МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. А.Н. КОСЫГИНА (ТЕХНОЛОГИИ.ДИЗАЙН.ИСКУССТВО)»
(ФГБОУ ВО «РГУ ИМ. А.Н.КОСЫГИНА»)

Кузьмина Т.М., Ветрова О.А.

Дискретная математика
Лабораторные работы

Учебное пособие

Допущены к изданию редакционно-издательским советом университета в качестве электронного учебного пособия для подготовки бакалавров по направлению 09.03.01 Информатика и вычислительная техника

Объем 1,62 МБ Тираж 10

Редакционно-издательский отдел ФГБОУ ВО «РГУ им. А.Н. Косыгина»
115035, Москва, ул. Садовническая, 33, стр. 1
тел.8-495-811-01-01 доб. 1099
e-mail: riomgudt@mail.ru

Москва
ФГБОУ ВО «РГУ им. А.Н. Косыгина»
2021

ISBN 978-5-87055-904-9

УДК 004.415

ББК 32.973.018.1

К89

К89 Кузьмина Т.М., Ветрова О.А.

Дискретная математика. Лабораторные работы. Учебное пособие. – М.: ФГБОУ ВО «РГУ им. А.Н. Косыгина», 2021– 1,62 мб.

Данное учебное пособие содержит рекомендации по выполнению лабораторных работ по основным вопросам дисциплины. Для каждой работы формулируется постановка задачи, описаны этапы ее выполнения, приведены индивидуальные задания и планы отчетов по ним. Изложен теоретический материал в объеме, необходимом для постижения сути задач и их решения.

Учебное пособие предназначено для обучающихся по направлению подготовки 09.03.01 Информатика и вычислительная техника всех форм обучения. Оно будет использовано при изучении дисциплины «Дискретная математика». Для выполнения лабораторных работ студенты должны иметь навыки написания программ на языке C# в среде Microsoft Visual Studio.

Минимальные системные требования: ПЭВМ, работающая под управлением Windows; оперативная память – 512 Мб; необходимо на винчестере – 512 Мб; операционные системы- Windows XP/ Windows Vista/ Windows 7/ Windows 8/ Windows 10;

дополнительные программные средства- Adobe Acrobat Reader.

Работа подготовлена на кафедре автоматизированных систем обработки информации и управления ФГБОУ ВО «РГУ им. А.Н. Косыгина».

Лабораторная работа № 1

Операции над множествами

Цель работы - изучение операций над множествами: объединения, пересечения, дополнения и разности.

Понятие множества относится к фундаментальным неопределяемым понятиям. Оно было введено в математику создателем теории множеств немецким ученым Георгом Кантором. Следуя ему, под множеством понимается совокупность объектов произвольной природы, которая рассматривается как единое целое.

Чтобы задать множество, нужно указать какие элементы ему принадлежат. Это можно сделать следующими способами:

1. Перечислением элементов. Например, M = {Ваня, Маня, Петя} или B = {2,4,9,80}.

Данный способ приемлем только для конечных множеств. Причем, если один и тот же элемент перечислен несколько раз, он все равно считается одним элементом.

2. Заданием порождающей процедуры, например A = {2,4,8,…,2x,…} или A ={2,4,8,…}, закон порождения можно не указывать, если он очевиден.

3. Указанием свойств элементов множества. Например, множество четных чисел.

В этом случае, записывают так: M = {x|x – четное число}

Отметим, что одно и тоже множество может быть задано разными способами. Так мы задали множество четных чисел и указанием свойства элементов, и порождающей процедурой.

Два множества называются равными, если они состоят из одних и тех же элементов.

Например, M = {2,6}, A = {2,7,2}. Поскольку каждый элемент в множество может входить только один раз, то множества M и A состоят из одних и тех же элементов, значит M=A.

Множество B называется подмножеством A (или множество A включает множество B), если любой элемент из B принадлежит множеству A. Обозначается включение так: $B \subseteq A$.

Для иллюстрации включения множества в другое множество и выполнения операций над множествами используются диаграммы Эйлера -Венна. На них множества изображаются некоторыми фигурами, например, овалами.

Если $B \subseteq A$, но на диаграмме множества A и B можно изобразить следующим образом:

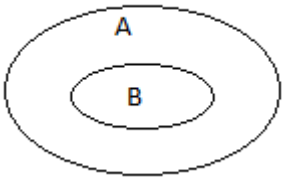


Рисунок 1. ***B⊆A***

Как правило, предполагается, что рассматриваемые множества содержатся в некотором большем множестве, называемым универсальным множеством, которое на диаграммах Эйлера-Венна, обозначается четырехугольником.

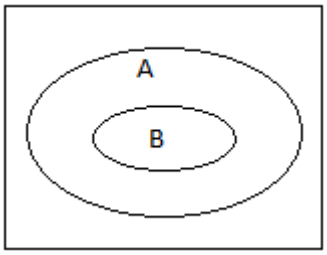


Рисунок 2. ***B⊆A, вместе с универсальным множеством***

Операции над множествами

Операция объединения.

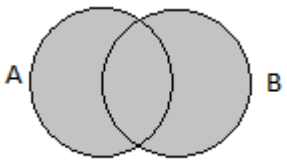


Рисунок 3. ***A∪B***

Объединением двух множеств *A,B* называют третье множество $A \cup B$, которое состоит из элементов входящих любо в множество *A*, либо в множество *B*.

$A \cup B = \{x \mid x \in A \text{ или } x \in B\}$.

На рисунке 3 построена диаграмма Эйлера- Венна объединения множеств.

Операция пересечения.

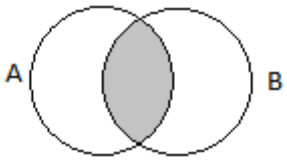


Рисунок 4. $A \cap B$

Пересечением двух множеств *A,B* называют третье множество $A \cap B$, которое состоит из элементов, входящих в множества *A,B* одновременно.

$A \cap B = \{x \mid x \in A \text{ и } x \in B\}$

На рисунке 4 построена диаграмма Эйлера- Венна пересечения множеств.

Операция разности.

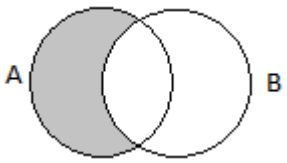


Рисунок 5. $A \setminus B$

Разностью двух множеств *A, B* называют третье множество $A \setminus B$, которое состоит из элементов, входящих в множество *A*, но не входящих в множество *B*.

$A \backslash B = \{x \mid x \in A \text{ и } x \notin B\}$.
На рисунке 5 построена диаграмма Эйлера- Венна разности множеств.

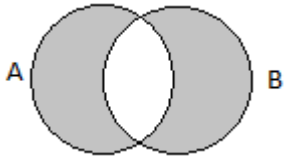


Рисунок 6. $A \oplus B$

Операция симметрической разности.
Симметрической разностью множеств *A, B* называют множество $A \oplus B$, которое состоит из элементов входящее в одно из этих множеств и не входящих в другое.
$A \oplus B = \{x \mid x \in A \text{ и } x \notin B \text{ или } x \in B \text{ и } x \notin A\}$
На рисунке 6 построена диаграмма Эйлера- Венна симметрической разности множеств.

Операция дополнения.

Эта операция определяется только при допущении, что существует универсальное множество. Универсальное множество, как правило, ограничивает некоторую область выбора объектов. Например, если мы рассматриваем множества людей, то универсальным множеством может быть множество всех людей или множество млекопитающих, или множество всех живых существ.

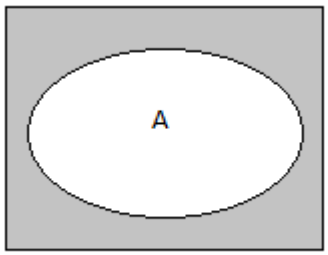


Рисунок 7. $\bar{A}$

Дополнением множества *A* называется множество $\bar{A}$, которое состоит из элементов, которые не входят в множество *A*.
$\bar{A} = \{x \mid x \notin A\}$
В определении не упоминается универсальное множество, но его существование заранее его оговаривается, поэтому можно сказать, что $\bar{A} = U \backslash A$. Если не рассматривать универсального множества, то мы можем получить парадоксальные (противоречивые) утверждения.
На рисунке 7 построена диаграмма Эйлера- Венна дополнения множеств A, универсальное множество, как правило, изображается прямоугольником.

