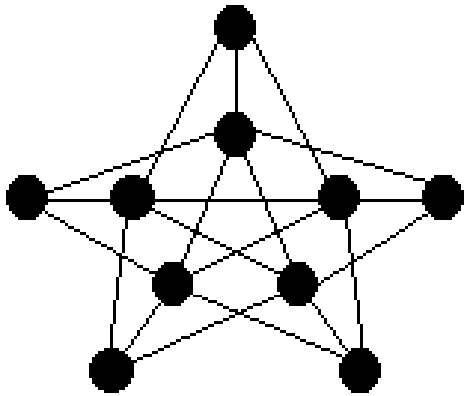
ГОУВПО «Воронежский государственный технический университет»

Кафедра компьютерных интеллектуальных технологий проектирования

232-2009

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ
по дисциплине «Дискретная математика»
для студентов специальности 230104 «Системы автоматизированного проектирования»
очной формы обучения



Воронеж 2009

Составители: канд. техн. наук О.В. Собенина,
ассистент Е.Н. Кордюкова

УДК 517.9

Методические указания к выполнению лабораторных работ по дисциплине «Дискретная математика» для студентов специальности 230104 «Системы автоматизированного проектирования» очной формы обучения / ГОУВПО «Воронежский государственный технический университет»; сост. О.В. Собенина, Е.Н. Кордюкова. Воронеж, 2009. 45 с.

Методические указания содержат необходимые для выполнения лабораторных работ теоретические сведения, варианты заданий и контрольные вопросы.

Предназначены для студентов 2 курса.

Табл. 1. Ил. 5. Библиогр.: 5 назв.

Рецензент канд. техн. наук, доц. Н.А. Ююкин

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. Е.Д. Федорков

Печатается по решению редакционно-издательского совета Воронежского государственного технического университета

Лабораторная работа №1

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМИЧЕСКИХ ПРОЦЕДУР ТЕОРИИ МНОЖЕСТВ

Цель работы: изучение основных понятий и определений теории множеств, свойств множеств и операций над ними. Получение практических навыков программной реализации алгоритмических процедур теории множеств.
Технические средства: IBM PC AT.
Программное средство: Delphi.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Основные понятия и определения теории множеств

К *теории множеств* в общем случае относятся аксиоматическая теория множеств и элементарная теория множеств. Любое понятие дискретной математики можно определить с помощью понятия множества, которое является одним из фундаментальных понятий и было сформулировано впервые немецким математиком Г. Кантором.

Под множеством понимается любое собрание определенных и различных между собой объектов, мыслимое как единое целое.

Множества будем обозначать заглавными буквами латинского алфавита; объекты, которые образуют множества, будем называть *элементами* множества и обозначать малыми буквами латинского алфавита. Если элемент *x* принадлежит множеству X, то этот факт записывается в виде $x \in X$, иначе $x \notin X$. Как правило, считается, что все элементы множества различны. Множество с повторяющимися элементами называется *мультимножеством.*

Множество, содержащее конечное число элементов, называется *конечным*; в противном случае множество называется

бесконечным. Количество элементов конечного множества называется *мощностью* и обозначается $|X|=n$, если множество X содержит *n* элементов. Если множество не содержит ни одного элемента, то оно называется *пустым* и обозначается $\varnothing$.

Для произвольных множеств X и Y можно определить два типа отношений – *отношение равенства и отношение включения.*

Два множества считаются равными, если они состоят из одних и тех же элементов. Принято обозначение X=Y, если X и Y равны, и $X \neq Y$- иначе.

Легко видеть, что для любых множеств X, Y, Z справедливы соотношения

$$X = X$$
$$X = Y \rightarrow Y = X$$
$$(X = Y \text{ и } Y = Z) \rightarrow X = Z$$

Если каждый элемент множества X является элементом множества Y, то говорят, что X включено в Y и обозначают $X \subseteq Y$:

$$X \subseteq Y \Leftrightarrow (x \in X \rightarrow x \in Y).$$

В этом случае говорят, что множество X является *подмножеством* множества Y. В частности X и Y могут совпадать, поэтому $\subseteq$ называется также отношение *нестрогого включения.* Некоторые свойства подмножества, вытекающие из его определения:

$$X \subseteq X$$
$$(X \subseteq Y \text{ и } Y \subseteq Z) \rightarrow X \subseteq Z.$$

Если $X \subseteq Y$ и $X \neq Y$, то говорят, что X есть *собственное подмножество* Y и обозначают $X \subset Y$, отношение между множествами в этом случае называется отношением *нестрогого включения.* Для отношения строгого включения справедливо

$$(X \subset Y \text{ и } Y \subset Z) \rightarrow X \subset Z.$$

Невключение подмножества X в множество Y обозначается $X \not\subset Y (X \not\subseteq Y)$.

Заметим, что если X является подмножеством Y и наоборот, то X и Y состоят из одних и тех же элементов, поэтому

$$X = Y \Leftrightarrow (X \subseteq Y \text{ и } Y \subseteq X).$$

Для каждого множества X существует множество, элементами которого являются различные подмножества множества X. Такое множество называется *семейством множества* и обозначается p(X). Так как $\varnothing$ включено в любое множество, то $\varnothing \subseteq p(X)$.

Пример.

Пусть $X = \{x_1, x_2, x_3\}$ Тогда

$$p(X) = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1, x_2, x_3\}, \varnothing\}.$$

Если в рамках некоторого рассуждения рассматриваются подмножества некоторого множества, то оно называется *универсальным,* или универсумом и обозначается U.

Множество может быть задано различными способами: перечислением элементов в скобках $\{\ \}$ (для конечных множеств) или указанием их свойств, однозначно определяющих принадлежность элементов данному множеству, при этом используется запись

$$X=\{\ x \mid x \text{ обладает свойством } P(x)\}$$

(выражение в скобках читается: множество всех элементов *x,* которые обладают свойством P(*x*). Так, множество натуральных чисел N={1,2,…} может быть описано следующим образом:

$$N=\{\ i \mid \text{ если целое } i \in N, \text{ то } i+1 \in N, i \geq 1\}.$$

Кроме того, множества можно задать с помощью характеристической функции, значения которой указывают, является ли (да или нет) *x* элементом множества *X :*

$$\mu_x(x) = \begin{cases} 1, x \in X \\ \\ 0, x \notin X \end{cases}$$

Заметим, что для любых элементов $\mu_\varnothing = 0$; $\mu_U = 1$.

Пример.

Пусть на универсуме U={a,b,c,d,e} определено множество X={a,c.d}, тогда

$$\mu_x(a)=1, \mu_x(b)=0, \mu_x(c)=1, \mu_x(d)=1, \mu_x(e)=0.$$

1.2. Операции над множествами

Для получения новых множеств из уже существующих используют операции над множествами. Рассмотрим основные из них [2].

Объединением множеств X и Y называется множество $X \cup Y$, все элементы которого являются элементами множества X или Y:

$$X \cup Y = \{x \mid x \in X \text{ или } x \in Y\}.$$

Пересечением множеств X и Y называется множество $X \cap Y$, элементы которого являются элементами обоих множеств X и Y:

$$X \cap Y = \{x \in X \text{ и } x \in Y\}.$$

Очевидно, что выполняются включения

$$X \cap Y \subseteq X \subseteq X \cup Y; \quad X \cap Y \subseteq Y \subseteq X \cup Y.$$

Разностью множеств X и Y называется множество X/Y всех тех элементов X, которые не принадлежат Y:

$$X \backslash Y = \{x \mid x \in X \text{ и } x \notin \overline{Y}.\}.$$

Симметричной разностью множества X и Y называется множество

$$X \oplus Y = (X \setminus Y) \cup (Y \setminus X).$$

Дополнением множества X называется множество $\overline{X}$ всех тех элементов x, которые не принадлежат множеству X:

$$\overline{X} = U \backslash X$$

Декартовым (прямым) произведением множеств X и Y называется множество упорядоченных пар вида

$$X \times Y = \{(x,y) \mid x \in X \text{ и } y \in Y\}.$$

Пример. Пусть $X=\{x_1, x_2, x_3\}$, $Y=\{y_1, y_2\}$.

