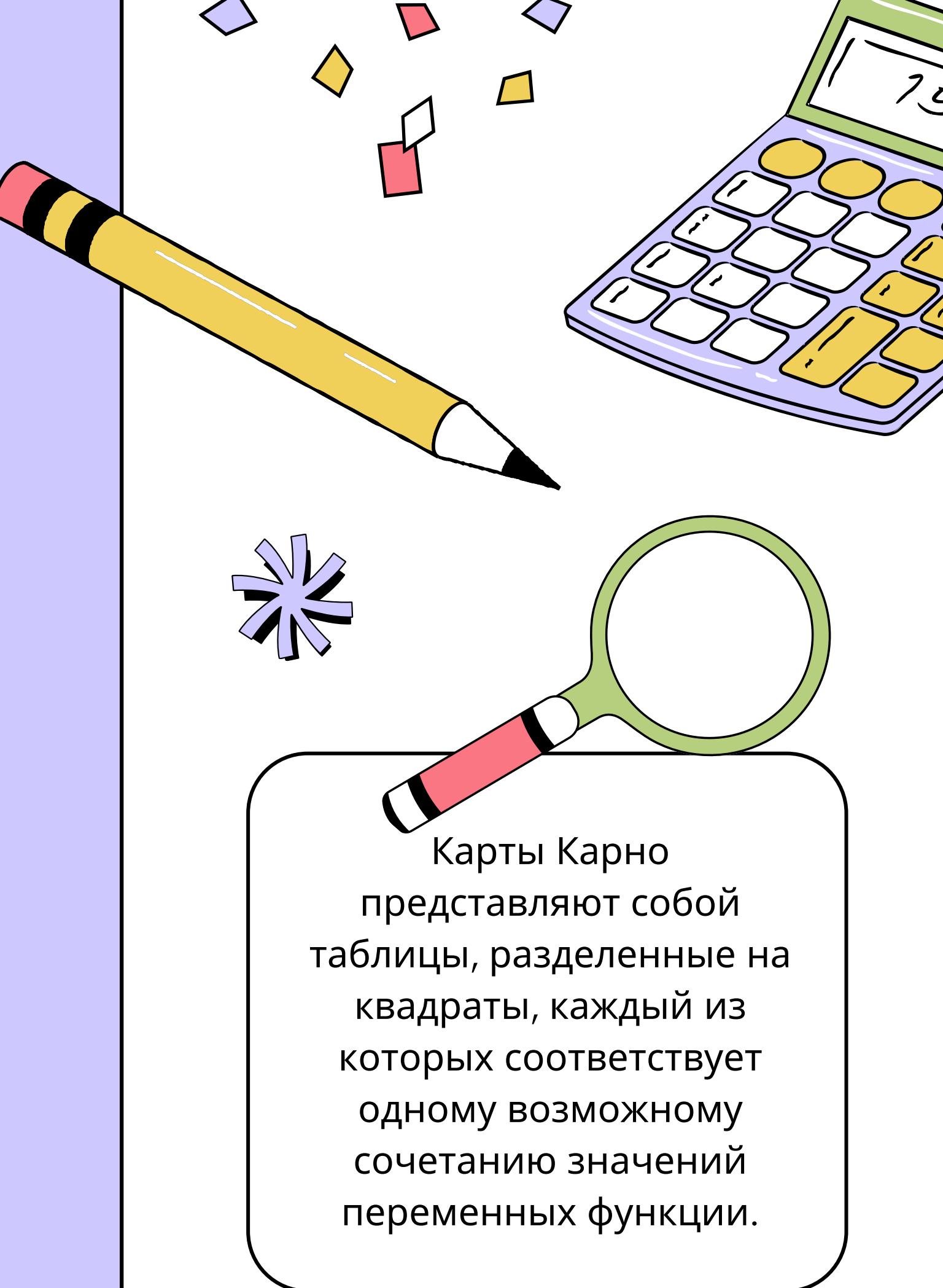
Разработать и предложить инструмент в виде телеграмм бота, интегрированного с методом карт Карно, для оптимизации булевых функций в цифровой логике. Это позволит улучшить процесс проектирования цифровых систем, делая его более удобным и эффективным.

