МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Классификация бинарных отношений и системы замыканий

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ПРИКЛАДНАЯ УНИВЕРСАЛЬНАЯ АЛГЕБРА»

студентки 3 курса 331 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Зиминой Ирины Олеговны

Преподаватель

профессор, д.ф.-м.н. ___________ В. А. Молчанов

подпись, дата

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В прикладной универсальной алгебре существует классификация бинарных отношений, исходя из того какими свойствами обладают эти отношения. Задача данной работы рассмотреть основные свойства бинарных отношений, определить классификацию бинарных отношений, изучить построение замыканий бинарных отношений.

1. Цель работы и порядок выполнения

Цель работы — изучение основных свойств бинарных отношений и операций замыкания бинарных отношений.

Порядок выполнения работы:

1. Разобрать основные определения видов бинарных отношений и разработать алгоритмы классификации бинарных отношений.
2. Изучить свойства бинарных отношений и рассмотреть основные системы замыкания на множестве бинарных отношений.
3. Разработать алгоритмы построения основных замыканий бинарных отношений.

2. Основные определения видов бинарных отношений

Подмножества декартова произведения $A \times B$ множеств A и B называют бинарными отношениями между элементами множеств A, B и обозначаются строчными греческими буквами: $\rho, \sigma, \dots, \rho_1, \rho_2, \dots$.

Для бинарного отношения $\rho \subset A \times B$ область определения D_ρ и множество значений E_ρ определяются как подмножества соответствующих множеств A и B по следующим формулам:

$$D_\rho = \{a : (a,b) \in p \text{ для некоторого } b \in B\},$$

$$E_\rho = \{b : (a,b) \in p \text{ для некоторого } a \in A\}.$$

Свойства бинарных отношений.

Бинарное отношение $\rho \subset A \times B$ называется:

- рефлексивным, если $(a,a) \in \rho$ для любого $a \in A$;
- антирефлексивным, если $(a,a) \notin \rho$ для любого $a \in A$;
- симметричным, если $(a,b) \in \rho \implies (b,a) \in \rho$, для любых $a, b \in A$;
- антисимметричным, если $(a,b) \in \rho$ и $(b,a) \in \rho \implies a = b$, для любых $a, b \in A$;
- транзитивным, если $(a,b) \in \rho$ и $(b,c) \in \rho \Rightarrow (a,c) \in \rho$ для любых $a, b, c \in A$.

3. Алгоритмы для определения свойств бинарных отношений

Пусть ρ – бинарное отношение на множестве $A = \{a_1, \dots, a_N\}$, мощности N. В этом случае матрица бинарного отношения p – это матрица $M(\rho)$ с размерностью $N \times N$, которая определяется следующим образом:

$$M(\rho)_{ij} = \begin{cases} 1, & \text{если } (a_i, a_j) \in \rho \\ 0, & \text{если } (a_i, a_j) \notin \rho \end{cases} \text{ для всех } i, j = \overline{1, N}$$

3.1 Проверка свойства рефлексивности.

Идея алгоритма: если свойство рефлексивности выполняется, то на главной диагонали матрицы смежности будут все единицы, в ином случае отношение не обладает свойством рефлексивности.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на рефлексивность: «отношение является рефлексивным» или «отношение не является рефлексивным».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
2) Если $M[i][i] = 1$, то увеличиваем счетчик q на единицу.
3) После выхода из цикла проводится проверка с счётчиками.
4) Если счётчик $q = N$, то на выход идет сообщение: «бинарное отношение ρ является рефлексивным».
5) Иначе на выход идет сообщение: «бинарное отношение ρ не является рефлексивным».

Сложность алгоритма: $O(n)$.

3.2 Проверка свойства антирефлексивности.

Идея алгоритма: если свойство антирефлексивности выполняется, то на главной диагонали матрицы смежности будут все нули, в ином случае отношение не обладает свойством антирефлексивности.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на антирефлексивность: «отношение является антирефлексивным» или «отношение не является антирефлексивным».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
2) Если $M[i][i] = 0$, то увеличиваем счетчик q на единицу.
3) После выхода из цикла проводится проверка с счётчиками.
4) Если счётчик $q = N$, то на выход идет сообщение: «бинарное отношение ρ является антирефлексивным».
5) Иначе на выход идет сообщение: «бинарное отношение ρ не является антирефлексивным».

Сложность алгоритма: $O(n)$.

3.3 Проверка свойства симметричности.

