Міністерство освіти і науки, молоді та спорту України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 3**

**«Застосування дерев. Властивості відношень»**

**(до тем «Дерева та їх застосування», «Відношення»**

**Лектор:**

проф. кафедри ПЗ

Журавчак Л.М.

**Виконав:**

студ. групи ПЗ-14

Губик А. С.

**Прийняв:**

асистент кафедри ПЗ

Курапов П. P.

«\_\_\_» \_\_\_\_\_\_ 2022 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2022

**ТЕМА РОБОТИ**: Застосування дерев. Властивості відношень.

**МЕТА РОБОТИ**: Ознайомитись на практиці з алгоритмом Краскала побудови мінімального остовного дерева, з алгоритмом побудови дерева за заданим кодом Прюфера, з поданням арифметичних виразів у вигляді математичного (бінарного) дерева та польськими записами, навчитися обчислювати значення виразів в обох цих записах, освоїти алгоритм бектрекінгу для розфарбування графа мінімально можливою кількістю кольорів. Реалізувати обчислення матриць операцій над двома бінарними відношеннями, а також визначення властивостей бінарного відношення.

#### Додаток

**graph.c:**

#include <stdio.h>

#include "graph.h"

void AddMatrix(int res[], int a[], int b[], int size)

{

for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

res[i \* size + j] = a[i \* size + j] || b[i \* size + j];

}

void MultMatrix(int res[], int a[], int b[], int size)

{

for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

for(int k = 0; k < size; k++)

res[i \* size + j] = res[i \* size + j] ||

a[i \* size + k] &&

b[k \* size + j];

}

void PrintMatrix(int g[], int size)

{

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

printf("\n");

for(int j = 0; j < size; j++)

printf("%d ", g[i \* size + j]);

}

printf("\n");

}

void CopyMatrix(int res[], int g[], int size)

{

for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

res[i \* size + j] = g[i \* size + j];

}

void ConstructComplementary(int GC[], int G[], int size)

{

for(int i = 0; i < size; i++)

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

GC[i \* size + j] = !G[i \* size + j];

}

}

void TranspondMatrix(int res[], int G[], int size)

{

for(int i = 0; i < size; i++)

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

res[i \* size + j] = G[j \* size + i];

}

}

void ConstructUnion(int U[],int G[],int H[], int size)

{

for(int i = 0; i < size; i++)

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

U[i \* size + j] = G[i \* size + j] || H[i \* size + j];

}

}

void ConstructCrossing(int U[], int G[], int H[], int size)

{

for(int i = 0; i < size; i++)

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

U[i \* size + j] = G[i \* size + j] && H[i \* size + j];

}

}

void ConstructSymetricalCrossing(int U[], int G[], int H[], int size)

{

for(int i = 0; i < size; i++)

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

U[i \* size + j] = G[i \* size + j] != H[i \* size + j];

}

}

int isUndirected(int G[], int size)

{

for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

if(G[i \* size + j] != G[j \* size + i])

return 0;

return 1;

}

int CountEdges(int G[], int size)

{

int res = 0;

for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

if(G[i \* size + j])

res++;

if(isUndirected(G, size))

return res / 2;

return res;

}

**graph.h:**

void AddMatrix(int res[], int a[], int b[], int size);

void MultMatrix(int res[], int a[], int b[], int size);

void PrintMatrix(int g[], int size);

void CopyMatrix(int res[], int g[], int n);

void ConstructComplementary(int GC[], int G[], int size);

void TranspondMatrix(int res[], int G[], int size);

void ConstructUnion(int U[], int G[], int H[], int size);

void ConstructCrossing(int X[], int G[], int H[], int size);

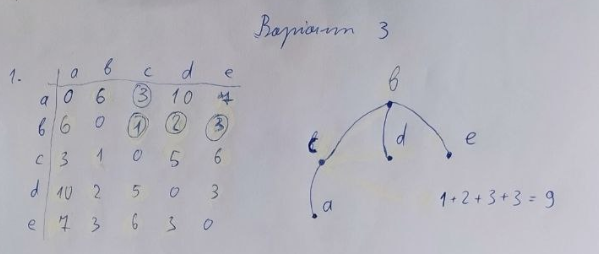
void ConstructSymetricalCrossing(int X[], int G[], int H[], int size);

int CountEdges(int G[], int size);

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 1

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує знаходження мінімального остовного (кістякового) дерева (МОД) неорієнтованого зваженого зв'язного графа, заданого ваговою матрицею (таблиця 3.1) за алгоритмом Краскала. Етапи розв'язання задачі виводити на екран.