A Karnaugh map for a three-variable function F with variables x_3 and x_4 . The columns represent x_3x_4 values: 00, 01, 11, 10. The rows represent x_1 values: 0, 1, 1, 0. The output F is shown in the top-left corner. The outputs S_1 , S_2 , and S_3 are indicated by red and blue lines connecting specific cells to the right. A green box highlights the cell for $x_1=1$ and $x_3=1$.

		$x_3 x_4$			
F		00	01	11	10
x_1	00	1	0	0	1
	01	1	0	0	1
11	0	1	1	0	
10	1	0	0	1	

Задачи

- 1 Изучение метода карт Карно
- 2 Рассмотреть применения метода карт Карно
- 3 Разработка кода с использованием карт Карно



Карты Карно представляют собой таблицы, разделенные на квадраты, каждый из которых соответствует одному возможному сочетанию значений переменных функции.

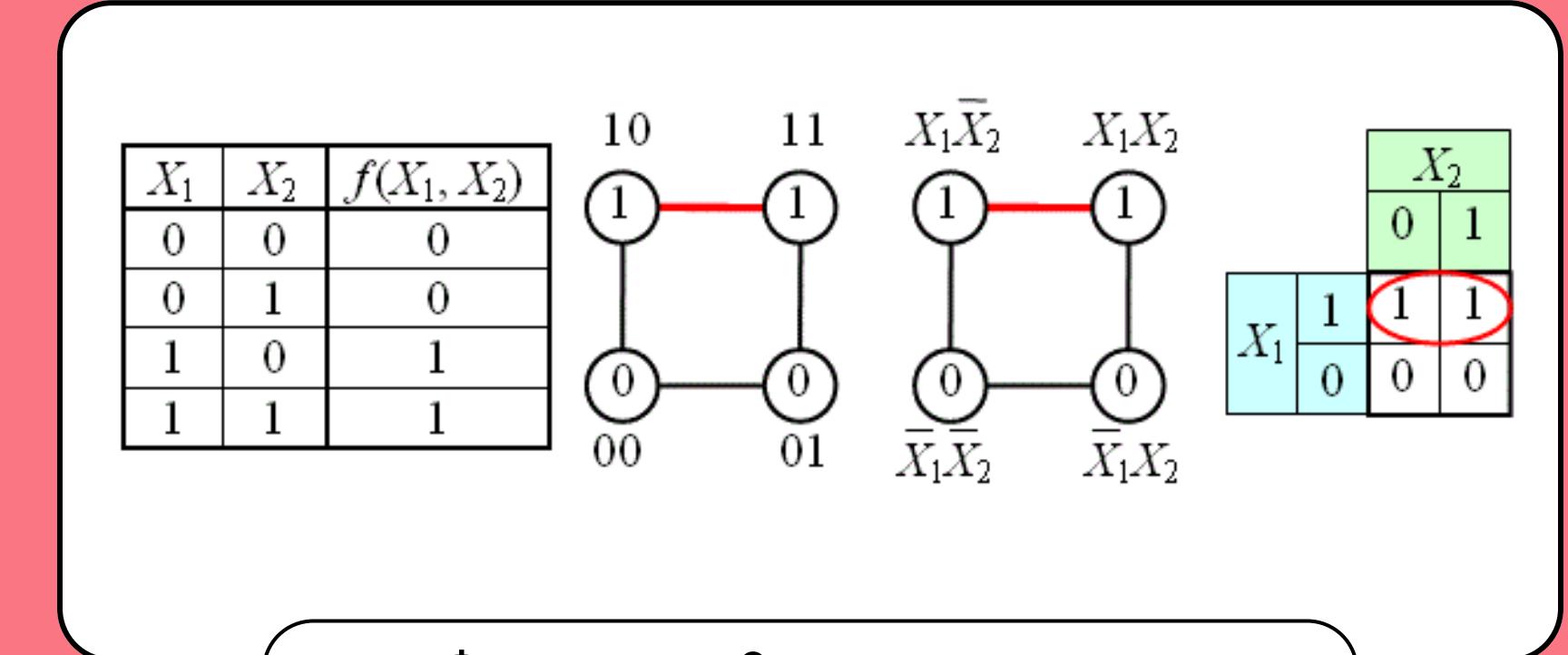


Актуальность

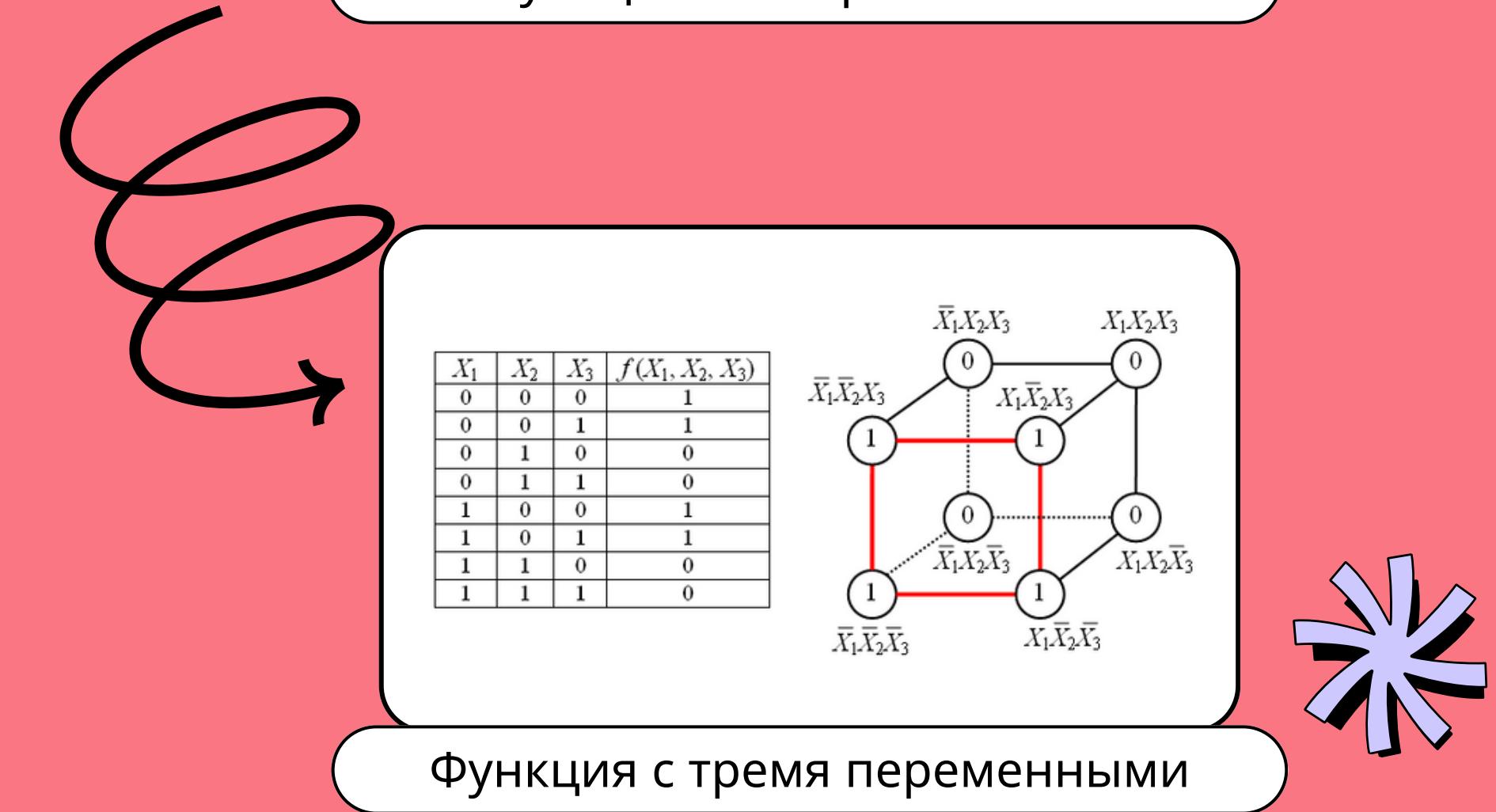
Метод карт Карно остается важным инструментом в области цифровой логики, несмотря на развитие современных методов проектирования. Его применение позволяет эффективно анализировать и упрощать сложные булевые функции, что актуально для разработки высокопроизводительных и надежных цифровых устройств. Обучение этому методу позволит инженерам и разработчикам повысить свои навыки и успешно решать задачи в области цифровой электроники и информационных технологий.

