Общие сведения

Программа, реализованная на языке программирования Python, решает задачу о размещении ферзей на шахматной доске с использованием бинарных диаграмм решений (BDD). BDD - это структура данных, которая позволяет компактно представлять и анализировать логические функции, в данном случае, ограничения на размещение ферзей на шахматной доске.

Программа позволяет пользователю ввести размер шахматной доски, а затем строит BDD, которая представляет все возможные решения, удовлетворяющие ограничениям задачи. После построения BDD программа выводит на экран несколько решений, представленных в виде шахматных досок с размещенными ферзями.

Ограничения

- Максимальный размер шахматной доски, который может обрабатывать программа, ограничен ресурсами компьютера, такими как оперативная память и процессорная мощность.
- Программа выводит на экран только несколько решений, так как количество возможных решений может быть очень большим.

Требования к аппаратному обеспечению

- Для выполнения программы требуется компьютер с достаточным объемом оперативной памяти для хранения BDD и процессором, способным эффективно обрабатывать рекурсивные структуры данных.
- Ресурсоемкость программы зависит от размера шахматной доски, так как количество переменных в BDD растет экспоненциально с увеличением размера доски.

Структура программного обеспечения

Программа написана на языке программирования Python, который был выбран за свою простоту, выразительность и наличие богатой экосистемы библиотек.

Основные компоненты:

- **Класс `BDD`:** Управляет построением и упрощением BDD.
- **Функция `create_queen_bdd`:** Создает BDD, которая представляет все возможные решения задачи о размещении ферзей.

- **Функция `print_solutions`:** Выводит на экран решения, представленные в виде шахматных досок.
- **Функция `main`:** Точка входа в программу, которая получает входные данные от пользователя, строит BDD и выводит результат.

Алгоритм работы программы

1. **Инициализация:**
- Пользователь вводит размер шахматной доски (целое число).
- Создается экземпляр класса BDD для управления построением и упрощением BDD.
- Вызывается функция `create_queen_bdd`, которая создает BDD, представляющую ограничения на размещение ферзей.
1. **Построение BDD:**
- В функции `create_queen_bdd` BDD строится рекурсивно, добавляя ограничения для каждой клетки шахматной доски.
- BDD включает в себя ограничения, которые гарантируют, что на каждой строке, каждом столбце и каждой диагонали шахматной доски может находиться не более одной ферзи.
1. **Вывод решений:**
- После построения BDD вызывается функция `print_solutions`, которая выводит на экран несколько найденных решений.
- Каждое решение представляется в виде шахматной доски с размещенными ферзями.

Используемые методы

Класс `BDD`:

- `declare(self, *vars)`: Объявляет переменные BDD.
- `add_expr(self, expr)`: Добавляет логическое выражение к BDD.
- `pick_iter(self, model)`: Возвращает итератор по всем решениям BDD, удовлетворяющим заданной модели.

Функция `create_queen_bdd`:

- `exactly_one(vars)`: Создает логическое выражение, которое гарантирует, что из списка переменных BDD ровно одна будет истинной.
- `create_queen_bdd(n)`: Создает BDD для задачи о размещении ферзей на шахматной доске размером n x n.

Функция `print_solutions`:

- `print_solutions(bdd, bdd_formula, n, max_solutions=10)`: Выводит на экран первые `max_solutions` решений задачи о размещении ферзей.

Структура программы с описанием функций составных частей

Класс BDD

from dd.autoref import BDD

Класс BDD импортируется из библиотеки dd, которая реализует бинарные диаграммы решений (BDD). Этот класс будет использоваться для хранения и манипулирования диаграммами решений.

Функция `create_queen_bdd(n)`

```
def create_queen_bdd(n):
    # Инициализация BDD и объявление переменных
    bdd = BDD()
    bdd.declare(*['Q_{}_{}'.format(i, j) for i in range(n) for j in range(n)])
    def exactly_one(vars):
        # Формирование условия, что ровно одна переменная из списка равна True
        return bdd.add_expr("({}) & {}".format(' | '.join(vars), ' & '.join(
            ['~({} & {})'.format(v1, v2) for i, v1 in enumerate(vars) for v2 in vars[i + 1:]])))
    # Начальная формула
    bdd_formula = bdd.true
    # Ограничение: по одной ферзи в строке
    for i in range(n):
        row_vars = ['Q_{}_{}'.format(i, j) for j in range(n)]
        bdd_formula &= exactly_one(row_vars)

    # Ограничение: по одной ферзи в столбце
    for j in range(n):
        col_vars = ['Q_{}_{}'.format(i, j) for i in range(n)]
```

```
    bdd_formula &= exactly_one(col_vars)


  # Ограничение: ферзи не могут находиться на одной диагонали
  for i in range(n):
    for j in range(n):
      for k in range(n):
        for l in range(n):
          if abs(i - k) == abs(j - l) and (i != k or j != l):
            bdd_formula &= bdd.add_expr('~Q_{}_{} | ~Q_{}_{}'.format(i, j, k, l))
  return bdd, bdd_formula
```

create_queen_bdd(n): Эта функция строит бинарную диаграмму решений (BDD) для задачи о размещении ферзей на доске размера n x n.

- **Инициализация BDD**: Создается объект BDD, и переменные (клетки доски) объявляются.
- **Функция exactly_one(vars)**: Формирует подформулу, которая гарантирует, что в каждой строке или столбце присутствует только одна ферзь.
- **Диагональные ограничения**: Добавляется ограничение, что ферзи не могут находиться на одной диагонали.

