МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

**Кафедра программного обеспечения вычислительной техники и автоматизированных систем**

**Лабораторная работа №3.2**

**Транзитивное замыкание отношения**

по дисциплине: Дискретная математика

Выполнил: ст. группы ВТ-231

Масленников Даниил Александрович

Проверил: Островский Алексей Мичеславович

Белгород 2024 г.

**1. Написать программу для реализации следующего алгоритма вычисления транзитивного замыкания отношения А, построенного на множестве М:**

Алгоритм 1.

Вход: А — отношение.

Выход: С — транзитивное замыкание отношения А.

1. С := А; S := А o А.

2. Пока S o C выполнять:

С := С ∪ S; S := C o S.

3. Конец алгоритма.

Реализация:

#include <stdio.h>

#include <stdbool.h>

#define SIZE 10

void composition(int A[SIZE][SIZE], int B[SIZE][SIZE], int result[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

result[i][j] = 0;

for (int k = 0; k < SIZE; k++) {

result[i][j] |= (A[i][k] && B[k][j]);

}

}

}

}

void unionRelations(int A[SIZE][SIZE], int B[SIZE][SIZE], int result[SIZE][SIZE])

{

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

result[i][j] = A[i][j] || B[i][j];

}

}

}

void copyMatrix(int source[SIZE][SIZE], int destination[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

destination[i][j] = source[i][j];

}

}

}

bool isSubset(int A[SIZE][SIZE], int B[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (A[i][j] && !B[i][j]) {

return false;

}

}

}

return true;

}

int transitiveClosure(int A[SIZE][SIZE], int C[SIZE][SIZE], int\* comparisons) {

int S[SIZE][SIZE];

int compositions = 0;

copyMatrix(A, C);

composition(A, A, S);

compositions++;

while (!isSubset(S, C)) {

unionRelations(C, S, C);

composition(S, A, S);

compositions++;

}

return compositions;

// компилятор ругается на название матрицы C2, использую S

// int S[SIZE][SIZE];

// int compositions = 0;

// copyMatrix(A, C);

// composition(C, C, S);

// while(!isSubset(S, C)){

// unionRelations(S, C, C);

// composition(C, C, S);

// compositions++;

// }

// return compositions;

}

void printMatrix(int matrix[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

**2. Привести пример отношения А на множестве М = {1,2,3,4,5,6,7,8,9,10}, при обработке которого алгоритмом 1 композиция выполнится не большее количество раз, чем при обработке любого другого отношения на множестве М. Сколько раз выполнится композиция при обработке отношения А? Проверить экспериментально. Определить, какое количество операций сравнения выполнится при обработке отношения А.**

Чтобы композиция выполнилась минимальное количество раз для данного алгоритма возьмем отношение A которое уже обладает свойством транзитивности.

Пример отношения: A={(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10)}

int main() {

int A[SIZE][SIZE] = {0};

for (int i = 0; i < SIZE; i++){

A[i][i] = 1;

}

int C[SIZE][SIZE];

printf("Исходная матрица A:\n");

printMatrix(A);

int comparisons = 0;

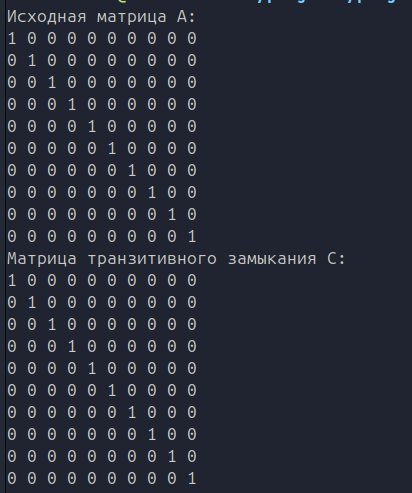
int comp = transitiveClosure(A, C, &comparisons);

printf("Матрица транзитивного замыкания C:\n");

printMatrix(C);

return 0;

}



**3. Привести пример отношения А на множестве М =**

**{1,2,3,4,5,6,7,8,9,10}, при обработке которого алгоритмом 1 композиция выполнится не меньшее количество раз, чем при обработке любого другого отношения на множестве М. Сколько раз выполнится композиция при обработке отношения А? Проверить**

**экспериментально. Определить, какое количество операций сравнения выполнится при обработке**

**отношения А.**

Чтобы композиция выполнилась максимальное количество раз для данного алгоритма возьмем отношение A которое состоит из максимальной цепочки дуг между двумя вершинами.