Тогда

$$X \times Y = \{(x_i, y_i), (x_1, y_2), (x_2, y_1), (x_2, y_2), (x_3, y_1), (x_3, y_2)\}.$$

Аналогично можно определить декартово произведение n множеств $X_1, X_2, ..., X_n$

$$X_1 \times X_2 \times .... \times X_n = \{(x_1, x_2, ..., x_n) | x_1 \in X_1, x_2 \in X_2, ..., x_n \in X_n \}.$$

Если $X_1 = X_2 = ... = X_n = X$, то n-я степень множества X определяется как

$$X^n = \underbrace{X \times X \times ... \times X}_{n}.$$

1.3. Представление множеств в ЭВМ

Применительно к множествам определение представления подразумевает описание способа хранение информации о принадлежности элементов множеству и описание алгоритмов для вычисления объединения, пересечения и других введённых операций.

Как правило, один и тот же объект может быть представлен многими разными способами, причем нельзя указать способ, который является наилучшим для всех возможных случаев. В одних случаях выгодно использовать одно представление, а в других - другое. Выбор представления зависит от целого ряда факторов: особенностей представляемого объекта, состава и относительной частоты использования операций в конкретной задаче и т.д.

Для записи алгоритмов в методических указаниях используется неспецифицированный язык программирования, похожий по синтаксису на Паскаль [1]. В программах используются математические обозначения, которые являются самоочевидными в конкретном контексте. Например, конструкция

for $x \in M$ do
  P(x)
end for

означает применение процедуры P ко всем элементам множества M.

Оператор

select $m \in M$

означает выбор произвольного элемента m из множества M.

Оператор

yield x

означает возврат значения x, но при этом выполнение функции не прекращается, а продолжается до следующего оператора. Этот оператор позволяет очень просто записать алгоритмы, результатом которых является некоторое заранее неизвестное множество значений.

1.3.1. Реализация операций над множествами заданного универсума U

Пусть универсум U – конечный, и число элементов в нём не превосходит разрядности ЭВМ: $|U| < n$. Элементы универсума нумеруются: $U = \{u_1,\ldots,u_n\}$. Подмножество A универсума U представляется кодом (машинным словом или битовой шкалой) C, в котором:

$$C[i] = \begin{cases} 1, \text{ если } u_i \in A, \\ 0, \text{ если } u_i \notin A, \end{cases}$$

где C[i] – это i-й разряд кода C.

Код пересечения множеств A и B есть поразрядное логическое произведение кода множества A и кода множества B. Код объединения множеств A и B есть поразрядная логическая сумма кода множества A и кода множества B. Код дополнения множества A есть инверсия кода множества A. В большинстве ЭВМ для этих операций есть соответствующие машинные команды. Таким образом, операции над небольшими множествами выполняются весьма эффективно.

Замечание. Если мощность универсума превосходит размер машинного слова, но не очень велика, то для представления множеств используются массивы битовых шкал. В этом случае операции над множествами реализуются с помощью циклов по элементам массива.

1.3.2. Представление множеств упорядоченными списками

Если универсум очень велик (или бесконечен), а рассматриваемые подмножества универсума не очень велики, то представление с помощью битовых шкал не является эффективным с точки зрения экономии памяти. В этом случае множества представляются списками элементов. Элемент списка при этом представляется записью с двумя полями: информационным и указателем на следующий элемент.

```
elem = record
   i: info; { информационное поле }
   n: ^ elem  {указатель на следующий элемент }
end record
```

Эффективная реализация операций над множествами, представленными в виде упорядоченных списков, основана на алгоритме, известном как алгоритм типа слияния.

Алгоритм типа слияния параллельно просматривает два множества, представленных упорядоченными списками, причем на каждом шаге продвижение происходит в том множестве, в котором текущий элемент меньше.

1.4. Алгоритмы

1.4.1. Генерация всех подмножеств универсума

Во многих переборных алгоритмах требуется последовательно рассмотреть все подмножества заданного множества. В большинстве компьютеров целые числа представляются кода-

ми в двоичной системе счисления, причем число $2^k - 1$ представляется кодом, содержащим k единиц. Таким образом, число 0 являеться представлением пустого множества $\varnothing$, число 1 является представлением подмножества, состоящего из первого элемента, и т.д. Следующий тривиальный алгоритм перечисляет все подмножества n – элементного множества.

Алгоритм. Алгоритм генерации всех подмножеств n – элементного множества.
Вход: $n \geq 0$ – мощность множества
Выход: последовательность кодов множеств i.

```
for i from 0 to 2^n – 1 do
  yield i
end for
```

Обоснование. Алгоритм выдаёт 2^n различных целых чисел, следовательно, 2^n различных кодов. С увеличением числа увеличивается количество двоичных разрядов, необходимых для его представления. Самое большое (из генерируемых) число 2^{n-1} требует для своего представления ровно n разрядов. Таким образом, все подмножества генерируются, причём ровно по одному разу.

Недостаток этого алгоритма состоит в том, что порядок генерации подмножеств никак не связан с их составом. Например, вслед за подмножествам с кодом 0111 будет перечислено подмножество с кодом 1000.

1.4.2. Проверка включения слиянием

Рассмотрим алгоритм типа слияния, который определяет, является ли множество A подмножеством множества B.

Алгоритм. Проверка включения слиянием.
Вход: проверяемые множества A и B, которые заданы указателями a и b.

Выход: 1, если A⊂B, в противном случае 0.

```
 pa:=a; pb:=b
 while pa≠nil & pb≠nil do
  if pa.i<pb.i then
     return 0 {элемент множества A отсутствует в множестве B}
  else if pa.i>pb.i then
     pb:=pb.n {элемент множества A, может быть, присутствовать в множестве B}
  else
     pa:=pa.n {здесь pa.i=pb.i, то есть }
     pb:=pb.n {элемент множества A точно присутствует в множестве B}
  end if
 end while
 return pa=nil
```

Обоснование. На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае текущий элемент множества A заведомо меньше, чем текущий и все последующие элементы множества B, а потому он не содержится в множестве B, и можно завершить выполнение алгоритма. Во втором случае происходит продвижение по множеству B в надежде отыскать элемент, совпадающий с текущим элементом множества A. В третьем случае найдены совпадающие элементы, и происходит продвижение сразу в обеих множествах. По завершении основного цикла возможны два случая: либо pa=nil, либо pa≠nil. Первый случай означает, что для всех элементов множества A удалось найти совпадающие элементы множества B. Второй случай означает, что множество B закончилось раньше, то есть не для всех элементов множества A удалось найти совпадающие элементы в множестве B.

1.4.3. Вычисление объединения слиянием

Рассмотрим алгоритм типа слияния, который вычисляет объединение двух множеств, представленных упорядоченными списками.

Алгоритм. Вычисление объединения слиянием.
Вход: объединяемые множества A и B, которые заданы указателями a и b.
Выход: объединение C=A∪B, заданным указателем c.

```
 pa:=a; pb:=b; c:=nil; e:=nil;
 while pa≠nil & pb≠nil do
  if pa.i<pb.i then
    d:=pa.i; pa:=pa.n {добавлению подлежит элемент множества A}
  else if pa.i>pb.i then
       d:=pb.i; pb:=pb.n {добавлению подлежит элемент множества B}
  else
       d:=pa.i {здесь pa.i=pb.i, и можно взять любой из элементов}
       pa:=pa.n; pb:=pb.n;
 end if
  Append(c,e,d) {добавление элемента d в конец списка c}
 end while
  p:=nil
  if pa≠nil then
      p:=pa {нужно добавить в результат оставшиеся элементы множества A}
   end if
  if pb≠nil then
      p:=pb {нужно добавить в результат оставшиеся элементы множества B}
   end if
  while p≠nil do
```

```
    Append(c,e,d,i)
    p:=p.n
  end while
```

Обоснование. На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае в результирующий список добавляется текущий элемент множества A и происходит продвижение в этом множестве, во втором аналогичная операция производится с множеством B, а в третьем случае найдены совпадающие элементы и происходит продвижение сразу в обоих множествах. Таким образом, в результат попадают все элементы обоих множеств, причем совпадающие элементы попадают ровно один раз. По завершении основного цикла один из указателей pa и pb (но не оба вместе!) может быть не равен nil. В этом случае остаток соответствующего множества без проверки добавляется в результат.

1.4.4. Вычисление пересечения слиянием

Рассмотрим алгоритм типа слияния, который вычисляет пересечение двух множеств, представленных упорядоченными списками.

Алгоритм. Вычисление пересечения слиянием.

