Минобрнауки России

Федеральное государственное автономное образовательное учреждение

высшего образования «Санкт-Петербургский государственный

электротехнический университет «ЛЭТИ»

им В. И. Ульянова (Ленина)»

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Зачётная работа № 1**

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

**на тему «Множества в памяти ЭВМ»**

Выполнили студенты группы 9311:

Малкин И. А.

Чалкин Д. Е.

Принял: старший преподаватель Колинько П. Г.

Санкт-Петербург

2020

Оглавление

[1.Цель работы 3](#_Toc58882862)

[2. Задание 3](#_Toc58882863)

[3. Формула для вычисления пятого множества 3](#_Toc58882864)

[4. Контрольные тесты 3](#_Toc58882865)

[5. Временная сложность 4](#_Toc58882866)

[6. Результат измерения времени обработки каждым из способов. 4](#_Toc58882867)

[7. Выводы 6](#_Toc58882868)

[8. Список используемых источников 6](#_Toc58882869)

[9. Приложение. Текст программы 7](#_Toc58882870)

# 1.Цель работы

Сравнительное исследование четырёх способов хранения множеств в памяти ЭВМ.

# 2. Задание

Множество прописных русских букв, содержащее все буквы множества A, которых нет во множествах B, C или D.

# 3. Формула для вычисления пятого множества

Формализация задания: E = A u (B and C) u D

# 4. Контрольные тесты

Ниже представлены примеры работы программы (рис. 1 – 4) . Используя генератор подмножеств заданной мощности, формируем подмножества с мощностью от 2 до 32 и считаем количество тиков процессора, произошедших за период обработки подмножеств в различных формах их представления. Результат выводится в консоль.

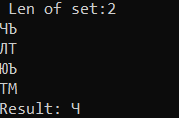
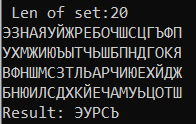
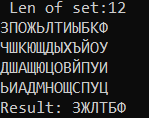
 

Рис. 1 Рис. 2 Рис. 3

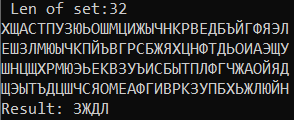


Рис. 4

# 5. Временная сложность

Таблица 1. Способы представления и временная сложность обработки

|  |  |  |
| --- | --- | --- |
| Способ представления | Временная сложность | |
| Ожидаемая | Фактическая |
| Массив символов | O() | O(n2) |
| Список | O(n2) |
| Универсум | O(|U|) | O(n) |
| Машинное слово | O(1) |

Пояснения:

Для множества, представленного набором элементов (массив символов или список) двуместная операция требует проверки всех комбинаций элементов множества, которых для множеств мощностью n будет O(n2). Общая сложность получится такой же, если вычисление заданного выражения свести к последовательности двуместных операций.

Для множеств, представленных отображением на универсум ожидаемое к‑во шагов двуместной операции равно мощности универсума.

Поскольку вычисление заданного выражения выполнено как последовательность двуместных операций, каждая из которых реализована двойным циклом по мощности множества, фактическая временная сложность алгоритма вычислений совпала с ожидаемой. Для отображения на универсум, мощность которого фиксирована, сложность вычисления можно считать константной

# 6. Результат измерения времени обработки для каждого из способов

Таблица 2. Результаты измерения времени обработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мощность множеств | Количество тиков процессора при обработке множеств при различных способах представления | | | |
| Массив символов | Список | Массив битов | Машинное слово |
| 2 | 1e-07 | 2e-06 | 2e-07 | 1e-07 |
| 3 | 7e-07 | 5.8e-06 | 7e-07 | 2e-07 |
| 4 | 2.1e-06 | 1.24e-05 | 9e-07 | 4e-07 |
| 5 | 2.8e-06 | 2.12e-05 | 1.3e-06 | 5e-07 |
| 6 | 4.4e-06 | 3.14e-05 | 1.7e-06 | 7e-07 |
| 7 | 6.2e-06 | 4.55e-05 | 2.2e-06 | 8e-07 |
| 8 | 8e-06 | 6.06e-05 | 2.5e-06 | 9e-07 |
| 9 | 1e-05 | 7.89e-05 | 3.2e-06 | 1e-06 |

Для получения более точных данных измеряемый процесс обработки множеств для каждого способа представления повторялся 10000 раз. При представлении множеств в виде массива символов и списка заметно, что время обработки множеств с увеличением мощности также увеличивается. Для универсума и машинного слова время обработки практически неизменно, т.е. не зависит от размера входа.

