**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Чувашский государственный университет имени И.Н. Ульянова»**

**Факультет информатики и вычислительной техники**

**Кафедра вычислительной техники**

***СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ***

**Расчетно-графическая работа**

**Исследование поиска с возвратом**

**Выполнил:**

студент группы ИВТ-41-22

Иванов В.С.

**Руководитель:**

доцент Павлов Л.А.

Чебоксары 2024

**Оглавление**

[Задание к РГР (вариант 13) 3](#_Toc444541366)

[Введение 4](#_Toc444541367)

[1. Формализация задачи 5](#_Toc444541368)

[1.1. Абстрактные структуры данных для представления объектов 5](#_Toc444541369)

[1.2. Анализ ограничений и усовершенствований 6](#_Toc444541370)

[1.3. Разработка алгоритмов решения задачи 6](#_Toc444541371)

[2. Исследование сложности выполнения алгоритмов 7](#_Toc444541372)

[3. Программная реализация алгоритмов 9](#_Toc444541373)

[3.1. Выбор языка и среды программирования 9](#_Toc444541374)

[3.2. Разработка структурной схемы программы 9](#_Toc444541375)

[3.3. Реализация структур данных и алгоритмов 10](#_Toc444541376)

[Заключение 10](#_Toc444541377)

[Список использованной литературы 11](#_Toc444541378)

Задание к РГР (вариант 13)

# Архиепископы, угроза всех полей

Архиепископ (слоноконь, кардинал, кентавр) – комбинированная шахматная фигура, сочетающая в себе возможности слона и коня. Найти все способы расстановки **минимального** числа архиепископов на шахматной доске размером *n* × *n* так, чтобы они держали под угрозой все поля доски.

Исследовать асимптотическую временную сложность решения задачи в зависимости от *n*.

Введение

***Цель работы*** – закрепление теоретических знаний, полученных по данному курсу и смежным дисциплинам, и приобретение практических навыков формализации поставленной задачи, создания и использования эффективных структур данных и алгоритмов в прикладных задачах, теоретических и экспериментальных оценок эффективности алгоритмов.

Поставленная задача об архмепископах относится к классу комбинаторных задач, которые требуют исчерпывающего поиска множества всех возможных решений, а алгоритмы решения имеют экспоненциальную вычислительную сложность. Одним из общих методов организации такого поиска является *поиск с возвратом* (backtracking), который можно взять за основу для решения поставленной задачи. Поскольку архиепископ атакует поля как в диагоналях, так и в форме буквы 'Г' (по шахматным правилам коня), необходимо определить возможность расстановки определенного количества архиепископов на доске так, чтобы они не находились под угрозой друг от друга. Если такая расстановка возможна, необходимо определить количество способов ее осуществления. В случае, если расстановка невозможна, решается аналогичная задача для уменьшенного количества архиепископов.

В процессе выполнения РГР необходимо:

* формализовать поставленную задачу (перейти от словесной неформальной постановки задачи к математической формулировке);
* приспосабливать общие методы и алгоритмы решения классов задач к решению конкретной задачи;
* проводить сравнительную оценку различных вариантов с целью выбора наиболее эффективных структур данных и алгоритмов их обработки;
* исследовать и оценивать теоретически (аналитически) и экспериментально методы сокращения перебора в комбинаторных задачах;
* оценивать аналитически и экспериментально эффективность предложенных в работе алгоритмов (временную и емкостную сложности);
* программно реализовать разработанные структуры данных и алгоритмы на одном из алгоритмических языков программирования.

# Формализация задачи

Поскольку стоит задача определения всех вариантов расстановки *n* не атакующих друг друга ферзей, можно взять за основу общий алгоритм поиска с возвратом (1):

1. Инициализация:

- Создать структуру данных для представления архиепископа (например, класс Archbishop с полями x и y).

- Создать пустую шахматную доску размером n x n.