Вход: пересекаемые множества A и B, которые заданы указателями a и b.
Выход: пересечение $C=A\cap B$, заданным указателем c.

```
pa:=a; pb:=b; c:=nil; e:=nil;
while pa≠nil & pb≠nil do
  if pa.i<pb.i then
```

```
      pa:=pa.n {элементы множества A не принадлежат пересечению}
    else if pa.i>pb.i then
      pb:=pb.n {элементы множества B не принадлежат пересечению}
    else
      {здесь pa.i=pb.i – данный элемент принадлежит пересечению}
      Append(c,e,pa.i); pa:=pa.n; pb:=pb.n;
    end if
  end while
```

Обоснование. На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае текущий элемент множества A не принадлежит пересечению, он пропускается и происходит продвижение в этом множестве, во втором то же самое производится с множеством B. В третьем случае найдены совпадающие элементы, один экземпляр элемента добавляется в результат и происходит продвижение сразу в обоих множествах. Таким образом, в результат попадают все совпадающие элементы обоих множеств, причем ровно один раз.

Вопросы для самопроверки

1. Что такое множество?
2. Какие спецификации множеств знаете?
3. Какими способами можно задать множество?
4. Что такое семейство множества?
5. Как определяется операция симметрической разности двух множеств ?
6. Какими свойствами обладают операции над множества?
7. Что такое декартово произведение множеств?

8. Какие операции над множествами знаете?
9. Как определяются операции над множествами.
10. Дать определение универсального множества.
11. Дать определение собственного подмножества.
12. Дать определение конечного множества.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Задания

Написать программу, реализующую следующую процедуру:

1. Даны два множества, заданные перечислением своих элементов. Получить симметрическую разность этих элементов.
2. Даны два множества, заданные перечислением своих элементов. Определить декартовое произведение этих множеств.
3. Получить семейство множества, заданного, перечислением своих элементов.
4. Выяснить является ли данное множество подмножеством множества.
5. Выяснить является ли множество собственным подмножеством.
6. Для произвольных множеств A, B и C определить $(A \setminus B) \cap (A \setminus C)$, $(A \cap \overline{B}) \cup C$.
7. Для произвольных множеств A, B и C определить $A \cup B \cup C$, $C \cap A \cup \overline{B}$.
8. Для произвольных множеств A, B и C определить $\overline{A \cup B \cup \overline{C}}$, $(A \cup B) \setminus C$.
9. Вычисление пересечения множеств слиянием.
10. Вычисление объединения множеств слиянием.
11. Проверка включения слиянием.
12. Генерация всех подмножеств универсума.

2.2. Порядок выполнения работы

1. Получить задание у преподавателя.
2. Разработать алгоритм решения задачи.
3. Реализовать полученный алгоритм.
4. Проанализировать результаты работы алгоритма.
5. Оформить отчет по лабораторной работе.

2.3. Содержание отчета

1. Номер и тема лабораторной работы.
2. Цель выполнения работы.
3. Схема алгоритма.
4. Исходные данные и результаты вычислений.
5. Анализ полученных результатов и вывод по работе.

2.4. Контрольные вопросы

1. С помощью каких структур данных можно представить множества в ЭВМ ?
2. Существует ли в Паскале специальный тип данных для задания множеств ?
3. С помощью какого опретатора можно определить пересечение множеств в Паскале ?
4. С помощью какого оператора можно определить объединение множеств в Паскале ?
5. Как в Паскале определяется включение элемента в множество ?
6. Можно ли задать множество с помощью массива? Как это сделать.
7. Как определит пересечение множеств слиянием ?
8. Как определить объединений множеств слиянием ?
9. Как определить генерацию всех подмножеств универсума ?

10. Как представить множество в Паскале с помощью списка ?
11. Обосновать алгоритм вычисления пересечения слиянием.
12. Обосновать алгоритм вычисления объединения слиянием.

Лабораторная работа №2

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМИЧЕСКИХ ПРОЦЕДУР ТЕОРИИ ОТНОШЕНИЙ

Цель работы: изучение основных понятий и определений теории отношений, свойств отношений, операций над ними и специальных типов бинарных отношений. Получение практических навыков программной реализации алгоритмических процедур теории отношений.
Технические средства: IBM PC AT.
Программное средство: Delphi.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В множестве X *n*-местным или *n-арным отношением* называется подмножество *R* *n*-й декартовой степени $X^n = X \times X \times .... \times X$ заданного множества, $R \subseteq X^n$, X называется носителем отношения. Будем говорить, что упорядоченные элементы $x_1, x_2, ..., x_n \in X$ находятся в отношении *R*, если $x_1, x_2, ..., x_n \in R$. Одноместное отношение называется *унарньм,* или свойством, и соответствует подмножеству множества X. Особую роль в приложениях играют *бинарные* отношения $R \subseteq X \times X$. Каждому бинарному отношению можно поставить в соответствие матрицу бинарного отношения, которую также обозначают через $R = \{r_{ij}\}_{n+n}$ $(n = |X|)$ и элементы которой r_{ij} оп-

ределяются следующим правилом:

$$r_{ij} = \begin{cases} 1, (x_i, x_j) \in R \\ 0, (x_i, x_j) \notin R \end{cases}$$

Рассмотрим свойства бинарных отношений:

Отношение $R \subseteq X \times X$ называется

рефлексивным, если для $\forall x \in X$ $(x,x) \in R$;

антирефлексивным, если $\forall x \in X$ $(x,x) \notin R$;

симметричным, если из условия $(x,y) \in R$ следует, что $(y,x) \in R$;

антисимметричным, если из условия $(x,y) \in R$ следует, что $(y,x) \notin R$;

транзитивным, если из условий $(x,y) \in R$ и $(y, z) \in R$ следует, что $(x,z) \in R$.

Матрица бинарного отношения содержит единицы на главной диагонали, если отношение является рефлексивным; такая матрица является симметричной относительно главной диагонали, если отношение симметрично; для антисимметричного отношения произведение элементов, расположенных симметрично относительно главной диагонали, равно нулю.

Пусть на множестве X задано отношение $U \subseteq X \times X$ тогда совокупности G = (X,U) называют *графом,* причем X - множество *вершин* графа, а U - множество линий, которые соединяют - все или часть из этих вершин. Если в образовании пары (x,y) играет роль порядок элементов, то эти линии называются *дугами* и изображаются направленным отрезком прямой (а граф G называется *ориентированным графом* иди орграфом), иначе - *ребрами* и изображаются просто отрезком прямой (граф G в этом случае называется *неориентированным графом* или неорграфом). Пару противоположно направленных дуг между двумя фиксированными вершинами в графе

часто заменяют ребром. Как правило, граф задается с помощью матрицы смежности $A = \{a_{ij}\}_{n+n}$ ($n=|X|$), элементы которой определяются следующим образом:

$$a_{ij} = \begin{cases} 1, (x_i, x_j) \in U \\ 0, (x_i, x_j) \notin U \end{cases}$$

Заметим, что матрица смежности графа совпадает с матрицей соответствующего бинарного отношения.

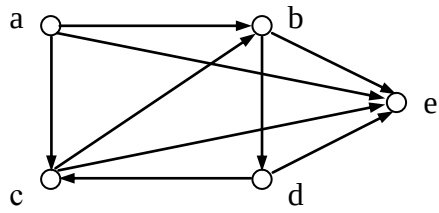
Пример.

Пусть матрица бинарного отношения *R*, заданного на универсальном множестве U - {a,b,c,d,e}, имеет вид

R	a	b	c	d	e
a	0	1	1	0	0
Ь	0	0	0	1	1
c	0	1	0	0	1
d	0	0	1	0	1
e	1	0	0	0	0

Тогда соответствующий граф будет иметь вид

Рис. 1



Так как отношение является прежде всего множеством упорядоченных пар, то для отношений можно ввести те же

операции, что и для множеств, то есть операции объединения, пересечения, разности и дополнения. Кроме того, для отношений существуют специальные операции:

инверсией отношения R называется отношение $R^{-1} = \{(x,y) \mid (y,x) \in R\}$;

пусть R_1, R_2 - отношения, заданные на множестве X, тогда *композицией отношений* R_1 и R_2 называется отношение, определяемое следующим образом:

$$R_1 \circ R_2 = \{(x,y) \mid \exists z : (x,z) \in R_1 \, u \, (z,y) \in R_2\}$$

Заметим, что $R_2 \circ R_1 \neq R_1 \circ R_2$.

Рефлексивное, симметричное, транзитивное отношение называется отношением эквивалентности или *эквивалентностью* (обозначение I).

