**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего 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-42-23

Кузнецов Д.А.

**Руководитель:**

доцент Павлов Л.А.

Чебоксары 2025

**Оглавление**

[Задание к РГР (вариант 77) 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 7](#_Toc444541371)

[2. Исследование сложности выполнения алгоритмов 8](#_Toc444541372)

[3. Программная реализация алгоритмов 10](#_Toc444541373)

[3.1. Выбор языка и среды программирования 10](#_Toc444541374)

[3.2. Разработка структурной схемы программы 11](#_Toc444541375)

[3.3. Реализация структур данных и алгоритмов 11](#_Toc444541376)

[Заключение 12](#_Toc444541377)

[Список использованной литературы 12](#_Toc444541378)

Задание к РГР (вариант 22)

**Оптимальный маршрут коня.**

Клетки шахматной доски размером *n* × *n* заполнены произвольными целыми числами (например, от 1 до 100). Необходимо найти такой маршрут коня из клетки (1, 1) до клетки (*n*, *n*), чтобы сумма чисел в клетках, через которые он пролегает, была минимальной.

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 оптимальном маршруте коня 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 *метод ветвей и границ* (branch and bound), который можно взять за основу для решения поставленной задачи.

В процессе выполнения РГР необходимо [4]:

* 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 поиска минимального пути коня, можно взять за основу общий алгоритм метода ветвей и границ, приведенный на рис. 1.