- Создать пустой вектор для хранения всех возможных решений.

2. Поиск (рекурсивная функция):

function search(Archbishop[] archbishops, int n):

if archbishops.size() == n: // Все архиепископы расставлены

if доска полностью покрыта:

добавить текущее решение в вектор решений

вернуть

else:

for каждая позиция (x, y) на доске:

создать новый архиепископ с координатами (x, y)

если новый архиепископ не атакует других архиепископов:

добавить его в список архиепископов

установить атаку на доске для нового архиепископа

вызвать рекурсивно функцию search с обновленным списком архиепископов

сбросить атаку на доске для нового архиепископа

удалить новый архиепископ из списка архиепископов

3. Запуск поиска:

вызвать функцию search с пустым списком архиепископов и размером доски n

4. Вывод результатов:

вывести найденные решения и их количество

В задаче расстановки архиепископов, решение представляет собой вектор

(𝑎1,𝑎2,…) неопределенной длины, удовлетворяющий определенным ограничениям. Каждый элемент 𝑎I является архиепископом с координатами (𝑥,𝑦) на шахматной доске. При исчерпывающем поиске рассматриваются все возможные комбинации координат архиепископов в качестве потенциальных решений. Переменная count\_of\_resh в алгоритме не имеет принципиального значения для поиска, она носит информативный характер и служит для подсчета числа исследованных вершин.

## 1.1 Абстрактные структуры данных для представления объектов

Сначала решим вопрос о представлении вектора решений. Очевидно, что все решения имеют одну и ту же фиксированную длину *n*, т. е. решение можно представить вектором (*a*1, …, *an*). На первый взгляд, элемент *ak* (1 ≤ *k* ≤ *n*) этого вектора должен представлять собой координату позиции, в которой размещается *k*-й архиепископ, т. е. упорядоченную пару чисел, определяющих соответственно номер строки и номер столбца. Однако поскольку в каждом столбце может находиться только один архиепископ, то решение можно представить более простым вектором (*a*1, …, *an*), в котором элемент *ak* (1 ≤ *k* ≤ *n*) есть номер строки архиепископа, расположенного в столбце с номером *k*, т. е. координатой позиции является пара (*ak*, *k*). Очевидно, что множества значений элементов *ak* совпадают, т. е. *A*1 = … = *An* = *A* = {1, …, *n*}.

Поскольку областью значений каждого элемента *ak* вектора решения является множество целых чисел от 1 до *n*, нет необходимости в вычислении и явном хранении подмножеств *Sk* и множества *A*. Проще хранить наименьшее значение из *Sk* и следующее значение вычислять по мере необходимости. Текущее значение элемента множества *Sk* обозначим через *sk*. Таким образом, *sk* (1 ≤ *k* ≤ *n*) является элементом вектора *S* = (*s*1, …, *sn*).

## 1.2 Анализ ограничений и усовершенствований

Рассмотрим свойственные задаче ограничения. Одно ограничение, связанное с тем, что в столбце может находиться только один ферзь, учтено представлением вектора решения. Другое ограничение заключается в том, что в каждой строке может быть только один ферзь, поэтому если *i* ≠ *j*, то *ai* ≠ *aj*. Наконец, поскольку ферзи могут атаковать друг друга по диагонали, мы должны иметь |*ai* − *aj* | ≠ |*i* − *j*|, если *i* ≠ *j*. Таким образом, для того чтобы определить, можно ли добавить *ak* для расширения частичного решения (*a*1, *a*2, …, *ak*–1) до (*a*1, *a*2, …, *ak–*1, *ak*), достаточно сравнить элемент *ak* с каждым *ai*, *i* < *k*. Эту проверку можно реализовать в виде функции *is\_uniq\_arch*, представленной на рис. 2, которая отвечает на вопрос, включить данную позицию в подмножество *Sk* кандидатов на выбор *ak* или нет.