Введение в метод КарноКарно

Метод карт Карно, разработанный Анри Жюлем Карно в 1854 году, является ключевым инструментом анализа и упрощения булевых функций в цифровой логике. Он заключается в представлении всех возможных комбинаций значений переменных булевой функции в виде таблицы, что упрощает анализ и упрощение сложных логических выражений.

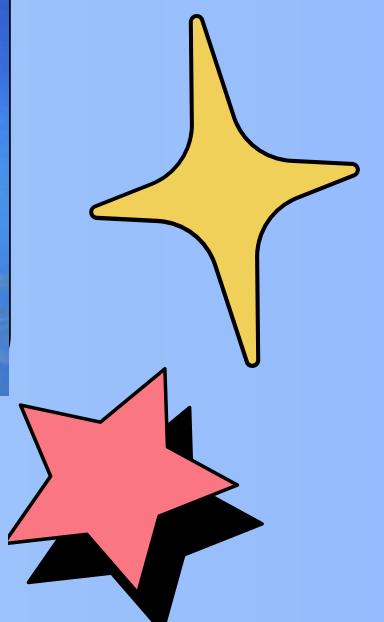
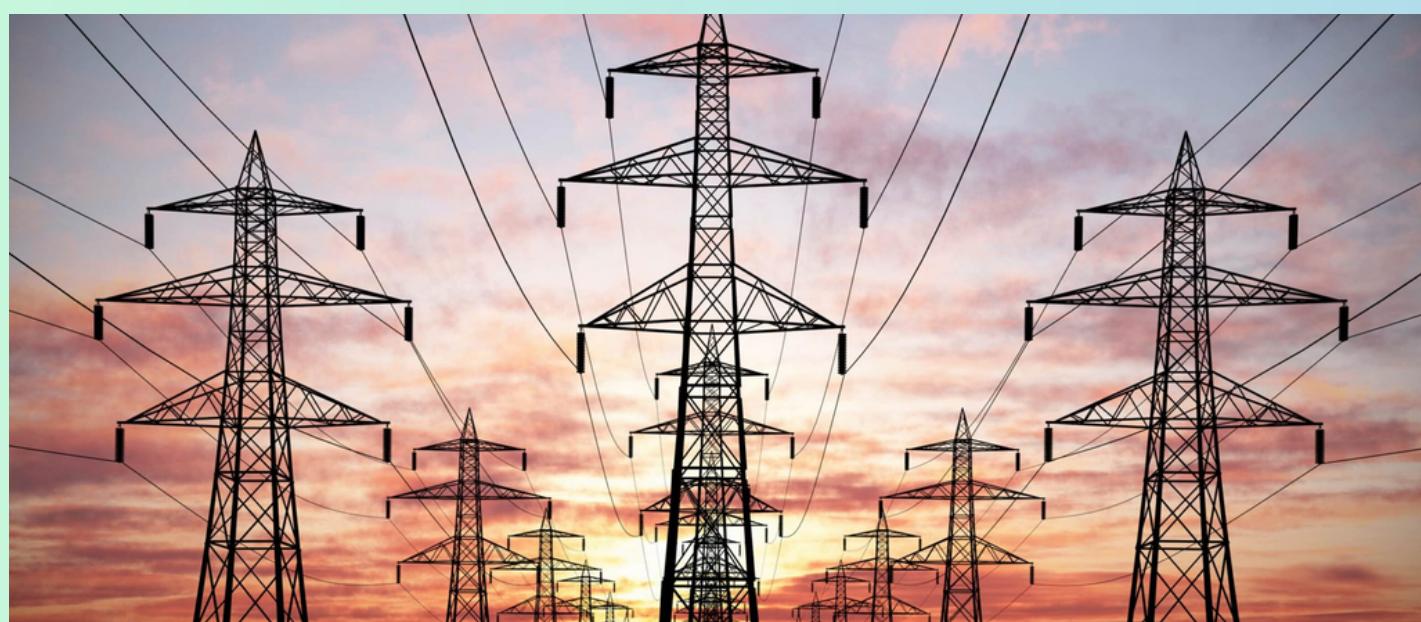