Linq -запросы, реализующие операции над множествами

Для использования Linq –запросов надо подключить библиотеку System.Linq.

При выполнении лабораторной работы могут потребоваться следующие операции:

Операция *Distinct*, которая удаляет дублированные элементы из входной последовательности.

```
IEnumerable<int> nums = arr.Distinct();
```

Операция *Union*, которая возвращает объединение множеств из двух исходных множеств, представленных в виде последовательностей. *IEnumerable<string> union = first.Union<string>(second);*

Операция *Intersect*, которая возвращает пересечение множеств из двух исходных множеств, представленных в виде последовательностей. *IEnumerable<string> auto = first.Intersect(second);*

Операция *Except*, которая возвращает последовательность, содержащую все элементы первой последовательности, которых нет во второй последовательности.
IEnumerable<int> nums = arr1.Except<int>(arr2);

Поскольку элементы множества будут храниться в массивах, то потребуется функция *ToArray();*
Mass= nums. ToArray();
Для вывода элементов массива на экран удобно использовать оператор *foreach.* Например.

```
foreach (int i in Mass)
 textBox5.Text+=i.ToString()+",";
```

Этапы выполнения лабораторной работы

Лабораторная работа состоит из двух заданий.
Этапы выполнения задания №1.

В индивидуальном задании дана формула алгебры множеств (формулу нельзя преобразовывать!).

1. Написать и отладить программу, которая позволяет выполнить действия, задаваемые формулой, определенной вариантом задания. Множества можно задать один раз внутри программного кода, универсальное множество определяется как объединение всех заданных множеств. На экран должны быть выведены сами множества и результаты вычислений «по действиям». Для каждого действия определяется отдельная кнопка и отдельное поле вывода.
2. Разработать тестовый пример. Для этого:
    a. Задать 4 множества натуральных чисел A, B, C, D. В формуле может присутствовать 3 множество, но задать надо все 4 множества. Значения, для задания множеств, нужно брать на интервале [10*n,10*n+20], где n - номер варианта. Задавать множества можно вручную, а можно с помощью генератора случайных чисел. Все множества должны попарно пересекаться, так же должны

пересекаться все тройки множеств. Пересечение всех 4-х множеств может быть пустым.
При поверке программы преподаватель может изменить любое множество.

b. Универсальное множество определяется, как объединение всех заданных множеств.

3. Запустить программу на тестовом примере, сделать скрины, демонстрирующие работу программы по вычислению формулы.
4. В отчете для каждого скрина, на котором показан результат вычислений по определенной формуле, построить диаграмму Эйлера-Венна.
5. Обязательно указать, каким способом закрашено итоговое множество. На диаграмму нанести числа – элементы тестовых множеств. Например, пусть множество A={1,2,3,4}, B={3,4,5,6}, тогда разность A\B на рисунке 6 закрашена серым цветом.

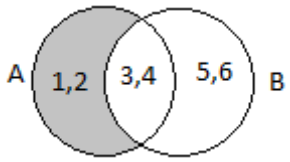


Рисунок 6. Диаграмма Эйлера-Венна с перечисленными элементами.

Этапы выполнения задания №2.

1. Построить диаграмму Эйлера-Венна для множества, заданного формулой расположенной слева от знака = (или $\subseteq$).
2. Построить диаграмму Эйлера-Венна для множества, заданного формулой расположенной справа от знака = (или $\subseteq$).
3. Сравнить итоговые множества обеих формул и сделать выводы.

Пример программы.

Если выполнить последовательные нажатия на кнопки button1, button2, button3, то получим вычисления по формуле $C \cup \overline{A \cap B}$. Отметим, что программа не имеет защиты от путаницы в порядке действий.

```
using System;
using System.Linq;

namespace Операции_над_множествами
{
   public partial class MainForm : Form
   {
      int[] A={1,2,3,7};
```

```
        int[] B={2,3,4,5,6};
        int[] C= {3,4,6,7,8};
        int[] U;
        int[] D;
         void MainFormLoad(object sender, EventArgs e)
        {
           foreach(int i in A)
              textBox1.Text+=i+",";
           foreach(int i in B)
              textBox2.Text+=i+",";
           foreach(int i in C)
              textBox3.Text+=i+",";
           U= A.Union(B).Union(C).ToArray();
            foreach (int i in U)
            textBox4.Text+=i+",";
        }
        void Button1Click(object sender, EventArgs e)
        {
           D=A.Intersect(B).ToArray();
           foreach (int i in D)
            textBox5.Text+=i+",";
        }
        void Button2Click(object sender, EventArgs e)
        {
           D=U.Except(D).ToArray();
           foreach (int i in D)
            textBox6.Text+=i+",";
        }
        void Button3Click(object sender, EventArgs e)
        {
           D=D.Union(C).ToArray();
           foreach (int i in D)
            textBox7.Text+=i+",";
        }
    }
}
```

Замечания по разработке интерфейса программы.

a. На форму должны выводиться элементы исходных множеств и элементы универсального множества.
b. Кнопки, выполняющие вычисления по определенной формуле, должны быть подписаны. Можно рядом с кнопкой поместить компонент PictureBox и на него вывести саму формулу.
c. Результаты всех вычислений должны выводиться на форму.

Содержание отчета

1. Задание №1.
2. Текст программы
3. Скрины, демонстрирующие работу программы.
4. Диаграммы Эйлера-Венна.
5. Задание №2.
6. Диаграммы Эйлера-Венна.
7. Выводы.

Варианты формул задания №1.

1	$C \cap B \cup \overline{A \cup B \cap C}$
2	$\bar{A} \cap \bar{B} \cap C \cup A \cap \bar{C}$
3	$\overline{A \cap B} \cap \overline{B \cup C}$
4	$(\bar{A} \cup B) \cap (B \cup \bar{C}) \cup \bar{A}$
5	$C \cap A \cup \overline{A \cap B \cup C}$
6	$C \cap B \cup \overline{A \cup B}$
7	$\bar{B} \cup A \cap \bar{C} \cup A \cap B$
8	$A \cap B \cup \overline{C \cup B}$
9	$D \cup \overline{D \cup B \cup C}$
10	$B \cup A \cap \bar{C} \cup \bar{B} \cap C$
11	$D \cap A \cup \overline{A \cup B \cup D}$

12	$A \cap (\overline{B} \cup \overline{C})$
13	$A \cap \bar{C} \cup \bar{B} \cap C \cap \bar{A}$
14	$A \cap \overline{B} \cap \overline{C}$
15	$\bar{A} \cap B \cup B \cap C$
16	$C \cup A \cap \overline{B \cup C}$
17	$\bar{C} \cup \bar{B} \cap C \cup \bar{B} \cap \bar{A}$
18	$A \cup B \cup \bar{A} \cap \bar{B} \cap C$
19	$\bar{A} \cap B \cup A \cap \bar{C} \cup \bar{B} \cap C$
20	$(B \cup \bar{C}) \cup (\bar{A} \cup \bar{B}) \cap (\bar{B} \cup C)$
21	$A \cup \overline{A \cap B \cup C}$
22	$A \cap B \cup \bar{A} \cap \bar{B} \cap C$

Варианты формул задания №2.

