**МИНОБРНАУКИ РОССИИ  
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 №2

Дисциплина: Дискретная математика

по теме «**Транзитивное замыкание отношения**»

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Рязанов Ю.Д

**Белгород 2018**

**Практическое занятие 3.2**

**Транзитивное замыкание отношения  
Цель занятия**: изучить и выполнить сравнительный анализ алгоритмов вычисления транзитивного замыкания отношения.

***Задания***

1. Изучить и программно реализовать алгоритмы 3.10 - 3.12, а также усовершенствованные АОС и АУ для вычисления транзитивного замыкания отношения.
2. Разработать и программно реализовать генератор отношений на множестве мощности N и содержащих заданное число пар.
3. Разработать и написать программу, которая генерирует 100 отношений на множестве мощности N с заданным числом пар, для каждого отношения вычисляет транзитивное замыкание пятью алгоритмами и подсчитываем кол-во выполнений операций сравнения. Выполнить программу при N = 5, 10 и 15. Результат для каждого N представить в виде таблицы.

**Задание 1**: Изучить и программно реализовать алгоритмы объединения степеней и Уоршалла для вычисления транзитивного замыкания отношения.

**Задание 2:** Разработать и программно реализовать генератор отношений на множестве мощности N и содержащих заданное число пар.

**Задание 3**: Разработать и написать программу, которая генерирует 1000 отношений на множестве мощности N с заданным числом пар.

Количество k выполнений операций сравнения для n=5

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Число пар в отношении | 1 | | (N^2)/4 | | (N^2)/2 | | (N^2)\*2/3 | | N^2 | |
| min | max | min | max | min | max | min | max | min | max |
| Алгоритм 3.10 | 1 | 7 | 1 | 10 | 4 | 7 | 4 | 7 | 1 | 1 |
| Алгоритм 3.11 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| Усовершенствованный АОС | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Алгоритм 3.12 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 | 250 |
| Усовершенствованный АУ | 50 | 120 | 60 | 120 | 130 | 240 | 170 | 250 | 250 | 250 |

Количество k выполнений операций сравнения для n=10

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Число пар в отношении | 1 | | (N^2)/4 | | (N^2)/2 | | (N^2)\*2/3 | | N^2 | |
| min | max | min | max | min | max | min | max | min | max |
| Алгоритм 3.10 | 1 | 10 | 7 | 10 | 4 | 7 | 4 | 7 | 1 | 1 |
| Алгоритм 3.11 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| Усовершенствованный АОС | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| Алгоритм 3.12 | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 | 2000 |
| Усовершенствованный АУ | 200 | 460 | 620 | 1460 | 1420 | 1960 | 1800 | 2000 | 2000 | 2000 |

Количество k выполнений операций сравнения для n=15

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Число пар в отношении | 1 | | (N^2)/4 | | (N^2)/2 | | (N^2)\*2/3 | | N^2 | |
| min | max | min | max | min | max | min | max | min | max |
| Алгоритм 3.10 | 4 | 13 | 7 | 10 | 4 | 7 | 4 | 7 | 1 | 1 |
| Алгоритм 3.11 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 26 |
| Усовершенствованный АОС | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 |
| Алгоритм 3.12 | 6750 | 6750 | 6750 | 6750 | 6750 | 6750 | 6750 | 6750 | 6750 | 6750 |
| Усовершенствованный АУ | 450 | 1380 | 3300 | 5730 | 5340 | 6600 | 6150 | 6750 | 6750 | 6750 |

#include <stdio.h>

#include <stdlib.h>

#define N 15

#define K 150

typedef int(\*t\_func)(int\*\*);

void print\_matr(int a[N][N]){

printf("\n");

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

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

printf("%i ", a[i][j]);

}

printf("\n");

}

}

void read\_matr(int a[N][N]){

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

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

scanf("%i", &a[i][j]);

}

}

}

void copy\_matr(int a[N][N], int b[N][N]){

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

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

b[i][j]=a[i][j];

}

}

}

void get\_I(int I[N][N]){

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

I[i][i]=1;

}

}

int equal(int a[N][N], int b[N][N]){

int f = 1;

int x = 1;

int y;

while (x<=N && f){

y=1;

while(y<=N && f){

f=(a[x-1][y-1] == b[x-1][y-1]);

y++;