Идея алгоритма: если свойство симметричности выполняется, то в такой матрице единицы симметричны относительно главной диагонали, в ином случае свойство симметричности не выполняется.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на симметричность: «отношение является симметричным» или «отношение не является симметричным».

Описание алгоритма:

1) Запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
2) Когда $i \neq j$, если $M[i][j] \neq M[j][i]$, то переменная q = false.
3) Если после выхода из цикла q = true, то выводится сообщение: «бинарное отношение ρ является симметричным».
4) Иначе на вход выводится сообщение: «бинарное отношение ρ не является симметричным».

Сложность алгоритма: $O(n^2)$.

3.4 Проверка свойства антисимметричности.

Идея алгоритма: если свойство антисимметричности выполняется, то в такой матрице отсутствуют единицы симметричные относительно главной диагонали, в ином случае свойство антисимметричности не выполняется.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на антисимметричность: «отношение является антисимметричным» или «отношение не является антисимметричным».

Описание алгоритма:

1) Запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
2) Когда $i \neq j$, если $M[i][j] = M[j][i] = 1$, то переменная q = false.
3) Если после выхода из цикла q = true, то выводится сообщение: «бинарное отношение ρ является антисимметричным».
4) Иначе на вход выводится сообщение: «бинарное отношение ρ не является антисимметричным».

Сложность алгоритма: $O(n^2)$.

3.5 Проверка свойства транзитивности.

Идея алгоритма: свойство транзитивности выполняется если квадрат матрицы является частью исходной матрицы бинарного отношения.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на транзитивность: «отношение является транзитивным» или «отношение не является транзитивным».

Описание алгоритма:

1) Строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения ρ.
2) Проводится сравнение матриц $M(N * N)$ и $W(N * N)$.
3) Для этого запускается цикл для прохода по двум одновременно: по матрице M от элемента $M[0][0]$ до элемента $M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.
5) Если $W[i][j] > M[j][i]$, то переменная q = false.
6) Если после выхода из цикла q = true, то выводится сообщение: «бинарное отношение ρ является транзитивным».
7) Иначе на вход выводится сообщение: «бинарное отношение ρ не является транзитивным».

Сложность алгоритма: $O(n^3)$.

4. Классификация бинарных отношений

Исходя из набора свойств, определенных для бинарного отношения это отношение можно отнести к одному из видов:

– отношение квазипорядка (бинарное отношение рефлексивное и транзитивное);
– отношение эквивалентности (бинарное отношение рефлексивное, симметричное и транзитивное);
– отношение частичного порядка (бинарное отношение рефлексивное, антисимметричное и транзитивное);
– отношение строгого порядка (бинарное отношение антирефлексивное, антисимметричное и транзитивное);
– отношение доминирования (бинарное отношение антирефлексивное и антисимметричное).

5. Алгоритмы для определения видов бинарных отношений

5.1 Проверка отношения на квазиподрядок.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на квазипорядок: «отношение является отношением квазипорядка» или «отношение не является отношением квазипорядка».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
2) Если $M[i][i] = 1$, то увеличиваем счетчик q на единицу.
3) Если счётчик $q = N$, то алгоритм переходит к проверке на отношения на транзитивность, иначе на вход идет сообщение: «отношение не является отношением квазипорядка».

4) Проверка на транзитивность: строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения ρ.
5) Проводится сравнение матриц $M(N * N)$ и $W(N * N)$.
6) Для этого запускается цикл для прохода по двум одновременно: по матрице M от элемента $M[0][0]$ до элемента $M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.
7) Если $W[i][j] > M[j][i]$, то переменная q = false.
8) Если после выхода из цикла q = true, то выводится сообщение: «отношение является отношением квазипорядка».
9) Иначе на вход выводится сообщение: «отношение не является отношением квазипорядка».

Сложность алгоритма: $O(n^2)$.

5.2 Проверка отношения на эквивалентность.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на эквивалентность: «отношение является отношением эквивалентности» или «отношение не является отношением эквивалентности».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Если $M[i][i] = 1$, то увеличиваем счетчик q на единицу.
3) Если счётчик $q = N$, то алгоритм переходит к проверке на отношения на симметричность, иначе на вход идет сообщение: «отношение не является отношением эквивалентности».
4) Проверка на симметричность: запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
5) Когда $i \neq j$, если $M[i][j] \neq M[j][i]$, то переменная q = false.