1	$A \cap (B \cup C) \subseteq A \cap B \cup C$
2	$\overline{A \cup B \cap \overline{C}} = \bar{A} \cap (\bar{B} \cup C)$
3	$(A-B) \oplus (B \cap (A \oplus C)) = (A-C) \oplus (C \cap (A \oplus B))$;
4	$(A-B) \oplus (B-A) = (B-A) \cup (A-B)$
5	$A \oplus B = (A \cup \bar{B}) \oplus (B \cup \bar{A})$
6	$(A \cup B) - C \subseteq A \cup (B-C)$
7	$(A \cap C) \oplus (B \cap D) \subseteq (A \oplus B) \cup (C \oplus D)$
8	$(A-B)-C = (A-C)-(B-C)$
9	$(A-B) \oplus (B-C) \oplus (C-A) = (A-C) \oplus (C-B) \oplus (B-A)$;
10	$(A \oplus B) - C = (A-(B \cup C)) \cup (B-(A \cup C))$;
11	$(A-B) \cap (A-C) = A-(B \cup C)$

12	$(A-B)\oplus(B-C)\oplus(B-A)\oplus(C-B)=A\oplus C$
13	$A-(B-C)\subseteq A\cup B\cap C$
14	$A\cap C=(A-(B-C))-(A-B-C)$
15	$(A\cup C)\oplus(B\cup D)\subseteq(A\oplus B)\cup(C\oplus D)$
16	$\overline{A\cup B\cap\overline{C}}\cap B=\overline{A}\cap B\cap C$
17	$A\cap B\oplus(A\cup B)=(\overline{A}\cup B)\oplus(A\cup\overline{B})$
18	$B\cap C=(\overline{B}\cap A\cup C)\cap(B\cup\overline{A\cup C})$
19	$(A-C)\cap(B-D)=A\cap B-(C\cup D)$
20	$(A-C)\cup(B-A)\subseteq A\cup B$
21	$(\overline{A}\cup B)\cap(\overline{B}\cup C)\subseteq\overline{A}\cup C$
22	$\overline{A\cap B\cap\overline{C}}\cap B=\overline{A}\cap B\cup C\cap B$
23	A - (B ⋂C) = (A - B) ⋂ (A - C)
24	A - (A - B) = A⋂ B

Лабораторная работа № 2

Бинарные отношения

Цель работы - изучение декартово произведения множеств и бинарных отношений.

Декартово произведение

Декартовым произведением двух множеств *A, B* называется множество упорядоченных пар <*a,b*>, таких, что первый элемент пары принадлежит *A,* а второй элемент пары множеству *B*. Декартово произведение множеств *A,B* обозначается *A*×*B*

Другими словами $A\times B=\{<a,b>\mid a\in A,\ b\in B\}$

Пример 1. Если *A={1,2,4}, B={1,7},* то
$A\times B=\{<1,1>,<2,7>,<1,7>,<2,1>,<4,1>,<4,7>\}$.

Пример 2. Если *R* – множество действительных чисел, то *R*×*R* множество всевозможных пар действительных чисел. Множество действительных чисел можно представить, как множество точек на прямой, в этом случае декартово произведение можно представить, как множество точек на плоскости.

Если рассматривать некоторый набор множеств $A_1,\ldots A_n$, то можно определить декартово произведение множеств $A_1,\ldots,A_n$, как множество последовательностей, содержащих n элементов, причем первый элемент должен быть из A_1, второй из A_2, …, n –элемент из A_n.

Другими словами,
$A_1\times A_2\times\ldots\times A_n=\{<a_1,\ldots>a_n>|a_1\in A_1,\ldots,a_n\in A_n\}$.

Пример 3. Если $A=\{2,4\}$, $B=\{1,7\}$, C={1,2},то $A\times B\times C$ $=\{<2,1,1>,<2,7,1>,<2,1,2>,<2,7,2><4,7,1>,<4,1,1>,<4,1,2>,<4,7,2>\}$.

Пример 4. Если R – множество действительных чисел, то $R\times R\times R$ будет множеством всевозможных троек чисел, а геометрически будет множеством точек в 3-х мерном пространстве.

Отношение на множестве

Отношение – это способ задания взаимосвязи между элементами либо одного множества, либо разных множеств. Например, «а больше в», «а равно в», «прямая а параллельна прямой в», «Петя учится лучше, чем Маша» и т. д. Элементы а и в могут принадлежать одному множеству, а могут принадлежать разным множествам. Отношение может связывать два элемента, в этом случае оно называется бинарным, а может связывать любое заранее оговоренное число элементов, в этом случае оно называется n- местным отношением, где n – это число элементов, связанных отношением. Рассматривается и одноместное (унарное) отношение, в этом случае оно будет определять некоторое свойство элементов.

Пусть даны множества А и В, если на элементах этих множеств определено отношение Р, то рассматривая различные пары <а, в> , где а элемент множества А, а в элемент множества В, мы можем определить находится ли элемент а в отношении Р к элементу в или нет. Поскольку множество всех пар <а, в> , где а элемент множества А, а в элемент множества В – это декартово произведение множеств А и В, (А×В) то отношение Р можно задать как подмножество А×В. Именно так дается строгое определение n- местного отношения.

Всякое подмножество L декартова произведения $A_1\times A_2\times\ldots\times A_n$ произвольных множеств $A_1,\ldots A_n$, называется отношением, определенным на множествах $A_1,\ldots,A_n$.

Если $<a,b>\in L$, то говорят, что элемент a находится в отношении L к элементу b или что отношение L для a,b выполняется.

Вместо $<a,b>\in L$ пишут также aLb или $L(a,b)$.

Поскольку отношения, заданные на фиксированной паре множеств A,B суть подмножества множества $A\times B$, то можно рассматривать их объединение, пересечение, дополнение.

Пусть L,M подмножества $A\times B$, тогда

1. $a(L\cup M)b \Leftrightarrow <a,b> \in L\cup M.. \Leftrightarrow. <a,b> \in L$ или $<a,b> \in M$, связка «или» в нашем случае не исключающая, т.е. логическая операция дизъюнкция. Поэтому вместо объединения отношений чаще говорят дизъюнкция отношений.

2. $a(L\cap M)b \Leftrightarrow <a,b> \in L\cap M.. \Leftrightarrow. <a,b> \in L$ *и* $<a,b> \in M$. Вместо пересечения отношений чаще говорят конъюнкция отношений.

3. $a\overline{L}b \Leftrightarrow <a,b> \notin L$. Вместо дополнения отношения, можно сказать отрицание отношения.

Если L – отношение, определенное на паре множеств A,B, то обратным отношением (символически L^{-1}) называется отношение определенное на паре множеств B, A, которое состоит из тех пар $<b,a>$, для которых $<a,b> \in L$, т.е. $bL^{-1}a \Leftrightarrow aLb$.

Если $B=A$, отношение называется бинарным (двуместным) отношением на множестве A.

Например, отношение равенства, определенное на множестве натуральных чисел N, можно понимать как совокупность всех диагональных пар $<1,1>,<,2,2>,...$ Отношение порядка $<$, есть множество пар $<a,b>$, таких, что $a<b$. Если S – множество людей, то множество супружеских пар будет подмножеством $S^{\times}S$, и будет отношением.

Свойства бинарных отношений.

1. Бинарное отношение L на множестве A называется *рефлексивны*м, если для любого a из A верно aLa (т.е. $<a,a> \in L$).
   Например. Если на числах рассмотреть отношение ($\leq$) нестрогого меньше, то оно будет рефлексивным, поскольку для любого числа a, верно $a \leq a$.

2. Бинарное отношение L на множестве A называется *антирефлексивн*ым, если для любого a из A не выполняется $a\ L\ a$.
   Примером антирефрексивного отношения будет отношение строгого меньше ($<$) на числах.

3. Бинарное отношение L на множестве A называется *симметричным*, если $bLa \Leftrightarrow aLb$ для любых элементов a, b из A.
   Отношение $\leq$ на числах не будет симметричным. А вот отношение супружеских пар на множестве людей будет симметричным.

4. Бинарное отношение *L* на множестве *A* называется *антисимметричным,* если *bLa & aLb =>b=a,* для любых элементов *a, b* из A
   Несложно проверить, что отношение нестрогого меньше будет антисимметричным отношением.

5. Бинарное отношение *L* на множестве *A* называется *транзитивным*, если

aLb& bLc =>aLc, для любых *a, b, c* из A.

Контрольные вопросы

1. Пусть множество A содержит 2 элемента, множество B содержит 3 элемента. Сколько существует бинарных отношений между множествами A и B?
2. Привести пример бинарного отношения, которое:
   a) рефлексивно, симметрично, не транзитивно;
   b) не рефлексивно, антисимметрично, не транзитивно;
   c) рефлексивно, не симметрично, транзитивно.
3. Определите, какие из следующих отношений на множестве людей рефлексивны, симметричны или транзитивны:
   a) «… учится в то же группе, что и …»;
   b) «… являться сестрой …»;
   c) «… старше, чем …»;
   d) «… не выше, чем …».
4. Существует ли отношение, которое одновременно не является симметричным и не является антисимметричным?
5. Существует ли отношение, которое одновременно не является рефлексивным и не является антирефлексивным?