ОПИС ВИКОНАННЯ РОБОТИ



ТЕКСТ ПРОГРАМИ

#include <stdio.h>

enum{

size = 5

};

int Kruskal(int G[][size])

{

int v[size] = {0};

int minId = 0, minRow = 0, res = 0;

for(int k = 0; k < size - 1; k++){

int min = (unsigned short)(-1) / 2;

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

for(int j = i + 1; j < size; j++){

if (v[j] && v[j - 1])

continue;

if (G[i][j] < min){

min = G[i][j];

minId = j;

minRow = i;

}

}

}

G[minRow][minId] = (unsigned short)(-1) / 2;

res += min;

v[minId] = 1;

printf("%c%c ", 'a' + minRow, 'a' + minId);

}

return res;

}

int main()

{

int G[size][size] = {

{0 , 6 , 3 , 10, 7 },

{6 , 0 , 1 , 2 , 3 },

{3 , 1 , 0 , 5 , 6 },

{10, 2 , 5 , 0 , 3 },

{7 , 3 , 6 , 3 , 0 }

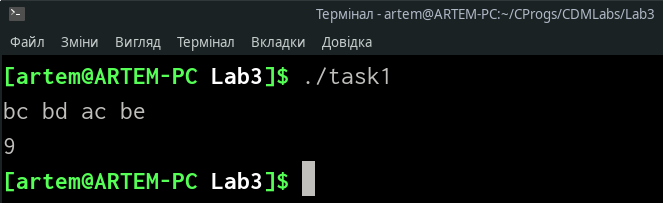
};

int res = Kruskal(G);

printf("\n%d\n", res);

}

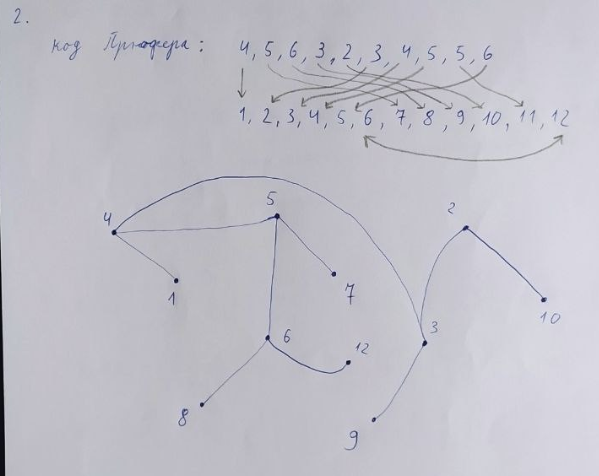
РЕЗУЛЬТАТИ



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 2

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує побудову дерева за заданим кодом Прюфера (таблиця 3.2). Етапи розв'язання задачі виводити на екран.

ОПИС ВИКОНАННЯ РОБОТИ



**ТЕКСТ ПРОГРАМИ**

#include <stdio.h>

enum{

size = 10

};

int inCode(int code[], int x)

{

for(int i = 0; i < size; i++)

if (code[i] == x)

return 1;

return 0;

}

void BuildTree(int code[])

{

for(int i = 0; i < size; i++)

for(int j = 1; j < size + 3; j++){

if(inCode(code, j))

continue;

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

code[i] = j;

break;

}

int FirstLeft, SecondLeft;

for(int i = 1; i < size + 3; i++)

if(!inCode(code, i)){

FirstLeft = i;

break;

}

for(int i = 1; i < size + 3; i++){

if(i == FirstLeft)

continue;

if(!inCode(code, i)){

SecondLeft = i;

break;

}

}

printf("%d-%d", FirstLeft, SecondLeft);

}

int main()

{

int code[size] = {4, 5, 6, 3, 2, 3, 4, 5, 5, 6};

