Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**Лабораторная работа**

**Тема:** Решение задачи о 8 ферзях

Семестр: 2

Выполнил студент ИВТ-23-2б:

Злыгостев Денис Николаевич

(дата, подпись)

Проверила доцент кафедры ИТАС:

Полякова Ольга Андреевна

(дата, подпись)

2024 г.

**Постановка задачи**

Создать программу, которая располагает ферзи в таком положении, чтобы они не били друг друга

**Анализ задачи**

1. Ферзь может ходить в любом направлении по горизонтали, вертикали, диагонали и на любое количество клеток, рубит он так же, как ходит.
2. Чтобы ферзи друг друга не «били», на каждой строке, диагонали и каждом столбце должен находиться один ферзь.
3. Для расстановки ферзей требуется:
4. Поставить первого ферзя на позицию а1 (первая клетка первой строки).
5. Перейти на следующую строку и поставить ферзя так, чтобы первый ферзь его не бил.
6. Если на какой-либо строке поставить ферзя невозможно(так, чтобы они не «били» друг друга), то возвращаемся на предыдущую строку и ставим ферзя на следующую клетку строки.
7. Повторяем пункты 2 и 3, пока не расставим всех ферзей.

**Подробный разбор**

Задача о восьми ферзях является классической задачей комбинаторной оптимизации, которая в своем простейшем виде заключается в расстановке восеми ферзей на шахматной доске 8x8 таким образом, чтобы они не находились под угрозой взятия друг друга. Под "угрозой взятия" понимается ситуация, когда два ферзя находятся на одной горизонтали, вертикали или диагонали.

1. Постановка задачи: Необходимо найти все возможные способы расстановки восеми ферзей таким образом, чтобы они не били друг друга.

2. Моделирование задачи: В задаче рассматривается доска размером 8x8, где позиции ферзей представляются парой (строка, столбец). Для поиска решения используется алгоритм обратной трассировки (backtracking), который позволяет эффективно итеративно перебирать все возможные варианты расстановки.

3. Подход к решению: В решении задачи строится рекурсивная функция solveNQueens, которая пытается разместить ферзя на каждой строке доски так, чтобы он не находился под угрозой других ферзей. При этом используется функция isSafe для проверки безопасности размещения на текущей позиции.

4. Алгоритм:

- Начиная с первой строки, для каждого столбца в текущей строке проверяется возможность размещения ферзя.

- Если размещение безопасно, ферзь помещается в это поле и рекурсивно вызывается solveNQueens для следующей строки.

- Если ни одно поле в строке не подходит, происходит возврат назад (backtracking).

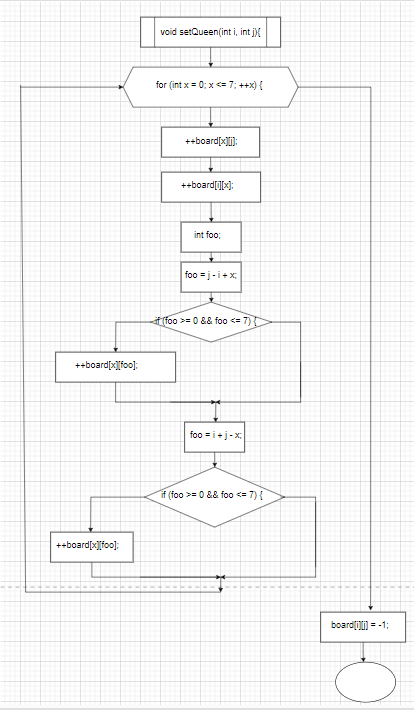
- После завершения всех рекурсивных вызовов, получаем все возможные решения.

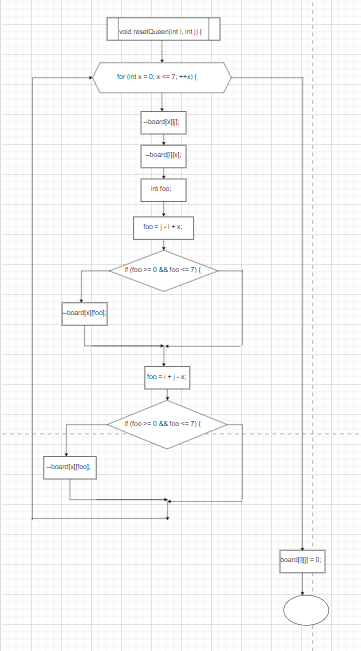
5. Вывод результатов: После нахождения всех решений, программа выводит на экран каждое расположение ферзей на доске, где Q обозначает ферзя, а точка - пустую клетку.