Примеры:

1. Отношение равенства на множестве целых чисел, отношения подобия на множестве треугольников являются отношениями эквивалентности.

Отношение эквивалентности имеет большое практическое значение. Так сущность моделирования заключается в том, что устанавливают отношение эквивалентности между двумя системами, каждая из которых может быль абстрактной или реально существующей. Если одна из систем оказывается проще для исследования, то ее рассматривают в качестве модели для другой.

Другим важным типом отношения является отношение *порядка.* Рефлексивное, антисимметричное, транзитивное отношение называется отношением нестрогого порядка и обозначается символом $\preceq$. Антирефлексивное, антисимметричное, транзитивное отношение называется отношением строгого порядка и обозначается символом $\prec$. Отношения строгого и нестрогого порядков иначе называют отношениями упорядоченности. Отношение, обратное отношению упорядоченности,

также является отношением упорядоченности, т.е. $(\prec)^{-1} = \succ$.

Пример.

Пусть Y - некоторое множество, тогда отношение включения $\subseteq$ на множестве всех подмножеств P(Y) является отношением нестрогого порядка.

Представление отношений в ЭВМ

Пусть $R \subset A^2$ и $|A|=n$. Перенумеруем элементы множества A. Тогда отношение R можно представить матрицей R: **array**[1..*n*,1..*n*] **of** 0..1, где

$$R[i,j]=\begin{cases} 1, (x_i, x_j) \in R \\ 0, (x_i, x_j) \notin R. \end{cases}$$

Вопросы для самопроверки

1. Что какое отношение?
2. Что необходимо для задания отношения?
3. Каким образом можно представить бинарное отношение с помощью матрицы?
4. Какими свойствами может обладать бинарное отношение?
5. В чем состоит отличие свойства антисимметричности от несимметричности?
6. Можно ли представить отношение в виде графа?
7. Какими свойствами обладает отношение эквивалентности?
8. Что такое отношение порядка?

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Задания

Написать программу, реализующую следующую процедуру:

1. Получить инверсию отношения в виде перечисления упорядоченных пар и матрицы отношения.
2. Даны два отношения A и B, заданные перечислением пар. Получить $A \circ B$ и $B \circ A$.
3. Определить какими из свойств (рефлективность, симметричность, транзитивность) обладает отношение, а каким нет.
4. Даны два отношения A и B. Определить A∪B, A∩B, A\B.
5. Даны два отношения A и B. Определить $A \cap \overline{B}$, $(A \cup B) \cap \overline{A}$.
6. Даны два отношения A и B. Определить $\overline{(A \cup B)} \cap \overline{A}$, $A \cup \overline{(\overline{A} \cap B)}$.
7. Даны три отношения A, B и C. Определить $\overline{A \cup B \cup C}$, $\overline{A \cap B} \cup C$.
8. Даны три отношения A, B и C. Определить $(A \cup B) \setminus (\overline{A} \cap C)$, $(A \setminus C) \cup B$.
9. Определить, является ли данное отношение отношением эквивалентности.
10. Определить, является ли данное отношение отношением строго порядка.
11. Определить, является ли данное отношение отношением нестрого порядка.
12. Приведением к блочно-диагональному виду матрицу отношения, определить, является ли данное отношение эквивалентностью.

2.2. Порядок выполнения работы

1. Получить задание у преподавателя.
2. Разработать алгоритм решения задачи.
3. Реализовать полученный алгоритм.
4. Проанализировать результаты работы алгоритма.
5. Оформить отчет по лабораторной работе.

2.3. Содержание отчета

1. Номер и тема лабораторной работы.
2. Цель выполнения работы.
3. Схема алгоритма.
4. Исходные данные и результаты вычислений.
5. Анализ полученных результатов и вывод по работе.

2.4. Контрольные вопросы

1. С помощью каких структур данных можно представить отношение в ЭВМ ?

2. Существует ли в Паскале специальный тип данных для задания отношений ?

3. Какие способы представления бинарных отношений знаете?

4. Бинарное отношение задано графом. Как представить такое отношение в ЭВМ?

5. Бинарное отношение задано перечисление пар. Как представить такое отношение в ЭВМ?

6. Бинарные отношения заданы матрицами. Как получить матрицу пересечения отношений ?

7. Бинарные отношения заданы матрицами. Как получить матрицу объединения отношений ?

8. Бинарные отношения заданы матрицами. Как получить матрицу композиции отношений ?

9. Бинарное отношение задано матрицей. Как получить матрицу инверсии отношения ?

10. Можно ли проверить бинарное отношение на эквивалентность с помощью операции композиция и отношения включения ?

Лабораторная работа №3

ПРЕДСТАВЛЕНИЕ ГРАФОВ В ЭВМ

Цель работы: изучение основных алгоритмов теории графов и получение практических навыков их программной реализации.

Технические средства: IBM PC AT.

Программное средство: Delphi.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Основу теории графов составляет совокупность методов и представлений, сформировавшихся при решении конкретных задач.

Граф есть совокупность точек и линий, соединяющих эти точки. Эти соединения могут обладать различными характеристиками. Теория графов занимается изучением этих характеристик.

Граф характеризует отношения между множествами объектов, и в теории графов сталкиваются главным образом с двумя формулировками задач. В первом случае требуется ответить на вопрос, существуют ли объекты, обладающие определенным образом, и если да, то сколько их и каково их максимальное количество. В другом случае нужно определить, как построить граф или подграф, обладающий некоторым свойством.

1.1. Основные понятия

Пусть задано некоторое конечное множество *X*, элементы которого будем называть вершинами, и множество *U* , состоящее из пар элементов (x_i, x_j) множества *X*. Упорядоченная пара множеств *G=(X,U)* называется *графом.*

Если в определении графа существенно в каком порядке выбираются вершины то есть пара (x_i, x_j) отлична от пары (x_j, x_i), то такой граф называют ориентированным или орграфом, а пару (x_i, x_j) - дугой, при этом считается, что x_i - начальная вершина, а x_j - конечная. В геометрической интерпретации дуге соответствует направленный отрезок. Часто орграф задают парой *G*=(X, Г), где X - множество вершин, а Г- неоднозначное отображение, ставящее в соответствие каждой вершине подмножество в X. $\Gamma(x_i)$ - множество вершин $x_j \in X$, для которых в графе *G* существует дуга (x_i, x_j). $\Gamma^{-1}(x_i)$ - множество вершин $x_j \in X$, для которых в графе *G* существует дуга (x_j, x_i).

Если в определении графа не существенен порядок вершин при образовании пары (x_i, x_j), то граф называют неориентированным или неорграфом**,** а пару (x_i, x_j) - ребром **.**

Путем в графе *G* называется такая последовательность дуг, в которой конец каждой предыдущей дуги совпадает с началом следующей. Для неорграфа такая последовательность ребер называется цепью.

Две вершины x_i и x_j называются смежными, если существует соединяющее их ребро (дуга), при этом вершины называются инцидентными этому ребру (дуге), а ребро (дуга) - инцидентным(-ой) этим вершинам. Аналогично, два различных

ребра (дуги) называются смежными, если они имеют по крайней мере одну общую вершину.

Если граф ориентированный, то говорят, что дуга (x_i, x_j) исходит из вершины x_i и заходит в вершину x_j. Число дуг, которые имеют вершину x_i своей начальной вершиной, называют полустепенью исхода вершины x_i и обозначают $d^-(x_i)$. Число дуг, которые имеют вершину x_i своей конечной вершиной, называют полустепенью захода вершины x_i и обозначают $d^+(x_i)$. Для неорграфа число ребер, инцидентных данной вершине x_i называется степенью (валентностью) вершины x_i, и обозначается $d(x_i)$.

Подграфом графа *G=(X,U)* называется граф $G' = (X', U')$, для которого $X' \subset X, U' \subset U$.

Остовным подграфом графа *G=(X,U)* называется граф *Go=(Xo,Uo),* для которого *Xo=X, Uo⊂ U.*

Порожденным подграфом графа *G=(X,U)* называется граф *Gs=(Xs,Us)*, для которого Xs⊂X и для $\forall x_i \in X_S (\Gamma_S(x_i) = \Gamma(x_i) \bigcap X_S)$.

Если в графе G существует путь, идущий от вершины x_i к вершине x_j то говорят, что вершина x_j *достижима* из вершины x_i.