BuildTree(code);

printf("\n");

#if 1

for(int i = 0; i < size; i++)

printf("%d ", code[i]);

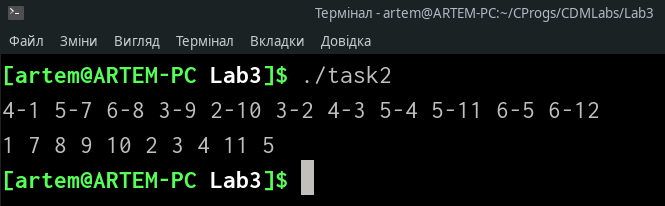
printf("\n");

#endif

return 0;

}

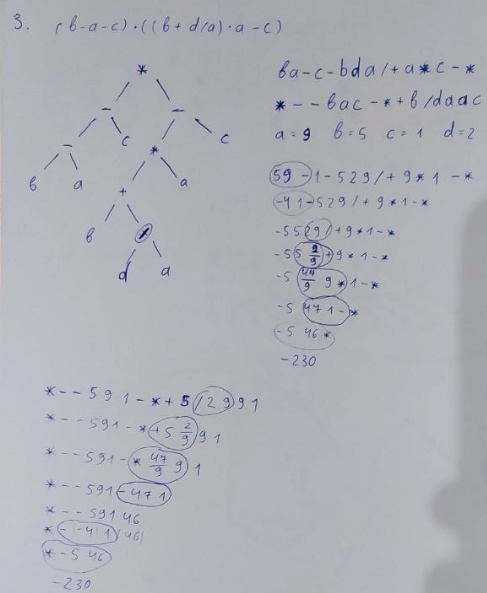
**РЕЗУЛЬТАТИ**



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 3

Подати у вигляді математичного (бінарного) дерева арифметичний вираз (таблиця 3.3). Отримати дві послідовності його обходу: префіксний (польський) та постфіксний (зворотний польський) записи. Відобразити динаміку обчислень та знайти значення виразів в обох цих записах.

ОПИС ВИКОНАННЯ РОБОТИ



ТЕКСТ ПРОГРАМИ

**calculator.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "structures.h"

#if 0

struct operator{

char\* symbol;

int precendence;

int associativity; // 0 - left, 1 - right;

}

#endif

enum{

OperatorNumber = 6

};

#if 0

void swap(char \*a, char \*b)

{

char tmp = \*a;

\*a = \*b;

\*b = tmp;

}

#endif

void InvertString(char \*res, const char \*s)

{

for(int i = 0, j = strlen(s) - 1; i < strlen(s); j--, i++){

if (s[j] == ')'){

res[i] = '(';

continue;

}

if (s[j] == '('){

res[i] = ')';

continue;

}

res[i] = s[j];

}

res[strlen(s)] = '\0';

}

const char\* OperatorList[] = {"-", "+", "\*", "/", "(", ")"};

int isOperator(const char \*token)

{

for(int i = 0; i < OperatorNumber; ++i)

if(!strcmp(token, OperatorList[i]))

return 1;

return 0;

}

int isVariable(const char \*token)

{

if(!strcmp(token, "a") ||

!strcmp(token, "b") ||

!strcmp(token, "c") ||

!strcmp(token, "d") )

return 1;

return 0;

}

int getPrecendence(const char \*operator)

{

switch(operator[0]){

case '(': return 0;

case '-': return 2;

case '+': return 2;

case '\*': return 3;

case '/': return 3;

}

}

double CalcTwoNum(double a, double b, char\* operator)

{

switch(operator[0]){

case '-': return a - b;

case '+': return a + b;

case '\*': return a \* b;

case '/': return a / b;

}

}

void ConvertToRPN(queue \*rpn, const char \*equasion)

{

char str[300] = "";

strcpy(str, equasion);

stack \*OperatorStack;

StackInit(&OperatorStack);

char\* token = strtok(str, " ");

do{

if(!isOperator(token)){

QueuePut(rpn, token);

continue;

}

if(StackIsEmpty(OperatorStack) || !strcmp(token, "(")){

StackPush(&OperatorStack, token);

continue;