6) Если после выхода из цикла r = true, то алгоритм переходит к проверке на отношения на транзитивность, иначе на вход идет сообщение: «отношение не является отношением эквивалентности».
7) Проверка на транзитивность: строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения ρ. Проводится сравнение матриц $M(N * N)$ и $W(N * N)$.
8) Для этого запускается цикл для прохода по двум одновременно: по матрице M от элемента $M[0][0]$ до элемента $M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.
9) Если $W[i][j] > M[j][i]$, то переменная q = false.
10) Если после выхода из цикла q = true, то выводится сообщение: «отношение является отношением эквивалентности».
11) Иначе на вход выводится сообщение: «отношение не является отношением эквивалентности».

Сложность алгоритма: $O(n^3)$.

5.3 Проверка отношения на частичный порядок.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на частичный порядок: «отношение является отношением частичного порядка» или «отношение не является отношением частичного порядка».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Если $M[i][i] = 1$, то увеличиваем счетчик q на единицу.
3) Если счётчик $q = N$, то алгоритм переходит к проверке на отношения на антисимметричность, иначе на вход идет сообщение: «отношение не является отношением частичного порядка».

4) Проверка на антисимметричность: запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
5) Когда $i \neq j$, если $M[i][j] = M[j][i] = 1$, то переменная q = false.
6) Если после выхода из цикла r = true, то алгоритм переходит к проверке на отношения на транзитивность, иначе на вход идет сообщение: «отношение не является отношением частичного порядка».
7) Проверка на транзитивность: строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения ρ.
8) Проводится сравнение матриц $M(N*N)$ и $W(N*N)$.
9) Для этого запускается цикл для прохода по двум одновременно: по матрице M от элемента $M[0][0]$ до элемента $M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.
10) Если $W[i][j] > M[j][i]$, то переменная q = false.
11) Если после выхода из цикла q = true, то выводится сообщение: «отношение является отношением частичного порядка».
12) Иначе на вход выводится сообщение: «отношение не является отношением частичного порядка».

Сложность алгоритма: $O(n^3)$.

5.4 Проверка отношения на строгий порядок.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на частичный порядок: «отношение является отношением строгого порядка» или «отношение не является отношением строгого порядка».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Если $M[i][i] = 0$, то увеличиваем счетчик q на единицу.

3) Если счётчик $q = N$, то алгоритм переходит к проверке на отношения на антисимметричность, иначе на вход идет сообщение: «отношение не является отношением строгого порядка».
4) Проверка на антисимметричность: запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
5) Когда $i \neq j$, если $M[i][j] = M[j][i]$, то переменная q = false.
6) Если после выхода из цикла r = true, то алгоритм переходит к проверке на отношения на транзитивность, иначе на вход идет сообщение: «отношение не является отношением строгого порядка».
7) Проверка на транзитивность: строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения ρ.
8) Проводится сравнение матриц $M(N * N)$ и $W(N * N)$.
9) Для этого запускается цикл для прохода по двум одновременно: по матрице M от элемента $M[0][0]$ до элемента $M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.
10) Если $W[i][j] > M[j][i]$, то переменная q = false.
11) Если после выхода из цикла q = true, то выводится сообщение: «отношение является отношением строго порядка».
12) Иначе на вход выводится сообщение: «отношение не является отношением строго порядка».

Сложность алгоритма: $O(n^3)$.

5.5 Проверка отношения на доминирование.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: результат проверки на частичный порядок: «отношение является отношением доминирования» или «отношение не является отношением доминирования».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Если $M[i][i]=0,$ то увеличиваем счетчик q на единицу.
3) Если счётчик $q=N$, то алгоритм переходит к проверке на отношения на антисимметричность, иначе на вход идет сообщение: «отношение не является отношением доминирования».
4) Проверка на антисимметричность: запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
5) Когда $i \neq j$, если $M[i][j]=M[j][i],$ то переменная q = false.
6) Если после выхода из цикла r = true, то на вход сообщение: «отношение является отношением доминированием», иначе на вход идет сообщение: «отношение не является отношением доминирования».

Сложность алгоритма: $O(n^2)$.

6. Замыкания бинарных отношений

Замыканием отношения R относительно свойства P называют такое множество R^*, что:

1) $R \subset R^*$.
2) R^* обладает свойством P.
3) R^* является подмножеством любого другого отношения, содержащего R и обладающего свойством P.