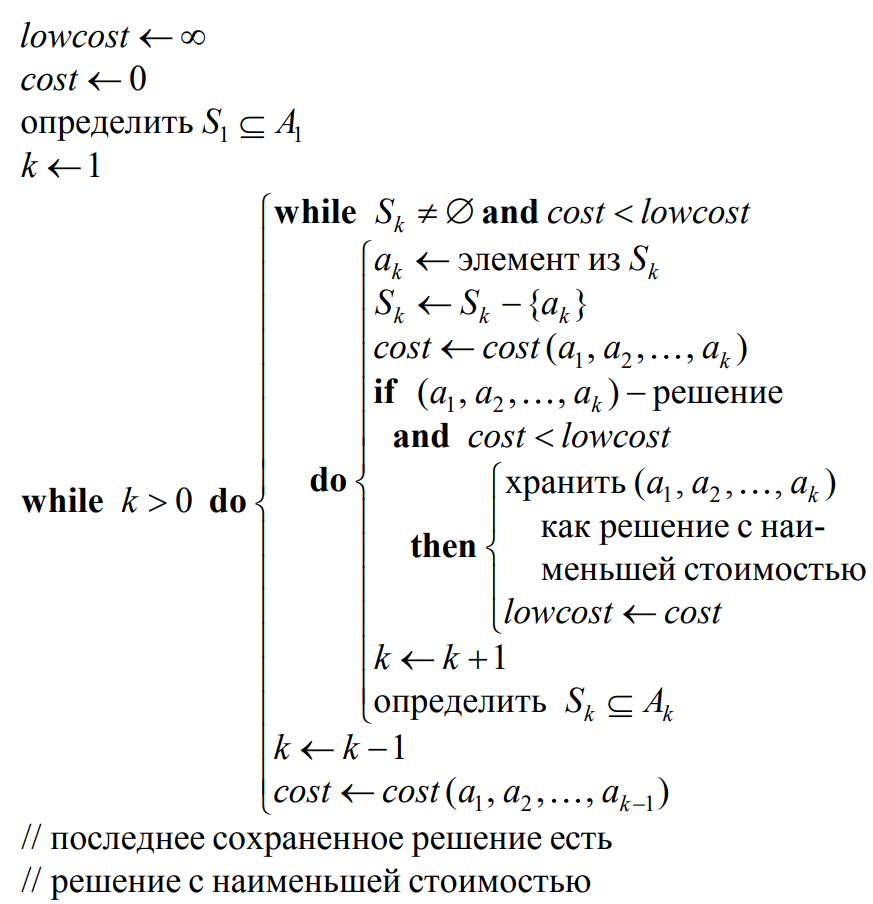


Рис. . Общий алгоритм метода ветвей и границ

В общем случае предполагается, что решение задачи представляет собой вектор (*a*1, *a*2, …) конечной, но не определенной длины, удовлетворяющий некоторым ограничениям. Каждый элемент *ai* является элементом конечного линейно упорядоченного множества *Ai*. Таким образом, при исчерпывающем поиске должны рассматриваться элементы множества *A*1 × *A*2 × … × *Ai* для *i* = 0, 1, 2, … в качестве возможных решений. В качестве исходного частичного решения выбирается пустой вектор ( ) и на основе имеющихся ограничений определяется, какие элементы из множества *A*1 являются кандидатами в *a*1; подмножество таких кандидатов обозначим через *S*1. В качестве *a*1 выбирается наименьший элемент множества *S*1; в результате получается частичное решение (*a*1). В общем случае различные ограничения, описывающие решения, определяют, из какого подмножества *Sk* множества *Ak* должны выбираться кандидаты для расширения частичного решения от (*a*1, *a*2, …, *ak–*1) до (*a*1, *a*2, …, *ak–*1, *ak*). Если частичное решение (*a*1, *a*2, …, *ak–*1) не предоставляет возможностей для выбора элемента *ak*, т. е. *Sk =*∅, то необходимо вернуться и выбрать новый элемент *ak–*1. Если новый элемент *ak–*1 выбрать невозможно, придется вернуться еще дальше и выбрать новый элемент *ak–*2 и т. д.

Метод ветвей и границ удобно представить в виде дерева поиска, в котором исследуемое подмножество множества *A*1 × *A*2 × … × *Ai* для *i* = 0, 1, 2, … представляется следующим образом. Корню дерева (нулевой уровень) ставится в соответствие пустой вектор. Его сыновья образуют множество *S*1 кандидатов для выбора *a*1. В общем случае вершины *k*-го уровня образуют множества *Sk* кандидатов на выбор *ak* при условии, что *a*1, *a*2, …, *ak–*1 выбраны так, как указывают предки этих вершин.

Переменная *cost* в алгоритме отвечает за текущую стоимость пути, а переменная *lowcost* – за минимальную стоимость.

## 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*). Элемент *ak* (1 ≤ *k* ≤ *n*) этого вектора должен представлять собой координату позиции, в которую перемещается конь на *k*-ом ходу, т. е. упорядоченную пару чисел, определяющих соответственно номер строки и номер столбца. Очевидно, что множества значений элементов *ak* совпадают, т. е. *A*1 = … = *An* = *A* = {1, …, *n*}.

## Анализ ограничений и усовершенствований

Рассмотрим свойственные задаче ограничения:

* Конь начинает свой путь из клетки (1, 1);
* Конь заканчивает свой путь в клетке (*n*, *n*).
* Ход коня – буква «Г».

## Разработка алгоритмов решения задачи

Будем хранить *Sk* в явном виде. В результате процедуру нахождения всех решений задачи о пути коня на доске размера *n* × *n* можно формально представить алгоритмом, приведенным на рис. 2.

|  |
| --- |
| // обнулить матрицу n x n visited  visited0,0 = 1;  currentPath = Stack()  currentPath += (0, 0)  notVisitedPoints += ((0, 0))  // получить ходы коня из клетки (0, 0)  currentSum = field0,0  minSum = inf  while |currentPath| > 0 {  while |notVisitedPoints.top()| > 0 AND currentSum < minSum {  nextPoint = notVisitedPoints.top().pop()  currentPath.push\_back(nextPoint)  currentSum += fieldnextPoint.x,nextPoint.y  visitednextPoint.x,nextPoint.y = 1  if nextPoint.x == n - 1 AND nextPoint.y == n - 1 AND currentSum < minSum {  minSum = currentSum  bestPath = currentPath  }  notVisitedPoints.push(пустой массив)  // получить ходы коня из последней клетки пути  }  lastPoint = currentPath.top()  visitedlastPoint.x,lastPoint.y = 0  currentSum -= fieldlastPoint.x,lastPoint.y  currentPath.pop()  notVisitedPoints.pop()  } |

Рис. 2. Алгоритм задачи о пути коня

# 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). Смысл этого метода в том, что исследуемый процесс моделируется путем многократного повторения его случайных реализаций. Каждая случайная реализация называется *статистическим испытанием*.

