Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Кубанский государственный технологический университет

(ФГБОУ ВПО «КубГТУ»)

Кафедра информационных систем и программирования

Факультет компьютерных технологий и автоматизированных систем

**КУРСОВАЯ РАБОТА**

По дисциплине дискретная математика

На тему : Задачи и алгоритмы дискретной математики

Выполнил студент группы 12-КБ-ПИ1 Грушин Егор Романович

Допущен к защите

Защищён \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Симоненко Е.А.\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Волик Е.Г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата, расшифровка подписи)

Краснодар

2013

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Кубанский государственный технологический университет

(ФГБОУ ВПО «КубГТУ»)

Кафедра информационных систем и программирования

Факультет компьютерных технологий и автоматизированных систем

УТВЕРЖДАЮ

Зав. кафедрой ИСП

профессор Л.А. Видовский

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись, расшифровка подписи)

**ЗАДАНИЕ**

Студенту Грушину Егору Романовичу группы 12-КБ-ПИ1

факультета Компьютерных технологий и автоматизированных систем управления специальности 230700 Прикладная информатика

Тема работы: Задачи и алгоритмы дискретной математики.

Содержание задания: Изучить темы «Размещения с повторениями и без» и «Фундаментальное множество циклов», провести исследования алгоритмов работы с этими объектами, составить программы, которые демонстрируют указанные алгоритмы, провести тестирование программ

Объём курсовой работы:

а) пояснительная записка стр. 18 ;

Рекомендуемая литература Окулов «Дискретная математика», Кристофидес «Теория графов».

Срок выполнения работы: с "2" сентября по "21" декабря 2013г.

Срок защиты: с "2" декабря по "21" декабря 2013г.

Дата выдачи задания: с "3" сентября по "10" сентября 2013г.

Дата сдачи работы на кафедру: с "2" декабря по "21" декабря 2013г.

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, ф.и.о., звание, степень)

Задание принял студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

**Реферат**

Пояснительная записка курсового проекта (работы) 21 с., 5 рис., 2 источника.

РАЗМЕЩЕНИЯ, ЦИКЛЫ, ОСТОВ, ФУНДАМЕНТАЛЬНОЕ МНОЖЕСТВО ЦИКЛОВ, ОБХОД В ГЛУБИНУ

В данной курсовой работе были рассмотрены вопросы о понятии размещений с и без повторений, алгоритме генерации следующего размещения в лексикографическом порядке, о понятии фундаментальное множество циклов, алгоритме их поиска, которые позволили написать программы на языке программирования C# для демонстрации алгоритма генерации следующего размещения в лексикографическом порядке и поиска фундаментального множества циклов.

Содержание

[Введение 5](#_Toc375683340)

[Глава 1 - Размещения 6](#_Toc375683341)

[1.1 Постановка задачи 6](#_Toc375683342)

[1.2 Описание алгоритма 6](#_Toc375683343)

[1.2.1 Без повторений 6](#_Toc375683344)

[1.2.2 С повторениями 7](#_Toc375683345)

[1.3 Результат ручного выполнения алгоритма 7](#_Toc375683346)

[1.3.1 Без повторений 7](#_Toc375683347)

[1.3.2 С повторениями 8](#_Toc375683348)

[1.4 Руководство по сборке и запуску на проверку программ, реализующих решение соответствующих задач 8](#_Toc375683349)

[1.4.1 Для размещений без повторений 8](#_Toc375683350)

[1.4.2 Для размещений с повторениями 8](#_Toc375683351)

[Глава 2 - Фундаментальное множество циклов 10](#_Toc375683352)

[2.1 Постановка задачи 10](#_Toc375683353)

[2.2 Описание алгоритма 10](#_Toc375683354)

[2.3 Результат ручного выполнения алгоритма 10](#_Toc375683355)

[2.4 Руководство по сборке и запуску на проверку программ, реализующих решение соответствующих задач 13](#_Toc375683356)

[Заключение 15](#_Toc375683357)

[Список используемых источников 16](#_Toc375683358)

[Приложение А 17](#_Toc375683359)

[Приложение Б 19](#_Toc375683360)

[Приложение B 20](#_Toc375683361)

# Введение

В данной курсовой работе рассмотрены два вопросы. Первый из них – генерация размещений с и без повторений. Второй – фундаментальное множество циклов и их нахождение.

Для реализации генерации размещений достаточно разработать метод для поиска следующего в лексико-графическом порядке размещения.