4 типа замыкания отношений:

- рефлексивное;
- симметричное;
- транзитивное;
- эквивалентное.

На множестве $P(A)$ всех бинарных отношений между элементами множества A следующие отображения являются операторами замыканий:

1) $f_r(\rho)=\rho \cup \Delta_A$ – наименьшее рефлексивное бинарное отношение, содержащее отношение $\rho \subset A^2$;

2) $f_s(\rho) = \rho \cup \rho^{-1}$ – наименьшее симметричное бинарное отношение, содержащее отношение $\rho \subset A^2$;
3) $f_t(\rho) = \bigcup_{n=1}^{\infty} \rho^n$ – наименьшее транзитивное бинарное отношение, содержащее отношение $\rho \subset A^2$;
4) $f_{e\rho}(\rho) = f_t f_s f_r(\rho)$ – наименьшее отношение эквивалентности, содержащее отношение $\rho \subset A^2$.

Примеры замыканий отношения:

Пусть на множестве $A = \{1, 2, 3, 4\}$ задано отношение $R = \{(1, 2), (3, 4), (4,2)\}$. Отношение не симметрично, ни рефлексивно и не транзитивно.

- Замыканием R относительно свойства рефлексивности является:

$$R^* = \{(1, 2), (3, 4), (4, 2), (1, 1), (2, 2), (3, 3), (4, 4)\}.$$

- Замыканием R относительно свойства симметричности является:

$$R^* = \{(1, 2), (3, 4), (4, 2), (2, 1), (4, 3), (2, 4)\}.$$

- Замыканием R относительно свойства транзитивности является:

$$R^* = \{(1, 2), (3, 4), (4, 2), (3, 2)\}.$$

7. Алгоритмы построения замыканий бинарных отношений

7.1 Построение исходного замыкания.

Для построения любых замыканий, сначала нужно вывести исходное отношение.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: исходное отношение.

Описание алгоритма:

1) Запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
2) Если $M[i][j] = 1$, то пара (i, j) добавляется в вектор пар $a(N^2)$.
3) После выхода из цикла, выводится построенный вектор пар.

7.2 Построение рефлексивного замыкания.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: замыкание относительно свойства рефлексивности.

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
2) Если $M[i][i] = 0,$ то пара (i, i) добавляется в вектор пар a (где уже хранится исходное отношение).
3) После выхода из цикла, выводится построенный вектор пар.

Сложность алгоритма: $O(n)$.

7.3 Построение симметричного замыкания.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: замыкание относительно свойства симметричности.

Описание алгоритма:

1) Запускается цикл прохода вектора пар a (где уже хранится исходное отношение) по индексу i от 0-го до $q - 1$, где q – переменная, равная размеру вектора пар исходного отношения.
2) В вектор пар a добавляются эти же пары, только в формате (j, i). Если рассматривать это в контексте матрицы, то добавляется элемент $M[j][i],$ и j – первый элемент пары, i – второй элемент пары.
3) С помощью сета убираются дубликаты пар.
4) Выводится построенный сет.

Сложность алгоритма: $O(n^2 * \log n)$.

7.4 Построение транзитивного замыкания.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: замыкание относительно свойства транзитивности.

Описание алгоритма:

1) Строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения p.
2) Создается матрица $copy\_M$, с размерностью $N \times N$, которая является копией исходной матрицы.
3) Пока бинарное отношение представленное матрицей $copy\_M$ не транзитивное.
4) Проводится сравнение матриц $copy\_M(N * N)$ и $W(N * N)$. Для этого запускается цикл для прохода по двум одновременно: по матрице $copy\_M$ от элемента $copy\_M[0][0]$ до элемента $copy\_M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.
5) Если $copy\_M[i][j] = 0$ и $W[i][j] = 1$, то пара (i, j) добавляется в вектор пар a.
6) Элемент $copy\_M[i][j]$ становится равным единице.
7) Если бинарное отношение представленное матрицей $copy\_M$ не транзитивное, то матрица W становится матрицей произведения матриц $copy\_M$ и M.
8) После выхода из цикла (т.е, бинарное отношение представленное матрицей $copy\_M$– транзитивное), выводится построенный вектор пар.

Сложность алгоритма: $O(n^4)$.

7.5 Построение эквивалентного замыкания.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: замыкание относительно свойства эквивалентности.

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Если $M[i][i] = 0$, то пара (i, j) добавляется в вектор пар a.
3) В вектор пар a, добавляются эти же пары, только в формате (j, i).
4) Строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения p.