Если для любой пары вершин существует цепь их соединяющая, то такой граф называется *связным.* Иначе граф называется *несвязным.*

Орграф называется *односторонне связным***,** если для любых двух различных вершин x_i и x_j существует по крайней мере один путь из x_i в x_j или из x_j в x_i.

Орграф называется *сильносвязным,* если для любых двух различных вершин существует по крайней мере один путь их соединяющий (это определение означает, что любые две вершины графа взаимно достижимы).

Максимальный связный подграф графа *G* называется *компонентой связности.*

Максимальный сильносвязный подграф графа *G* называется *сильной компонентой* связности (СК).

Граф $G^*=(X^*,U^*)$ называется *конденсацией* графа $G=(X,U)$, если каждая его вершина $x_i^* \in X^*$ соответствует $СК(x_i)$ графа G, а $(x_i^*,x_j^*) \in U^*$ тогда и только тогда , когда в G существует $(x_i,x_j) \in U$, причем $x_i \in СК(x_i)$, $x_j \in СК(x_j)$.

Базой графа G называется минимальный подграф B, из которого достижима любая вершина графа G.

Матрицей достижимости графа $G=(X,U)$, $|X|=n$ называется матрица $R = [\ r_{ij}\ ]_{nxn}$, элементы которой определяются следующим образом:

$$r_{ij}= \begin{cases} 1, \text{ если вершина } x_j \text{ достижима из } x_i; \\ 0 \text{ в противном случае.} \end{cases}$$

Матрицей контрдостижимости графа $G = (X, U)$, $|X| = n$ называется матрица $Q = [\ q_{ij}\ ]_{nxn}$, элементы которой определяются следующим образом:

$$q_{ij}= \begin{cases} 1, \text{ если вершина } x_i \text{ достижима из } x_j; \\ 0 \text{ в противном случае.} \end{cases}$$

Множество достижимости вершины x_i $R(x_i)$ состоит из таких вершин x_j , для которых $r_{ij}=1$, определяется формулой:

$$R(x_i) = \{ x_i \} \cup \Gamma(x_i) \cup \Gamma^2(x_i) \cup \ldots \cup \Gamma^p(x_i), \quad p < n.$$

Множество контрдостижимости вершины x_i $Q(x_i)$ состоит из таких вершин x_j, для которых $q_{ij}=l$, определяется формулой:

$$Q(x_{ij}) = \{ x_i \} \cup \Gamma^{-1}(x_i) \cup \Gamma^{-2}(x_i) \cup \ldots \cup \Gamma^{-p}(x_i), \quad p < n.$$

Основные операции над графами

1. *Объединение гра*фов. Объединением графов $G_1 = (X_1, U_1)$ и $G_2 = (X_2, U_2)$ называется граф $G_3 = (X_1 \cup X_2, U_1 \cup U_2)$.

2. *Пересечение графов*. Пересечением графов $G_1 = (X_1, U_1)$ и $G_2 = (X_2, U_2)$ называется граф $G_3 = (X_1 \cap X_2, U_1 \cap U_2)$.

3. *Удаление вершины*. При удалении вершины из графа удаляются и все инцидентные ей ребра (дуги).

4. *Удаление ребра (дуги)*. При удалении ребра (дуги) его концевые вершины не удаляются. Операцией, являющейся обратной к удалению ребра, является добавление ребра.

5. *Слияние или отождествление вершин*. Говорят, что вершины x_i и x_j в графе G отождествляются (сливаются), если они заменяются такой новой вершиной x_k, что все ребра (дуги) графа, инцидентные x_i и x_i, становятся инцидентными новой вершине x_k.

6. *Стягивание реб*ра. Эта операция означает удаление ребра и отождествление его концевых вершин. Граф G называется стягиваемым к графу H, если граф H может быть получен из G в результате некоторой последовательности стягиваний ребер.

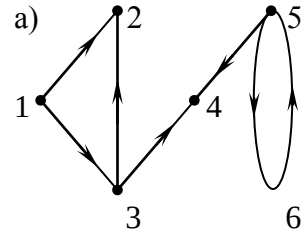
1.2. Представление графов в ЭВМ

Очевидно, что наиболее понятный и полезный для человека способ представления графа – изображение графа на плоскости в виде точек и соединяющих их линий – будет совершенно бесполезным, если решать задачи, связанные с графами, с помощью ЭВМ. Выбор соответствующей структуры данных для представления графов имеет принципиальное влияние на эффективность алгоритмов. Рассмотрим различные способов представления графов [1,5].

В теории графов классическим способом представления графа служит *матрица инциденций.* Это матрица B с *n* строками, соответствующими вершинам, и *m* столбцами, соответствующими ребрам. Для ориентированного графа столбец, соответствующий дуге (x, *y)*$\in U$, содержит –1 в строке, соответствующей вершине *x,* 1 в строке, соответствующей вершине *y,* и нули во всех остальных строках *(петлю,* т. е. дугу вида (*x*, *x*) удобно представлять иным значением в строке *x,* например, 2). В случае неориентированного графа столбец, соответствующий ребру *(x, y),* содержит 1 в строках, соответствующих *x* и *y,* и нули в остальных строках. Это проиллюстрировано на рис. 2. С алгоритмической точки зрения матрица инциденций является, вероятно, самым худшим способом представления графа, который только можно себе представить. Во-первых, он требует *nm* ячеек памяти, причем большинство этих ячеек вообще занято нулями**.** Неудобен также доступ к информации. Ответ на элементарные вопросы типа «существует ли дуга (*x*, *y*)?», «к каким вершинам ведут ребра из *x?»* требует в худшем случае перебора всех столбцов матрицы**,** а следовательно, *m* шагов.

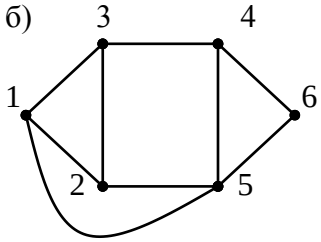
Лучшим способом представления графа является матрица смежности, определяемая как матрица $A = [a_{ij}]$ размера $n \times n$**,** где $a_{ij} = 1$, если существует ребро, идущее из вершины *x* в вершину *y,* и $a_{ij} = 0$ в противном случае. Здесь мы подразумеваем, что ребро (*x*, y*)* неориентированного графа идет как от *x к y,*

так и от *у к х,* так что матрица смежности такого графа всегда является симметричной. Это проиллюстрировано на рис. 3.



$B=$

	(1,2)	(1,3)	(3,2)	(3,4)	(5,4)	(5,6)	(6,5)
1	1	1	0	0	0	0	0
2	-1	0	-1	0	0	0	0
3	0	-1	1	1	0	0	0
4	0	0	0	-1	-1	0	0
5	0	0	0	0	1	1	-1
6	0	0	0	0	0	-1	1



$B=$

	(1,2)	(1,3)	(1,5)	(2,3)	(2,5)	(3,4)	(4,5)	(4,6)	(5,6)
1	1	1	1	0	0	0	0	0	0
2	1	0	0	1	1	0	0	0	0
3	0	1	0	1	0	1	0	0	0
4	0	0	0	0	0	1	1	1	0
5	0	0	1	0	1	0	1	0	1
6	0	0	0	0	0	0	0	1	1

Рис.2.

а) ориентированный граф и его матрица инцидентности,

б) неориентированный граф и его матрица инцидентности

(*а*)

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

(*б*)

	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	1	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	1
6	0	0	0	1	1	0

Рис. 3. Матрицы смежности для графов на рис.2

Основным преимуществом матрицы смежности является тот факт, что за один шаг можно получить ответ на вопрос «существует ли ребро из *x* в *y?»*. Недостатком же является тот факт, что независимо от числа ребер объем занятой памяти составляет n^2. На практике это неудобство можно иногда уменьшить, храня целую строку (столбец) матрицы в одном машинном слове – это возможно для малых *n*.

Более экономным в отношении памяти (особенно в случае неплотных графов, когда *m* гораздо меньше n^2) является метод представления графа с помощью списка пар, соответствующих его ребрам. Пара (x, *y)* соответствует дуге (*x*,*y*), если граф ориентированный, и ребру (*x*, *y*) в случае неориентированного графа (рис. 4). Очевидно, что объем памяти в этом случае составляет 2*m*. Неудобством является большое число шагов – порядка *m* в худшем случае, – необходимое для получения множества вершин, к которым ведут ребра из данной вершины.