Для поиска фундаментального множества цикла графа используется обход в глубину с последующим просмотром пройденных вершин графа.

# Глава 1 - Размещения

# 1.1 Постановка задачи

В комбинаторике размеще́нием (из n по k) называется упорядоченный набор из k различных элементов из некоторого множества различных n элементов. Лексикографический порядок — отношение линейного порядка на множестве слов длины n над некоторым упорядоченным алфавитом S.

Размещения бывают с повторениями, когда допускаются одинаковые элементы внутри одного размещения, и без повторений – когда такое не допускается.

# 1.2 Описание алгоритма

# 1.2.1 Без повторений

Будем последовательно перебирать элементы размещения справа налево. Если на каком-то этапе для текущего элемента размещения [j] можно найти элемент F[k] из свободных членов, который больше его, то заменяем [j] на F[k]. При этом F[k] должен быть минимально возможным. При этом может возникнуть две ситуации:

1) Это сделать удалось. При это нужно дополнить размещение так, чтобы ее размер стал равным m. Для этого на свободные места последовательно добавляем элементы множества F в порядке возрастания начиная с самого маленького.

2) Этого сделать не удалось. Текущий элемент [j] удаляем из текущего размещения и добавляем в множество F. Берем следующий элемент [j-1] и повторяем итерацию. Если же не удалось найти такой элемент, значит на текущей итерации мы работали с последним размещением в лексикографическом порядке.

# 1.2.2 С повторениями

Для генерации следующего размещения будем пользоваться простым правилом:

Пусть задан алфавит, состоящих из первых k цифр, начиная с 0. Также имеется текущее размещение с повторениями длины n. Найдем следующее размещение с повторениями в лексикографическом порядке.

Перебираем все элементы справа налево. Если текущий элемент равен k-1, тогда обнуляем его и переходим к элементу левее. В противном случае увеличиваем текущий элемент на 1 и заканчиваем генерацию. Если на текущем шаге были перебраны все элемента размещения, но не получилось инкрементировать никакой элемент, значит на текущем шаге мы работали с последним размещением для заданных n и k.

# 1.3 Результат ручного выполнения алгоритма

# 1.3.1 Без повторений

Пусть n = 9, k = 6. Возьмем любое размещение из 9 по 6, например = {1, 3, 6, 5, 9, 8}. Найдем следующее размещение . Алфавитом в данном случае будет являться набор числе от 1 до 9.

Для начала разделим все элементы алфавита на две группы: те, которые входят в данное размещение и те, что нет (множество F):

**136598247**

=**136598247 –** Нет свободного элемента больше 8, поэтому ставим 8 в множество F так, чтобы оно было упорядочнено по возрастанию.

**136592478 –** аналогично с 9.

**136524789 -** Есть 3 свободных элемента больше 5: 7,8,9. Выбираем 7(наименьший) и меняем их местами

**136724589 –** первые 4 элемента размещения определенны. Осталось взять первые 6 элементов, и мы получим наше : **136724.**

# 1.3.2 С повторениями

Пусть n = 5, k = 10. Возьмем любое размещение из 9 по 5, например = {1, 3, 6, 5, 9}. Найдем следующее размещение . Алфавитом в данном случае будет являться набор числе от 0 до 9.

Справа налево:

Первый элемент равен 9, равен k – 1, значит обнуляем его:

{1, 3, 6, 5, 0}.

Второй элемент равен 5, что не равно k – 1, значит увеличиваем его на единицу и заканчиваем генерацию:= {1, 3, 6, 6, 0}.

# 1.4 Руководство по сборке и запуску на проверку программ, реализующих решение соответствующих задач

# 1.4.1 Для размещений без повторений

Язык программирования: C#.

Среда разработки: Microsoft Visual Studio 2012

Тестирование: ручное.

Важные параметры:

1. Статическое поле letters, представляющее собой алфавит множества.
2. Параметром для метода GetNext является целочисленный массив, являющийся представлением текущего размещения

Пример инициализации и декларации данных:

letters = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

GetNext(new int[]{1,3,6,5,9,8});

Результат работы:

136724

# 1.4.2 Для размещений с повторениями

Язык программирования: C#.

Среда разработки: Microsoft Visual Studio 2012

Тестирование: ручное.