Общее задание

1. Написать программу, которая демонстрирует бинарное отношение на множестве как некоторое подмножество декартово квадрата этого множества (A×A). Для этого задать некоторое множество чисел. Это множество можно задать один раз внутри программного кода, но при защите лабораторной работы, преподаватель может попросить изменить его. На экран вывести элементы декартова квадрата исходного множества, причем элементы, составляющие заданное вариантом отношение должны быть каким-то образом выделены (например, цветом).
2. Определить свойства исследуемого отношения.

Варианты индивидуальных заданий

Множество A – конечное множество целых чисел, a, b ϵ A.

1. Элемент a находится в отношении P к элементу b, если a + 2 < b;
2. Элемент a находится в отношении P к элементу b, если a ≤ b+2;
3. Элемент a находится в отношении P к элементу b, если произведение a*b является четным числом;
4. Элемент a находится в отношении P к элементу b, если произведение a*b является нечетным числом;
5. Элемент a находится в отношении P к элементу b, если $a + b$ – нечетное число;
6. Элемент a находится в отношении P к элементу b, если $a + b$ – четное число;
7. Элемент a находится в отношении P к элементу b, если ***a*** делится на ***b*** без остатка***;***
8. Элемент a находится в отношении P к элементу b, если ***a*** не делится на b без остатка;
9. Элемент a находится в отношении P к элементу b, если ***a*** равно остатку от деления b на 10;
10. Элемент a находится в отношении P к элементу b, если ***a*** равно остатку от деления ***b*** на 8;
11. Элемент a находится в отношении P к элементу b, если $a^2 + b^2 \leq 4$;
12. Элемент a находится в отношении P к элементу b, если $| a - b | = 2$;
13. Элемент a находится в отношении P к элементу b, если $a^2 + b^2 \geq 4$;
14. Элемент a находится в отношении P к элементу b, если ***a*** при делении на b дает в остатке 3;
15. Элемент a находится в отношении P к элементу b, если ***a*** при делении на b не дает в остатке 3;
16. Элемент a находится в отношении P к элементу b, если ***a*** и b одного знака;
17. Элемент a находится в отношении P к элементу b, если ***a*** и b разного знака;
18. Элемент a находится в отношении P к элементу b, если ***a*** и b***-2*** одного знака;
19. Элемент a находится в отношении P к элементу b, если ***a*** является квадратом ***b***;
20. Элемент a находится в отношении P к элементу b, если ***a*** является кубом ***b***;
21. Элемент a находится в отношении P к элементу b, если разность ***a – b*** отрицательна;

Пример программы.

Использование таблицы для вывода элементов декартового произведения. В комментариях записывается то, что делает та или иная команда.

```
namespace Таблица
{    public partial class MainForm : Form
```

```
   {
      int[] A={2,4,5,6,8,7};
   int[] B={11,2,-3,-5,-6};
      public MainForm()
      {          InitializeComponent();
        }
      void MainFormLoad(object sender, EventArgs e)
      {
//Задание числа столбцов и числа строк таблицы
          dataGridView1.ColumnCount=A.Length;
          dataGridView1.RowCount=B.Length;
//Определения ширины столбцов
          dataGridView1.AutoSizeColumnsMode= DataGridViewAutoSizeCol-
umnsMode.Fill;
          dataGridView1.RowHeadersWidth=60;
//Запреты на добавление и удаление строк пользователем
          dataGridView1.AllowUserToAddRows=false;
          dataGridView1.AllowUserToDeleteRows=false;
//Заголовки столбцов
          for(int i=0;i< A.Length;i++)
             dataGridView1.Columns[i].HeaderText=A[i].ToString();
//Заголовки строк
          for(int i=0;i< B.Length-1;i++)
             dataGridView1.Rows[i].HeaderCell.Value=B[i].ToString();
//Запись в ячейки таблицы
          data-
GridView1.Rows[0].Cells[0].Value=A[0].ToString()+","+B[0].ToString();
          dataGridView1[1,0].Value=5;
dataGridView3.Rows[0].Cells[0].Style.ForeColor = Color.Aqua; //закраска
      }
   }
}
```

Содержание отчета

1. Задание.
2. Текст программы
3. Скрины, демонстрирующие работу программы.
4. Описание свойств, исследуемого отношения.
5. Выводы.

Лабораторная работа № 3

Матричные способы представления графов

Цель работы - изучить способы задания графов, которые часто используются при написании программ.

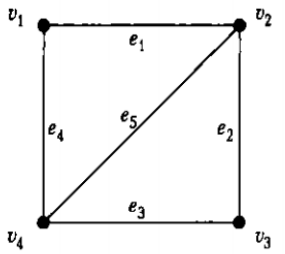
Матрица смежности

Матрица смежности – это квадратная матрица размерности $n \times n$, значения элементов которой характеризуются смежностью вершин графа. (n-это количество вершин графа). Элементы матрицы определяются следующим образом:

$$s_{i,j} = \begin{cases} 1, \textit{если вершины} v_i \textit{ и } v_j \textit{ смежные}; \\ 0, \textit{в противном случае}; \end{cases}$$

Если рассматривается псевдограф, то вместо единиц ставится число кратных ребер, соединяющих вершины v_i и v_j.

Пример. На рисунке 32 дан граф и его матрица смежности.



	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	1
v_3	0	0	1	1
v_4	1	1	1	0

Рисунок 3. Граф и его матрица смежности.

Матрица смежности графа симметрична относительно главной диагонали, поэтому достаточно хранить только верхнюю (или нижнюю) треугольную матрицу.

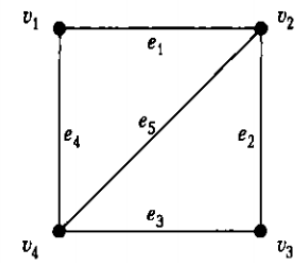
Матрица инцидентности

В матрице инцидентности хранятся связи между инцидентными элементами – вершинами и ребрами. Размерность этой матрицы $m \times n$, где m - количество ребер, n - количество вершин. Столбцы матрицы соответствуют вершинам, строки – ребрам. Ненулевое значение в ячейке матрицы указывает связь между вершиной и ребром. Данный способ является самым емким для хранения, но облегчает нахождение циклов в графе.

Если рассматривается неориентированный граф, то матрица инцидентности определяется так:

$$s_{i,j} = \begin{cases} 1, \text{если } i-\text{тое ребро инцидентно } j-\text{той вершине}; \\ 0, \text{в противном случае}; \end{cases}$$

Пример. На рисунке 33 представлены граф и его матрица инцидентности.



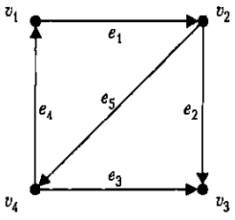
	v_1	v_2	v_3	v_4
e_1	1	1	0	0
e_2	0	1	1	0
e_3	0	0	1	1
e_4	1	0	0	1
e_5	0	1	0	1

Рисунок 4. Граф и его матрица инцидентности.

При рассмотрении орграфов учитывается направление дуг:

$$s_{i,j} = \begin{cases} 1, \text{если } i-\text{тая дуга заканчивается в } j-\text{той вершине}; \\ 0, i-\text{тая дуга и } j-\text{тая вершина не инцидентны}; \\ -1, \text{если } i-\text{тая дуга начинается в } j-\text{той вершине}; \end{cases}$$

Пример. На рисунке 34 представлен орграф и его матрица инцидентности.



	v_1	v_2	v_3	v_4
e_1	-1	1	0	0
e_2	0	-1	1	0
e_3	0	0	1	-1
e_4	-1	0	0	1
e_5	0	-1	0	1

Рисунок 5. Ориентированный граф и его матрица инцидентности.

