МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра інформаційних систем та мереж



**ЗВІТ**

до виконання лабораторної роботи № 1

***“Комп’ютерне подання множин”***

з дисципліни “*Методи та системи штучного інтелекту*”

Виконав студент групи КН-38

***Пахолюк Степан***

Прийняла:

*асистент* ***Савчук В.В.***

Львів 2017

**Мета роботи**: Вивчити способи комп’ютерного подання множин та виконання комп’ютерних операцій над ними.

**Індивідуальне завдання:**

Написати програму, яка дозволяє розв’язувати наступну задачу.

1. Реалізувати операції A, AB ∪ , AB ∩ , AB \ , BA \ поданням заданих множин A та B , які є підмножинами n- елементної універсальної множини.

2. З допомогою бітових рядків виконати обчислення складних виразів з множинами. Для обчислення складних виразів з множинами використати побудовані операції.

**Короткі теоретичні відомості:**

Існують різні способи зображення множин у комп’ютері. Один із способів – залишати елементи множини у невпорядкованому вигляді. Проте, якщо так зробити, то операції із множинами вимагатимуть значних витрат часу через необхідність щоразу здійснювати перегляд елементів. Ми розглянемо інший спосіб зображення множин у комп’ютері. Упорядкуємо довільним способом елементи універсальної множини. Підмножина AU ⊂ зображатиметься у комп’ютері бітовим рядком, що складається із 0 та 1 та має довжину n, де i-й біт цього рядка дорівнює 1, якщо ai ∈ А, та дорівнює 0, якщо ai ∉ А.

**Результати роботи:**

***Код програми:***

#include <iostream>

#include <string>

namespace SetRelation

{

class Relation

{

static const size\_t relation\_size = 13;

static const std::string U;

static char charset;

char symbol;

std::string SET;

bool\* relation;

void SetToNull();

void SymbolToBinary();

void BinaryToSymbol();

public:

Relation();

Relation(const Relation&);

Relation& operator=(const Relation&);

friend Relation Intersection(const Relation&, const Relation&);

friend Relation Association(const Relation&, const Relation&);

friend Relation Difference(const Relation&, const Relation&);

friend Relation ExclusiveOr(const Relation&, const Relation&);

friend Relation Inverse(const Relation&);

friend std::ostream& operator<<(std::ostream&, const Relation&);

friend std::istream& operator >> (std::istream&, Relation&);

void Show() const;

~Relation();

};

}

namespace SetRelation {

char Relation::charset = '@';

const std::string Relation::U = "abcdefghklmnq";

Relation::Relation() : SET("") {

relation = new bool[relation\_size];

SetToNull();

symbol = ++charset;

}

Relation::Relation(const Relation& r) {

SET = r.SET;

symbol = r.symbol;

relation = new bool[relation\_size];

for (size\_t i(0); i < relation\_size; ++i)

relation[i] = r.relation[i];

}

Relation& Relation::operator=(const Relation& r) {

if (this != &r) {

delete[] relation;

SET = r.SET;

symbol = r.symbol;

relation = new bool[relation\_size];

for (size\_t i(0); i < relation\_size; ++i)

relation[i] = r.relation[i];

}

return \*this;

}

void Relation::Show() const {

std::cout << "Universal set U :\n";

std::cout << "{ ";

for (size\_t i(0); i < U.length(); ++i) {

if (i + 1 == U.length()) {

std::cout << U[i];

break;

}

std::cout << U[i] << ", ";

}

std::cout << " }" << std::endl;

std::cout << "Subset " << symbol << " :\n";

std::cout << "{ ";

for (size\_t i(0); i < SET.length(); ++i) {

if (i + 1 == SET.length()) {

std::cout << SET[i];

break;

}

std::cout << SET[i] << ", ";

}

std::cout << " }" << std::endl;

std::cout << "String of bits :\n";

for (size\_t i(0); i < relation\_size; ++i)

std::cout << relation[i] << " ";

std::cout << std::endl;

}

void Relation::SetToNull() {

for (size\_t i(0); i < relation\_size; ++i)

relation[i] = (bool)0;

}

void Relation::SymbolToBinary() {

for (size\_t i(0), j(0); i < Relation::U.length(); ++i)

if (Relation::U[i] == SET[j])

{

relation[i] = (bool)1;

++j;