bool is\_uniq\_arch(vector<archiepiskop\*>& all\_arch, int x, int y) {

for (auto& a : all\_arch) {

if (a->pos\_x == x && a->pos\_y == y) {

return false;

}

}

return true;

}

Оценим, как влияют эти ограничения на процесс поиска. Если нет никаких ограничений, то на доске размером 𝑛×𝑛 существует n возможных способов расстановки

n архиепископов. Тот факт, что в каждом столбце может находиться только один архиепископ, дает n расстановок. То, что в строку можно поставить только одного архиепископа, говорит о том, что вектор (a1 ,…,an ) может быть решением только тогда, когда он является перестановкой элементов (1,2,…,n), что дает n! возможных расстановок. Требование, что на диагонали может находиться только один архиепископ, еще больше сокращает число возможных расстановок.

## Разработка алгоритмов решения задачи

Поскольку нет необходимости в явном хранении подмножеств Sk, вычисляемое текущее значение элемента множества Sk, обозначенное через sk, является элементом вектора S=(s1,…,sn ). Тогда проверке условия Sk !=∅ в общем алгоритме (см. 1) будет соответствовать условие sk≤n. В результате процедуру нахождения всех решений задачи о неатакующих друг друга архиепископах на доске размера n×n можно формально представить алгоритмом, приведенным ниже.

1. Инициализация:

- Создать пустой вектор resheniya для хранения всех решений.

- Создать пустой вектор all\_arch для хранения всех архиепископов.

- Создать двумерный массив doska размером n × n и заполнить его нулями.

2. Генерация решений:

- Пока количество архиепископов all\_arch.size() < n:

- Создать нового архиепископа с случайными координатами (pos\_x, pos\_y) в диапазоне [0, n-1].

- Проверить уникальность архиепископа среди всех архиепископов в all\_arch.

- Если архиепископ уникален, добавить его в all\_arch.

- Инициализировать переменную count\_of\_resh = 0 для подсчета количества решений.

- Пока count\_of\_resh < 5 (или другое желаемое количество решений):

- Для каждого архиепископа в all\_arch:

- Установить все поля, которые атакуются данным архиепископом, в doska.

- Если все поля на доске покрыты архиепископами:

- Проверить уникальность текущего решения среди уже найденных решений в resheniya.

- Если решение уникально:

- Добавить текущее решение в resheniya.

- Увеличить count\_of\_resh на 1.

- Сбросить положение архиепископов в all\_arch до случайных координат.

- Сбросить все поля на доске doska до нулей.

3. Вывод решений:

- Вывести решения из вектора resheniya.

Следует отметить, что данный алгоритм корректно обрабатывает и ситуацию, когда

k=n+1, поскольку вычисляемое значение sn+1 не меньше, чем n+1, и, следовательно, множество Sn+1 всегда пусто. Включение во внутренний цикл специальной проверки для предотвращения ситуации, когда значение k становится больше n, будет слишком дорогостоящим с точки зрения времени работы алгоритма. Поскольку в соответствии с алгоритмом возможна ситуация, когда k=n+1, следует увеличить размер вектора S = (s1, …, sn) на единицу, т. е. S = (s1, …, sn+1), с учетом описанного ранее алгоритма.

# Исследование сложности выполнения алгоритмов

Аналитическое выражение для оценки вычислительной сложности алгоритмов решения комбинаторных задач удается получить редко, так как трудно предсказать, как взаимодействуют различные ограничения по мере появления их при продвижении вглубь дерева поиска. В подобных случаях, когда построение аналитической модели является трудной или вовсе неосуществимой задачей, можно применить *метод Монте-Карло* (метод статистических испытаний). Этот метод заключается в моделировании исследуемого процесса путем многократного проведения его случайных реализаций, называемых статистическими испытаниями.