}

x++;

}

return f;

}

int inclusion(int a[N][N], int b[N][N]){

int f = 1;

int x = 1;

int y;

while (x<=N && f){

y=1;

while(y<=N && f){

f=(a[x-1][y-1] <= b[x-1][y-1]);

y++;

}

x++;

}

return f;

}

void unite(int a[N][N], int b[N][N], int (\*c)[N][N]){

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

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

(\*c)[i][j]=a[i][j] || b[i][j];

}

}

}

void composition(int a[N][N], int b[N][N], int (\*c)[N][N]){

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

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

(\*c)[i][j]=0;

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

(\*c)[i][j] = (\*c)[i][j] || (a[i][z] && b[z][j]);

}

}

}

}

int transit\_closure(int a[N][N]){

int count=1;

int c\_tran[N][N];

int c2[N][N];

int tmp[N][N];

copy\_matr(a, c\_tran);

composition(c\_tran, c\_tran, &c2);

while(!(inclusion(c2, c\_tran))){

unite(c\_tran, c2, &tmp);

copy\_matr(tmp, c\_tran);

composition(c\_tran, c\_tran, &c2);

count+=3;

}

//printf("1:");

//print\_matr(c2);

return count;

}

int AOC(int a[N][N]){

int count=0;

int c\_tran[N][N];

int a\_i[N][N];

int tmp[N][N];

copy\_matr(a, c\_tran);

copy\_matr(a, a\_i);

for(int i=2; i<N; i++){

composition(a\_i, a\_i, &tmp);

copy\_matr(tmp, a\_i);

unite(c\_tran, a\_i, &tmp);

copy\_matr(tmp, c\_tran);

count+=2;

}

//printf("AOC:");

//print\_matr(c\_tran);

return count;

}

int AOC\_improved(int a[N][N]){

int count=1;

int I[N][N];

int tmp[N][N];

int tmp1[N][N];

get\_I(I);

unite(I, a, &tmp);

for (int i=2; i<N-2; i++){

composition (tmp, tmp, &tmp1);

copy\_matr(tmp1, tmp);

count++;

}

composition(a, tmp, &tmp1);

//printf("AOC imp:");

//print\_matr(tmp1);

return ++count;

}

int AY(int a[N][N]){

int count=0;

int c[N][N];

int tmp[N][N];

copy\_matr(a, c);

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

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

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

c[x][y]=c[x][y] || c[x][z] && c[z][y];

count+=2;

}

}

}

//printf("AY:");

//print\_matr(c);

return count;

}

int AY\_improved (int a[N][N]){

int count=0;

int c[N][N];

int tmp[N][N];

copy\_matr(a, c);

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

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

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

if(c[x][z]){

c[x][y]=c[x][y] || c[z][y];

count+=2;

}

}

}

}

//printf("AY imp:");

//print\_matr(c);

return count;

}

void randomize\_matr (int a[N][N]){

int i, j;

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

i=rand()%N;

j=rand()%N;

if (a[i][j]){

k--;

} else {

a[i][j]=1;

}

}

}

void clean\_matr(int a[N][N]){

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

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

a[i][j]=0;

}

}

}

void get\_rev(int a[N][N], t\_func func){

int max\_k=0, min\_k=10000000, tmp;

int min\_rev[N][N];

int max\_rev[N][N];

int tmp\_rev[N][N];

for (int k=0; k<100; k++){

randomize\_matr(a);

copy\_matr(a, tmp\_rev);

tmp=func(tmp\_rev);

if (tmp<min\_k){

min\_k=tmp;

copy\_matr(a, min\_rev);

} else {

if (tmp>max\_k){

max\_k=tmp;

copy\_matr(a, max\_rev);

}

}

clean\_matr(a);

}

printf ("%i - max counter\n", max\_k);

//print\_matr(max\_rev);

printf ("%i - min counter\n", min\_k);

//print\_matr(min\_rev);

}

int main(){

int a[N][N]={};

printf("transitive closure\n");

get\_rev(a,transit\_closure);

printf("AY\n");

get\_rev(a,AY);

printf("AY improve\n");

get\_rev(a,AY\_improved);

printf("AOC\n");

get\_rev(a,AOC);

printf("AOC improve\n");

get\_rev(a,AOC\_improved);

}