}

else

relation[i] = (bool)0;

}

void Relation::BinaryToSymbol() {

for (size\_t i(0); i < Relation::U.length(); ++i)

if (relation[i])

{

SET += Relation::U[i];

}

}

std::ostream& operator<<(std::ostream& os, const Relation& r)

{

os << "Subset " << r.symbol << " :\n";

os << "{ ";

for (size\_t i(0); i < r.SET.length(); ++i)

{

if (i + 1 == r.SET.length())

{

os << r.SET[i];

break;

}

os << r.SET[i] << ", ";

}

os << " }" << std::endl;

for (size\_t i(0); i < r.relation\_size; ++i)

os << r.relation[i] << " ";

os << std::endl;

return os;

}

std::istream& operator >> (std::istream& is, Relation& r)

{

bool check = true;

do

{

check = true;

std::cout << "Enter subset " << r.symbol << " :\n"; is >> r.SET;

if (r.SET.length() > Relation::U.length())

{

check = false;

std::cout << "Re-enter subset " << r.symbol << "!" << std::endl;

continue;

}

for (size\_t i(0); i < r.SET.length(); ++i)

for (size\_t j(0); j < Relation::U.length(); ++j)

{

if (r.SET[i] == Relation::U[j])

break;

if (j + 1 == Relation::U.length())

{

check = false;

goto m;

}

}

m:

if (!check)

std::cout << "Re-enter subset " << r.symbol << "!" << std::endl;

} while (check == false);

r.SymbolToBinary();

return is;

}

Relation Intersection(const Relation& l\_r, const Relation& r\_r)

{

Relation temp;

for (size\_t i(0); i < temp.relation\_size; ++i)

temp.relation[i] = l\_r.relation[i] & r\_r.relation[i];

temp.BinaryToSymbol();

return temp;

}

Relation Association(const Relation& l\_r, const Relation& r\_r)

{

Relation temp;

for (size\_t i(0); i < temp.relation\_size; ++i)

temp.relation[i] = l\_r.relation[i] | r\_r.relation[i];

temp.BinaryToSymbol();

return temp;

}

Relation Difference(const Relation& l\_r, const Relation& r\_r)

{

Relation temp;

for (size\_t i(0); i < l\_r.relation\_size; ++i)

{

if ((l\_r.relation[i] == r\_r.relation[i]) == (bool)1)

{

temp.relation[i] = (bool)0;

continue;

}

if (l\_r.relation[i] == (bool)1)

{

temp.relation[i] = (bool)1;

continue;

}

temp.relation[i] = (bool)0;

}

temp.BinaryToSymbol();

return temp;

}

Relation ExclusiveOr(const Relation& l\_r, const Relation& r\_r)

{

Relation temp;

for (size\_t i(0); i < temp.relation\_size; ++i)

temp.relation[i] = l\_r.relation[i] ^ r\_r.relation[i];

temp.BinaryToSymbol();

return temp;

}

Relation Inverse(const Relation& r)

{

Relation temp(r);

temp.SET = "";

for (size\_t i(0); i < temp.relation\_size; ++i)

if (temp.relation[i] == (bool)1)

temp.relation[i] = (bool)0;

else

temp.relation[i] = (bool)1;

temp.BinaryToSymbol();

return temp;

}

Relation::~Relation()

{

delete[] relation;

}

}

int main(int argc, char\*\* argv) {

SetRelation::Relation A, B;

std::cout << "\n#### U:{ a,b,c,d,e,f,g,h,k,l,m,n,q } ####\n" << std::endl;

std::cin >> A;

std::cin >> B;

std::cout << A;

std::cout << B;

int choice;

do {

std::cout <<

"\t####\_---\_Please, choose operation\_---\_####\n"

"Menu:\n"

"1 - Intersection\n"

"2 - Union\n"

"3 - Difference\n"

"4 - XOR\n"

"5 - Custom operation\n"

"0 - Exit\n";

std::cout << "-> "; std::cin >> choice;

switch (choice) {

case 1: std::cout << SetRelation::Intersection(A, B);

break;

case 2: std::cout << SetRelation::Association(A, B);

break;

case 3: std::cout << SetRelation::Difference(A, B);

break;

case 4: std::cout << SetRelation::ExclusiveOr(A, B);

break;

case 5: std::cout << SetRelation::Inverse(SetRelation::Intersection(A,B)) << '\n';

break;