# 7. Выводы

В ходе данной работы пришли к выводу, что наиболее быстрым способом представления множеств для их обработки является машинное слово. Этот способ рекомендуется использовать, когда есть простая функция для отображения элемента множества в соответствующий ему порядковый номер бита и размер универсума не превышает разрядности слова.

Написанное ранее о представлении множеств в виде машинного слова также относится к вектору битов, однако мощность универсума в данном случае тоже должна быть ограничена.

Наиболее медленной оказалась обработка списков. Данный способ представления следует использовать, когда мощность создаваемого множества неизвестна, и выделить память сразу под всё множество не представляется возможным.

Представлять множество в виде набора элементов в массиве следует в том случае, если можно с достаточной точностью знать размер массива, а мощность универсума слишком велика для использования вектора битов или машинного слова.

# 8. Список используемых источников

1. Колинько, П.Г. Пользовательские структуры данных [Текст]: методические указания по дисциплине «Алгоритмы и структуры данных, часть 1» /П.Г. Колинько, Н. Т. – Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020. – 64 с.

2. Множества в памяти ЭВМ // Алгоритмы и структуры данных. Лекция от 11.09.2020.

3. Студент Медведев О. Частное сообщение.

# 9. Приложение. Текст программы

# 

#include <iostream>

#include <set>

#include <Windows.h>

#include <stdlib.h>

#include <time.h>

#include <ctime>

#include <fstream>

const short SHIFT = 192;

const short LENOFLETTERS = 32;

const int ONE = 1;

using namespace std;

#pragma region set

bool isInSet(unsigned char item, unsigned char\* set) {

    for (size\_t i = 0; set[i]; ++i) {

        if (set[i] == item) {

            return true;

        }

    }

    return false;

}

void InputSet(unsigned char\* set) {

    char symbol;

    int i = 0;

    do {

        cin >> symbol;

        if (symbol >= 'А' && symbol <= 'Я' && symbol != 'Ё' && !isInSet(symbol, set)) {

            set[i++] = symbol;

        }

    } while (cin.peek() != '\n');

}

void DeleteItem(int index, unsigned char\* set) {

    for (size\_t i = index; set[i]; ++i) {

        set[i] = set[i + 1];

    }

}

void AddItem(unsigned char item, unsigned char\* set) {

    size\_t i;

    for (i = 0; set[i]; ++i);

    set[i] = item;

    set[i + 1] = 0;

}

void UnionSets(unsigned char\* source, unsigned char\* destination) {

    for (size\_t i = 0; source[i]; ++i) {

        if (!isInSet(source[i], destination)) {

            AddItem(source[i], destination);

        }

    }

}

void ExclusionSets(unsigned char\* source, unsigned char\* destination) {

    for (size\_t i = 0; destination[i]; ++i) {

        if (isInSet(destination[i], source)) {

            DeleteItem(i, destination);

        }

    }

}

void PrintSet(unsigned char\* set) {

    for (size\_t i = 0; set[i]; ++i) {

        cout << set[i];

    }

}

#pragma endregion all what neded for set

#pragma region list

struct ST {

    unsigned char letter;

    ST\* next;

    ST(unsigned char l, ST\* n) : letter(l), next(n) {}

    ~ST() { delete next; }

};

ST\* ToList(unsigned char\* set) {

    ST\* x = nullptr;

    for (size\_t i = 0; set[i]; ++i) {

        x = new ST(set[i], x);