}

if(!strcmp(token, ")")){

char tmp[50];

while(strcmp(StackTop(OperatorStack), "(")){

StackPop(&OperatorStack, tmp);

QueuePut(rpn, tmp);

}

StackPop(&OperatorStack, tmp);

continue;

}

while(!StackIsEmpty(OperatorStack) &&

(getPrecendence(token) <= getPrecendence(StackTop(OperatorStack)))){

char tmp[50];

StackPop(&OperatorStack, tmp);

QueuePut(rpn, tmp);

}

StackPush(&OperatorStack, token);

}while(token = strtok(NULL, " "));

while(!StackIsEmpty(OperatorStack)){

char tmp[50];

StackPop(&OperatorStack, tmp);

QueuePut(rpn, tmp);

}

}

float CalculateRPN(queue \*rpn, double variable[])

{

StackOfDouble \*numbers;

SODInit(&numbers);

while(!QueueIsEmpty(\*rpn)){

char token[50];

QueueGet(rpn, token);

if(!isOperator(token)){

if(isVariable(token))

SODPush(&numbers, variable[token[0] - 'a']);

else

SODPush(&numbers, atof(token));

continue;

}

double res = CalcTwoNum(SODPop(&numbers), SODPop(&numbers), token);

SODPush(&numbers, res);

}

return SODPop(&numbers);

}

int main()

{

queue rpn, pn;

char equasion[300], inv[300];

double var[4];

QueueInit(&rpn);

printf("Enter an equasion:\n");

fgets(equasion, 300, stdin);

equasion[strlen(equasion) - 1] = '\0';

InvertString(inv, equasion);

printf("Enter the a, b, c and d: ");

for(int i = 0; i < 4; i++)

scanf("%lf", &var[i]);

ConvertToRPN(&rpn, equasion);

ConvertToRPN(&pn, inv);

PrintQueue(rpn);

PrintReverseQueue(pn.first);

printf("\n");

double res = CalculateRPN(&rpn, var);

printf("%lf\n", res);

return 0;

}

**structures.c:**

typedef struct item stack;

typedef struct ItemOfDouble StackOfDouble;

struct node{

struct item \*first;

struct item \*last;

};

typedef struct node queue;

void StackInit(stack \*\*s);

void StackPush(stack \*\*s, const char\* str);

char\* StackTop(stack \*s);

void StackPop(stack \*\*s, char res[]);

int StackIsEmpty(stack \*s);

void PrintStack(stack \*s);

void SODInit(StackOfDouble \*\*s);

void SODPush(StackOfDouble \*\*s, double n);

double SODPop(StackOfDouble \*\*s);

double SODTop(StackOfDouble \*s);

int SODIsEmpty(StackOfDouble \*s);

void QueueInit(queue \*q);

void QueuePut(queue \*q, char \*n);

void QueueGet(queue \*q, char res[]);

int QueueIsEmpty(queue q);

void PrintQueue(queue q);

void PrintReverseQueue(struct item \*qFirst);

**structures.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct item{

char data[50];

struct item \*next;

};

struct ItemOfDouble{

double data;

struct ItemOfDouble \*next;

};

struct node{

struct item \*first;

struct item \*last;

};

typedef struct item stack;

typedef struct ItemOfDouble StackOfDouble;

typedef struct node queue;

/\*======= Stack =======\*/

void StackInit(stack \*\*s)

{

\*s = NULL;

}

void StackPush(stack \*\*s, const char\* str)

{

stack \*tmp = malloc(sizeof(stack));

strcpy(tmp->data, str);

tmp->next = \*s;

\*s = tmp;

}

char\* StackTop(stack \*s)

{

return s->data;

}

void StackPop(stack \*\*s, char res[])

{

strcpy(res, (\*s)->data);

stack \*tmp = \*s;

\*s = (\*s)->next;

free(tmp);

}

int StackIsEmpty(stack \*s)

{

return !s;

}

void PrintStack(stack \*s)

{

stack \*tmp = s;

while(tmp){

printf("%s ", tmp->data);

tmp = tmp->next;

}

printf("\n");

}

/\*======= StackOfDouble =======\*/

void SODInit(StackOfDouble \*\*s)