Пример отношения: {(1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10)}

int main() {

int A[SIZE][SIZE] = {0};

for (int i = 0; i < SIZE - 1; ++i) {

A[i][i + 1] = 1; // x -> y для смежных x и y

}

int C[SIZE][SIZE];

printf("Исходная матрица A:\n");

printMatrix(A);

int comparisons = 0;

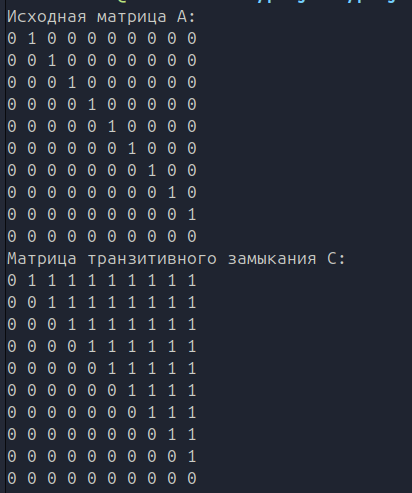
int comp = transitiveClosure(A, C, &comparisons);

printf("Матрица транзитивного замыкания C:\n");

printMatrix(C);

return 0;

}



**4. Написать программу для реализации следующего алгоритма вычисления транзитивного замыкания отношения А, построенного на множестве М:**

Алгоритм 2.

Вход: А — отношение.

Выход: С — транзитивное замыкание отношения А.

1. С := А; С2 := С o С.

2. Пока C2 ∈̅C выполнять:

С := С ∪ С2; С2 := С o С.

3. Конец алгоритма.

**Реализация:**

int transitiveClosure(int A[SIZE][SIZE], int C[SIZE][SIZE], int\* comparisons) {

// компилятор ругается на название матрицы C2, использую S

int S[SIZE][SIZE];

int compositions = 0;

copyMatrix(A, C);

composition(C, C, S);

while(!isSubset(S, C)){

unionRelations(S, C, C);

composition(C, C, S);

compositions++;

}

return compositions;

}

**5. Привести пример отношения А на множестве**

**М = {1,2,3,4,5,6,7,8,9,10}, при обработке которого алгоритмом 2 композиция выполнится не большее количество раз, чем при обработке любого другого отношения на множестве М. Сколько раз выполнится композиция при обработке отношения А? Проверить экспериментально. Определить, какое количество операций сравнения выполнится при обработке отношения А.**

Чтобы композиция выполнилась минимальное количество раз для данного алгоритма возьмем отношение A которое уже обладает свойством транзитивности.

Пример отношения: A={(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10)}

int main() {

int A[SIZE][SIZE] = {0};

for (int i = 0; i < SIZE; i++){

A[i][i] = 1;

}

int C[SIZE][SIZE];

printf("Исходная матрица A:\n");

printMatrix(A);

int comparisons = 0;

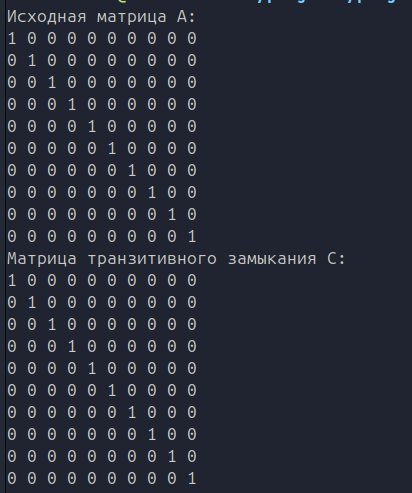
int comp = transitiveClosure(A, C, &comparisons);

printf("Матрица транзитивного замыкания C:\n");

printMatrix(C);

return 0;

}



**6. Привести пример отношения А на множестве**

**М = {1,2,3,4,5,6,7,8,9,10}, при обработке которого алгоритмом 2 композиция выполнится не меньшее количество раз, чем при обработке любого другого отношения на множестве М.**

**Сколько раз выполнится композиция при обработке отношения А? Проверить экспериментально.**

**Определить, какое количество операций сравнения выполнится при обработке отношения А.**

Чтобы композиция выполнилась максимальное количество раз для данного алгоритма возьмем отношение A которое состоит из максимальной цепочки дуг между двумя вершинами.

Пример отношения: {(1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10)}

int main() {

int A[SIZE][SIZE] = {0};

for (int i = 0; i < SIZE-1; i++){

A[i][i+1] = 1;

}

int C[SIZE][SIZE];

printf("Исходная матрица A:\n");

printMatrix(A);

int comparisons = 0;

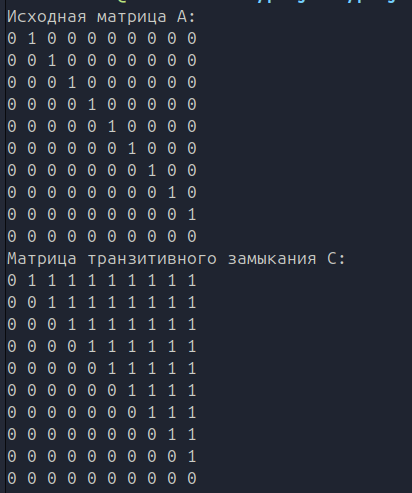
int comp = transitiveClosure(A, C, &comparisons);

printf("Матрица транзитивного замыкания C:\n");

printMatrix(C);

return 0;

}



**7. Написать программу для реализации следующего алгоритма вычисления транзитивного замыкания отношения А, построенного на множестве М:**

Алгоритм 3.

Вход: А — отношение.

Выход: С — транзитивное замыкание отношения А.

1. С := А

2. Для всех z ∈ М

выполнить: Для всех х ∈

М выполнить:

Если (x,z) ∈ C то

Для всех у ∈ М выполнить:

Если (z,y) ∈ C то С := С ∪ {(х,y)}.

