1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**ЛАБОРАТОРНАЯ РАБОТА**

1. «**Решение задачи “Кубик” с помощью алгоритма перебора с возвратом**»
2. по дисциплине «Алгоритмизация и программирование»

Выполнили студенты гр. 5131001/30003:

1. Утеев Владислав Юрьевич \_\_\_\_\_\_\_\_\_\_\_\_

<*подпись*>

Шевчук Ника Евгеньевна \_\_\_\_\_\_\_\_\_\_\_\_

<*подпись*>

1. Преподаватель: Семьянов Павел Валентинович, \_\_\_\_\_\_\_\_\_\_\_\_
2. Старший преподаватель <*подпись*>
3. Санкт-Петербург
4. 2023
5. СОДЕРЖАНИЕ

[Цель 3](#_Toc156418070)

[Задачи 3](#_Toc156418071)

[Введение 3](#_Toc156418072)

[1. Условие задачи 3](#_Toc156418073)

[2. Актуальность 3](#_Toc156418074)

[Ход работы 4](#_Toc156418075)

[1. Идея алгоритма 4](#_Toc156418076)

[2. Алгоритм 4](#_Toc156418077)

[3. Проблемы и их решения 6](#_Toc156418078)

[Выводы 7](#_Toc156418079)

[Результат 9](#_Toc156418080)

Цель

Адаптировать и изучить существующие алгоритмы с возвратом для нахождения пути кубика с учетом поставленных в задаче условий.

Задачи

1. Реализовать данный алгоритм на языке программирования Си.
2. Проанализировать результаты.

Введение

1. Условие задачи

Написать программу, которая строит маршрут кубика по полю N\*N из левой верхней клетки в правую верхнюю, удовлетворяющий следующим условиям (см. пример маршрута для поля 6x6 на рис. 1):

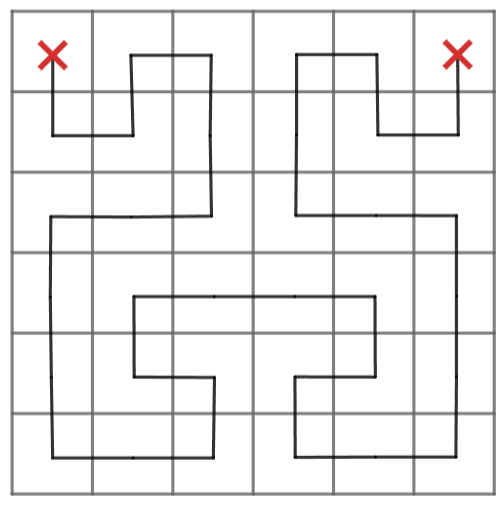
* У кубика есть запрещенная грань. В процессе всего маршрута кубик не должен вступить на эту грань.
* В изначальном положении, в левой верхней клетке, запрещенная грань находится вверху.
* В конечном положении, в правой верхней клетке, запрещенная грань также должна находиться вверху.
* Кубик должен обойти все клетки поля.
* Кубик не может побывать на одной клетке дважды.

Рис. 1

1. Актуальность

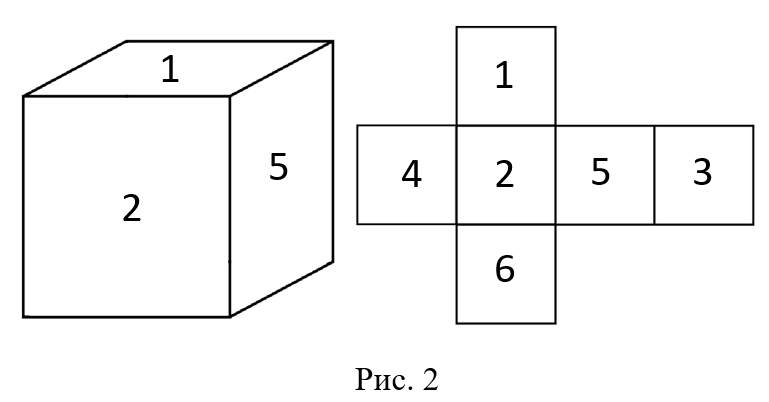
Существуют задачи, которые человек не научился решать каким-либо другим методом, кроме перебора. Примером такой задачи может послужить задача о расстановке шахматных фигур на доске. Однако, часто полный перебор требует очень много времени. Поэтому в таких задачах основная цель состоит в сокращении перебора путём отбрасывания заведомо проигрышных вариантов. Задача о кубике относится к данной категории задач. Практически при каждом ходе необходимом просматривать все четыре варианта направления перемещения, что ведет за собой экспоненциальный рост количества вызовов рекурсивной функции. Соответственно, с помощью специальных дополнительных условий необходимо отсеивать заранее неудачные ходы.

Ход работы

1. Идея алгоритма

Заранее пропишем, как меняются грани кубика в зависимости от его хода вправо/влево/вниз/вверх в следующую клетку. Алгоритм должен последовательно строить маршрут (заполнять двумерный массив последовательностью ходов), на каждом шаге проверяя выполнение поставленных в описании задачи условий, отсекая заведомо неподходящие варианты. Каждый следующий шаг маршрута строится за счет рекурсивного вызова.

