**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе № 1**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Множества»**

Студент гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Нукусси Ф.М

## Студент гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Петров М.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Колинько П.Г.

Санкт-Петербург

2024

**Цель работы:**

Исследование четырёх способов хранения множеств в памяти ЭВМ.

**Задание 14:**

Множество, содержащее все буквы, множества A, которых нет во миожествах B,C или в D

**Формализация задания:**

Программа будет работать по следующей формуле:

**E = A ~** B ∩ C ∩ D

**Контрольные тесты:**

**Ниже приведены контрольные тесты работы программы:**

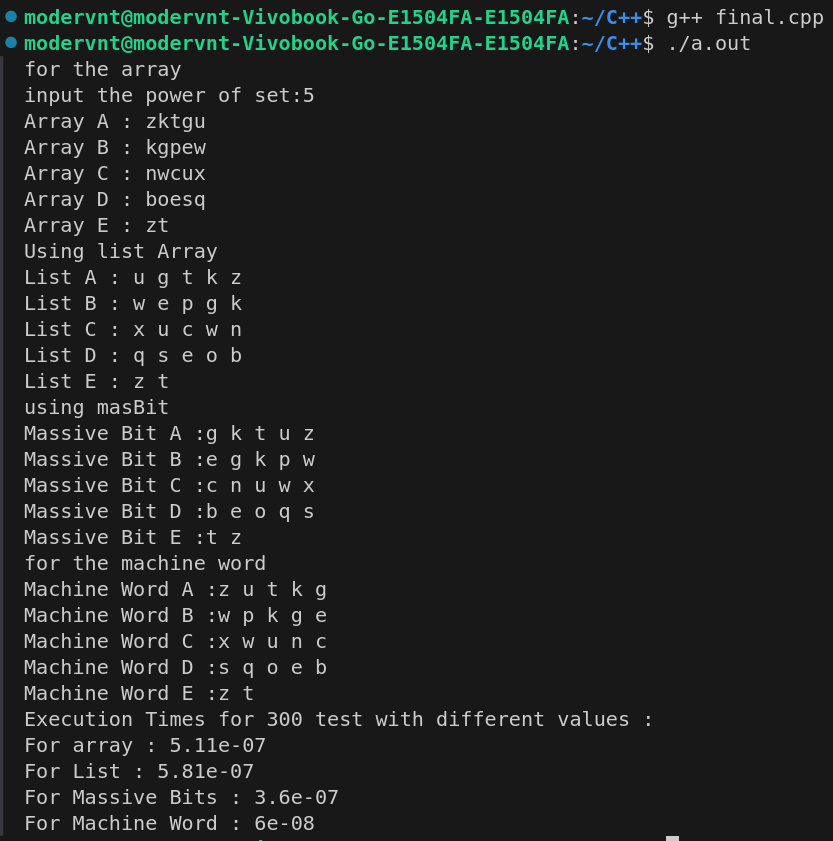