5) Создается матрица $copy\_M$, с размерностью $N \times N$, которая является копией исходной матрицы.
6) Пока бинарное отношение представленное матрицей $copy\_M$ не транзитивное.
7) Проводится сравнение матриц $copy\_M(N * N)$ и $W(N * N)$. Для этого запускается цикл для прохода по двум одновременно: по матрице $copy\_M$ от элемента $copy\_M[0][0]$ до элемента $copy\_M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.
8) Если $copy\_M[i][j] = 0$ и $W[i][j] = 1$, то пара (i, j) добавляется в вектор пар a.
9) Элемент $copy\_M[i][j]$ становится равным единице.
10) Если бинарное отношение представленное матрицей $copy\_M$ не транзитивное, то матрица W становится матрицей произведения матриц $copy\_M$ и M.
11) После выхода из цикла (т.е, бинарное отношение представленное матрицей $copy\_M$– транзитивное), выводится построенный вектор пар.
12) С помощью сета убираются дубликаты пар.
13) Выводится построенный сет.

Сложность алгоритма: $O(n^4)$.

8. Реализация рассмотренных алгоритмов

8.1 Алгоритмы для определения свойств бинарных отношений:

– **Проверка свойства рефлексивности.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
```

```
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    int q = 0;

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 1) {
            ++q;
        }
    }

    if (q == n) {
        cout << "бинарное отношение ρ является рефлексивным";
    } else {
        cout << "бинарное отношение ρ не является рефлексивным";
    }

    return 0;

}
```

Результаты тестирования:

```
5
1 0 0 0 0
0 1 0 0 0
0 0 1 1 1
0 0 0 1 1
0 0 0 1 1
бинарное отношение ρ является рефлексивным
```

– **Проверка свойства антирефлексивности.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
```

```
    }

    int q = 0;

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 0) {
            ++q;
        }
    }

    if (q == n) {
        cout << "бинарное отношение ρ является антирефлексивным";
    } else {
        cout << "бинарное отношение ρ не является антирефлексивным";
    }

    return 0;

}
```

Результаты тестирования:

```
5
0 0 0 0 1
1 0 0 0 0
1 0 0 1 1
0 1 0 0 1
0 1 1 1 0
бинарное отношение ρ является антирефлексивным
```

- **Проверка свойства симметричности.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    bool q = true;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
```

```
            if (i != j && m[i][j] != m[j][i]) {
                q = false;
            }
        }
    }

    if (q) {
        cout << "бинарное отношение ρ является симметричным";
    } else {
        cout << "бинарное отношение ρ не является симметричным";
    }

    return 0;

}
```

Результаты тестирования:

```
5
0 1 0 0 1
1 1 0 0 0
0 0 1 1 1
0 0 1 0 0
1 0 1 0 1
бинарное отношение ρ является симметричным
```

– **Проверка свойства антисимметричности.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    bool q = true;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i != j && m[i][j] == m[j][i] && m[i][j] == 1) {
                q = false;
            }
        }
```

```
    }

    if (q) {
        cout << "бинарное отношение ρ является антисимметричным";
    } else {
        cout << "бинарное отношение ρ не является антисимметричным";
    }

    return 0;

}
```

Результаты тестирования:

```
5
1 0 0 0 1
0 1 0 0 0
1 0 1 0 0
0 0 1 0 0
0 0 0 1 1
бинарное отношение ρ является антисимметричным
```

– **Проверка свойства транзитивности.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    int w[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            w[i][j] = 0;
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int l = 0; l < n; ++l) {
            int s = 0;
            for (int j = 0; j < n; ++j) {
```

```
                s += m[i][j] * m[j][l];
            }
            w[i][l] += s;
            if (w[i][l] > 1) {
                w[i][l] = 1;
            }
        }
    }

    bool q = true;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if ((w[i][j] > m[i][j]) {
                q = false;
            }
        }
    }

    if (!q) {
        cout << "бинарное отношение ρ не является транзитивным";
    } else {
        cout << "бинарное отношение ρ является транзитивным";
    }

    return 0;

}
```

Результаты тестирования:

```
5
0 0 0 0 0
1 0 0 0 0
1 1 0 0 0
1 1 0 0 0
1 1 1 0 0
бинарное отношение ρ является транзитивным
```

8.2 Алгоритмы для определения видов бинарных отношений:

– **Проверка отношения на квазипорядок.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];
```

```
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    int q = 0;

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 1) {
            ++q;
        }
    }

    if (q != n) {
        cout << "отношение не является отношением квазипорядка";
        return 0;
    } else {
        int w[n][n];

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                w[i][j] = 0;
            }
        }

        for (int i = 0; i < n; ++i) {
            for (int l = 0; l < n; ++l) {
                int s = 0;
                for (int j = 0; j < n; ++j) {
                    s += m[i][j] * m[j][l];
                }
                w[i][l] += s;
                if (w[i][l] > 1) {
                    w[i][l] = 1;
                }
            }
        }

        bool r = true;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (w[i][j] > m[i][j]) {
                    r = false;
                }
            }
        }

        if (!r) {
            cout << "отношение не является отношением квазипорядка";
        } else {
            cout << "отношение является отношением квазипорядка";
        }
    }

    return 0;

}
```

Результаты тестирования:

```
5
1 0 0 0 0
0 1 0 0 1
0 0 1 0 0
0 0 1 1 0
0 0 0 0 1
отношение является отношением квазипорядка
```

– **Проверка отношения на эквивалентность.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    int q = 0;

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 1) {
            ++q;
        }
    }

    if (q != n) {
        cout << "отношение не является отношением эквивалентности";
        return 0;
    } else {

        bool r = true;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (i != j && m[i][j] != m[j][i]) {
                    r = false;
                }
            }
        }

        if (!r) {
            cout << "отношение не является отношением эквивалентности";
            return 0;
```

```
        } else {
            int w[n][n];

            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    w[i][j] = 0;
                }
            }

            for (int i = 0; i < n; ++i) {
                for (int l = 0; l < n; ++l) {
                    int s = 0;
                    for (int j = 0; j < n; ++j) {
                        s += m[i][j] * m[j][l];
                    }
                    w[i][l] += s;
                    if (w[i][l] > 1) {
                        w[i][l] = 1;
                    }
                }
            }

            r = true;

            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    if (w[i][j] > m[i][j]) {
                        r = false;
                    }
                }
            }

            if (!r) {
                cout << "отношение не является отношением эквивалентности";
            } else {
                cout << "отношение является отношением эквивалентности";
            }
        }
    }

    return 0;
}
```

Результаты тестирования:

```
5
1 0 0 0 0
0 1 0 1 0
0 0 1 0 0
0 1 0 1 0
0 0 0 0 1
отношение является отношением эквивалентности
```

– **Проверка отношения на частичный порядок.**

Программный код:

```
#include <iostream>
```

```
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    int q = 0;

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 1) {
            ++q;
        }
    }

    if (q != n) {
        cout << "отношение не является отношением частичного порядка";
        return 0;
    } else {

        bool r = true;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (i != j && m[i][j] == m[j][i] && m[i][j] == 1) {
                    r = false;
                }
            }
        }

        if (!r) {
            cout << "отношение не является отношением частичного порядка";
            return 0;
        } else {
            int w[n][n];

            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    w[i][j] = 0;
                }
            }

            for (int i = 0; i < n; ++i) {
                for (int l = 0; l < n; ++l) {
                    int s = 0;
                    for (int j = 0; j < n; ++j) {
                        s += m[i][j] * m[j][l];
                    }
                    w[i][l] += s;
                    if (w[i][l] > 1) {
                        w[i][l] = 1;
                    }
```

```
                }
            }

            r = true;

            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    if (w[i][j] > m[i][j]) {
                        r = false;
                    }
                }
            }

            if (!r) {
                cout << "отношение не является отношением частичного порядка";
            } else {
                cout << "отношение является отношением частичного порядка";
            }
        }
    }

    return 0;
}
```

Результаты тестирования:

```
5
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 1 1 0
1 0 0 0 1
отношение является отношением частичного порядка
```

– **Проверка отношения на строгий порядок.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    int q = 0;
```

```
for (int i = 0; i < n; ++i) {
    if (m[i][i] == 0) {
        ++q;
    }
}

if (q != n) {
    cout << "отношение не является отношением строгого порядка";
    return 0;
} else {

    bool r = true;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i != j && m[i][j] == m[j][i] && m[i][j] == 1) {
                r = false;
            }
        }
    }

    if (!r) {
        cout << "отношение не является отношением строгого порядка";
        return 0;
    } else {
        int w[n][n];

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                w[i][j] = 0;
            }
        }

        for (int i = 0; i < n; ++i) {
            for (int l = 0; l < n; ++l) {
                int s = 0;
                for (int j = 0; j < n; ++j) {
                    s += m[i][j] * m[j][l];
                }
                w[i][l] += s;
                if (w[i][l] > 1) {
                    w[i][l] = 1;
                }
            }
        }

        r = true;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (w[i][j] > m[i][j]) {
                    r = false;
                }
            }
        }

        if (!r) {
            cout << "отношение не является отношением строгого порядка";
        } else {
            cout << "отношение является отношением строгого порядка";
        }
    }
```

```
    }

    return 0;
}
```