Допустим, мы рассматриваем применение метода Монте-Карло для оценки размера доски, на которой необходимо разместить архиепископов. Суть метода заключается в проведении нескольких испытаний, где каждое испытание представляет собой случайное размещение архиепископов. Пусть у нас есть частичное решение, и для каждого следующего элемента решения мы выбираем случайным образом одно из доступных значений. Если доступных значений нет, то испытание завершается. Таким образом, мы повторяем процесс выбора элементов решения многократно, чтобы оценить средний размер доски, необходимый для размещения архиепископов.

Общий алгоритм метода Монте-Карло может быть адаптирован для реализации таких испытаний: если для текущего элемента решения доступных значений нет, то испытание просто завершается. Алгоритм оценки размера доски проводит N испытаний для определения среднего размера доски, необходимого для размещения архиепископов.

1. Инициализация:

- Задать количество испытаний N.

- Задать начальный размер доски size.

- Задать количество архиепископов k.

- Задать счётчик успешных испытаний success\_count = 0.

- Задать счётчик общего числа испытаний total\_count = 0.

2. Для каждое испытание от 1 до N:

2.1. Создать пустой список архиепископов archiepiscopes.

2.2. Пока размер списка archiepiscopes меньше k:

2.2.1. Сгенерировать случайные координаты x и y в диапазоне от 0 до size-1.

2.2.2. Если архиепископ с такими координатами уже существует в списке archiepiscopes, перейти к шагу 2.2.1.

2.2.3. Добавить нового архиепископа с координатами (x, y) в список archiepiscopes.

2.3. Проверить, покрывают ли архиепископы в списке archiepiscopes всю доску:

2.3.1. Если да, увеличить success\_count на 1.

2.4. Увеличить total\_count на 1.

3. Вычисление среднего размера доски:

- Средний размер доски avg\_size = size \* (success\_count / total\_count).

1. Вернуть avg\_size как результат.

Таким образом, каждое испытание представляет собой случайную расстановку архиепископов на доске. Затем проверяется, покрывают ли эти архиепископы всю доску. Оценка размера доски выполняется за счет статистической обработки результатов множества таких случайных расстановок в методе Монте-Карло.

Вычисление по методу Монте-Карло можно использовать для оценки эффективности алгоритма поиска с возвратом путем сравнения его с эталоном, полученным для задачи с меньшей размерностью.

Результаты экспериментальных исследований алгоритма решения задачи об архиепископах представлены в табл. 1. В качестве эталона взят размер задачи 8\*8, для которого выполнен как поиск с возвратом (определен фактический размер дерева поиска), так и оценка размеров дерева поиска методом Монте-Карло. Для размера задачи 10\*10 применен только метод Монте-Карло, который позволил определить, что ожидаемое время выполнения поиска для доски размера 10\*10 составит примерно 196 мс. Для каждого из исследованных методом Монте-Карло размеров задачи проведено *N* = 1000 испытаний.

Таблица 1

**Оценка времени выполнения**

|  |  |  |  |
| --- | --- | --- | --- |
| Размер  задачи | Метод Монте-Карло | Фактически | |
| Время | Время | Порядок роста |
| 6\*6 | – | 450 | – |
| 7\*7 | – | 619 | в 1.37 раза |
| 8\*8 | – | 1992 | в 4.42 раза |
| 9\*9 | 66 | 3675 | в 8.16 раза |
| 10\*10 | 196 | – | – |

График, построенный по полученным экспериментальным данным, показан на рис.1 (тонкой линией изображена аппроксимирующая функция). Ось абсцисс – размер задачи, ось ординат – размер дерева поиска. Наиболее близкой аппроксимирующей функцией является функция *y* =e0.7469x+1.4842 величиной достоверности аппроксимации *R*2 = 0,9917. Аппроксимирующая функция показана черным цветом. Полученные результаты подтверждают экспоненциальную вычислительную сложность решения задачи об архиепископах.