Рисунок 1 — результат работы со случайными множествами из 5 элементов

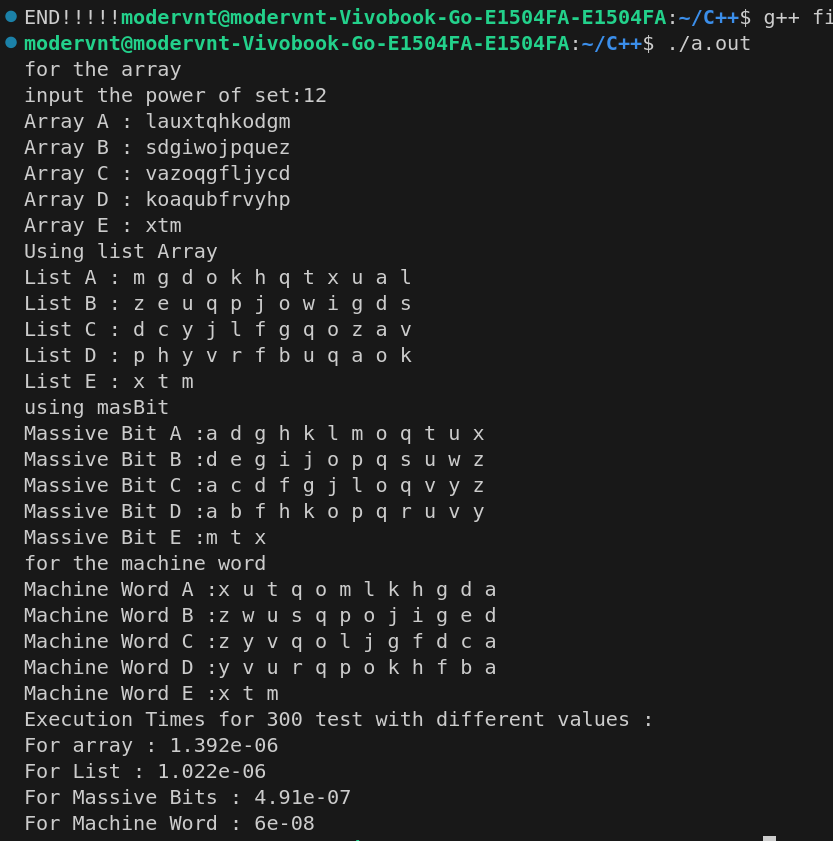
****

Рисунок 2 — результат работы со случайными множествами из 12 элементов

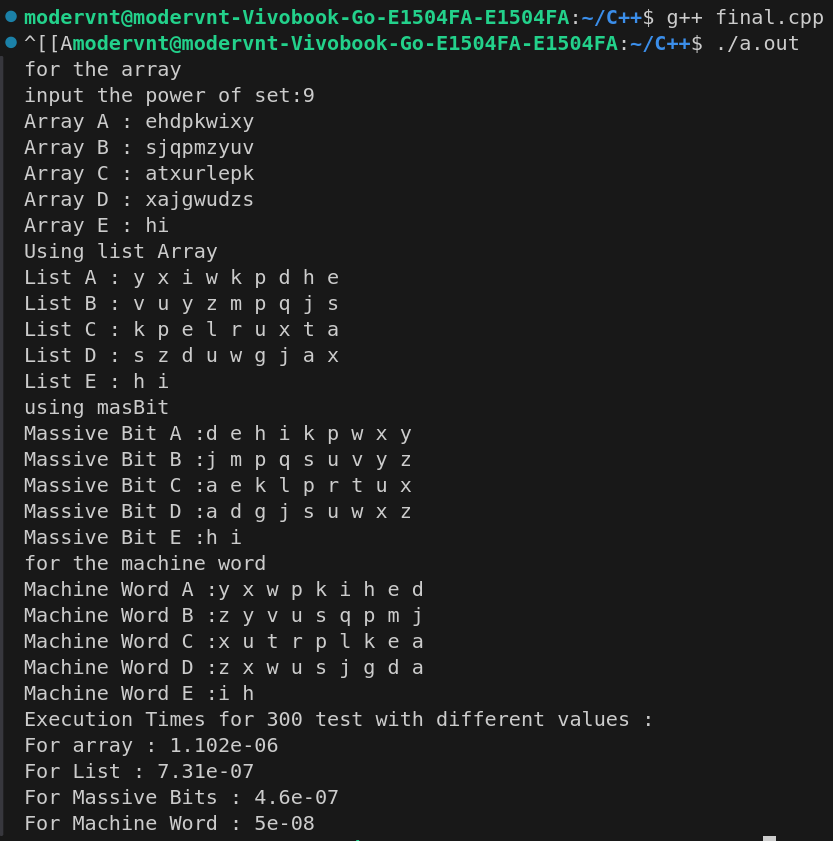
****

Рисунок 3 — результат работы со случайными множествами из 9 элементов

**Временная сложность:**

В представлении множества в виде массива ожидалась квадратичная сложность, фактическая — кубическая.

Для множеств в виде списков ожидалась квадратичная сложность, фактическая — кубическая.

Множества, представленные в виде массива битов и машинных слов, ожидались и вышли с константными сложностями соответственно. Увидеть это можно в следующем пункте.