(а)

1	2
1	3
3	2
3	4
5	4
5	6
6	5

(б)

1	2
1	3
1	5
2	3
2	5
3	4
4	5
4	6
5	6

Рис. 4. Списки ребер, соответствующие графам на рис. 2

Ситуацию можно значительно улучшить, упорядочив множество пар лексикографически и применяя двоичный поиск, но лучшим решением во многих случаях оказывается структура данных, которую называют *списками инцидентности.* Она содержит для каждой вершины $v \in V$ список вершин *u,* таких что $v \to u$ (или $v - u$ в случае неориентированного графа). Точнее, каждый элемент такого списка является запи-

сью r, содержащей вершину r.*строка* и указатель r.*след* на следующую запись в списке *(r.след=nil* для последней записи в списке). Начало каждого списка хранится в таблице *НАЧАЛО;* точнее, *НАЧАЛО[v]* является указателем на начало списка, содержащего вершины из множества *(u: v → u) ((u: v – u)* для неориентированного графа). Весь такой список обычно неформально будем обозначать *ЗАПИСЬ[v],* а цикл, выполняющий определенную операцию для каждого элемента *u* из этого списка в произвольной, но четко установленной последовательности, соответствующей очередности элементов в списке, будем записывать «for $u \in$ *ЗАПИСЬ[v]* do ...».

Отметим, что для неориентированных графов каждое, ребро *(u, v)* представлено дважды: через вершину *v* в списке *ЗАПИСЬ[u]* и через вершину *u* в списке *ЗАПИСЬ[v]*. Во многих алгоритмах структура графа динамически модифицируется добавлением и удалением ребер. Каждый элемент списка может содержать указатель не только к следующему элементу, но и к предыдущему. Аналогичным способом определяем для каждой вершины *v* неориентированного графа список *ПРЕДШ[v],* содержащий вершины из множества *(u: v → u).*

Число ячеек памяти, необходимое для представления графа с помощью списков инцидентности, будет, очевидно, иметь порядок *m+n.* На рис. 5 представлены списки инцидентности, соответствующие графам на рис. 2.

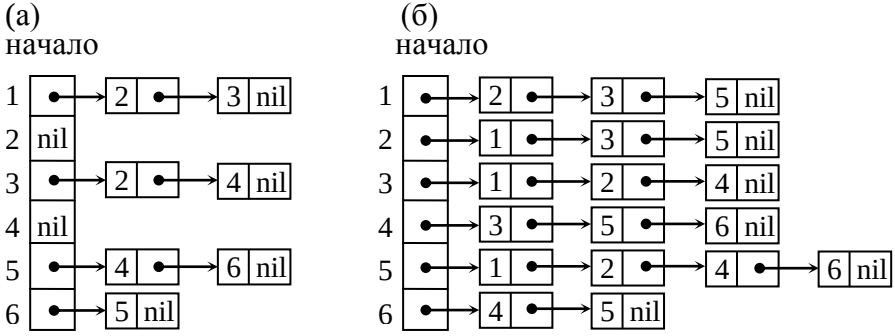


Рис. 5. Списки инцидентности ЗАПИСЬ, соответствующие графам на рис. 2

1.3. Алгоритмы

1. Алгоритм построения простого графа, имеющего заданную последовательностьстепеней

Шаг 1. $\{d_1, \ldots, d_n\}$- последовательность степеней, упорядоченная по невозрастанию. Выберем произвольное $d_k \neq 0$ и "изымем" d_k из последовательности, соединяя вершину x_k с первыми d_k вершинами, не считая саму вершину x_k.

Шаг 2. Упорядочим остаточную последовательность в порядке невозрастания.

Шаг 3. Шаги 1-2 выполнять до тех пор, пока не возникнет одна из следующих ситуаций:

а) все остаточные степени равны 0. В этом случае последовательность степеней является графической. Искомый граф получается в результате выполнения шагов, соответствующих порядку изъятия степеней;

б) одна из остаточных степеней отрицательна - это означает, что последовательность $\{d_1, \ldots, d_n\}$ не является графической, т. е. не существует простого графа, который ее реализует.

2. Алгоритм проверки связности неорграфа

Шаг 1. $G = (X, U)$ - данный неорграф. Для произвольной вершины $x_0 \in X$ найти множество $R(x_0)$. Перейти к шагу 2.

Шаг 2. Если $R(x_0) = X$, то граф является связным, иначе граф не является связным. Выдать соответствующее сообщение. Останов.

3. Алгоритм нахождения сильных компонент графа

Шаг 1. $G = (X, U)$ - данный граф. Определение сильных компонент графа (СК) начать с произвольной вершины $x_i \in X$. Найти $R(x_i)$ и $Q(x_i)$. Положить $СК(x_i) = R(x_i) \cap Q(x_i)$.

Шаг 2. Рассмотреть множество $\overline{X} = X \setminus (R(x_i) \cap Q(x_i))$ и для произвольной вершины $x_k \in \overline{X}$ найти $СК(x_i)$ на $\overline{X}$. Перейти к шагу 3.

Шаг 3. Если $\overline{X} \neq 0$, то перейти к шагу 2, иначе останов, так как все сильные компоненты определены.

Вопросы для самопроверки

1. Какие способы задания и представления графов знаете?
2. Какие бинарные операции над графами знаете?
3. Какие унарные операции над графами знаете?
4. Дать определение конденсации графа?
5. Что такое подграф?
6. Какой граф является дополнением полного графа?
7. Что представляет собой степень вершины?
8. Как определяется матрица смежности графа ?
9. Как определяется матрица инцидентности графа ?
10. Как построить матрицу достижимостей графа ?
11. Дать определение сильной компоненты графа.
12. Дать определение связного графа.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Задания

1. Написать программу, позволяющую осуществлять переход от матрицы смежности к матрице инциденций для ориентированного графа.
2. Написать программу, позволяющую осуществлять переход от матрицы смежности к матрице инциденций для неориентированного графа.
3. Написать программу, позволяющую строить простой граф с заданной последовательностью степеней, если он существует.
4. Написать программу, позволяющую для произвольного графа определять количество путей заданной длины из каждой вершины в каждую.
5. Написать процедуру для определения, является ли данный граф связным.
6. Написать программу, позволяющую находить сильные компоненты связного графа.
7. Написать программу, позволяющую получать матрицы достижимости и контрдостижимости для произвольного графа.
8. Написать программу, реализующую алгоритм определения уровней графа без контуров.
9. Написать программу, которая для заданных графов $G_1 = (X_1, U_1)$ и $G_2 = (X_2, U_2)$ строит объединение этих графов.
10. Написать программу, которая для заданных графов $G_1 = (X_1, U_1)$ и $G_2 = (X_2, U_2)$ строит пересечение этих графов.
11. Написать программу, которая переводит матрицу смежности в список ребер и список инцидентности для ориентированного графа

12. Написать программу, которая переводит матрицу инцидентности в список ребер и список инцидентности для неориентированного графа.

2.2. Порядок выполнения работы

1. Получить задание у преподавателя.
2. Разработать алгоритм решения задачи.
3. Реализовать полученный алгоритм.
4. Проанализировать результаты работы алгоритма.
5. Оформить отчет по лабораторной работе.

2.3. Содержание отчета

1. Номер и тема лабораторной работы.
2. Цель выполнения работы.
3. Схема алгоритма.
4. Исходные данные и результаты вычислений.
5. Анализ полученных результатов и вывод по работе.

2.4. Контрольные вопросы

1. С помощью какой структуры данных можно представить список инцидентности в Паскале ?
2. Как в Паскале представить список ребер ?
3. Какой способ представления графов является лучшим в смысле экономии памяти ?
4. Какой способ представления графов является худшим в смысле экономии памяти ?
5. Чем отличается матрица смежности неориентированного графа от матрицы смежности ориентированного ?
6. Как в матрице смежности представить информацию о кратных ребрах ?
7. Как в матрице инцидентности представит информацию о петлях ?

8. Как получить матрицу пересечения графов по матрицам смежности исходных графов ?
9. Как получить матрицу объединения графов по матрицам смежности исходных графов ?
10. Как получить матрицу пересечения графов по матрицам инцидентности исходных графов ?
11. Как получит матрицу объединения графов по матрицам инцидентности графов ?
12. Как с помощью матрицы смежности построит матрицу достижимостей ?

Лабораторная работа №4

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРОЦЕДУРЫ СОСТАВЛЕНИЯ СКНФ И СДНФ ДЛЯ ПФ

Цель работы: изучение специальных разложений формул переключательных функций (ПФ), нормальных и совершенных нормальных форм. Получение практических навыков программной реализации алгоритмических процедур.