{

\*s = NULL;

}

void SODPush(StackOfDouble \*\*s, double n)

{

StackOfDouble \*tmp = malloc(sizeof(StackOfDouble));

tmp->data = n;

tmp->next = \*s;

\*s = tmp;

}

double SODTop(StackOfDouble \*s)

{

return s->data;

}

double SODPop(StackOfDouble \*\*s)

{

double res = (\*s)->data;

StackOfDouble \*tmp = \*s;

\*s = (\*s)->next;

free(tmp);

return res;

}

int SODIsEmpty(StackOfDouble \*s)

{

return !s;

}

/\*======= Queue =======\*/

#if 0

void QueueInit(queue \*\*q)

{

(\*q)->first = NULL;

(\*q)->last = NULL;

}

#else

void QueueInit(queue \*q)

{

q->first = NULL;

q->last = NULL;

}

#endif

void QueuePut(queue \*q, const char \*str)

{

if(!q->first){

q->first = malloc(sizeof(struct item));

q->last = q->first;

}else{

q->last->next = malloc(sizeof(struct item));

q->last = q->last->next;

}

strcpy(q->last->data, str);

q->last->next = NULL;

}

void QueueGet(queue \*q, char res[])

{

strcpy(res, q->first->data);

struct item \*tmp = q->first;

q->first = q->first->next;

if (!q->first)

q->last = NULL;

free(tmp);

}

int QueueIsEmpty(queue q)

{

return !q.first;

}

void PrintQueue(queue q)

{

struct item \*tmp = q.first;

while(tmp){

printf("%s ", tmp->data);

tmp = tmp->next;

}

printf("\n");

}

void PrintReverseQueue(struct item \*qFirst)

{

if (!qFirst)

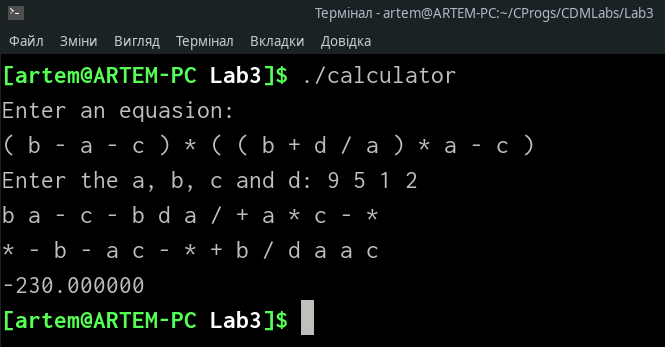
return;

PrintReverseQueue(qFirst->next);

printf("%s ", qFirst->data);

}

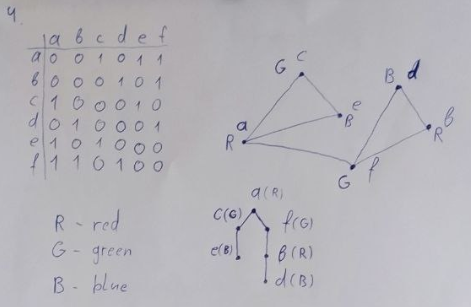
**РЕЗУЛЬТАТИ**



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 4

Розфарбувати граф, заданий матрицею суміжності (таблиця 3.4), мінімально можливою кількістю кольорів. Використати алгоритм бектрекінгу, результат подати у вигляді дерева.

ВИКОНАННЯ РОБОТИ



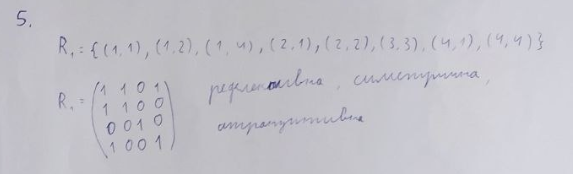
ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 5

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує

визначення властивостей бінарного відношення R1, заданого переліком елементів на множині

А={1,2,3,4}, R1 є A2 (таблиця 3.5). Вказати, чи є дане відношення відношенням еквівалентності, толерантності, порядку.