Задание

На форму нужно поместить 2 таблицы (компоненты DataGridView) одну для матрицы смежности, другую для матрицы инцидентности. Одна из таблиц предназначена для ввода значений пользователем, другая для вывода расчетных данных. Кроме таблиц на форме должны быть компонент для вывода рисунка графа и кнопки управления.

Поскольку количество вершин фиксировано и определяется вариантом задания, то вершины графа можно нарисовать при запуске программы.

Пользователь задает только ребра графа. У студентов с чётными номерами варианта задания в качестве входной матрицы должна быть матрица смежности графа. У студентов с нечётными номерами - матрица инцидентности. Каждый студент работает с графом, у которого количество вершин вычисляется по формуле (n/2)+4, где n-вариант задания.

Пользователь задаёт матрицу смежности (инцидентности) графа. Программа должна проверить корректность введённой матрицы. Если пользователь ошибётся, то программа должна сообщить об этом. Построить матрицу инцидентности (смежности) графа и вывести рисунок графа на экран.

Пример проверки корректности ввода данных в таблицу

Пользователь должен вводить данные в таблицу, и эти данные должны быть проверены программой, поэтому часть проверок можно сделать в момент ввода. Поскольку пользователь должен вводить в таблицы только цифры 0 и 1, то приводится пример программы, которая не позволяет ввести в таблицу никаких других символов, кроме 1 и 0. Заголовки столбцов и строк хранятся в массивах A и B, соответственно. Поскольку вершины и ребра графа нумеруются (вершины – четными числами, а ребра - нечетными), то массивы имеют тип int.

```
namespace ТаблицаДМ
{
   public partial class MainForm : Form
   {
   int[] A={1,3,5,7,9};
   int[] B={2,4,6,8};
   int nom;
      public MainForm()
      {          InitializeComponent();
        }

      void MainFormLoad(object sender, EventArgs e)
      {
         dataGridView1.ColumnCount=A.Length;
         dataGridView1.RowCount=B.Length;
         dataGridView1.AutoSizeColumnsMode=     DataGridViewAutoSizeCol-
umnsMode.Fill;
         data-
GridView1.RowHeadersWidth=60;          for(int i=0;i< A.Length;i++)
             dataGridView1.Columns[i].HeaderText=A[i].ToString();
```

```
            for(int i=0;i< B.Length;i++)
               dataGridView1.Rows[i].HeaderCell.Value=B[i].ToString();
            data-
GridView1.Rows[0].Cells[0].Value=A[0].ToString()+","+B[0].ToString();
            dataGridView1[1,0].Value=5;
            nom=dataGridView1.RowCount;
        }

        TextBox tb;
        void DataGridView1EditingControlShowing(object sender, DataGridView
EditingControlShowingEventArgs e)
        {
              tb = (TextBox)e.Control;
              tb.KeyPress += new KeyPressEventHandler(tb_KeyPress);

              if(dataGridView1.RowCount>nom) dataGridView1.Rows[dataGridView
1.RowCount-1].HeaderCell.Value=textBox1.Text;
              if(dataGridView1.RowCount<nom) nom=dataGridView1.RowCount;
        }
        void tb_KeyPress(object sender, KeyPressEventArgs e)
        {
    if (( e.KeyChar != 49||tb.Text.Length>0) && e.KeyChar != 8 && e.KeyChar!=1
27)
               { e.Handled = true;}
        }
     }
}
```

Содержание отчета

1. Задание
2. Текст программы
3. Скрины, демонстрирующие работу программы.
4. Выводы.

Контрольные вопросы

1. Определить матрицу смежности графа.
2. Определить матрицу инцидентности графа.
3. Что такое граф?
4. Чем отличается граф от мультиграфа?

5. Чем отличается граф от псевлографа?
6. Привести пример псевдографа, не являющегося мультиграфом.
7. Играет ли роль место положение вершин, при определении графа?
8. Играет ли роль длина и форма ребер, при определении графа?
9. Привести пример графа, содержащего 5 вершин и 7 ребер.

Лабораторная работа № 4

Работа с деревьями

Цель работы: изучить библиотечные классы языка C#, предназначенные для работы с деревьями и научится их использовать в программе.

Дерево, как модель иерархических отношений

Мы определили, дерево, как связный граф без циклов. Все вершины дерева равноправны. Но для моделирования иерархических отношений, которые часто встречаются как в практической жизни, так и в математике, и в программировании, используется дополненное определение дерева. В этом случае вершины называют, как правило, узлами, а ребра задают отношения, которые называют родительскими. Один узел выделен как корень.

Если рассматривать дерево в старом определении, то для того что бы получить дерево в новом определении, надо одну вершину выделить, назвав ее корнем, а смежные ей вершины назвать сыновьями. Для этих узлов корень будет родителем. Корень не имеет родителя. Т.е. вершины, получив название узлы, некоторым образом упорядочиваются. Если дерево представить веревочной моделью, в которой вершины узелки, а ребра нитки, то в этом случае, взяв за узелок, названный корнем и встряхнув за нее все дерево, мы получим дерево в новом определении. Итак, корень может иметь сыновей.

Каждый из его сыновей может иметь своих сыновей, т.е. быть родителем для этих узлов и т.д. Каждый узел, кроме корня, имеет одного родителя. Узлы, не имеющие сыновей, называются листьями.

Итак, новое определение дерева отличается от старого, тем, что узлы упорядочиваются отношением, как правило, называемым родительским.
Примеры деревьев.

1. Структура вложенности каталогов и файлов в современных операционных системах.
2. Дерево синтаксического разбора.
3. Иерархическая структура вложенных элементов данных.
4. Иерархия классов.

Библиотечные классы, предназначенные для работы с деревьями

Для работы с деревьями в библиотеке языка C# введен класс *TreeNode*. Объекты, которого имеют двойственную природу, их можно рассматривать или как поддерево, или как узел, который совпадает с корнем этого поддерева. Свойство *Text* определяет имя корня. Свойство *Nodes* – является списком всех сыновей корня этого поддерева. Каждый сын – это в свою очередь объект типа *TreeNode* Удаление узла приводит к удалению всех его потомков, т.е. всего поддерева, корнем которого является удаляемый узел. Реализация такой структуры возможна, поскольку в списке *Nodes* хранятся ссылки на узлы, являющиеся сыновьями.

Рассмотрим пример создания дерева.

TreeNode t=new TreeNode("Корень");

Данная команда создает узел – объект типа *TreeNode* , который будет корнем нашего дерева.

TreeNode t1=new TreeNode("C1");

t.Nodes.Add(t1);

Данная команда добавляет узел *t1* к дереву *t.* Теперь к узлу *t1* можно обращаться и по ссылке *t1* и по номеру *0* в списке *t.Nodes,* кроме того, к этому узлу можно обратиться через свойство *t.FirstNode* поскольку данный узел стоит в списке первым. Свойство *t.LastNode* возвращает ссылку на последний элемент в списке *t.Nodes*. Свойство *t.Parent* возвращает ссылку на родительский узел, если t корень, она будет равна *null*.

Свойство *t1.Index* возвращает номер узла *t1*, в списке *Nodes.*

Для просмотра дерева и работы с ним создан компонент *TreeView,* который является контейнером для объектов *TreeNode* Котпонент *TreeView* имеет свойство *Nodes*, которое является списком объектов *TreeNode*. Компонент *TreeView* визуальный - это прямоугольник, в котором нарисовано дерево, похожее на дерево папок и файлов. Каждый узел может быть свернутым и развернутым, может иметь имя и значок. Свойства для чтения *IsExpanded, IsVisible; IsSelected;* имеют булев тип и сообщают, является ли узел развернутым, видимым и выделенным, соответственно.

Чаще всего работа происходит с выделенным узлом, поэтому класс *TreeView* имеет свойство *SelectedNode*, которое указывает на выделенный узел, это свойство предназначено и для чтения, и для записи.

Команда *treeView1.SelectedNode.Remove();* удалит выделенный узел, а *treeView1.SelectedNode.Text="op";* переименует выделенный узел.

У класса *TreeView* есть события, связанные с действиями над деревьями :

AfterSelect, BeforeSelect, AfterCollapse BeforeCollapse, AfterExpand BeforeExpand, Первые два события происходят при выделении узла. Событие *BeforeSelect* до выделения, *AfterSelect* - после выделения. Следующие два события связаны со сворачивание узла, а последние два с разворачиванием узла.