1. Алгоритм

 Сначала пропишем отдельную функцию turn\_cube, которая фиксирует изменение положений граней кубика при ходе вправо/влево/вверх/вниз на следующую клетку доски. На вход функция получает номер текущей верхней грани и направление движения, возвращает номер верхней грани после выполнения хода. Изменение положения граней кубика рассматриваем относительно нумерации на модели, изображенной на рис. 2. Запрещенная грань – номер 1.

Начнём писать основную рекурсивную функцию перебора backtraking. На каждом шаге перебора в нее передаётся:

* n – размерность поля;
* двумерный массив m[100][100], отвечающий за отметку прохождения клетки;
* номер верхней грани кубика на текущий момент (restricted\_face\_position);
* номер строки i, номер столбца j, однозначно определяющие клетку на поле;
* переменная vis, отвечающая за номер текущего хода.
* двумерный массив patch[100][100], отвечающий за запись хода и последующий вывод решения;
* указатель на переменную found, фиксирующую найден маршрут или нет.

В качестве условия выхода из рекурсивной функции прописываем условие, сравнивающее номер текущего хода с общим количество клеток доски, а также дополнительных условий задачи - верхняя грань запрещенная, кубик находится в правом верхнем углу доски. Если все условия выполнены, то с помощью дополнительной процедуры result\_match выводим на экран полный маршрут и счетчик количества вызовов рекурсии для оценки эффективности алгоритма. Также изменяем значение переменной found, отмечая, что маршрут найден.

Далее для ускорения перебора прописываем условие, отсекающее ходы, при которых кубик находится в правой верхней клетке, однако счетчик ходов не равен общему количеству клеток.

Затем начинаем перебирать варианты следующего хода. Присваиваем текущей клетке (значению двумерного массива m[i][j]) номер текущего хода. Далее прописываем условия, при которых возможен следующий ход:

* проверка на то, что данная клетка (куда планируется сделать ход) еще не пройдена (значение двумерного массива m для этой клетки равно 0);
* после выполнения хода кубик не стоит на запрещенной грани (выполняется с помощью функции turn\_cube).
* кубик не находится в правом крайнем столбце (j=n-1) перед ходом вправо, в левом крайнем столбце (j=0) перед ходом в лево, в верхней строке (i=0) перед ходом вверх, в нижней строке (i=n-1) перед ходом вниз.

Если условия выполнены, то начинается проверка варианта – запись хода в двумерный массив-маршрут patch[100][100] и рекурсивный вызов функции, в котором изменяются положение граней кубика (функцией turn\_cube), клетка поля (переменными i, j), а также увеличивается счетчик ходов.

В конце рекурсивной функции прописаны действия стирания хода, если он оказался неудачным: уменьшаем счетчик ходов на единицу, обнуляем значение клетки в двумерном массиве m[100][100].

При первом вызове рекурсивной функции в основной функции main() в функцию передаются следующие входные значения:

* n – размерность поля, введенная пользователем с клавиатуры;
* двумерный массив m[100][100] изначально обнулен;
* верхняя грань – запрещенная (грань под номером 1);
* кубик находится в верхнем левом углу (i=0, j=0);
* текущий ход – первый;
* двумерный массив patch[100][100] изначально обнулен;
* указатель на переменную found, которая изначально равна нулю.

Также в функции main() после вызова рекурсивной функции прописываем с помощью переменной found условие, проверяющее найден ли маршрут для данного n, и выводим “no solution”, если нет.

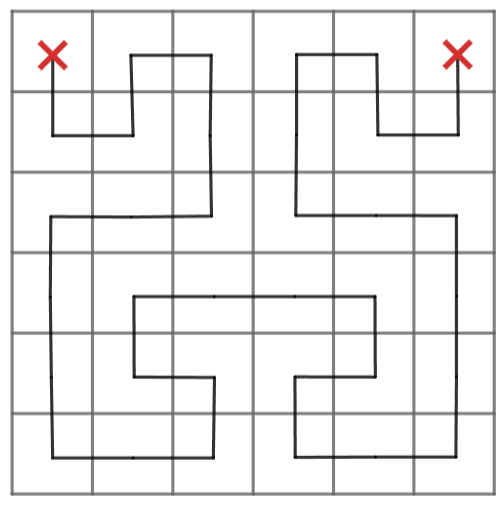
1. Проблемы и их решения

Первая проблема – учет сменяемости граней кубика при выполнении хода. Необходимо было придумать модель, четко определяющую положение граней кубика в зависимости от его предыдущего состояния. Для решения этой проблемы была создана отдельная функция turn\_cube, в которой с помощью оператора множественного выбора switch прописано, как изменяются все грани кубиков относительно его предыдущего положения. За относительную модель принята фиксированная нумерация граней (см. рис. 2).

Вторая проблема заключается в том, что время выполнения рекурсивной функции в зависимости от величины n растет экспоненциально, также она потребляет много памяти из-за хранения локальных переменных и контекста вызова для каждого уровня рекурсии. Поэтому программа обрабатывает все значения n большие 9 очень долго. Однако, несмотря на это, производительность рекурсии, хоть и несущественно, можно увеличить путем отсечения заведомо лишних вызовов.

