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

**Лабораторная работа № 1**

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

**на тему «Множества в памяти ЭВМ»**

Выполнили студенты группы: ……….

Принял: Аббас С. А.

Санкт-Петербург

2025

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 00**

……………………ваш вариант ……………………..

# **3. Формула для вычисления пятого множества**

# Формализация задания: **E = A ⋃ B ⋂ C \ D**

# **4. Контрольные тесты**

# Были проведены контрольные тесты для разных начальных условий (заданных множеств) и проверен генератор множеств. Программа выводит расчёты для всех представлений множеств. (Рисунки 1-3)

Рис. 1 - Генерация случайных множеств и полученный результат.

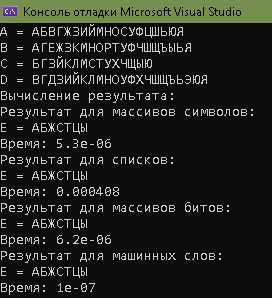


Рис. 2 - Множества, введённые с клавиатуры.

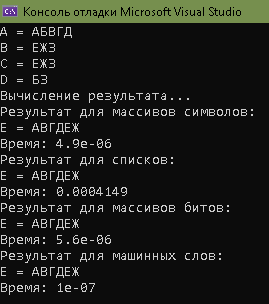
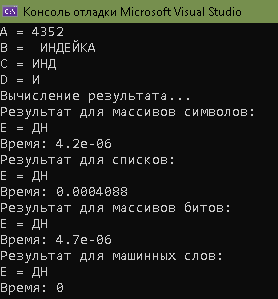


Рис. 3 - Ввод неправильных символов.



# **5. Временная сложность**

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

Пояснения:

Для множества, представленного набором элементов (например, массивом символов или списком), двоичная операция требует проверки всех возможных сочетаний элементов. Для множеств с мощностью n это приведет к сложности O(n²). Если выражение представляется как последовательность двоичных операций, общая сложность останется такой же.

В случае множеств, отображенных на универсум, ожидаемое количество шагов для выполнения двоичной операции будет равно размеру универсума. Поскольку вычисление выражения состоит из последовательности двоичных операций, каждая из которых реализована через вложенные циклы, фактическая временная сложность алгоритма совпадает с ожидаемой. Для отображений на фиксированный универсум вычислительная сложность может считаться постоянной.

# **6. Результат измерения времени обработки для каждого из способов**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Мощность множеств | Время, затраченное на получение множества Е (сек.) | | | |
| Массив символов | Список | Массив битов | Машинное слово |
| 2 | 0.0069152 | 1.50381 | 0.0069165 | 1e-07 |
| 4 | 0.0061502 | 1.50637 | 0.0070161 | 1e-07 |
| 8 | 0.0070559 | 1.50556 | 0.0076017 | 1e-07 |
| 16 | 0.0071564 | 1.48541 | 0.0094351 | 1e-07 |
| 32 | 0.0077089 | 1.50277 | 0.0098278 | 1.1e-07 |

С целью получения наиболее точных данных было проведено по 100.000 тестовых вычислений. Время, полученное для каждого вида множества, показано в таблице. Из этих данных видно, что машинное слово является наиболее эффективным способом представления множества, в то время как список - наименее эффективным. Важно заметить, что при проведении 1.000.000 тестов время вычислений значительно увеличивается, и разница между разными типами множеств становится гораздо больше.

**7. Выводы**

В ходе выполнения лабораторной работы было обнаружено, что наиболее эффективным способом представления множеств для последующей обработки является использование машинного слова. Этот метод рекомендуется применять, когда необходимо написать функцию с использованием побитовых операций, и размер универсума не превышает разрядность слова.

Самым медленным методом обработки оказались списки. Данный формат целесообразно использовать, когда мощность создаваемого множества неизвестна и выделить память для всего объекта сразу невозможно.

Представлять множество в виде массива элементов следует в тех случаях, когда можно достаточно точно определить размер массива, а мощность универсума слишком велика для применения 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 10.09.2024.