Важные параметры:

1) Параметрами для метода GetNext является целочисленный массив, являющийся представлением текущего размещения и целое число k, являющееся мощностью алфавита размещения

Пример инициализации и декларации данных:

GetNext(new int[] {1, 3, 6, 5, 9}, 10);

Результат работы:

13660

# Глава 2 - Фундаментальное множество циклов

# 2.1 Постановка задачи

Если при обходе в глубину связного неориентированного графа } сохранять пройденные ребра, то множество этих ребер представляет собой остов данного графа. При добавлении одного ребра к S из множества ребер образуется один цикл, называемый фундаментальным. Множество таких циклов будем искать.

# 2.2 Описание алгоритма

Алгоритм поиска циклов заключается в проходе графа в глубину, запоминая номера вершин в порядке прохода в массив. После идет проход по этому массиву. Смотрим инцидентные вершины текущей (здесь и далее) вершине по ма трице инцидентности, исключая следующую и предыдущую из массива вершин остова. Если номер инцидентной (здесь и далее) вершины меньше, чем номер текущей, значит эти две вершины образуют ребро, которое закрывает цикл. Закрыв цикл есть можно получить номера его вершин – начиная с вершины инцидентной считывать вершины из массива вершин остова до вершины текущей.

# 2.3 Результат ручного выполнения алгоритма

Дан граф:

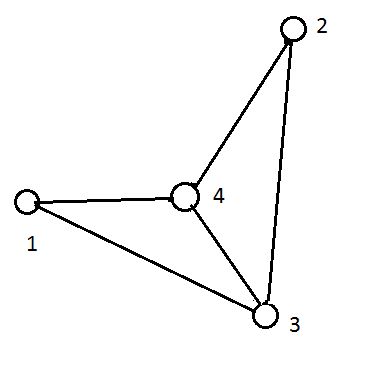


Рисунок 1 - Граф

Проход в глубину с сохранением пройденных ребер даст остов:

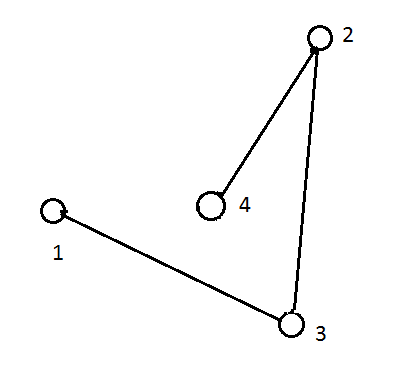


Рисунок 2 - Остов

Каждое ребро из множества ребер, невошедших в остов ( , будет давать цикл:

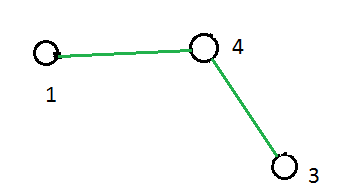


Рисунок 3 – Множество

Добавим одно ребро из множества к остову S:

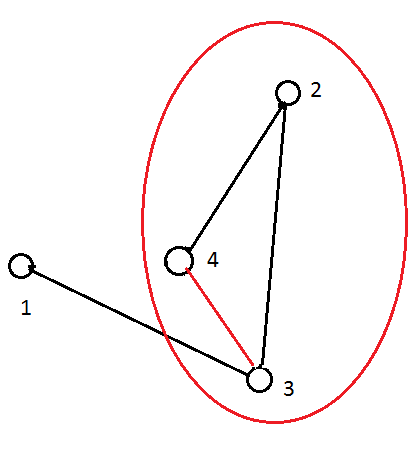


Рисунок 4 – Первый цикл

Получили цикл из вершин 2-3-4-2.

Добавим к остову второе ребро из множества:

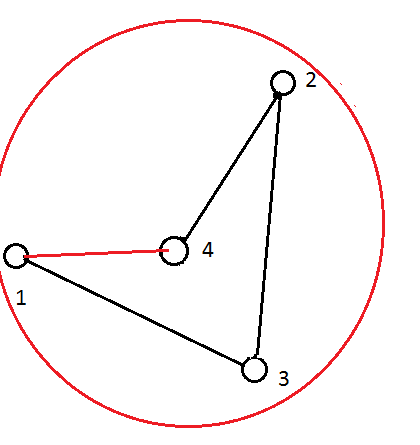


Рисунок 5 – Второй цикл

Получен цикл 1-4-2-3-1.

Все фундаментальные циклы найдены.

# 2.4 Руководство по сборке и запуску на проверку программ, реализующих решение соответствующих задач

Язык программирования: C#.

Среда разработки: Microsoft Visual Studio 2012

Тестирование: ручное.

Важные параметры:

1. Статическое поле baseGraph, представляющее матрицу инцидентности n x n.
2. Статическое поле colors, представляющее собой массив длиной n.