**Результаты измерения времени обработки.**

Таблица 1. Массивы.

|  |  |
| --- | --- |
| Количество элементов в исходном множестве | Среднее время обработки (в миллисекундах) |
| 5 | 5.11e-07 |
| 9 | 1.102e-06 |
| 12 | 1.392e-06 |

Таблица 2. Списки.

|  |  |
| --- | --- |
| Количество элементов в исходном множестве | Среднее время обработки (в миллисекундах) |
| 5 | 5.81e-07 |
| 9 | 7.31e-07 |
| 12 | 1.022e-06 |

Таблица 3. Massive Bits.

|  |  |
| --- | --- |
| Количество элементов в исходном множестве | Среднее время обработки (в миллисекундах) |
| 5 | 3.6e-07 |
| 9 | 4.6e-07 |
| 12 | 4.91e-07 |

Таблица 4. Масhine Word.

|  |  |
| --- | --- |
| Количество элементов в исходном множестве | Среднее время обработки (в миллисекундах) |
| 5 | 6e-08 |
| 9 | 5e-08 |
| 12 | 6e-08 |

**Вывод.**

В зависимости от специфики задачи, наиболее подходящей структурой данных для представления и обработки множеств могут быть массивы, связные списки, битовые массивы или машинные слова. Проведенный анализ и тесты показали, что битовые массивы и машинные слова являются наиболее эффективными с точки зрения времени выполнения операций, тогда как массивы и связные списки могут быть удобнее для реализации и управления динамическими данными.

**Список использованных источников.**

Колинько П. Г. Пользовательские структуры данных: Методические указания по дисциплине “Алгоритмы и структуры данных, часть 1”. - СПб.: СПбГЭТУ “ЛЭТИ”, 2023. - 64 c. (вып.2309).

**Приложение. Исходный код:**

#include <iostream>

#include <algorithm>

#include <chrono>

#include <cstring>

#include <ratio>

#include <ctime>

using namespace std;

const int MAX\_SIZE = 26;

char U[MAX\_SIZE+1] = {"abcdefghijklmnopqrstuvwxyz"}; //UNIVERSUM

// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

// all about array

bool exist(char str, char A [])

{

    bool f = false;

    for(int i = 0; A[i] && !f; ++i ){

        f |= (str == A[i]);

    }

    return f;

}

void Calculation(char A[], char B[], char C[], char D[], char \* E){

    int count = 0;

    for(int i = 0; A[i]; i++){

        if(exist(A[i], B) && exist(A[i],C) && !exist(A[i],D)){

            E[count++] = A[i];

        }

    }

}

// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

// all about set and list

struct Set {

    char el;

    Set \* next;

    Set(char  e, Set \* n = nullptr) : el(e), next(n) { }

    ~Set( ) { delete next; }

};

Set \* AssigningValuesToEls(char Arr[], int i, int size)

{

    Set \*nxt;

    if (i < size){

        nxt = new Set(Arr[i], AssigningValuesToEls(Arr, i+1, strlen(Arr)));

    }

    else if (Arr[i] != '\0'){

        nxt = new Set(Arr[i++], nullptr);

    }

    else {

        nxt = nullptr;

    }

    return nxt;

}

void AddElsToE(Set \*&List\_E, char e){

    Set \*nxt;

    if (List\_E->el == '\0') {

            List\_E = new Set(e, nullptr);

    }

    else {

        for (nxt = List\_E; nxt->next; nxt = nxt->next);

        nxt->next = new Set(e, nullptr);

    }

}

bool exist\_list(char s, Set \*List\_A)

{

    bool f{false};

    for(Set \*p = List\_A; p && p->el && !f; p = p->next)

    {

        f |= (s == p->el);

    }

    return f;

}

void DEL\_list(Set \*&List\_E, Set \*List\_C, Set \*List\_D){

    Set \*p = List\_E;

    Set \*m = nullptr;

    Set \*g = nullptr;

    while (p != nullptr){

        bool f = false;

        g = List\_D;

        while (g != nullptr){

            if (p->el == g->el){

                if (m != nullptr){ // not first element to delete{

                    m->next = p->next;

                    p->next = nullptr;

                    delete p;

                    p = m->next;

                }

                else{

                    Set\* temp = p; // first element to delete

                    List\_E = p->next;

                    p = List\_E;

                    temp->next = nullptr;

                    delete temp;

                }

                f = true;

                break;

            }

            g = g->next;

        }

        if (!f){

            m = p;

            p = p->next;

        }

    }

}

void AND\_list(Set \*&List\_E, Set \*List\_A, Set \*List\_B){

    for (Set \*p = List\_A; p; p = p->next){

        for (Set \*g = List\_B; g; g = g->next){

            if (p->el == g->el){

                AddElsToE(List\_E, p->el);