Все обработчики этих событий имеют формальный параметр e, который, через свойство *Node* передает ссылку на узел, с которым происходит соответствующее событие.

Например,

```
private void treeView1_BeforeCollapse(object sender, Sys-
tem.Windows.Forms.TreeViewCancelEventArgs e)
        {
        label2.Text=e.Node.Text;
        }
```

Данный фрагмент программы в момент сворачивания узла будет выводить его имя на экран.

Задание

Общее задание.

На форму помещен пустой компонент treeView1 и 3 кнопки.

При нажатии на первую кнопку в компоненте treeView1 появляется дерево, определенное вариантом.

Задание состоит из 4 частей. В первой части указывается обработчик, какого события должен быть создан. Вторая часть задания определяет работу второй кнопки, третья часть задания – работу третьей кнопки. Четвертая часть задания определяет дерево, которое выводится при нажатии первой кнопки.

Первая часть задания. Работа с событиями свертывания, развертывания и выделения узлов.

1. В момент свертывания узла, его имя выводится на форму.
2. В момент развертывания узла, его имя выводится на форму.
3. В момент выделения узла, его имя выводится на форму.

Вторая часть задания. Это задание определяет работу первой кнопки

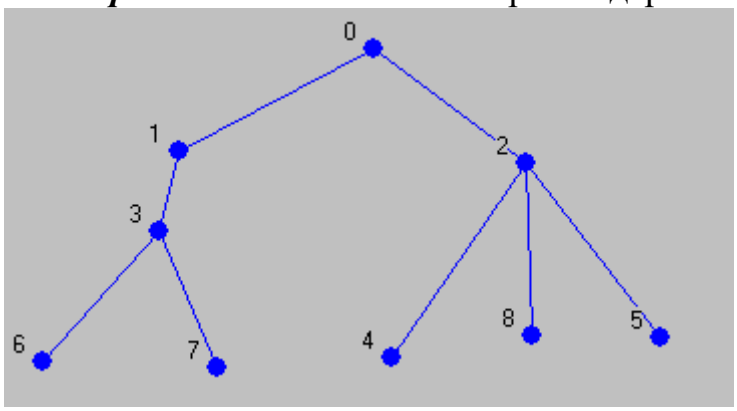
1. выводит на форму имя выделенного узла.
2. удаляет выделенный узел.
3. добавляет сына к выделенному узлу. Имя сына задается пользователем.

Третья часть задания. Кнопка, которая выполняет, (если возможно, если не возможно, то выводит сообщение об этом) следующее действие:

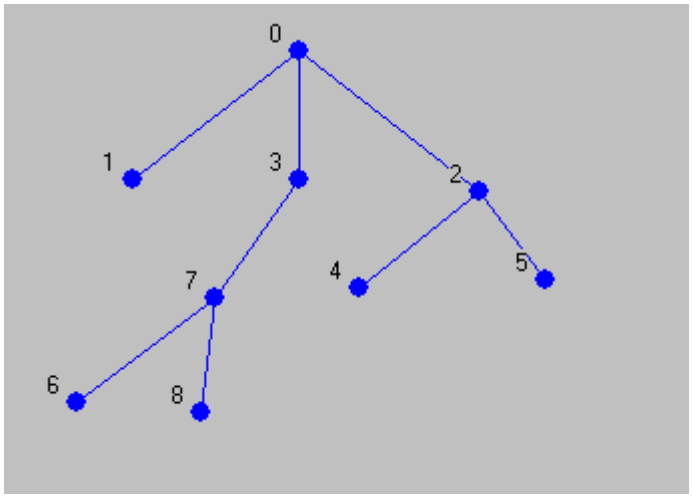
1. На форму выводит список имен сыновей выделенного узла.
2. На форму выводит имя родителя выделенного узла.
3. На форму выводит имя первого сына выделенного узла.
4. На форму выводит имя последнего сына выделенного узла.
5. На форму выводит имя левого брата выделенного узла.
6. На форму выводит имя правого брата выделенного узла.
7. Сообщает, является ли выделенный узел листом.
8. Выясняет, имеет ли выделенный узел братьев.
9. Удаляет первого сына выделенного узла.
10. Удаляет последнего сына выделенного узла.
11. Удаляет всех сыновей выделенного узла, сам узел оставляет на месте.

12. Удаляет левого брата выделенного узла.
13. Удаляет правого брата выделенного узла.

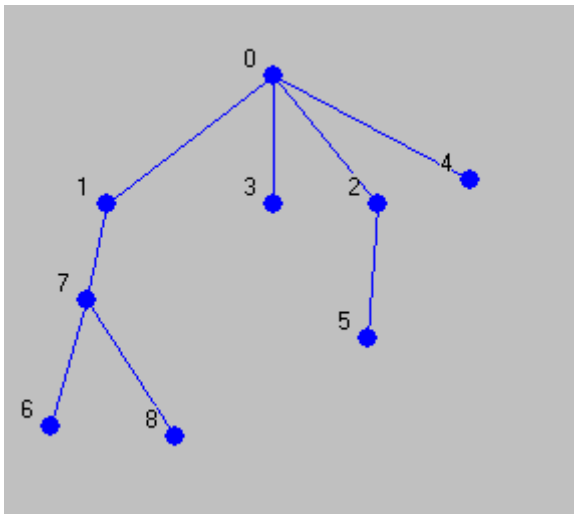
Четвертая часть задания. Вариант дерева.