    }

    return x;

}

bool isInSet(unsigned char item, ST\* set) {

    ST\* x = set;

    while (x) {

        if (x->letter == item) {

            return true;

        }

        x = x->next;

    }

    return false;

}

void DeleteItem(unsigned char item, ST\*& set) {

    ST\* x = set;

    if (x->letter == item) {

        set = x->next;

        x->next = nullptr;

        delete x;

    }

    else {

        ST\* tmp = x;

        x = x->next;

        while (x) {

            if (x->letter == item) {

                tmp->next = x->next;

                x->next = nullptr;

                delete x;

                x = nullptr;

            }

            else {

                x = x->next;

                tmp = tmp->next;

            }

        }

    }

}

void AddItem(unsigned char item, ST\*& set) {

    set = new ST(item, set);

}

void UnionSets(ST\* source, ST\*& destination) {

    while (source) {

        if (!isInSet(source->letter, destination)) {

            AddItem(source->letter, destination);

        }

        source = source->next;

    }

}

void ExclusionSets(ST\* source, ST\*& destination) {

    while (source) {

        if (isInSet(source->letter, destination)) {

            DeleteItem(source->letter, destination);

        }

        source = source->next;

    }

}

void PrintSet(ST\* set) {

    while (set) {

        cout << set->letter;

        set = set->next;

    }

}

#pragma endregion all what needed for list

#pragma region bool

void ToBool(unsigned char\* set, bool\* universum) {

    for (size\_t i = 0; set[i]; ++i) {

        universum[set[i] - SHIFT] = true;

    }

}

void UnionSets(bool\* source, bool\* destination) {

    for (size\_t i = 0; i < LENOFLETTERS; ++i) {

        destination[i] = destination[i] | source[i];

    }

}

void ExclusionSets(bool\* source, bool\* destination) {

    for (size\_t i = 0; i < LENOFLETTERS; ++i) {

        destination[i] = destination[i] & ~source[i];

    }

}

void PrintSet(bool\* set) {

    for (size\_t i = 0; i < LENOFLETTERS; ++i) {

        if (set[i]) {

            cout << (unsigned char)(i + SHIFT);

        }

    }

}

#pragma endregion all what neded for bool

#pragma region word

int ToMachineWord(unsigned char\* set) {

    int mword = 0;

    for (size\_t i = 0; set[i]; ++i) {

        mword = mword | ONE << (set[i] - SHIFT);

    }

    return mword;

}

void UnionSets(int source, int& destination) {

    destination = source | destination;

}

void ExclusionSets(int source, int& destination) {

    destination = ~source & destination;

}

void PrintSet(int mword) {

    int one = 1;

    for (size\_t i = 0; i < LENOFLETTERS; ++i) {

        if (mword & one << i) {

            cout << (unsigned char)(i + SHIFT);

        }

    }

}

void SetByWord(int mword, unsigned char\* arr) {

    int one = 1;

    for (size\_t i = 0; i < LENOFLETTERS; ++i) {

        if (mword & one << i) {

            AddItem((unsigned char)(i + SHIFT), arr);

        }

    }

}

#pragma endregion

#pragma region Random generators

void Generator(int len, unsigned char\* arr)

{

    unsigned char uni[LENOFLETTERS + 1] = { '\0' };

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

    {

        uni[i] = 'А' + i;

    }

    for (int i = 0; i < len; i++)

    {

        int p = rand() % (LENOFLETTERS - i);

        if (p)

        {

            swap(uni[p + i], uni[i]);

        }

    }

    for (int i = 0; i < len; i++)

    {

        AddItem(uni[i], arr);

    }

}

void Discharge(unsigned char\* arr)

{

    for (size\_t i = 0; arr[i]; i++)

    {

        arr[i] = 0;

    }

}

#pragma endregion Genarators for tests

int main()