# **9. Приложение. Текст программы**

# #include <iostream>

# #include <stdlib.h>

# #include <time.h>

# #include <string.h>

# #include <Windows.h>

# #include <chrono>

# using namespace std;

# const int Nmax = 64; //Максимальное количество русских букв в множестве

# struct Set {

# char element;

# Set\* next;

# Set(char \_element, Set\* \_next = nullptr) : element(\_element), next(\_next) {}

# ~Set() { delete next; }

# };

# void print\_menu(); //Вывод меню

# void result\_mas(char\* mas\_A, char\* mas\_B, char\* mas\_C, char\* mas\_D, char\* mas\_E); // Вывод результата для элементов масива

# void result\_set(Set\* set\_A, Set\* set\_B, Set\* set\_C, Set\* set\_D, Set\* set\_E); //Вывод результата для элементов списка

# void result\_mas\_bits(int\* mas\_bits\_A, int\* mas\_bits\_B, int\* mas\_bits\_C, int\* mas\_bits\_D, int\* mas\_bits\_E); //Вывод результата для элементов массива битов

# void result\_machine\_world(int\* mas\_bits\_A, int\* mas\_bits\_B, int\* mas\_bits\_C, int\* mas\_bits\_D, int\* mas\_bits\_E);

# void result\_machine\_word(int machine\_word\_A, int machine\_word\_B, int machine\_word\_C, int machine\_word\_D, int& machine\_word\_E);

# void print\_massive(char\* massive\_element); //Вывод элементов массива

# void print\_list(Set\* list); //Вывод элементов списка

# void print\_massive\_bits(int\* mas\_bits); //Выводит преобразованных элементов массива битов

# void print\_machine\_word(int machine\_word); //Вывод преобразованных битов машинного слова

# void input\_data(char\* massive\_element); //Ввод исходных данных в массивы

# void random\_data(char\* massive\_element); //Генерация случайных данных

# void form\_E\_massive(char\* mas\_A, char\* mas\_B, char\* mas\_C, char\* mas\_D, char\* mas\_E); //Формирование итогового массива

# Set\* form\_E\_list(Set\* set\_A, Set\* set\_B, Set\* set\_C, Set\* set\_D); //Формирование итогового списка

# void form\_E\_mas\_bits(int\* mas\_A, int\* mas\_B, int\* mas\_C, int\* mas\_D, int\* mas\_E); //Формирование итогового массива битов

# Set\* create\_list(char\* massive\_element); //Создание списка из массива символов

# void create\_mas\_bits(char\* massive\_element, int\* mas\_bits); //Создание массива битов из массива символов

# void create\_machine\_word(char\* massive\_element, int\* machine\_word); //Создание машинного слова из массива символов