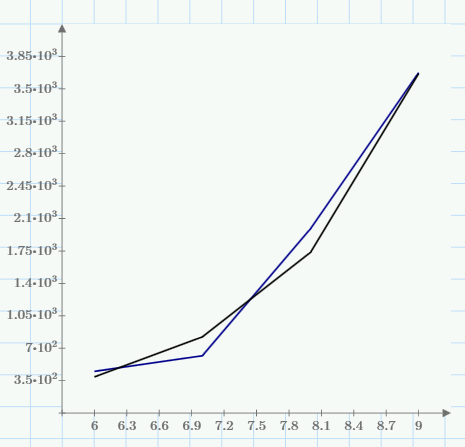


Рис. 1. График функции вычислительной сложности

# 3.Программная реализация алгоритмов

## Выбор языка и среды программирования

В качестве среды разработки и языка программирования выбран язык Visual Studio и C++ соответственно.

В того, что язык используется для проведения лабораторных работ и очень нравится лично, то он был выбран для данной расчётно-графической работы.

## Разработка структурной схемы программы

Разработанная программа исследования алгоритма задачи о архиепископах включает в себя процедуру решения самой задачи. Таким образом, программа настолько проста, что нет необходимости в разработке специальной структурной схемы программы.

## Реализация структур данных и алгоритмов

Программа реализует класс reshenie, который используется для нахождения решения задачи о расстановке архиепископов на доске.

Структуры данных:

vector<pair<int, int>> move\_slon: вектор пар, представляющих возможные ходы слона на доске.

vector<pair<int, int>> move\_kon: вектор пар, представляющих возможные ходы коня на доске.

struct archiepiskop: структура, представляющая архиепископа с его координатами на доске.

vector<archiepiskop\*> all\_arch: вектор указателей на архиепископов.

vector<vector<int>> doska: двумерный вектор, представляющий доску.

Алгоритмы:

random\_int(int left, int right): генерирует случайное целое число в заданном диапазоне.

set\_rand\_coord(archiepiskop\* arch): устанавливает случайные координаты для архиепископа.

set\_rand\_coord(vector<archiepiskop\*> arch\_): устанавливает случайные координаты для всех архиепископов в векторе.

set\_attack(archiepiskop\* arch): устанавливает все поля, которые покрываются архиепископом.

reset\_attack(): сбрасывает доску до нулей.

all\_covered(): проверяет, покрывают ли архиепископы всю доску.

check\_is\_uniq(vector<archiepiskop\*>& arch): проверяет уникальность текущего решения.

check\_for\_all\_resh\_is\_uniq(): дополнительная проверка на уникальность каждого решения.

Заключение

Результатами вычислений для доски размером 8 × 8 (*n* = 8) являются: всего решений 24. В качестве примера на рис. 2 приведены 3 решения.