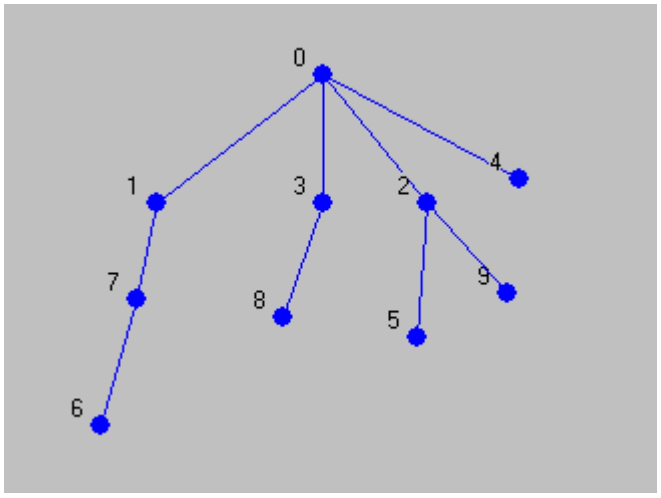
Вариант 1



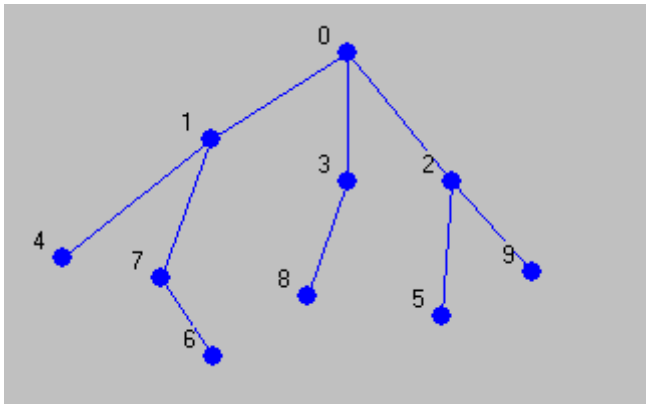
Вариант 2



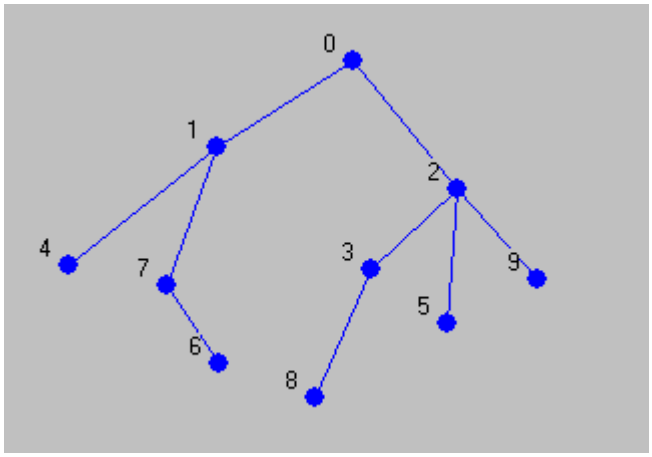
Вариант 3



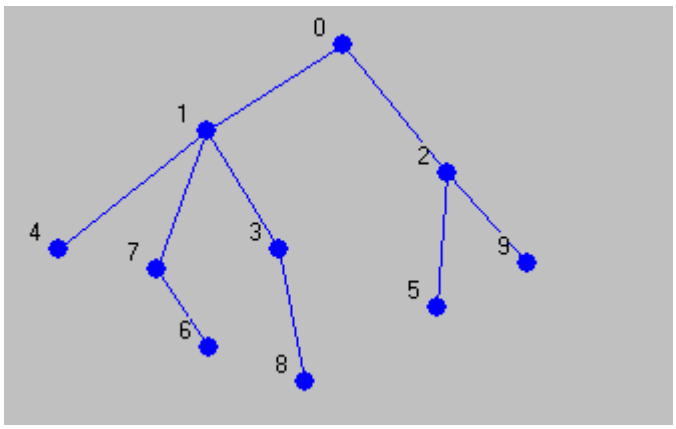
Вариант 4



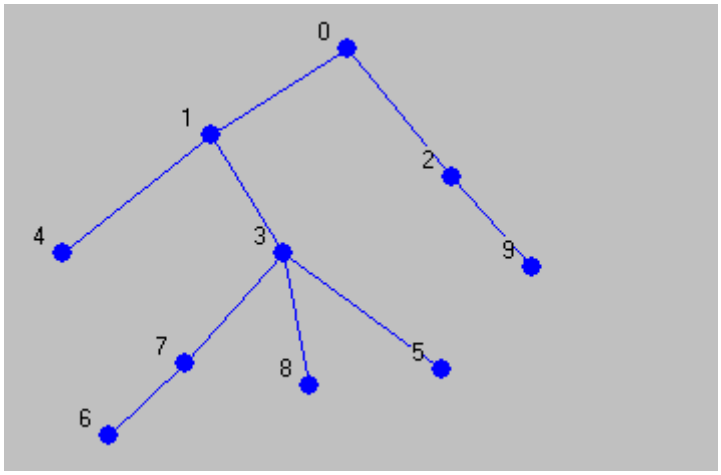
Вариант 5



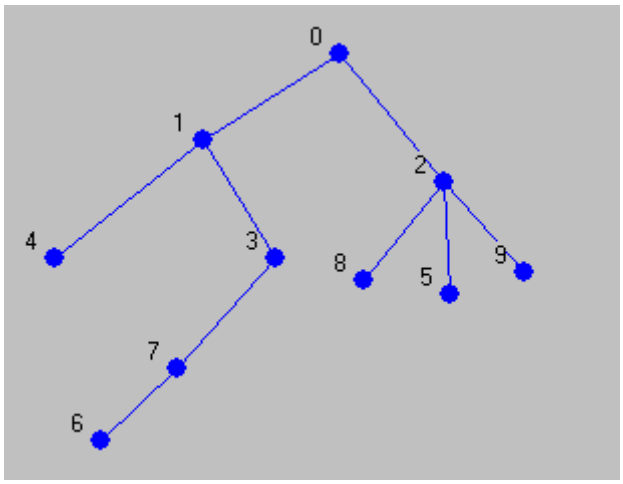
Вариант 6



Вариант 7



Вариант 8



Вариант 9

Индивидуальные задания.

Вариант полного задания состоит из вариантов четырех частей.

Вариант 1. 1.1, 2.1, 3.1, 4.1.
Вариант 2. 1.2, 2.1, 3.2, 4.2.
Вариант 3. 1.3, 2.1, 3.3, 4.3.
Вариант 4. 1.1, 2.2, 3.4, 4.4.
Вариант 5. 1.2, 2.2, 3.5, 4.5.
Вариант 6. 1.3, 2.2, 3.6, 4.6.
Вариант 7. 1.1, 2.3, 3.7, 4.7.
Вариант 8. 1.2., 2.3, 3.8, 4.8.
Вариант 9. 1.3, 2.3, 3.9, 4.9.
Вариант 10. 1.1, 2.3, 3.10, 4.1.
Вариант 11. 1.1, 2.2, 3.11, 4.2.
Вариант 12. 1.1, 2.2, 3.12, 4.3.
Вариант 13. 1.1, 2.1, 3.13, 4.4.
Вариант 14. 1.2, 2.1, 3.1, 4.5.
Вариант 15. 1.2, 2.2, 3.2, 4.6.
Вариант 16. 1.2, 2.3, 3.3, 4.7.
Вариант 17. 1.3, 2.1, 3.4, 4.8.
Вариант 18. 1.3, 2.3, 3.5, 4.9.

Контрольные вопросы

1. Как определяется дерево?
2. Какой узел называется корнем?
3. Какой узел называется листом?
4. У какого узла нет родителя?
5. У какого узла нет сыновей?
6. Как можно рассматривать объекты класса `TreeNode`?
7. Какое свойство класса `TreeNode` позволяет перейти от родителя к сыновьям?
8. Как проверить, имеет ли узел сыновей?
9. Как проверить имеет ли узел родителя?
10. Как обратится к первому сыну узла t?
11. Какой компонент предназначен для просмотра дерева?
12. Как поместить дерево в компонент *TreeView?*
13. Какие события компонента *TreeView* позволяют работать с включенным в него деревом?
14. Какое свойство компонента позволяет *TreeView* обратиться к выделенному узлу?

План отчета

1. Титульный лист.
2. Цель работы.
3. Задание. При формулировке задания, рисунок дерева обязателен.

4. Теоретический материал, используемый при выполнении задания.
5. Распечатка программы.

Этапы выполнения лабораторной работы

1. Изучить теоретический материал.
2. Ответить на контрольные вопросы.
3. Написать программу, решающую поставленную задачу.
4. Отладить программу
5. Написать отчет.

Лабораторная работа № 5

Обходы дерева

Цель: Изучить алгоритмы самых распространенных обходов дерева и на основе одного из алгоритмов написать программу.

Обходы дерева

Обход дерева – это список, в который заносятся узлы по мере их прохождения.

Наиболее известны являются три следующих обхода:

1) прямой;
2) обратный;
3) симметричный.

Дадим определения этих обходов.

Если дерево T есть нулевое дерево, то в список обхода заносится пустая запись.

Если дерево состоит из одного узла, то в список записывается этот узел.

Рассмотрим более сложное дерево T, n-корень дерева T, T_1,…, $T_к$, - поддеревья дерева T, корнями которых, являются сыновья узла n.

1.При прохождении в прямом порядке узлов дерева T сначала посещается корень n, затем узлы поддерева T_1,…, T_k, так же в прямом порядке.

2. При прохождении в обратном порядке узлов дерева T, сначала посещается в обратном порядке все узлы поддерева T_1, затем T_2, затем T_3, …, $T_к$, так же в обратном порядке и затем корень n.

3. При прохождении в симметричном порядке узлов дерева T, сначала посещаются в симметричном порядке все узлы поддерева T_1, далее корень n,

затем последовательно в симметричном порядке все узлы поддеревьев $T_2,\ldots T_к$.

Помеченные деревья

Часто каждому узлу дерева ставят в соответствие метку-значение. Дерево, у которого узлами в соответствие поставлены метки называются помеченными деревом.

Метка узла – это не имя узла, а значение, которое храниться в узле. Рассмотрим дерево синтаксического разбора арифметического выражения: (a+b)*(a+c). Оно представлено на Рисунке 3

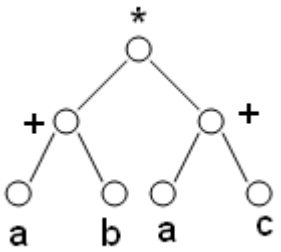


Рисунок 6

Часто при обходе дерева составляется список не имён узлов, а их меток.
Прямой обход : *,+,a,b,+,a,c
Обратный обход : a,b,+,a,c,+,*
Симметричный обход : a,+,b,*,a,+,c

В случае прямого обхода дерева мы получаем префиксную форму выражения, где оператор предшествует левому и правому операндам. Такая форма выражения называется прямой польской записью.