            }

        }

    }

}

//A \* not(B)

void NOT\_AND\_list(Set \*&List\_E, Set \*List\_A, Set \*List\_B){

    for (Set \*p = List\_A; p; p = p->next){

        if(!exist\_list(p->el, List\_B)){

            AddElsToE(List\_E,p->el);

        }

    }

}

//E = (A \* B \* C) \* not(D)

void Count\_List(char A[], char B[], char C[], char D[]){

    Set \*List\_E = new Set('\0', nullptr);

    Set \*List\_E1 = new Set('\0', nullptr);

    Set \*List\_E2 = new Set('\0', nullptr);

    Set \*List\_A = new Set(A[0], AssigningValuesToEls(A, 1, strlen(A)));

    Set \*List\_B = new Set(B[0], AssigningValuesToEls(B, 1, strlen(B)));

    Set \*List\_C = new Set(C[0], AssigningValuesToEls(C, 1, strlen(C)));

    Set \*List\_D = new Set(D[0], AssigningValuesToEls(D, 1, strlen(D)));

    auto t1 = std::chrono::high\_resolution\_clock::now( );

    //E = (A \* B \* C) \* not(D)

    AND\_list(List\_E1, List\_A, List\_B);

    AND\_list(List\_E2, List\_E1, List\_C);

    NOT\_AND\_list(List\_E, List\_E2, List\_D);

    auto t2 = std::chrono::high\_resolution\_clock::now( );

    auto dt = chrono::duration\_cast<chrono::duration<double>>(t2-t1);

    if (List\_E != nullptr){

        cout<< "Count with list:  E = ";

        for (Set \*p = List\_E; p; p = p->next){

                cout<< p->el;

        }

        cout << "  time: ";

        cout << chrono::duration<double, nano> (dt).count() << " ns \n";

    }

    else{

        cout<< "Count with list: E = " << "empty set;" << " time: " << chrono::duration<double, nano> (t2-t1).count() << " ns \n";

    }

    delete List\_E;

    delete List\_E1;

    delete List\_E2;

    delete List\_A;

    delete List\_B;

    delete List\_C;

    delete List\_D;

}

// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

// all about Bit vector

void Bit\_Vector(char A[], char B[], char C[], char D[]){

    char E[MAX\_SIZE+1] = {};

    int bit\_vec\_A[MAX\_SIZE+1]{};

    int bit\_vec\_B[MAX\_SIZE+1]{};

    int bit\_vec\_C[MAX\_SIZE+1]{};

    int bit\_vec\_D[MAX\_SIZE+1]{};

    int bit\_vec\_E[MAX\_SIZE+1]{};

    for (int i = 0; A[i]; i++){

        bit\_vec\_A[A[i]-'a'] = 1;

    }

    for (int i = 0; B[i]; i++){

        bit\_vec\_B[B[i]-'a'] = 1;

    }

    for (int i = 0; C[i]; i++){

        bit\_vec\_C[C[i]-'a'] = 1;

    }

    for (int i = 0; D[i]; i++){

        bit\_vec\_D[D[i]-'a'] = 1;

    }

    auto t1 = std::chrono::high\_resolution\_clock::now( );

    for (int i = 0; i < MAX\_SIZE+1; i++){

        bit\_vec\_E[i] = bit\_vec\_A[i] && bit\_vec\_B[i] && bit\_vec\_C[i] && !(bit\_vec\_D[i]);

    }

    auto t2 = std::chrono::high\_resolution\_clock::now( );

    auto dt = chrono::duration\_cast<chrono::duration<double,nano>>(t2-t1);

    for (int i = 0, k = 0; i < MAX\_SIZE; ++i) {

        if (bit\_vec\_E[i]){

            E[k++] = (i+'a');

        }

    }

    if (U[0] != '\0'){

        cout << "Count with bit vector: E = " << E << " time: " << chrono::duration<double, nano> (dt).count() << " ns\n";

    }

    else{

            cout << "Count with bit vector: E = " << "empty set;" << " time: " << chrono::duration<double, nano> (dt).count() << " ns\n";

    }

}

// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

// all about Machine Word

void Mach\_Word(char A[], char B[], char C[], char D[]){

    int Mach\_W\_A = 0, Mach\_W\_B = 0, Mach\_W\_C = 0, Mach\_W\_D = 0, Mach\_W\_E = 0;

    char E[MAX\_SIZE+1];

    for (int i = 0; A[i]; ++i){

        Mach\_W\_A = Mach\_W\_A | (1 << (A[i]-'a'));