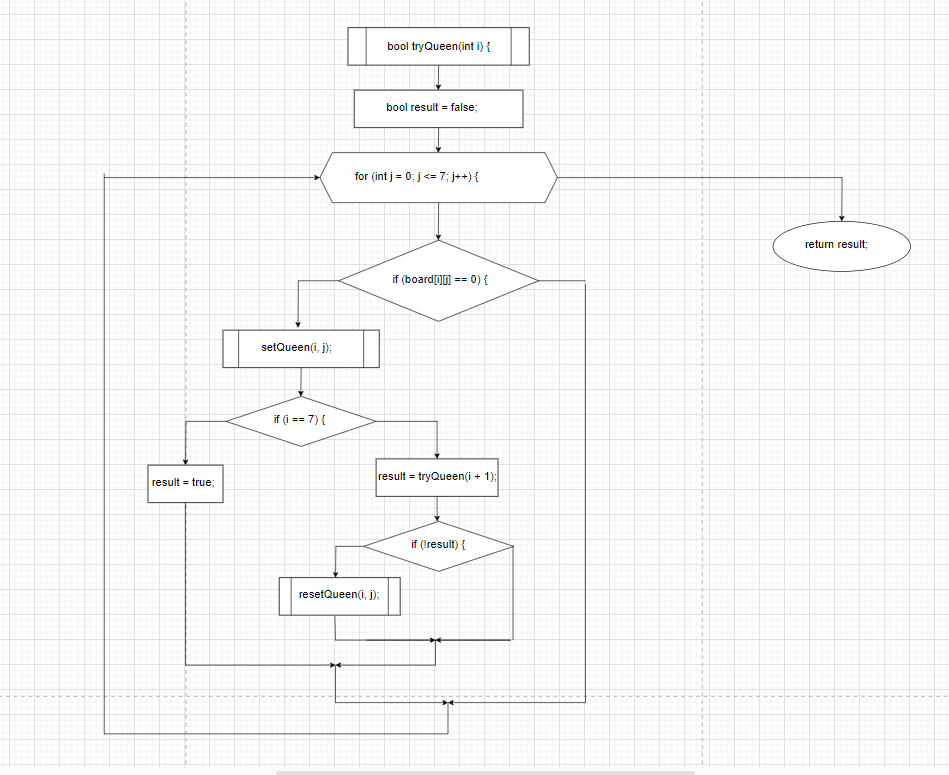
6. Сложность: В худшем случае сложность данного алгоритма составляет примерно O(n!), где n - количество ферзей. Это связано с экспоненциальным ростом вариантов размещения ферзей на доске.

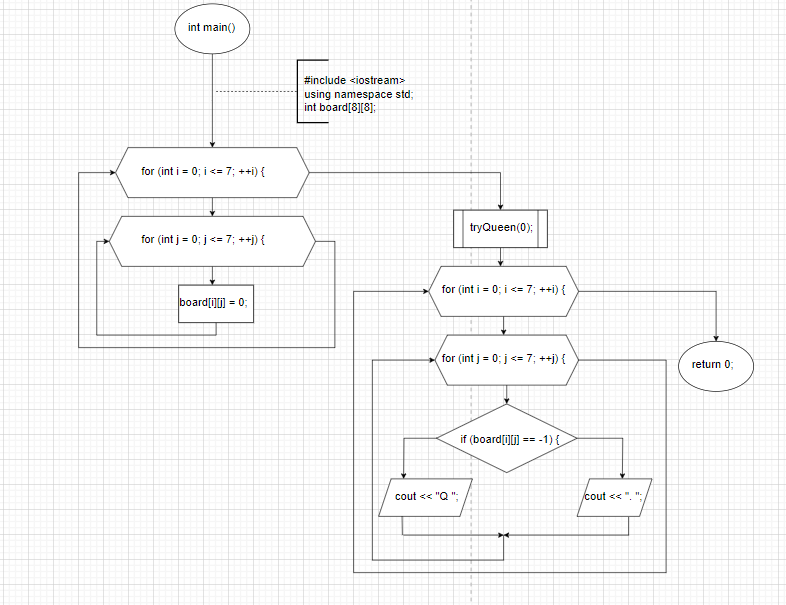
Таким образом, решение задачи о восьми ферзях требует применения алгоритма обратной трассировки и является классическим примером комбинаторной задачи, которая может быть эффективно решена при помощи правильно спроектированного рекурсивного алгоритма.

**Блок схема**









**Код**

﻿#include <iostream>

#include <vector>

using namespace std;

const int N = 8; // Размер шахматной доски

vector<vector<int>> solutions;

// Функция для проверки, можно ли поставить ферзя на данную позицию

bool isSafe(vector<int>& board, int row, int col) {

for (int i = 0; i < row; ++i) {

if (board[i] == col || abs(board[i] - col) == abs(i - row)) {

return false;

}

}

return true;

}

// Рекурсивная функция для размещения ферзей

void solveNQueens(vector<int>& board, int row) {

if (row == N) {

solutions.push\_back(board);

return;

}

for (int col = 0; col < N; ++col) {

if (isSafe(board, row, col)) {

board[row] = col;

solveNQueens(board, row + 1);

}

}

}

// Функция для вывода решений

void printSolutions() {

int count = 1;

for (vector<int>& solution : solutions) {

cout << "Решение " << count << ":\n";

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

if (solution[i] == j) {

cout << "Q ";

}

else {

cout << ". ";

}

}

cout << "\n";

}

cout << "\n";

count++;

}

}

int main() {

setlocale(LC\_ALL, "RU");

vector<int> board(N, 0); // Создание доски

solveNQueens(board, 0); // Нахождение всех решений

printSolutions(); // Вывод результатов

return 0;

}

**Скрин из GitHub**