Функция print_solutions(bdd, bdd_formula, n, max_solutions=10)

```
def print_solutions(bdd, bdd_formula, n, max_solutions=10):
  solutions = bdd.pick_iter(bdd_formula)
  count = 0
  for sol in solutions:
    if count >= max_solutions:
      break
    board = [['.' for _ in range(n)] for _ in range(n)]
    for var, val in sol.items():
      if val:
        i, j = map(int, var[2:].split('_'))
```

```
            board[i][j] = 'Q'
        for row in board:
            print(' '.join(row))
        print('')
        count += 1
    if count == 0:
        print("No solutions found")
```

print_solutions(bdd, bdd_formula, n, max_solutions=10): Эта функция выводит до max_solutions решений задачи. Решения выбираются с помощью итератора по возможным значениям, которые удовлетворяют ограничениям.

- **Визуализация доски**: Решение отображается в виде шахматной доски, где ферзи помечены символом 'Q', а пустые клетки — '.'.

Функция main()

```
def main():
    while True:
        try:
            if sys.version_info[0] < 3:
                n = int(raw_input("Введите размер шахматной доски (целое число): "))
            else:
                n = int(input("Введите размер шахматной доски (целое число): "))
            if n <= 0:
                raise ValueError("Размер доски должен быть положительным целым числом.")
            break
        except ValueError as e:
            print("Ошибка: ", e)

    bdd, bdd_formula = create_queen_bdd(n)
    print_solutions(bdd, bdd_formula, n, max_solutions=4)
```

main(): Главная функция программы.

- Запрашивает размер доски у пользователя и вызывает функции для создания BDD и вывода решений.

Функция main

```
def main():
    print("Добро пожаловать в программу решения задачи размещения ферзей на шахматной доске с помощью BDD!")

    # Ввод размера доски
    while True:
        try:
            if sys.version_info[0] < 3:
                n = int(raw_input("Введите размер шахматной доски (целое число): "))
            else:
                n = int(input("Введите размер шахматной доски (целое число): "))
            if n <= 0:
                raise ValueError("Размер доски должен быть положительным целым числом.")
            break
        except ValueError as e:
            print(e)
```

main: Функция приветствует пользователя и запрашивает размер шахматной доски. Цикл продолжается до тех пор, пока не будет введено корректное положительное число.

Вызов функций для создания и вывода решений:

```
    # Построение BDD для задачи о ферзях
    bdd, bdd_formula = create_queen_bdd(n)

    # Вывод решений
    print("\nНайденные решения:")
    print_solutions(bdd, bdd_formula, n, max_solutions=4)
```

bdd, bdd_formula: Функция создает бинарную диаграмму решений (BDD) для задачи о размещении ферзей с помощью функции create_queen_bdd(n). Далее вызывается функция print_solutions(bdd, bdd_formula, n, max_solutions=4), которая выводит до 4 решений.

Завершение

```
if __name__ == "__main__":
    main()
```

Запуск программы: Функция main() будет запущена при запуске скрипта.

Методика тестирования

Для тестирования программы, решающей задачу размещения ферзей на шахматной доске с использованием бинарной диаграммы решений (BDD), была применена следующая методика:

Подготовка тестовых данных:

- **Разные размеры доски:** Были выбраны шахматные доски разных размеров — 4x4, 5x5, 6x6 и 8x8.
- **Проверка правильности:** Для каждого размера доски ожидалось нахождение правильных решений задачи о размещении ферзей, где ни один ферзь не угрожает другому.
- **Тестовые сценарии:** Были протестированы как корректные решения (для меньших досок с ограниченным количеством решений), так и большие доски, где решений больше.

Запуск программы на тестовых данных:

- **Запуск для каждого размера доски:** Программа запускалась для каждого выбранного размера доски. Пользователь вводит размер, и программа строит и упрощает бинарную диаграмму решений для задачи о размещении ферзей.
- **Проверка корректности решения:** Для каждого размера доски было проверено корректное построение BDD.

Анализ результатов:

- **Эквивалентность решений:** Для каждого теста проверялось, что BDD, построенная программой, соответствует возможным решениям задачи размещения ферзей.

Примеры тестовых данных

Доска 4x4:

- **Размер доски:** 4x4
- **Ожидаемый результат:** 2 решения, где каждый ферзь размещен таким образом, чтобы не угрожать другим ферзям.

Результаты тестирования

Для доски размером 4x4 программа должна вывести следующие два корректных решения размещения ферзей:

Первое решение:

. . Q .

Q . . .

. . . Q

. Q . .

Второе решение:

. Q . .

. . . Q

Q . . .

. . Q .

Эти схемы показывают, как расставить 4 ферзя на доске 4x4 так, чтобы ни один ферзь не угрожал другим.

Заключение

В процессе разработки и тестирования программы для решения задачи размещения ферзей на шахматной доске с использованием бинарной диаграммы решений (BDD) были получены следующие результаты:

Разработанная программа доказала свою работоспособность и пригодность для решения задачи N-ферзей, демонстрируя корректное построение бинарной диаграммы решений для представления допустимых расстановок ферзей на доске. Программа корректно справляется с размещением ферзей на доске любого размера, что делает её универсальным инструментом для решения подобных комбинаторных задач.

Программа предоставляет возможность расширения для работы с досками больших размеров.

Таким образом, можно сделать вывод, что разработка и тестирование программы для размещения ферзей с использованием BDD прошли успешно.