**ОПИС ВИКОНАННЯ РОБОТИ**



ТЕКСТ ПРОГРАМИ

#include <stdio.h>

#include "graph.h"

enum{

size = 4

};

void MakeMatrix(int res[size][size], int r[][2])

{

for(int i = 0; i < 8; i++)

res[r[i][0] - 1][r[i][1] - 1] = 1;

}

int isTransitive(const int r[size][size])

{

int square[size][size] = {0};

MultMatrix(square, r, r, size);

int isTransitive = 1, isAntitransitive = 1;

for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

if(!(r[i][j] >= square[i][j])){

isTransitive = 0;

break;

}

for(int i = 0; i < size; i++)

for(int j = 0; j < size; j++)

if(r[i][j] >= square[i][j]){

isAntitransitive = 0;

break;

}

if(isTransitive)

return 0;

else if(isAntitransitive)

return 1;

else

return -1;

}

int isSymetric(const int r[size][size])

{

int isSymetric = 1, isAntisymetric = 1;

for(int i = 0; i < size; i++)

for(int j = i + 1; j < size; j++)

if(r[i][j] == r[j][i]){

isAntisymetric = 0;

break;

}

for(int i = 0; i < size; i++)

for(int j = i + 1; j < size; j++)

if(r[i][j] != r[j][i]){

isSymetric = 0;

break;

}

if(isSymetric)

return 1;

else if(isAntisymetric)

return 0;

else

return -1;

}

int isReflective(const int r[size][size])

{

for(int i = 1; i < size; i++)

if(r[0][0] != r[i][i])

return -1;

return r[0][0];

}

int main()

{

int R[8][2] = {

{1, 1},

{1, 2},

{1, 4},

{2, 1},

{2, 2},

{3, 3},

{4, 1},

{4, 4}

};

int m[size][size] = {0};

int q[size][size] = {0};

MakeMatrix(m, R);

PrintMatrix(m, size);

int ref = isReflective(m);

int sym = isSymetric(m);

int trans = isTransitive(m);

if (ref == -1)

printf("Realation is neither reflective nor anti\n");

else if (ref == 0)

printf("Realation is antireflective\n");

else

printf("Realation is reflective\n");

if (sym == -1)

printf("Realation is neither symtric nor anti \n");

else if (sym == 0)

printf("Realation is antisymetric\n");

else

printf("Realation is symetric\n");

if (trans == -1)

printf("Realation is neither transitive nor anti \n");

else if (trans == 0)

printf("Realation is antitransitive\n");

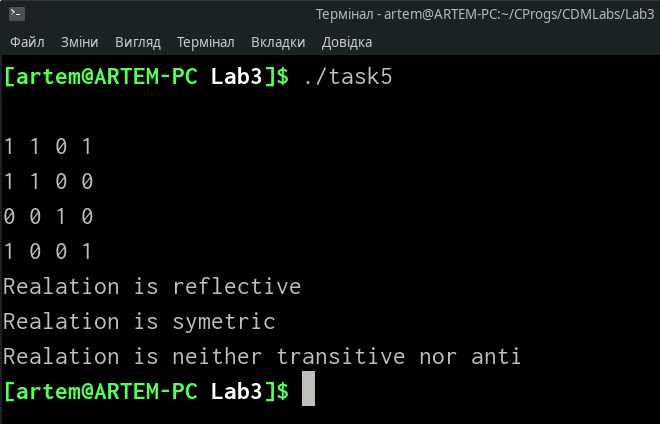
else

printf("Realation is transitive\n");

return 0;

}

**РЕЗУЛЬТАТИ**



ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 6

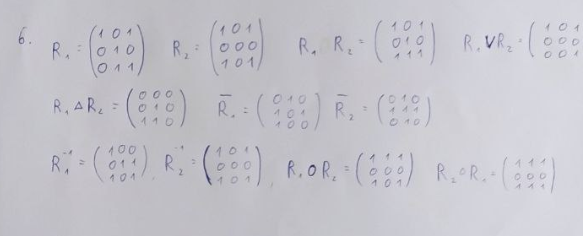
Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує обчислення

матриць перетину, об’єднання, різниці, симетричної різниці, доповнення, обернених, композицій

R1 o R2, R1 o R2 двох бінарних відношень, заданих на множині А = {a, b, c} (таблиця 3.6). У випадку

подання відношень переліком елементів спочатку записати їх матрицями.