3. Конец алгоритма.

Реализация:

void relationUnionWithPair(int A[SIZE][SIZE], int x, int y) {

A[x][y] = 1;

}

void transitiveClosure(int A[SIZE][SIZE], int C[SIZE][SIZE], int\* comparisons) {

copyMatrix(A, C);

for (int z = 0; z < SIZE; z++){

for (int x = 0; x < SIZE; x++){

if (C[x][z] == 1){

for (int y = 0; y < SIZE; y++){

if(C[z][y] == 1){

relationUnionWithPair(C, x, y);

}

}

}

}

}

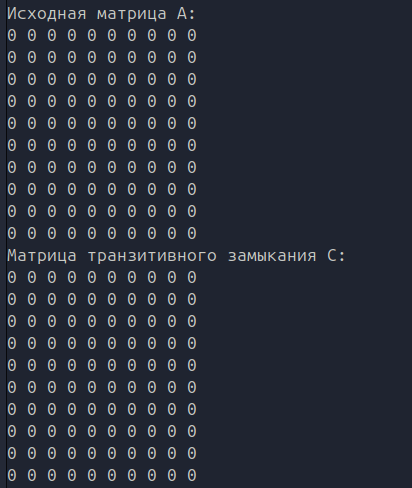
}

**8. Привести пример отношения А на множестве М =**

**{1,2,3,4,5,6,7,8,9,10}, при обработке которого алгоритмом 3 количество операций сравнения выполнится не большее количество раз, чем при обработке любого другого отношения на множестве М. Определить, какое количество операций сравнения выполнится при обработке отношения А. Проверить экспериментально.**

Чтобы операции сравнения выполнились минимальное количество раз для данного алгоритма возьмем пустое отношение A. Тогда количество операций сравнения будет равно N^2, где N мощность множества М, так как не будет пройдена проверка и к перебору y программа не приступит.

Пример отношения: {}



**9. Привести пример отношения А на множестве М =**

**{1,2,3,4,5,6,7,8,9,10}, при обработке которого алгоритмом 3 количество операций сравнения выполнится не меньшее количество раз, чем при обработке любого другого отношения на множестве М. Определить, какое количество операций сравнения выполнится при обработке отношения А. Проверить экспериментально.**

Для того чтобы операции сравнения выполнились максимальное количество раз для данного алгоритма возьмем отношение A включающее в себя все возможные пары.

Пример отношения: {(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7), (1,8), (1,9), (1,10), (2,1), (2,2), (2,3), (2,4), (2,5), (2,6), (2,7), (2,8), (2,9), (2,10), (3,1), (3,2), (3,3), (3,4),

(3,5), (3,6), (3,7), (3,8), (3,9), (3,10), (4,1), (4,2), (4,3), (4,4), (4,5), (4,6), (4,7), (4,8),

(4,9), (4,10), (5,1), (5,2), (5,3), (5,4), (5,5), (5,6), (5,7), (5,8), (5,9), (5,1 0), (6,1), (6,2), (6,3), (6,4), (6,5), (6,6), (6,7), (6,8), (6,9), (6,10), (7,1), (7,2), (7,3), (7,4), (7,5),

(7,6), (7,7), (7,8), (7,9), (7,10), (8,1), (8,2), (8,3), (8,4), (8,5), (8,6), (8,7), (8,8), (8,9),

(8,10), (9,1), (9,2), (9,3), (9,4), (9,5), (9,6), (9,7), (9,8), (9,9), (9,10), (10,1), (10,2), (10,3), (10,4), (10,5), (10,6), (10,7), (10,8), (10,9), (10,10)}

int main() {

int A[SIZE][SIZE] = {0};

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

A[i][j] = 1;

}

}

int C[SIZE][SIZE];

printf("Исходная матрица A:\n");

printMatrix(A);

int comparisons = 0;

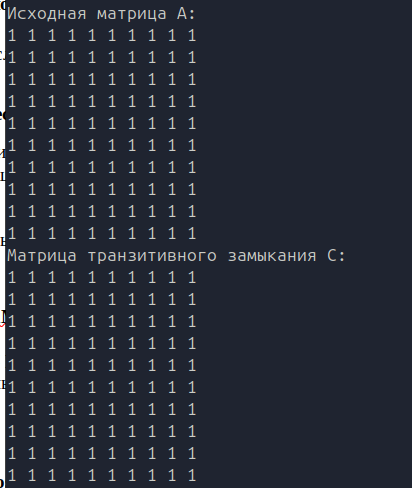
int comp = transitiveClosure(A, C, &comparisons);

printf("Матрица транзитивного замыкания C:\n");

printMatrix(C);

return 0;

}



**10. Определить порядок функции временной сложности алгоритмов**

**вычисления транзитивного замыкания.**

Алгоритм 1: O(𝑛 + (𝑛 − 2) ∗ (𝑛2 + 𝑛3))

Алгоритм 2: O(𝑛 + (𝑛 − 2) ∗ (𝑛2 + 𝑛3))  
Алгоритм 3: O(𝑛3)