    }

    for (int i = 0; B[i]; ++i){

        Mach\_W\_B = Mach\_W\_B | (1 << (B[i]-'a'));

    }

    for (int i = 0; C[i]; ++i){

        Mach\_W\_C = Mach\_W\_C | (1 << (C[i]-'a'));

    }

    for (int i = 0; D[i]; ++i){

        Mach\_W\_D = Mach\_W\_D | (1 << (D[i]-'a'));

    }

    auto t1 = std::chrono::high\_resolution\_clock::now( );

    Mach\_W\_E = Mach\_W\_A & Mach\_W\_B & Mach\_W\_C & ~Mach\_W\_D;

    auto t2 = std::chrono::high\_resolution\_clock::now( );

    auto dt = chrono::duration\_cast<chrono::duration<double,nano>>(t2-t1);

    for (int i = 0, k = 0; i < MAX\_SIZE+1; ++i){

         if ((Mach\_W\_E >> i) & 1){

            E[k++] = (i+'a');

         }

    }

    bool flag = false;

    for (int i = 0; E[i]; i++){

        for (int j = 0; U[j]; j++){

            if (E[i] == U[j]){

                flag = true;

            }

        }

        if (!flag){

            E[i] = '\0';

        }

        flag = false;

    }

    if (E[0] != '\0' && E[0] != 'e') {

        cout << "Count with machine word: E = " << E << " time: " << chrono::duration<double, nano> (dt).count() << " ns\n";

    }

    else {

        cout << "Count with machine word: E = " << "empty set;" << " time: " <<  chrono::duration<double, nano> (dt).count() << " ns\n";

    }

}

// ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

//E = (A \* B \* C) \* not(D)

int main(){

    srand(time(0));

    int expression,len\_rand;

    char A[MAX\_SIZE+1] = {};

    char B[MAX\_SIZE+1] = {};

    char C[MAX\_SIZE+1] = {};

    char D[MAX\_SIZE+1] = {};

    char E[MAX\_SIZE+1] = {};

    cout << "1 - Enter from keyboard" << '\n' << "2 - random arrays " << '\n' << "3 - ready array" << '\n';

    cout << "Enter type of array: ";

    cin >>  expression;

    switch (expression){

    case 1:

        cout << "A = ";

        cin >> A;

        cout << "B = ";

        cin >> B;

        cout << "C = ";

        cin >> C;

        cout << "D = ";

        cin >> D;

        break;

    case 2:

        cout << "Enter length of array: ";

        cin >> len\_rand;

        cout << '\n';

        for ( int i = 0; i < len\_rand; ++i) {

            A[i] = char(rand()%26+0x61);

        }

        cout << "A = "<< A << '\n';

        for ( int i = 0; i < len\_rand; ++i) {

            B[i] = char(rand()%26+0x61);

            }

        cout << "B = "<< B << '\n';

        for ( int i = 0; i < len\_rand; ++i) {

            C[i] = char(rand()%26+0x61);

        }

        cout << "C = "<< C << '\n';

        for ( int i = 0; i < len\_rand; ++i) {

            D[i] = char(rand()%26+0x61);

        }

        cout << "D = "<< D << '\n';

        break;

    case 3:

        strcpy(A, "abcdfi");

        cout << "A = "<< A << '\n';

        strcpy(B, "bdfi");

        cout << "B = "<< B << '\n';

        strcpy(C, "dfghi");

        cout << "C = "<< C << '\n';

        strcpy(D, "aci");

        cout << "D = "<< D << '\n';

        break;

    default:

         cout << "Something goes wrong!";

        break;

    }

    auto t1 = std::chrono::high\_resolution\_clock::now( );

    Calculation(A,B,C,D,E);

    auto t2 = std::chrono::high\_resolution\_clock::now( );

    auto dt = chrono::duration\_cast<chrono::duration<double>>(t2-t1);

    cout << "E = " << E ;

    cout << "  "<< "time: ";

    cout << chrono::duration<double, milli> (dt).count() << " ms \n ";

    cout << '\n';

    Count\_List(A,B,C,D);

    Bit\_Vector(A,B,C,D);

    Mach\_Word(A,B,C,D);

    return 0;

}