Рассмотрим применение метода Монте-Карло для экспериментальной оценки размеров дерева поиска. Идея метода состоит в проведении нескольких испытаний, при этом каждое испытание представляет собой поиск с возвратом со случайно выбранными значениями *ai*. Предположим, что имеется частичное решение (*a*1, *a*2, …, *ak*–1) и что число выборов для *ak*, основанное на том, вводятся ли ограничения или осуществляется склеивание, равно *xk* = ⎪*Sk*⎪. Если *xk* ≠ 0, то *ak* выбирается случайно из *Sk* и для каждого элемента вероятность быть выбранным равна 1/*xk*. Если *xk* = 0, то испытание заканчивается. Таким образом, если *x*1 = ⎪*S*1⎪, то *a*1 ∈ *S*1 выбирается случайно с вероятностью 1/*x*1; если *x*2 = ⎪*S*2⎪, то при условии, что *a*1 было выбрано из *S*1, *a*2 ∈ *S*2 выбирается случайно с вероятностью 1/*x*2 и т. д. Математическое ожидание *x*1 + *x*1*x*2 + *x*1*x*2*x*3 + *x*1*x*2*x*3*x*4 + … равно числу вершин в дереве поиска, отличных от корня, т. е. оно равно числу случаев, которые будут исследованы алгоритмом поиска с возвратом. Существует доказательство этого утверждения [5].

Общий алгоритм поиска с возвратом легко преобразуется для реализации таких испытаний; для этого при *Sk* = ∅ вместо возвращения просто заканчивается испытание. Алгоритм оценки размера дерева поиска [3; 4; 5] приведен на рис. 3. Он осуществляет *N* испытаний для вычисления числа вершин в дереве. Операция *ak* ← *rand*(*Sk*) реализует случайный выбор элемента *ak* из множества *Sk*.



Рис. 3. Метод Монте-Карло для поиска с возвратом

Таким образом, каждое испытание представляет собой продвижение по дереву поиска от корня к листьям по случайно выбираемому на каждом уровне направлению. Поскольку в методе Монте-Карло отсутствует возврат, оценка размеров дерева выполняется за полиномиальное время.

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.

Конкретизация этого алгоритма для задачи о ферзях представлена на рис. 5.

Результаты экспериментальных исследований алгоритма решения задачи о ферзях (см. рис. 3) представлены в табл. 2. В качестве эталона взят размер задачи 11×11, для которого выполнен как поиск с возвратом (определен фактический размер дерева поиска), так и оценка размеров дерева поиска методом Монте-Карло. Для размера задачи 12×12 применен только метод Монте-Карло, который позволил определить, что ожидаемое время выполнения поиска для доски размера 12 × 12 составит примерно 451 мс. Для каждого из исследованных методом Монте-Карло размеров задачи проведено *N* = 1000 испытаний.



Рис. . Конкретизация метода Монте-Карло для задачи о ферзях

Таблица

**Оценка времени выполнения**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер  задачи | Метод Монте-Карло | | Фактически | | |
| Число узлов | Порядок роста | Число узлов | Время | Порядок роста |
| 8×8 | – | – | 2 056 | 1,14 мс | – |
| 9×9 | – | – | 8 393 | 4,57 мс | в 4 раза |
| 10×10 | – | – | 35 538 | 19,40 мс | в 4,2 раза |
| 11×11 | 161 124 | в 4,5 раза | 166 925 | 92,04 мс | в 4,7 раза |
| 12×12 | 825 987 | в 4,9 раза | – | – | – |