Функция с 2 переменными



Функция с тремя переменными

Применение



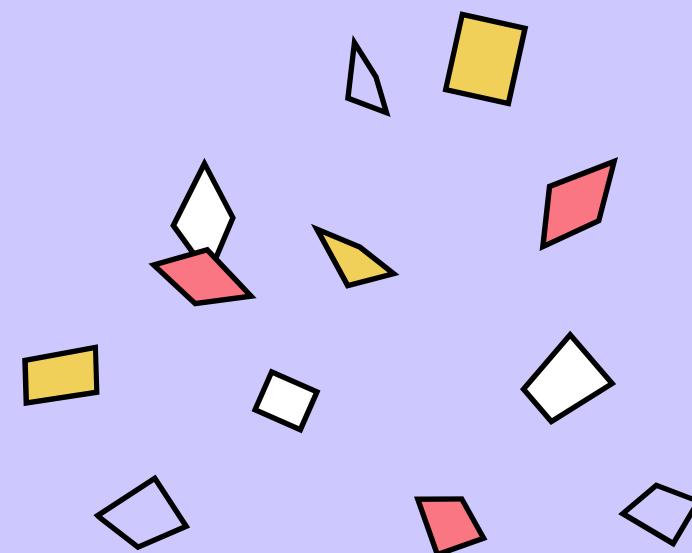
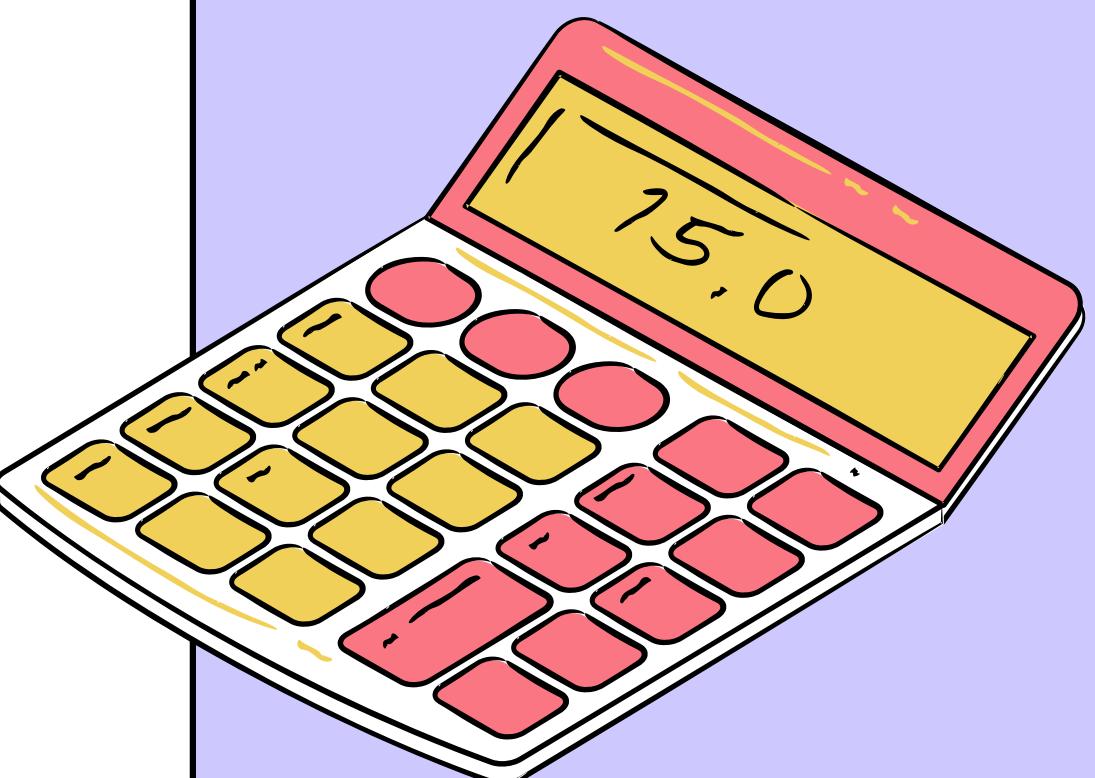
Оптимизация