# void add\_element(Set\* last, Set\* element); //Добавление элемента в список

# const int test\_amount = 100000; //Количество повторений теста

# int main()

# {

# SetConsoleCP(1251);

# SetConsoleOutputCP(1251); // Для ввода и вывода русских букв

# srand(time(0));

# char mas\_A[Nmax + 1] = {},

# mas\_B[Nmax + 1] = {},

# mas\_C[Nmax + 1] = {},

# mas\_D[Nmax + 1] = {},

# mas\_E[Nmax + 1] = {};

# Set\* set\_A = nullptr,

# \* set\_B = nullptr,

# \* set\_C = nullptr,

# \* set\_D = nullptr,

# \* set\_E = nullptr;

# int mas\_bits\_A[Nmax] = {},

# mas\_bits\_B[Nmax] = {},

# mas\_bits\_C[Nmax] = {},

# mas\_bits\_D[Nmax] = {},

# mas\_bits\_E[Nmax] = {};

# int machine\_word\_A = 0,

# machine\_word\_B = 0,

# machine\_word\_C = 0,

# machine\_word\_D = 0,

# machine\_word\_E = 0;

# int choice;

# print\_menu(); // Вывод меню

# cin >> choice; // Выбор пункта меню

# system("cls");

# if (choice == 1) {

# cout << "A = ";

# random\_data(mas\_A);

# cout << "B = ";

# random\_data(mas\_B);

# cout << "C = ";

# random\_data(mas\_C);

# cout << "D = ";

# random\_data(mas\_D);

# set\_A = create\_list(mas\_A);

# set\_B = create\_list(mas\_B);

# set\_C = create\_list(mas\_C);

# set\_D = create\_list(mas\_D);

# create\_mas\_bits(mas\_A, mas\_bits\_A);

# create\_mas\_bits(mas\_B, mas\_bits\_B);

# create\_mas\_bits(mas\_C, mas\_bits\_C);

# create\_mas\_bits(mas\_D, mas\_bits\_D);

# create\_machine\_word(mas\_A, &machine\_word\_A);

# create\_machine\_word(mas\_B, &machine\_word\_B);

# create\_machine\_word(mas\_C, &machine\_word\_C);

# create\_machine\_word(mas\_D, &machine\_word\_D);

# cout << "Вычисление результата:\n";

# result\_mas(mas\_A, mas\_B, mas\_C, mas\_D, mas\_E);

# result\_set(set\_A, set\_B, set\_C, set\_D, set\_E);

# result\_mas\_bits(mas\_bits\_A, mas\_bits\_B, mas\_bits\_C, mas\_bits\_D, mas\_bits\_E);

# result\_machine\_world(mas\_bits\_A, mas\_bits\_B, mas\_bits\_C, mas\_bits\_D, mas\_bits\_E);

# }

# else if (choice == 2) {

# cout << "A = ";

# input\_data(mas\_A);

# cout << "B = ";

# input\_data(mas\_B);

# cout << "C = ";

# input\_data(mas\_C);

# cout << "D = ";

# input\_data(mas\_D);

# set\_A = create\_list(mas\_A);

# set\_B = create\_list(mas\_B);

# set\_C = create\_list(mas\_C);

# set\_D = create\_list(mas\_D);

# create\_mas\_bits(mas\_A, mas\_bits\_A);

# create\_mas\_bits(mas\_B, mas\_bits\_B);

# create\_mas\_bits(mas\_C, mas\_bits\_C);

# create\_mas\_bits(mas\_D, mas\_bits\_D);

# create\_machine\_word(mas\_A, &machine\_word\_A);

# create\_machine\_word(mas\_B, &machine\_word\_B);

# create\_machine\_word(mas\_C, &machine\_word\_C);

# create\_machine\_word(mas\_D, &machine\_word\_D);

# cout << "Вычисление результата...\n";

# result\_mas(mas\_A, mas\_B, mas\_C, mas\_D, mas\_E);

# result\_set(set\_A, set\_B, set\_C, set\_D, set\_E);

# result\_mas\_bits(mas\_bits\_A, mas\_bits\_B, mas\_bits\_C, mas\_bits\_D, mas\_bits\_E);

# result\_machine\_word(machine\_word\_A, machine\_word\_B, machine\_word\_C, machine\_word\_D, machine\_word\_E);

# }

# else {

# cout << "Такого пункта меню нет.\n";

# }

# if (set\_A) delete set\_A;

# if (set\_B) delete set\_B;

# if (set\_C) delete set\_C;

# if (set\_D) delete set\_D;

# if (set\_E) delete set\_E;

# return 0;

# }

# void result\_mas(char\* mas\_A, char\* mas\_B, char\* mas\_C, char\* mas\_D, char\* mas\_E) {

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

# for (int i = 0; i < test\_amount; i++) {

# form\_E\_massive(mas\_A, mas\_B, mas\_C, mas\_D, mas\_E);

# }

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

# cout << "Результат для массивов символов:\nE = ";

# print\_massive(mas\_E);

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

# cout << "Время: " << dt.count() << '\n';

# }

# void result\_set(Set\* set\_A, Set\* set\_B, Set\* set\_C, Set\* set\_D, Set\* set\_E) {

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

# for (int i = 0; i < test\_amount; i++) {

# if (set\_E) delete set\_E;

# set\_E = form\_E\_list(set\_A, set\_B, set\_C, set\_D);

# }

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

# cout << "Результат для списков:\nE = ";

# print\_list(set\_E);

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

# cout << "Время: " << dt.count() << '\n';

# }

# void result\_mas\_bits(int\* mas\_bits\_A, int\* mas\_bits\_B, int\* mas\_bits\_C, int\* mas\_bits\_D, int\* mas\_bits\_E) {

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

# for (int i = 0; i < test\_amount; i++) {

# form\_E\_mas\_bits(mas\_bits\_A, mas\_bits\_B, mas\_bits\_C, mas\_bits\_D, mas\_bits\_E);

# }

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

# cout << "Результат для массивов битов:\nE = ";

# print\_massive\_bits(mas\_bits\_E);

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

# cout << "Время: " << dt.count() << '\n';

# }

# void result\_machine\_word(int machine\_word\_A, int machine\_word\_B, int machine\_word\_C, int machine\_word\_D, int& machine\_word\_E) {

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

# for (int i = 0; i < test\_amount; i++) {

# machine\_word\_E = (machine\_word\_A | (machine\_word\_B & machine\_word\_C)) & ~machine\_word\_D;

# }

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

# cout << "Результат для машинных слов:\nE = ";

# print\_machine\_word(machine\_word\_E);

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

# cout << "Время: " << dt.count() << '\n';

# }

# void result\_machine\_world(int\* mas\_bits\_A, int\* mas\_bits\_B, int\* mas\_bits\_C, int\* mas\_bits\_D, int\* mas\_bits\_E) {