{

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    unsigned char choose;

    unsigned char A[LENOFLETTERS + 1] = { '\0' };

    unsigned char B[LENOFLETTERS + 1] = { '\0' };

    unsigned char C[LENOFLETTERS + 1] = { '\0' };

    unsigned char D[LENOFLETTERS + 1] = { '\0' };

    unsigned char E[LENOFLETTERS + 1] = { '\0' };

    bool A\_Bool[LENOFLETTERS] = { false };

    bool B\_Bool[LENOFLETTERS] = { false };

    bool C\_Bool[LENOFLETTERS] = { false };

    bool D\_Bool[LENOFLETTERS] = { false };

    bool E\_Bool[LENOFLETTERS] = { false };

    int A\_Mword = 0;

    int B\_Mword = 0;

    int C\_Mword = 0;

    int D\_Mword = 0;

    int E\_Mword = 0;

    srand(static\_cast<unsigned int>(time(0)));

#pragma region Время длителности

    clock\_t t, sumt1 = 0, sumt2 = 0, sumt3 = 0, sumt4 = 0;

    int countIteartion = 1;

    for (size\_t j = 2; j <=32; j+=2)

    {

        cout << " Len of set:" << j<<endl;

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

        {

            Generator(j, A);

            Generator(j, B);

            Generator(j, C);

            Generator(j, D);

            cout << A << endl;

            cout << B << endl;

            cout << C << endl;

            cout << D << endl;

#pragma region Array

            t = clock();

            UnionSets(A, E);

            ExclusionSets(B, E);

            ExclusionSets(C, E);

            ExclusionSets(D, E);

            sumt1 += clock() - t;

            cout << "Result: " << E << endl;

#pragma endregion

#pragma region List

            ST\* A\_List = nullptr;

            ST\* B\_List = nullptr;

            ST\* C\_List = nullptr;

            ST\* D\_List = nullptr;

            ST\* E\_List = nullptr;

            A\_List = ToList(A);

            B\_List = ToList(B);

            C\_List = ToList(C);

            D\_List = ToList(D);

            t = clock();

            UnionSets(A\_List, E\_List);

            ExclusionSets(B\_List, E\_List);

            ExclusionSets(C\_List, E\_List);

            ExclusionSets(D\_List, E\_List);

            delete A\_List;

            delete B\_List;

            delete C\_List;

            delete D\_List;

            delete E\_List;

            sumt2 += clock() - t;

#pragma endregion

#pragma region Bool array

            ToBool(A, A\_Bool);

            ToBool(B, B\_Bool);

            ToBool(C, C\_Bool);

            ToBool(D, D\_Bool);

            t = clock();

            UnionSets(A\_Bool, E\_Bool);

            ExclusionSets(B\_Bool, E\_Bool);

            ExclusionSets(C\_Bool, E\_Bool);

            ExclusionSets(D\_Bool, E\_Bool);

            sumt3 += clock() - t;

#pragma endregion

#pragma region Bool array

            A\_Mword = ToMachineWord(A);

            B\_Mword = ToMachineWord(B);

            C\_Mword = ToMachineWord(C);

            D\_Mword = ToMachineWord(D);

            t = clock();

            UnionSets(A\_Mword, E\_Mword);

            ExclusionSets(B\_Mword, E\_Mword);

            ExclusionSets(C\_Mword, E\_Mword);

            ExclusionSets(D\_Mword, E\_Mword);

            sumt4 += clock() - t;

#pragma endregion

            Discharge(A);

            Discharge(B);

            Discharge(C);

            Discharge(D);

            Discharge(E);

            A\_Mword = 0;

            B\_Mword = 0;

            C\_Mword = 0;

            D\_Mword = 0;

            E\_Mword = 0;

        }

        cout << " Array: " << (double)sumt1 / CLK\_TCK / countIteartion << endl;

        cout << " List: " << (double)sumt2 / CLK\_TCK / countIteartion << endl;

        cout << " Boolean array: " << (double)sumt3 / CLK\_TCK / countIteartion << endl;

        cout << " Machine word: " << (double)sumt4 / CLK\_TCK / countIteartion << endl;

    }

    return 0;

}