ОПИС ВИКОНАННЯ РОБОТИ



ТЕКСТ ПРОГРАМИ

#include <stdio.h>

#include "graph.h"

enum{

size = 3

};

int main()

{

int R1[size][size] = {

{1, 0, 1},

{0, 1, 0},

{0, 1, 1}

};

int R2[size][size] = {

{1, 0, 1},

{0, 0, 0},

{1, 0, 1}

};

int U[size][size], X[size][size], SX[size][size],

R1C[size][size], R2C[size][size],

R1T[size][size], R2T[size][size],

Comp1[size][size], Comp2[size][size];

ConstructUnion(U, R1, R2, size);

ConstructCrossing(X, R1, R2, size);

ConstructSymetricalCrossing(SX, R1, R2, size);

ConstructComplementary(R1C, R1, size);

ConstructComplementary(R2C, R2, size);

TranspondMatrix(R1T, R1, size);

TranspondMatrix(R2T, R2, size);

MultMatrix(Comp1, R1, R2, size);

MultMatrix(Comp2, R2, R1, size);

printf("R1 | R2:\n");

PrintMatrix(U, size);

printf("R1 & R2:\n");

PrintMatrix(X, size);

printf("R1 ^ R2:\n");

PrintMatrix(SX, size);

printf("!R1:\n");

PrintMatrix(R1C, size);

printf("!R2:\n");

PrintMatrix(R2C, size);

printf("R1 o R2:\n");

PrintMatrix(Comp1, size);

printf("R1 o R2:\n");

PrintMatrix(Comp2, size);

printf("R1^-1:\n");

PrintMatrix(R1T, size);

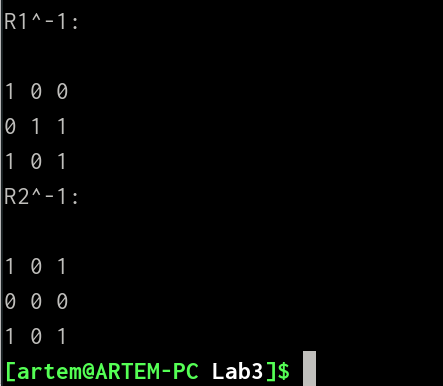
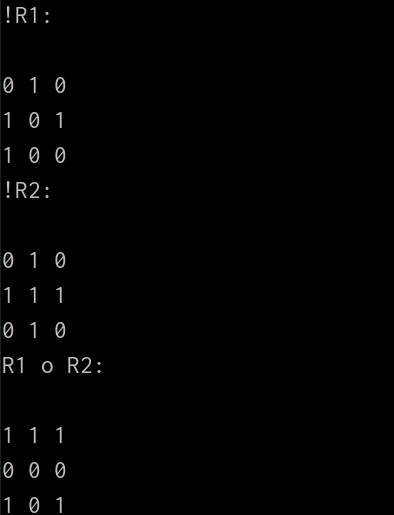
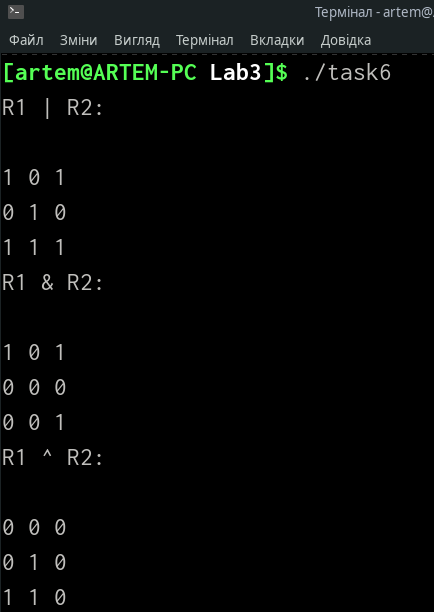
printf("R2^-1:\n");

PrintMatrix(R2T, size);

return 0;

}

**РЕЗУЛЬТАТИ**



ВИСНОВКИ

Дерева відіграють важливу роль в структурах даних.