Технические средства: IBM PC AT.

Программное средство: Delphi.

Теоретические сведения

Покажем, что для каждой ПФ существует ей равносильная специального вида. Пусть v есть 0 или 1. Введем обозначение $X_1^v = \begin{cases} x_1, v = 1 \\ \overline{x_1}, v = 0 \end{cases}$

Заметим, что ПФ $X_1^v = 1$ тогда и только тогда, когда $X_1 = v$, то есть $v^v = 1$.

Следовательно, ПФ $X_1^{v_1} \wedge X_2^{v_2} \wedge ... \wedge X_n^{v_n}$ на наборе $(v_1, v_2, ..., v_n)$ принимает значение 1, а на любом другом – значение 0. Аналогично ПФ $X_1^{v_1} \vee X_2^{v_2} \vee ... \vee X_n^{v_n}$ принимает значение 0 только на наборе $(\overline{v_1}, \overline{v_2}, ..., \overline{v_n})$, а на всех остальных наборах – 1.

Теорема 1. Для любой ПФ имеет место равносильность

$$F(X_1, X_2, ..., X_n) \equiv X_1 \wedge F(1, X_2, ..., X_n) \vee \overline{X}_1 \wedge$$
$$\wedge F(0, X_2, ..., X_n),$$

называемая *дизъюнктивным разложением по переменной* X_1.

Пример.

Для ПФ (X1→ X2 ^X3) определить дизъюнктивное разложение по переменной X1.

$$X_1 \to X_2 \wedge X_3 \equiv X_1 \wedge (1 \to X_2 \wedge X_3) \vee \overline{X}_1 \wedge (0 \to X_2 \wedge X_3) \equiv$$
$$\equiv X_1 \wedge X_2 \wedge X_3 \vee \overline{X}_1.$$

Следствие. Для любой ПФ $F(X_1, X_2, ..., X_n)$ и любого натурального **k** имеет место равносильность

$$F(X_1, X_2, ..., X_n) \equiv \bigvee_{(v_1, v_2, ..., v_k)} X_1^{v_1} \wedge X_2^{v_2} \wedge ... \wedge X_k^{v_k} \wedge$$
$$\wedge F(v_1, ..., v_k, X_{k+1}, ..., X_n),$$

называемая дизъюнктивным разложением $F(X_1, X_2, ..., X_n)$ *по переменным* $X_1, X_2, ..., X_n$.

Теорема 2. Для любой ПФ имеет место равносильность

$$F(X_1, X_2, ..., X_n) \equiv (X_1 \vee F(0, X_2, ..., X_n)) \wedge$$
$$\wedge (\overline{X}_1 \vee F(1, X_2, ..., X_n))$$

называемая конъюнктивным разложением по переменной X_1.

Пример. Для ПФ

определить конъюнктивное разложение по переменной X1.

$$X_1 \wedge X_2 \to \overline{X}_3 \equiv (X_1 \vee (0 \wedge X_2 \to \overline{X}_3)) \wedge (\overline{X}_1 \vee (1 \wedge X_2 \to$$
$$\to \overline{X}_3) \equiv \overline{X}_1 \vee (X_2 \to \overline{X}_3))$$

Следствие. Для любой ПФ $F(X_1, X_2, ..., X_n)$ и любого натурального **k** имеет место равносильность

$$F(X_1, X_2, ..., X_n) \equiv \bigwedge_{(v_1, v_2, ..., v_k)} (\overline{X_1^{v_1}} \vee \overline{X_2^{v_2}} \vee ... \vee \overline{X_k^{v_k}} \vee$$
$$\vee F(v_1, ..., v_k, X_{k+1}, ..., X_n)),$$

называемая конъюнктивным разложением $F(X_1, X_2, ..., X_n)$ *по переменным* $X_1, X_2, ..., X_n$.

Таким образом, для любой ПФ существует равносильная ей, содержащая только константы 0 и 1, символы $\wedge, \vee, \bar{}$ и переменные.

Определим некоторые канонические представления ПФ.

ПФ называется *элементарной конъюнкцией (дизъюнкцией),* если она является конъюнкцией (дизъюнкцией) переменных и отрицаний переменных.

Пример.

$X \wedge Y \wedge \overline{Z}$ - элементарная конъюнкция.

$\overline{X} \vee Z$ - элементарная дизъюнкция.

Говорят, что ПФ задана в *дизъюнктивной нормальной форме (ДНФ),* если она является дизъюнкцией элементарных конъюнкций.

Пример. $X \wedge \overline{Y} \wedge Z \vee X \wedge Y \vee \overline{Y} \wedge Z$ - ДНФ.

Говорят, что ПФ задана в *конъюнктивной нормальной форме (КНФ),* если она является конъюнкцией элементарных дизъюнкций.

Пример. $(\overline{X} \vee Y \vee Z) \wedge (X \vee \overline{Y})$ - КНФ.

На основе равносильных преобразований любая формула может быть приведена к нормальной форме (ДНФ или КНФ).

Совершенной дизъюнктивной нормальной формой (СДНФ) данной ПФ называется ДНФ, в которой каждая элементарная конъюнкция содержит все переменные – без отрицания или с отрицанием, но не вместе.

Совершенной конъюнктивной нормальной формой (СКНФ) данной ПФ называется КНФ, в которой каждая элементарная дизъюнкция содержит все переменные – без отрицания или с отрицанием, но не вместе.

Существует два способа перехода к совершенным формам табличный и аналитический.

Алгоритм 1

(табличный способ приведения к СДНФ)

1. Составить таблицу истинности данной ПФ.
2. Рассмотреть те строки, в которых формула принимает истинностное значение 1. Каждой такой строке поставить в соответствие элементарную конъюнкцию, причем переменная, принимающая значение 1, входит в нее без отрицания, а 0 – с отрицанием.
3. Образовать дизъюнкцию всех полученных элементарных конъюнкций, которая и составляет СДНФ.

Алгоритм 2

(табличный способ приведения к СКНФ)

1. Составить таблица истинности данной ПФ.
2. Рассмотреть те строки, в которых формула принимает истинностное значение 0. Каждой такой строке поставить в соответствие элементарную дизъюнкцию, причем переменная, принимающая значение 1, входит в нее с отрицанием, а 0 – без отрицания.
4. Образовать конъюнкцию всех полученных элементарных дизъюнкций, которая и составляет СКНФ.

Пример. Привести ПФ $X \vee \overline{Y} \leftrightarrow Z$ к совершенным нормальным формам.

Для приведения к совершенным нормальным формам воспользуемся алгоритмами 3 и 4. Построим таблицу истинности и на ее основе составим СДНФ и СКНФ.

X	Y	Z	$X \vee \overline{Y} \leftrightarrow Z$	Элементарные конъюнкции	Элементарные дизъюнкции
0	0	0	0		$X \vee Y \vee Z$
0	0	1	1	$\overline{X} \wedge \overline{Y} \wedge Z$	
0	1	0	1	$\overline{X} \wedge Y \wedge Z$	
0	1	1	0		$X \vee \overline{Y} \vee \overline{Z}$
1	0	0	0		$\overline{X} \vee Y \vee Z$
1	0	1	1	$X \wedge \overline{Y} \wedge Z$	
1	1	0	0		$\overline{X} \vee \overline{Y} \vee Z$
1	1	1	1	$X \wedge Y \wedge Z$	

СДНФ : $\overline{X} \wedge \overline{Y} \wedge Z \vee \overline{X} \wedge \overline{Y} \wedge Z \vee X \wedge \overline{Y} \wedge Z \vee X \wedge Y \wedge Z$

СКНФ :

$(X \vee Y \vee Z) \wedge (X \vee \overline{Y} \vee \overline{Z}) \wedge (\overline{X} \vee Y \vee Z) \wedge (\overline{X} \vee \overline{Y} \vee Z)$

Построение СДНФ

СДНФ булевой функции может быть построена по заданной таблицы истинности с помощью следующего алгоритма.

Алгоритм. Построение СДНФ

Вход: Вектор X : array [1..*n*] of string идентификаторов переменных,

матриц V:array [1..2^n, 1..*n*] of 0..1 всех различных наборов значения переменных, вектор *F*:array [1..2^n] of 0..1 соответствующих значений функции.