Оптимизация с картами Карно - упрощение булевых функций путём группировки минтермов, что помогает выявить общие части функции и уменьшить количество необходимых логических элементов.

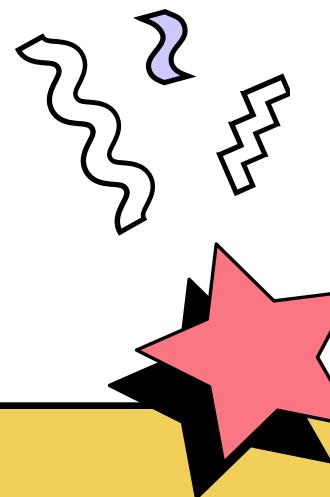
$$4 \times 1 = 4$$

$$23 \times 1 = 23$$

$$105 \times 1 = 105$$



Алгоритм



01

Начните с таблицы истинности для вашей логической функции.

02

Создайте таблицу Карно с количеством строк и столбцов, равным 2 в степени числа переменных.

03

Запишем результаты вашей функции в ячейки этой таблицы.

04

Обведите соседние единицы в таблице Карно прямоугольниками.

05

Попробуйте объединить прямоугольники, чтобы получить минимальное количество больших прямоугольников.

06

Каждый большой прямоугольник будет соответствовать одному слагаемому в минимизированной функции.

07

Запишите минимизированную функцию, используя слагаемые из больших прямоугольников.

08

Проверьте упрощенное выражение с помощью исходной таблицы.

Алгоритм

```
def simplify_kmap(values):
    simplified_expression = []

    # Проверка на полное покрытие и полное отсутствие
    if all(v == 1 for v in values):
        return "1"
    if all(v == 0 for v in values):
        return "0"
```

```
# Функция для проверки пар и добавления упрощений
def check_pairs():
    # Проверка горизонтальных и вертикальных пар
    if values[0] == values[1] == 1:
        simplified_expression.append("NOT A")
    if values[2] == values[3] == 1:
        simplified_expression.append("A")
    if values[0] == values[2] == 1:
        simplified_expression.append("NOT B")
    if values[1] == values[3] == 1:
        simplified_expression.append("B")

# Проверка диагональных пар для дополнительного упрощения
def check_diagonals():
    if values[0] == values[3] == 1 and not (values[1] or values[2]):
        simplified_expression.append("(A XOR B)")
    if values[1] == values[2] == 1 and not (values[0] or values[3]):
        simplified_expression.append("NOT (A XOR B)")

# Проверка одиночных единиц
def check_singles():
    if values.count(1) == 1:
        index = values.index(1)
        # Добавляем условия в зависимости от положения единицы
        if index == 0:
            simplified_expression.append("NOT A AND NOT B")
        elif index == 1:
            simplified_expression.append("NOT A AND B")
        elif index == 2:
            simplified_expression.append("A AND NOT B")
        elif index == 3:
            simplified_expression.append("A AND B")

check_pairs()
check_diagonals()
check_singles()
```

Заключение