Дадим точное описание префиксной формы выражений.

Считаем, что префиксным выражением, одиночного операнда является сам операнд; далее префиксная форма для выражения (E1)q(E2), где q – бинарный оператор имеет вид q,P1,P2, где P1,P2-префиксные формы для E1,E2, а префиксная форма выражения q(E1), где q – унарный оператор имеет вид q,P1, где P1 -префиксная форма для E1.

Отметим, что в префиксных формах нет необходимости отделять или выделять отдельные выражения скобками, т.к. всегда можно просмотреть префиксное выражение qP1P2 и определить единственным образом P1.

Обратное упорядочивание меток даёт постфиксное представление выражения, в этом представлении оператор следует сразу за операндами.

Дадим точное описание постфиксной формы или обратной польской записи выражений.

Считаем, что постфиксным выражением, одиночного операнда является сам операнд; далее постфиксная форма для выражения (E1)q(E2), где q – бинарный оператор имеет вид P1,P2, q, где P1,P2-постфиксные формы для

E1,E2, а префиксная форма выражения q(E1), где q – унарный оператор имеет вид P1, q, где P1 -префиксная форма для E1.
Обратная польская запись так же не требует скобок.
При симметричном обходе мы получим стандартную запись выражения, хотя и без скобок.

Алгоритмы прямого и обратного обходов дерева, использующие рекурсивные функции

Procedure PR (n-Узел)
Begin
(1)занести в список обхода узел n
(2)for для каждого сына с узла n в порядке с лева на право do вызвать процедуру PR(do_RR)
End;

В этом схематически показанном наброске процедуры PR(прямое упорядочивание)

Составляющей список узлов дерева при обходе его в прямом порядке достаточно поменять местами строки 1 и 2 чтобы получить процедуру выполняющую обход дерева обратном порядке.

Алгоритм симметричного обхода дерева, использующий рекурсивную функцию

Procedure IN(n-узел)
Begin
If n-листок, then занести в список обхода узел n
else begin
(вызываем процедуру для самого левого)
In(самый левый сын узла n)
Заносим в список обхода n;
For для каждого сына e узла n, исключая самый левый, в порядке слева на право выполняется do In(c);
End;
End;

Алгоритм прямого обхода дерева, не использующий рекурсивную функцию

В этом алгоритме используется стек.

Шаг 0:заносим в стек корень дерева

Шаг 1:Если на вершине стека находится узел а, удаляем узел а из стека, помещаем его в список обхода и если у а есть сыновья, то заносим их в стек.

Шаг 2:если стек не пуст, то переходим к Шагу 1, если пуст - завершаем работу.

Алгоритм прямого обхода дерева, не использующий рекурсивную функцию.

В этом алгоритме так же используется стек, кроме того узлы, попавшие в стек метятся.

Шаг 0:заносим в стек корень дерева и метим его

Шаг 1: Пусть на вершине стека находится узел а. Если узел а имеет непомеченных сыновей, то их помещаем в стек и метим. Если узел а не имеет непомеченных сыновей, то удаляем узел а из стека и помещаем его в список обхода.

Шаг 2:Если стек не пуст, то переходим к Шагу 1,если пуст, то завершаем работу.

Алгоритм симметричного обхода дерева, не использующий рекурсивную функцию

Шаг 0. Помещаем в стек корень дерева и метим его.

Шаг 1. Пусть узел *N* находится на вершине стека. Если *N* – лист, то удаляем его из стека и помещаем в список обхода. Если *N* – не лист, то рассматриваем его левого сына *‘c’*. Если *‘c’* не помечен, то помещаем его в стек и метим, если *‘c’* помечен, то узел *N* удаляем из стека и помещаем в список обхода, а в стек помещаем всех сыновей узла *N* кроме левого и метим их.

Шаг 2. Проверяем стек. Если он не пуст переходим к шагу 1, если пуст – завершаем программу.

Программа, построенная по алгоритму симметричного обхода дерева, не использующего рекурсивную функцию:

```
string f(TreeNode t) // t- это ссылка на корень дерева
{
  string List=” ”// список обхода будем формировать, как список меток его узлов
ArrayList  St=new ArrayList(); // список St мы будем использовать, как стек
TreeNode p=t;
```

```
St.Add(p); p.Tag=1;// помещаем корень дерева в стек и метим его
while (p!=null)
{
if (p.Nodes.Count = = 0) {St.Remove(p); List=List+p.Text+" ";}
else { TreeNode c=p.Nodes[0];
if (c.Tag = = null) {St.Add(c); c.Tag=1;}
else {St.Remove(p); List=List+p.Text+" ";

for(int i=1; i<p.Nodes.Count; i++) // помещает в стек всех сыновей кроме левого
{ St.Add(p.Nodes[i]) p.Nodes[i]. Tag=1;}
if(St.Count = = 0) return List; // если стек пуст, то мы завершаем работу, возвращая в качестве ответа, список меток всех узлов
p=(TreeNode) St.LastNode();
}
return List;
}
```

Задание

1. Написать арифметическое выражение, содержащее 15 символов. Это выражение должно обязательно содержать скобки.
2. Построить дерево синтаксического разбора придуманного выражения.
3. На этапе проектирования программы ввести построенное дерево синтаксического разбора в компонент TreeView.
4. Написать и отладить функцию, выполняющую обход дерева по алгоритму, определенному вариантом задания.
5. Поместить на форму кнопку, выполняющую обход введенного в компонент TreeView дерева. Результат обхода должен выводиться на форму.

Варианты алгоритмов обхода дерева:

1. Алгоритм прямого обхода, не использующий рекурсивные функции.
2. Алгоритм обратного обхода, не использующий рекурсивные функции.
3. Алгоритм прямого обхода, использующий рекурсивную функцию.
4. Алгоритм обратного обхода, использующий рекурсивную функцию.
5. Алгоритм симметричного обхода, использующий рекурсивную функцию.

Контрольные вопросы

1. Что такое обход дерева?
2. Какой обход называется прямым?
3. Какой обход называется обратным?
4. Какой обход называется симметричным?

5. Выполнить обход заданного дерева в прямом порядке.
6. Выполнить обход заданного дерева в обратном порядке.
7. Выполнить обход заданного дерева в симметричном порядке.
8. При каком обходе первым проходится корень?
9. При каком обходе корень проходится в последнюю очередь?
10. Продемонстрировать, как строится дерево на этапе проектирования программы.

План отчета

1. Титульный лист.
2. Цель работы.
3. Придуманное арифметическое выражение.
4. Дерево синтаксического разбора арифметического выражения.
5. Теоретический материал, используемый при выполнении задания.
6. Распечатка программы.

Оглавление

Литература

1. Алексеев В.Е., Киселева Л.Г., Смирнова Т.Г. Сборник задач по дискретной математике., Электронное учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2012, 80 с.
2. Кузьмина Т.М. Конспект лекций по дисциплине «Математическая логика и теория алгоритмов»., М.: МГТУ им. А.Н.Косыгина., 2012, 80с.
3. Кузьмина Т.М., Ветрова О.А. Дискретная математика. М.: МГУДТ, 2016 г. 80 с.
4. Новиков Ф. А. Дискретная математика для программистов: Учебник для вузов. 3-е изд. — СПб.: Питер, 2009, 384 с.
5. Хаггарти Р., Дискретная математика для программистов., Москва: «Техносфера», 2003, 320с

Учебная литература

Кузьмина Тамара Михайловна

Ветрова Ольга Авенировна

Дискретная математика
Лабораторные работы

Учебное пособие

Объем 1,62 МБ Тираж 10 экз

Редакционно-издательский отдел ФГБОУ ВО «РГУ им. А.Н. Косыгина»

115035, Москва, ул. Садовническая, 33, стр. 1
тел. 8-495-811-01-01 доб. 1099
e-mail: riomgudt@mail.ru