График, построенный по полученным экспериментальным данным, показан на рис. 6 (тонкой линией изображена аппроксимирующая функция). Ось абсцисс – размер задачи, ось ординат – размер дерева поиска. Наиболее близкой аппроксимирующей функцией является функция *y* = 0,012*e*1,4982*n* = *O*(*cn*) с величиной достоверности аппроксимации *R*2 = 0,9992. Полученные результаты подтверждают экспоненциальную вычислительную сложность решения задачи о ферзях. Что касается емкостной сложности, то она очевидна: для хранения вектора решений (*a*1, …, *an*) требуется *n* ячеек памяти, для хранения вектора *S* = (*s*1, …, *sn*+1) требуется *n* + 1 ячеек и одна ячейка для *k*, т.е. всего требуется памяти *n* + *n* + 1 + 1 = 2*n* + 2 = *O*(*n*).

**Примечание для студента**. В качестве оси ординат можно использовать время вычислений. В данном отчете используется размер дерева поиска, поскольку время вычислений при поиске с возвратом примерно пропорционально числу вершин в дереве поиска.

Рис. . График функции вычислительной сложности

# Программная реализация алгоритмов

## Выбор языка и среды программирования

В качестве среды разработки и языка программирования выбрана система программирования Visual Studio C++. Этот выбор сделан исходя из следующих соображений.

Прежде всего Visual Studio C++предназначен для профессиональных разработчиков, желающих очень быстро разрабатывать приложения. C++ производит небольшие по размерам высокоэффективные исполняемые модули (.exe и .dll). С другой стороны небольшие по размерам и быстро исполняемые модули означают, что требования к клиентским рабочим местам существенно снижаются.

Преимущества Visual Studio C++ по сравнению с аналогичными программными продуктами: высокая производительность разработанного приложения;

* низкие требования разработанного приложения к ресурсам компьютера;
* широкие возможности в отладке на языке C++.

На выбор среды повлияло также то, что имеется опыт разработки других приложений в среде Visual Studio C++.

## Разработка структурной схемы программы

Разработанная программа исследования алгоритма решения задачи о ферзях представляет собой единственный программный модуль (**main.cpp**), включающий в себя процедуру решения самой задачи (findSolution) и процедуру исследования методом Монте-Карло (MonteCarlo). Эти процедуры между собой не взаимодействуют и запускаются отдельно при выборе соответствующего режима исследования. Таким образом, программа настолько проста, что нет необходимости в разработке специальной структурной схемы программы.

## Реализация структур данных и алгоритмов

Программа реализована в виде единственного модуля (**main.cpp**). Модуль включает в себя независимые процедуры:

**void** findSolution – решение задачи о пути коня;

**void** getNextPoints – получение ходов коня;

**bool** isOnField – проверка, что клетка находится на поле;

Заключение

Результатами вычислений для доски размером 8 × 8 (*n* = 8) являются: всего решений 92; количество исследованных вершин дерева поиска 2056; время решения 1,14 мс. В качестве примера на рис. 7 приведены 3 решения из 92.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 72 | 80 | 83 | 43 | 12 | 87 | 61 | 9 |  | 72 | 80 | 83 | 43 | 12 | 87 | 61 | 9 |
| 66 | 74 | 34 | 42 | 98 | 77 | 60 | 56 |  | 66 | 74 | 34 | 42 | 98 | 77 | 60 | 56 |
| 33 | 84 | 40 | 25 | 86 | 30 | 34 | 42 |  | 33 | 84 | 40 | 25 | 86 | 30 | 34 | 42 |
| 96 | 47 | 56 | 54 | 45 | 11 | 23 | 86 |  | 96 | 47 | 56 | 54 | 45 | 11 | 23 | 86 |
| 97 | 11 | 13 | 26 | 60 | 6 | 61 | 66 |  | 97 | 11 | 13 | 26 | 60 | 6 | 61 | 66 |
| 40 | 72 | 56 | 34 | 90 | 37 | 6 | 79 |  | 40 | 72 | 56 | 34 | 90 | 37 | 6 | 79 |
| 94 | 77 | 44 | 19 | 20 | 39 | 75 | 99 |  | 94 | 77 | 44 | 19 | 20 | 39 | 75 | 99 |
| 74 | 20 | 80 | 11 | 92 | 58 | 13 | 25 |  | 74 | 20 | 80 | 11 | 92 | 58 | 13 | 25 |