Выводы

Программа находит и выводит маршрут для n = {2, 6, 9}. Для n = 1 решение не имеет физического смысла. При n = {3, 4, 5, 7, 8} решений нет. При больших n даже с учетом ускорения перебора он занимает слишком много времени (маршрут для n=10 (рис. 7) программа вычисляла практически 7 часов). Ниже приведены маршруты, построенные программой для n = 2 (рис. 3), n = 6 (рис. 1) и n = 9 (рис. 4), а также таблица (рис. 5) и график (рис. 6) зависимости n от времени работы программы.



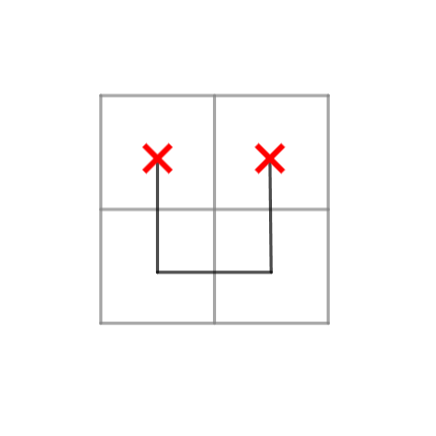


Рис. 1

Рис. 3

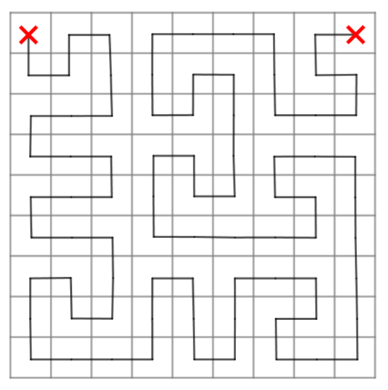
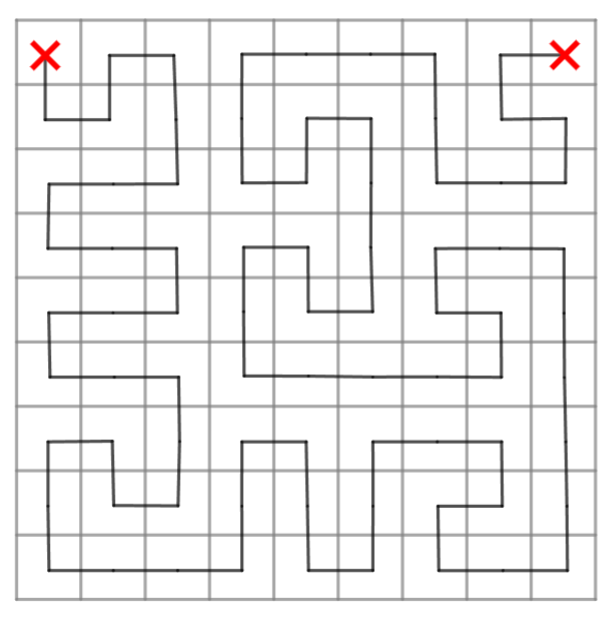
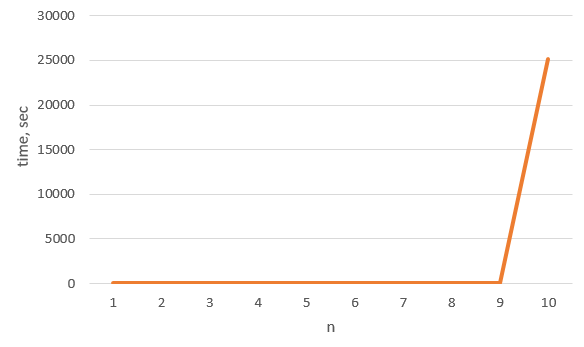


Рис. 4



|  |  |
| --- | --- |
| n | time, sec |
| 2 | 0,002000 |
| 3 | 0,001000 |
| 4 | 0,0000001 |
| 5 | 0,0000001 |
| 6 | 0,007000 |
| 7 | 0,022000 |
| 8 | 0,960000 |
| 9 | 94,852000 |
| 10 | 25149,698000 |

Рис. 6

Рис. 5

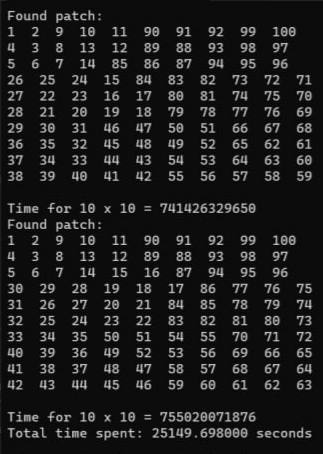


Рис. 7

Результат

Нам удалось адаптировать алгоритм перебора под конкретную задачу и разработать программу на языке Си, которая находит маршрут кубика по полю заданного размера, выполняя все условия поставленной задачи. Также мы исследовали полученные результаты.