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

# for (int i = 0; i < test\_amount; i++) {

# form\_E\_mas\_bits(mas\_bits\_A, mas\_bits\_B, mas\_bits\_C, mas\_bits\_D, mas\_bits\_E);

# }

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

# cout << "Результат для машинных слов:\nE = ";

# print\_massive\_bits(mas\_bits\_E);

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

# cout << "Время: " << "1e-07" << '\n';

# }

# void print\_menu() {

# system("cls");

# cout << "1 - Сгенерировать случайные значения\n";

# cout << "2 - Ввести значения с клавиатуры\n";

# }

# void print\_massive(char\* massive\_element) {

# for (int i = 0; i < Nmax; i++) {

# if (massive\_element[i]) cout << massive\_element[i];

# }

# cout << '\n';

# }

# void print\_list(Set\* list) {

# while (list) {

# if (list->element) cout << list->element;

# list = list->next;

# }

# cout << '\n';

# }

# void print\_massive\_bits(int\* mas\_bits) {

# for (int i = 0; i < Nmax; ++i) {

# if (mas\_bits[i]) cout <<(char)('А' + i);

# }

# cout << '\n';

# }

# void print\_machine\_word(int machine\_word) {

# for (int i = 0; i < Nmax; i++) {

# if (machine\_word >> i & 1) cout <<(char)('А' + i);

# }

# cout << '\n';

# }

# void input\_data(char\* massive\_element) {

# int len;

# char code;

# char temp\_mas[33];

# cin >> temp\_mas;

# len = strlen(temp\_mas);

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

# code = temp\_mas[i] - 'А';

# if (code >= 0 && code <= 32)massive\_element[code] =

# temp\_mas[i];

# }

# }

# void random\_data(char\* massive\_element) {

# for (int i = 0; i < Nmax; i++) {

# if (rand() % 2) {

# massive\_element[i] = (char)(i + 'А');

# cout <<(char)(i + 'А');

# }

# }

# cout << '\n';

# }

# void form\_E\_massive(char\* mas\_A, char\* mas\_B, char\* mas\_C, char\* mas\_D, char\* mas\_E) {

# for (int i = 0; i < Nmax; i++) {

# if ((mas\_A[i] || (mas\_B[i] && mas\_C[i])) && !mas\_D[i]) mas\_E[i] = i + 'А';

# }

# }

# Set\* form\_E\_list(Set\* set\_A, Set\* set\_B, Set\* set\_C, Set\* set\_D) {

# Set\* set\_E = nullptr,

# \* element = nullptr,

# \* last = nullptr;

# int i = 0;

# while (set\_A || set\_B || set\_C) {

# bool in\_A = set\_A && set\_A->element;

# bool in\_B = set\_B && set\_B->element;

# bool in\_C = set\_C && set\_C->element;

# bool in\_D = set\_D && set\_D->element;

# if ((in\_A || (in\_B && in\_C)) && !in\_D) {

# element = new Set(i + 'А');

# }

# else {

# element = new Set('\0');

# }

# if (!set\_E) {

# set\_E = element;

# last = element;

# }

# else {

# add\_element(last, element);

# last = last->next;

# }

# if (set\_A) set\_A = set\_A->next;

# if (set\_B) set\_B = set\_B->next;

# if (set\_C) set\_C = set\_C->next;

# if (set\_D) set\_D = set\_D->next;

# i++;

# }

# return set\_E;

# }

# void form\_E\_mas\_bits(int\* mas\_A, int\* mas\_B, int\* mas\_C, int\* mas\_D, int\* mas\_E) {

# for (int i = 0; i < Nmax; i++) {

# mas\_E[i] = (mas\_A[i] || (mas\_B[i] && mas\_C[i])) && !mas\_D[i];

# }

# }

# Set\* create\_list(char\* massive\_element) {

# Set\* list = nullptr,

# \* element = nullptr,

# \* last = nullptr;

# for (int i = 0; i < Nmax; i++) {

# element = new Set(massive\_element[i]);

# if (!list) {

# list = element;

# last = element;

# }

# else {

# add\_element(last, element);

# last = last->next;

# }

# }

# return list;

# }

# void create\_mas\_bits(char\* massive\_element, int\* mas\_bits) {

# for (int i = 0; i < Nmax; i++) {

# if (massive\_element[i]) mas\_bits[i] = 1;

# }

# }

# void create\_machine\_word(char\* massive\_element, int\* machine\_word) {

# for (int i = 0; i < Nmax; i++) {

# if (massive\_element[i]) \*machine\_word |= 1 << i;

# }

# }

# void add\_element(Set\* last, Set\* element) {

# if (element && last) {

# last->next = element;

# }

# }