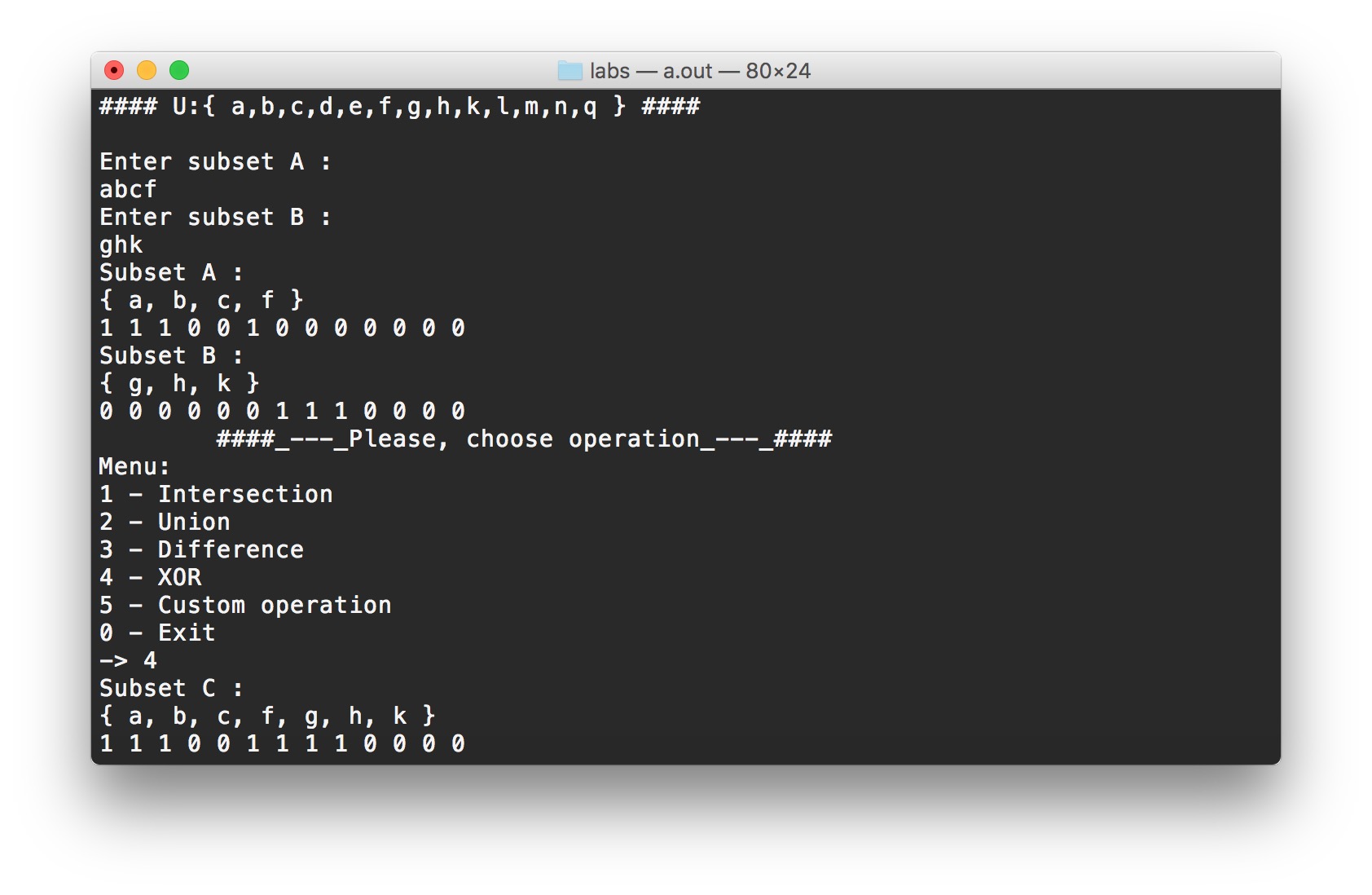
}

} while (choice != 0);

system("pause");

return 0;

}



*Рис. 1. Результат виконання програми.*

**Висновок:** Під час виконання даної лабораторної роботи було розглянуто способи комп’ютерного подання множин та реалізовано основні операцій над ними з використанням засобів комп’ютерної реалізації.