Рис. 6. Пример решения задачи о пути коня для размера 8 × 8

В процессе выполнения расчетно-графической работы:

* формализована поставленная задача;
* общий алгоритм поиска с возвратом приспособлен к решению задачи о пути коня;
* 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. *Кормен*, *Т*. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – 2-е изд. – М.: Вильямс, 2011. – 1296 с.

2. *Липский*, *В*. Комбинаторика для программистов / В. Липский. – М.: Мир, 1988. – 213 с.

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

4. 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.

5. *Рейнгольд*, *Э*. Комбинаторные алгоритмы. Теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М.: Мир, 1980. – 476 с.

**Приложение**

#include <ctime>

#include <iostream>

#include <random>

#include <vector>

using namespace std;

struct Point {

int x, y;

};

const int max\_size = 20;

void findSolution(int field[max\_size][max\_size], int n);

void getNextPoints(Point point, vector<Point>& dest, bool visited[max\_size][max\_size], int n);

bool isOnField(Point point, int n);

int field[max\_size][max\_size] = {};

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

int n = 8;

#if 1

cout << "Введите длину поля: ";

cin >> n;

cout << endl;

#endif

cout << "Доска:" << endl;

for (int y = 0; y < n; y++) {

for (int x = 0; x < n; x++) {

field[x][y] = rand() % 100 + 1;

cout << field[x][y] << "\t";

}

cout << endl;

}

findSolution(field, n);

return 0;

}

void findSolution(int field[max\_size][max\_size], int n)

{

bool visited[max\_size][max\_size] = {};

visited[0][0] = 1;

vector<Point> currentPath(1, { 0, 0 });

vector<vector<Point>> notVisitedPoints = { vector<Point>() };

getNextPoints(currentPath.back(), notVisitedPoints.back(), visited, n);

int currentSum = field[0][0];

vector<Point> bestPath;

int minSum = INT\_MAX;

while (!currentPath.empty()) {

while (!notVisitedPoints.back().empty() && currentSum < minSum) {

Point nextPoint = notVisitedPoints.back().back();

notVisitedPoints.back().pop\_back();

currentPath.push\_back(nextPoint);

currentSum += field[nextPoint.x][nextPoint.y];

visited[nextPoint.x][nextPoint.y] = 1;

if (nextPoint.x == n - 1 && nextPoint.y == n - 1 && currentSum < minSum) {

minSum = currentSum;

bestPath = currentPath;

}

notVisitedPoints.push\_back(vector<Point>());

getNextPoints(nextPoint, notVisitedPoints.back(), visited, n);

}

Point& lastPoint = currentPath.back();

visited[lastPoint.x][lastPoint.y] = 0;

currentSum -= field[lastPoint.x][lastPoint.y];

currentPath.pop\_back();

notVisitedPoints.pop\_back();

}

cout << "Сумма: " << minSum << endl;

cout << "Путь: ";

for (auto& point : bestPath) {

cout << "(" << point.x + 1 << ", " << point.y + 1 << ") ";

}

}

void getNextPoints(Point point, vector<Point>& dest, bool visited[max\_size][max\_size], int n)

{

for (int scaleX = 1; scaleX <= 2; scaleX++) {

for (int dx = -scaleX; dx <= scaleX; dx += 2 \* scaleX) {

int scaleY = 3 - scaleX;

for (int dy = -scaleY; dy <= scaleY; dy += 2 \* scaleY) {

Point nextPoint = { point.x + dx, point.y + dy };

if (isOnField(nextPoint, n) && !visited[nextPoint.x][nextPoint.y]) {

dest.push\_back(nextPoint);

}

}

}

}

}

bool isOnField(Point point, int n)

{

return 0 <= point.x && point.x < n && 0 <= point.y && point.y < n;

}