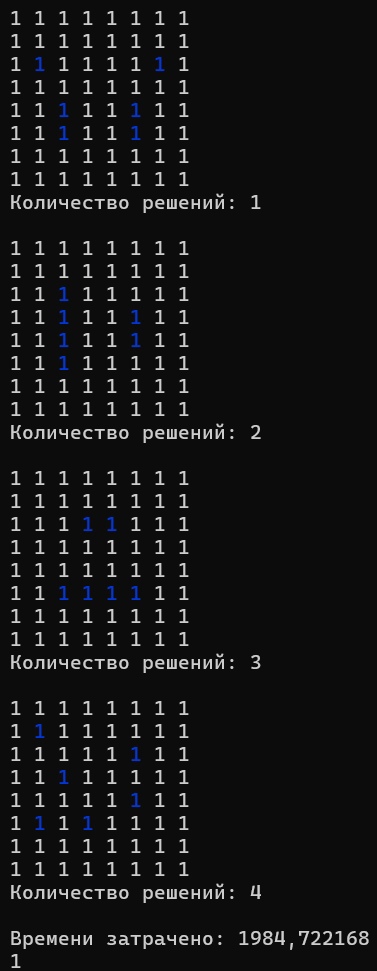
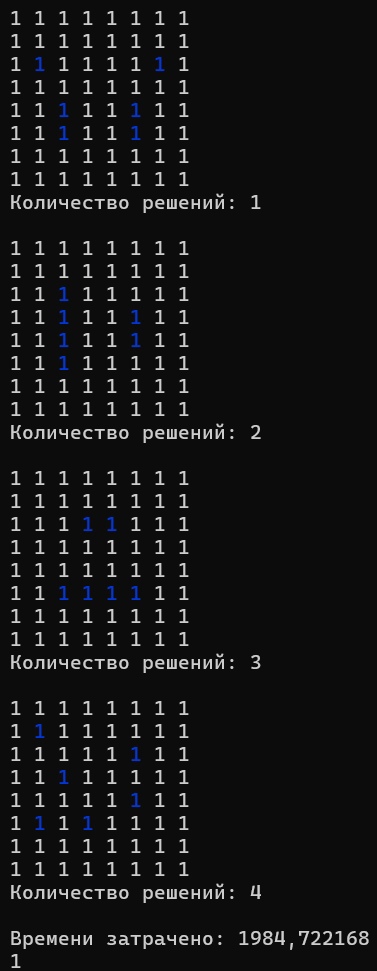
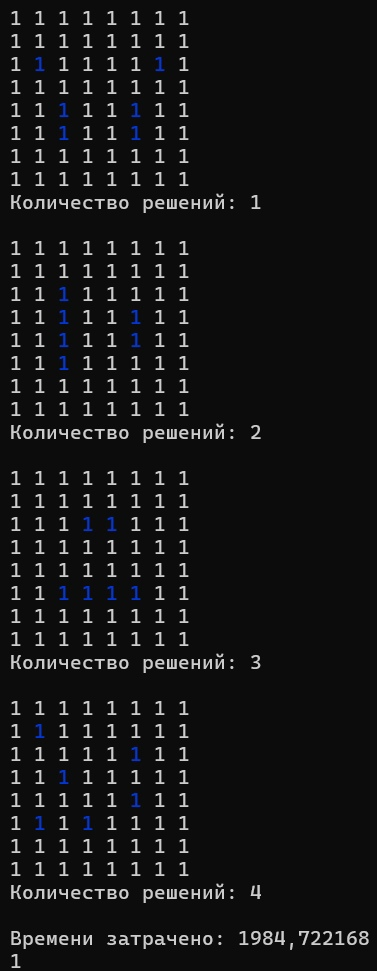
  

Рис. 2. Примеры трех решений задачи об архиепископах для размера 8 × 8

В процессе выполнения расчетно-графической работы:

* формализована поставленная задача;
* общий алгоритм поиска с возвратом приспособлен к решению задачи о ферзях;
* проведена сравнительная оценка различных вариантов с целью выбора наиболее эффективных структур данных и алгоритмов их обработки;
* исследованы и оценены теоретически (аналитически) и экспериментально использованные методы сокращения перебора;
* экспериментально оценена эффективность предложенных в работе алгоритмов;
* программно реализованы разработанные структуры данных и алгоритмы.

В результате выполнения работы закреплены теоретические знания, полученных по данному курсу и смежным дисциплинам, приобретены практические навыки формализации задач, создания и использования эффективных структур данных и алгоритмов, теоретических и экспериментальных оценок эффективности алгоритмов.

Список использованной литературы

1. *Павлов, Л.А*. Структуры и алгоритмы обработки данных: учеб. пособие / Л.А. Павлов. – Чебоксары: Изд-во Чуваш. ун-та, 2008. – 252 с.

2. Структуры и алгоритмы обработки данных: метод. указания к выполнению расчетно-графической работы / сост. Л.А. Павлов. – Чебоксары: Изд-во Чуваш. ун-та, 2014. – 24 с.