Результаты тестирования:

```
5
0 0 0 0 0
1 0 0 0 0
1 1 0 0 0
1 1 0 0 0
1 1 1 0 0
отношение является отношением строгого порядка
```

- **Проверка отношения на доминирование.**

Программный код:

```
#include <iostream>
#include <windows.h>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    int m[n][n];

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    int q = 0;

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 0) {
            ++q;
        }
    }

    if (q != n) {
        cout << "отношение не является отношением доминирования";
        return 0;
    } else {

        bool r = true;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (i != j && m[i][j] == m[j][i] && m[i][j] == 1) {
                    r = false;
                }
```

```
            }
        }

        if (r) {
            cout << "отношение является отношением доминирования";
        } else {
            cout << "отношение не является отношением доминирования";
        }

        return 0;

    }
}
```

Результаты тестирования:

```
5
0 0 0 0 0
1 0 0 0 0
1 1 0 0 0
1 1 0 0 0
1 1 1 0 0
отношение является отношением доминирования
```

8.3 Алгоритмы построения замыканий бинарных отношений:

– **Получение исходного отношения.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    vector<vector<int>> m(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    vector<pair<int, int>> a;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (m[i][j] == 1) {
```

```
                a.push_back({i + 1, j + 1});
            }
        }
    }

    for (int i = 0; i < size(a); ++i) {
        if (i > 0) {
            cout << ", ";
        }
        cout << "(" << a[i].first << ", " << a[i].second << ")";
    }

    return 0;
}
```

Результаты тестирования:

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

– **Построение рефлексивного замыкания.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    vector<vector<int>> m(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    vector<pair<int, int>> a;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (m[i][j] == 1) {
                a.push_back({i + 1, j + 1});
            }
        }
    }

    for (int i = 0; i < n; ++i) {
```

```
        if (m[i][i] == 0) {
            a.push_back({i + 1, i + 1});
        }
    }

    for (int i = 0; i < size(a); ++i) {
        if (i > 0) {
            cout << ", ";
        }
        cout << "(" << a[i].first << ", " << a[i].second << ")";
    }

    return 0;
}
```

Результаты тестирования:

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

– **Построение симметричного замыкания.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <set>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    vector<vector<int>> m(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    vector<pair<int, int>> a;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (m[i][j] == 1) {
                a.push_back({i + 1, j + 1});
            }
        }
    }
```

```
    int q = size(a);

    for (int i = 0; i < q; ++i) {
        a.push_back({a[i].second, a[i].first});
    }

    set<pair<int, int>> rrr(begin(a), end(a));

    bool isFirst = true;
    for (auto [i, j] : rrr) {
        if (!isFirst) {
            cout << ", ";
        }
        cout << "(" << i << ", " << j << ")";
        isFirst = false;
    }