Пример инициализации полей:

baseGraph = new int[4, 4]

{

{0,0,1,1},

{0,0,1,1},

{1,1,0,1},

{1,1,1,0}

};

colors = new int[4];

Результат:

Циклы, представленные номерами вершин:

1324

324

# Заключение

В данной курсовой работе мы познакомились с понятием размещения, алгоритмом генерации следующего в лексикографическом порядке размещения с повторением и без, с фундаментальным множество циклов, а так же поиском этих циклов.

Зная все или хотя бы большую часть тонкостей алгоритмов, мы способны создать программы, которые будут более устойчивы к ошибкам и оптимально настроены под ту или иную ситуацию.

Благодаря понимаю основных алгоритмов, человек имеет возможность на их базе создать более сложные системы, которые базируются на простых.

Список используемых источников

1. Окулов С.М. Дискретная математика. Теория и практика решения задач по информатике: учебное пособие. – М.: БИНОМ. Лаборатория знаний, 2008. – 422 с.

2. Кристофидес Н. Теория графов. Алгоритмический подход. – Пер. с англ. – М.: Мир, 1978. – 432 с.

Приложение А

Листинг программы, демонстрирующую работу алгоритма получения следующего в лексикографическом порядке размещения без повторений (С#)

class Program

{

static int[] letters = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

static int[] GetNext(int[] current)

{

List<int> vacant = new List<int>();

foreach (int elem in letters)

{

if (!current.Contains(elem))

{

vacant.Add(elem);

}

}

int j = current.Length - 1;

while (j >= 0)

{

bool flag = false;

int vacantIndex = 0;

for (int i = 0; i < vacant.Count - 1; i++)

{

if (vacant[i] > current[j] && !flag)

{

flag = true;

vacantIndex = i;

}

}

if (!flag)

{

vacant.Add(current[j]);

vacant.Sort();

}

else

{

current[j] ^= vacant[vacantIndex];

vacant[vacantIndex] ^= current[j];

current[j] ^= vacant[vacantIndex];

int l = j + 1;

for (int k = 0; k < current.Length - j - 1; k++)

{

current[l] = vacant[k];

l++;

}

return current;

}

j--;

}

return current;

}

static void Main(string[] args)

{

int[] next = GetNext(new int[]{1,3,6,5,9,8});

foreach (var elem in next)

{

Console.Write(elem);

}

Console.WriteLine();

}

}

Приложение Б

Листинг программы, демонстрирующую работу алгоритма получения следующего в лексикографическом порядке размещения с повторениями (С#)

class Program

{

static int[] GetNext(int[] current, int k)

{

bool flag = true;

int position = current.Length - 1;

while (position >= 0 && flag)

{

if (current[position] == k - 1)

{

current[position] = 0;

position--;

}

else

{

current[position]++;

flag = false;

}

}

return current;

}

static void Main(string[] args)

{

int[] next = GetNext(new int[] {1, 3, 6, 5, 9}, 10);

foreach (var elem in next)

{

Console.Write(elem);

}

Console.WriteLine();

}

}

Приложение B

Листинг программы, демонстрирующий работу алгоритма поиска фундаментальных циклов (С#)

class Program

{

static int[,] baseGraph = new int[4, 4]

{

{0,0,1,1},

{0,0,1,1},

{1,1,0,1},

{1,1,1,0}

};

static List<int> skeleton = new List<int>();

static int[] colors = new int[4];

static void DFS(int currentVertex)

{

skeleton.Add(currentVertex);

colors[currentVertex - 1] = 1;

for (int i = 1; i < 5; i++)

{

if (baseGraph[currentVertex - 1, i - 1] == 1 && colors[i - 1] == 0)

{

DFS(i);

}

}

colors[currentVertex - 1] = 2;

}

static void Loops()

{

for (int i = 1; i < 5; i++)

{

int currentVertex = skeleton[i];

for (int j = 0; j < 4; j++)

{

if (baseGraph[currentVertex - 1, j] == 1 && j != skeleton[i - 1] - 1 && j != skeleton[i + 1] - 1 && j < currentVertex - 1)

{

for (int k = 1; k < 6; k++)

{

if (skeleton[k] == j + 1)

{

for (int m = k; m <= i; m++)

{

Console.Write(skeleton[m]);

}

Console.WriteLine();

}

}

}

}

}

}

static void Main(string[] args)

{

skeleton.Add(-1);

DFS(1);

skeleton.Add(-1);

Loops();

}

}