Выход: последовательность символов, образующих запись формул СДНФ для заданной функции.

f:=false { признак присутствия левого оператора дизъюнкции}

```
        for i from 1 to 2^n do
        if F[i]=1 then
            if f  then
              yield '∨' {добавление в формулу знака дизъюнкции}
            else
                f:=true
            end if
        g:=false  {признак присутствия левого оператора конъюнкции}
        for j from 1 to n do
            if g then
                yield '∧' {добавление в формулу знака конъюнкции}
            else
                g:=true
            end if
            if V[i,j] = 0 then
        yield '¬' {добавление в формулу знака отрицания}
        end if
        yield X[j] {добавление в формулу идентификатора переменной}
      end for
   end if
end for
```

Замечание. Если зафиксировать порядок перечисления переменных в таблице истинности и порядок перечисления кортежей значений, то алгоритм построения СДНФ дает синтаксически однозначный результат. Переменные всегда перечисляются в лексикографическом порядке, а кортежи булевских значений – в порядке возрастания целых чисел, задаваемых кортежами как двоичными шкалами. Такой порядок считается установленным.

Алгоритм вычисления значения булевой функции

Некоторые классы формул допускают более эффективную интерпретацию по сравнению с алгоритмом Eval. Рассмотрим алгоритм вычисления значения булевой функции, заданной в виде СДНФ, для заданных значений переменных x_1, ... , x_n. В этом алгоритме используется следующее представление данных. СДНФ задана массивом f array [1..k,1..n] of 0..1, где строка f [i,*] содержит набор значений $\nu_1,...,\nu_n$, для которого $f(\nu_1,...,\nu_n)=1$, $i=\overline{1,k}$, $k \le n$.

Быстрое вычисление значения СДНФ имеет не только теоретическое, но и большое практическое значение. Например, во многих современных программах с графическим интерфейсом для составления сложных логических условий используется наглядный бланк в виде таблицы: в клетках записывается условие, причем клетки одного столбца считаются соединенными конъюнкциями, а столбцы – дизъюнкцией, т.е. образуют ДНФ (или наоборот, в таком случае получается КНФ). В частности, так устроен графический интерфейс QBE (Query-by-Example), применяемый для формулировки логических условий при запросе к СУБД.

Алгоритм. Алгоритм вычисления СДНФ

Вход: массив, представляющий СДНФ: f : **array** [1..k, 1..n] **of** 0..1;

множество значений переменных x : **array** [1..n] **of** 0..1.

Выход: 0..1 – значение булевой функции.

for i from 1 to k do
 for j from 1 to n do
  if $f[i,j] \neq x[j]$ then
   next for i { $x_j^{\nu_j} = 0 \Rightarrow x_1^{\nu_1} \wedge ... \wedge x_n^{\nu_n} = 0$ }
  end if
 end for

return 1

$\{ x_1^{\nu_1} \vee \ldots \vee x_n^{\nu_n} = 1 \Rightarrow \vee_{(\nu_1,\ldots,\nu_n)} x_1^{\nu_1} \wedge \ldots \wedge x_n^{\nu_n} = 1 \}$

end for

return 0 {все слагаемые в дизъюнкции = 0}

Замечание. В алгоритме использован оператор **next**, которому здесь придается следующая семантика: выполнение текущего цикла прерывается, а выполнение программы продолжается со следующего шага цикла, указанного в оператора **next**. Такого рода операторы называются *операторами структурного перехода*. Операторы структурного перехода присутствуют в некоторых реальных языках программирования (например оператор **continue** в языке С), хотя обычно имеют более ограниченную семантику по сравнению с использованным здесь оператором **next.**

Этот алгоритм в худшем случае выполняет *kn* сравнений, а в среднем – гораздо меньше, т.е. он существенно эффективнее общего алгоритма интерпретации.

Вопросы для самопроверки

1. Что понимают под высказыванием? Привести пример высказывания.
2. Какое предложение не будет высказыванием? Привести пример такого предложение.
3. Привести пример сложного высказывания.
4. Какие операции над высказываниями существуют ?
5. Каков приоритет операций над высказываниями ?
6. Дать определение пропозициональной формулы.
7. Дать определение булевой функции
8. Какие способы задания булевых функций существуют ?
9. Дать определение таблицы истинности.

10. Сколько существует различных булевых функций от n переменных?
11. Дать определение импликации двух высказываний.
12. Дать определение эквиваленции двух высказываний.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ
Задание

Написать программу для получения интерпретации формулы.

1. $(X \to Y) \wedge (Y \to Z) \to (X \to Z)$.
2. $(X \to (Y \to Z)) \to (Y \to (X \to Z))$.
3. $X \wedge Y \vee \overline{X} \leftrightarrow \overline{X} \vee \overline{Y}$.
4. $(X \leftrightarrow Y) \wedge (Y \vee Z) \to (Z \wedge X)$.
5. $\overline{X} \vee \overline{X \to Z} \vee \overline{X} \wedge Z \vee X \wedge Y$.
6. $\overline{X} \wedge \overline{Y} \wedge Z \vee \overline{\overline{X} \to \overline{Y}} \wedge Z \vee Y \to Z$.
7. $\overline{(X \wedge Y \wedge Z)} \wedge \overline{(X \to Z)}$.
8. $\overline{X \wedge Y \wedge \overline{Z}} \vee X \to Y \vee \overline{X} \wedge Y$.
9. $(X \vee \overline{Y}) \wedge \overline{(Y \vee Z)} \wedge (Z \wedge X)$.
10. $\overline{((\overline{X} \wedge \overline{Y}) \to X)} \wedge (Z \to \overline{(\overline{X} \wedge \overline{Y})})$.
11. $\overline{(\overline{X} \vee \overline{Y})} \to \overline{(X \leftrightarrow Z)}$.
12. $(Y \wedge X) \leftrightarrow Z \vee \overline{(\overline{X} \wedge \overline{Y})}$.

2.2. Порядок выполнения работы

1. Получить задание у преподавателя.
2. Разработать алгоритм решения задачи.
3. Реализовать полученный алгоритм.
4. Проанализировать результаты работы алгоритма.
5. Оформить отчет по лабораторной работе.

2.3. Содержание отчета

1. Номер и тема лабораторной работы.
2. Цель выполнения работы.
3. Схема алгоритма.
4. Исходные данные и результаты вычислений. Проделать все необходимые преобразования формул вручную и сравнить с результатом программы.
5. Анализ полученных результатов и вывод по работе.

2.4. Контрольные вопросы

1. Какие символы используют для записи формул?
2. Как определить тип пропозициональной формулы ?
3. Какие логические значения знаете ?
4. Какие функции в Паскале возвращают логические значения ?
5. Как в Паскале описать логический тип данных ?
6. Какой оператор в Паскале соответствует логической операции конъюнкция ?
7. Какой оператор в Паскале соответствует логической операции дизъюнкция ?
8. Какой оператор в Паскале соответствует логической операции отрицание ?
9. Каким образом в программе предусмотрен приоритет операций ?
10. Какое максимальное количество переменных может содержать формула в разработанной программе ?
11. Учитывается ли в программе возможность наличия скобок в записи формулы ?
12. Как учитывается в программе возможность наличия скобок в записи формулы ?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Новиков Ф.А. Дискретная математика для программистов /Ф.А. Новиков. СПб.: Питер, 2004. 364 с.

2. Судоплатов С.В. Элементы дискретной математики / С.В. Судоплатов, Е.В. Овчинникова. М.: ИНФРА-М, 2002. 280 с.

3. Судоплатов С.В. Математическая логика и теория алгоритмов: учебник / С.В. Судоплатов, Е.В. Овчинникова. М.: ИНФРА-М, 2004. – 224 с.

4. Иванов Б.Н. Дискретная математика. Алгоритмы и программы / Б.Н. Иванов. М.: Лаборатория базовых знаний, 2003. – 288 с.

5. Кузнецов О.П. Дискретная математика для инженера / О.П. Кузнецов. СПб.: Лань, 2005. – 400 с.

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ
по дисциплине «Дискретная математика»
для студентов специальности 230104 «Системы автоматизированного проектирования» очной формы обучения

Составители

Собенина Ольга Валерьевна
Кордюкова Елена Николаевна

В авторской редакции

Компьютерный набор О.В. Собениной

Подписано в печать 12.05.2009.
Формат 60x84/16. Бумага для множительных аппаратов.
Усл. печ. л. 2,8. Уч.-изд. л. . Тираж 70 экз. «С»
Зак. №