    return 0;
}
```

Результаты тестирования:

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

– **Построение транзитивного замыкания.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>

using namespace std;

bool transitivity(vector<vector<int>>& m) {

    vector<vector<int>> w(size(m), vector<int> (size(m)));

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            w[i][j] = 0;
        }
    }

    for (int i = 0; i < size(m); ++i) {
        for (int l = 0; l < size(m); ++l) {
            int s = 0;
            for (int j = 0; j < size(m); ++j) {
                s += m[i][j] * m[j][l];
            }
            w[i][l] += s;
```

```
            if (w[i][l] > 1) {
                w[i][l] = 1;
            }
        }
    }

    bool q = true;

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            if (w[i][j] > m[i][j]) {
                q = false;
            }
        }
    }

    return q;

}

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    vector<vector<int>> m(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    vector<pair<int, int>> a;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (m[i][j] == 1) {
                a.push_back({i + 1, j + 1});
            }
        }
    }

    vector<vector<int>> w(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            w[i][j] = 0;
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int l = 0; l < n; ++l) {
            int s = 0;
            for (int j = 0; j < n; ++j) {
                s += m[i][j] * m[j][l];
            }
            w[i][l] += s;
            if (w[i][l] > 1) {
                w[i][l] = 1;
            }
        }
```

```
    }

    vector<vector<int>> copy_m(size(m), vector<int> (size(m)));

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            copy_m[i][j] = m[i][j];
        }

    }

    while (!transitivity(copy_m)) {
        for (int i = 0; i < size(m); ++i) {
            for (int j = 0; j < size(m); ++j) {
                if (copy_m[i][j] == 0 && w[i][j] == 1) {
                    a.push_back({i + 1, j + 1});
                    copy_m[i][j] = 1;
                }
            }
        }
        if (!transitivity(copy_m)) {
            for (int i = 0; i < size(m); ++i) {
                for (int l = 0; l < size(m); ++l) {
                    int s = 0;
                    for (int j = 0; j < size(m); ++j) {
                        s += copy_m[i][j] * m[j][l];
                    }
                    w[i][l] += s;
                    if (w[i][l] > 1) {
                        w[i][l] = 1;
                    }
                }
            }
        }
    }

    for (int i = 0; i < size(a); ++i) {
        if (i > 0) {
            cout << ", ";
        }
        cout << "(" << a[i].first << ", " << a[i].second << ")";
    }

    return 0;
}
```

Результаты тестирования:

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

– **Построение эквивалентного замыкания.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <set>

using namespace std;

bool transitivity(vector<vector<int>>& m) {

    vector<vector<int>> w(size(m), vector<int> (size(m)));

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            w[i][j] = 0;
        }
    }

    for (int i = 0; i < size(m); ++i) {
        for (int l = 0; l < size(m); ++l) {
            int s = 0;
            for (int j = 0; j < size(m); ++j) {
                s += m[i][j] * m[j][l];
            }
            w[i][l] += s;
            if (w[i][l] > 1) {
                w[i][l] = 1;
            }
        }
    }

    bool q = true;

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            if (w[i][j] > m[i][j]) {
                q = false;
            }
        }
    }

    return q;

}

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    vector<vector<int>> m(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    vector<pair<int, int>> a;
```

```
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (m[i][j] == 1) {
                a.push_back({i + 1, j + 1});
            }
        }
    }

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 0) {
            a.push_back({i + 1, i + 1});
        }
    }

    int q = size(a);

    for (int i = 0; i < q; ++i) {
        a.push_back({a[i].second, a[i].first});
    }

    vector<vector<int>> w(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            w[i][j] = 0;
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int l = 0; l < n; ++l) {
            int s = 0;
            for (int j = 0; j < n; ++j) {
                s += m[i][j] * m[j][l];
            }
            w[i][l] += s;
            if (w[i][l] > 1) {
                w[i][l] = 1;
            }
        }
    }

vector<vector<int>> copy_m(size(m), vector<int> (size(m)));

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            copy_m[i][j] = m[i][j];
        }

    }

    while (!transitivity(copy_m)) {
        for (int i = 0; i < size(m); ++i) {
            for (int j = 0; j < size(m); ++j) {
                if (copy_m[i][j] == 0 && w[i][j] == 1) {
                    a.push_back({i + 1, j + 1});
                    copy_m[i][j] = 1;
                }
            }
        }
        if (!transitivity(copy_m)) {
            for (int i = 0; i < size(m); ++i) {
                for (int l = 0; l < size(m); ++l) {
                    int s = 0;
```

```
                for (int j = 0; j < size(m); ++j) {
                    s += copy_m[i][j] * m[j][l];
                }
                w[i][l] += s;
                if (w[i][l] > 1) {
                    w[i][l] = 1;
                }
            }
        }
    }
}

set<pair<int, int>> rrr(begin(a), end(a));

bool isFirst = true;
for (auto [i, j] : rrr) {
    if (!isFirst) {
        cout << ", ";
    }
    cout << "(" << i << ", " << j << ")";
    isFirst = false;
}

return 0;
}
```

Результаты тестирования:

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

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены основные свойства бинарных отношений, определены классификации бинарных отношений и изучено построение замыканий отношений. Результатом работы являются: алгоритмы для определения свойств бинарных отношений, алгоритмы для определения видов бинарных отношений, алгоритмы построения замыкания бинарных отношений. Также была осуществлена программная реализация описанных алгоритмов.