МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ЛАБОРАТОРНАЯ РАБОТА**

**«Структура хранения множества»**

**Выполнил:** студент группы 381706-2

Чирков Роман Дмитриевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ,

Шестакова Наталья Валерьевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Оглавление

[Введение 2](#_Toc502266332)

[Постановка задачи 3](#_Toc502266333)

[Руководство пользователя 4](#_Toc502266335)

[Руководство программиста 6](#_Toc502266336)

[Алгоритмы сортировок 10](#_Toc502266337)

[Заключение 16](#_Toc502266338)

[Литература 17](#_Toc502266339)

[Приложение 18](#_Toc502266340)

# Введение

*Теория множеств* — это раздел логики и математики, в рамках которого изучаются классы (множества) элементов произвольной природы. Множество при этом понимается как произвольная совокупность определённых и различимых объектов, мысленно объединённых в единое целое и называемых его элементами. Методы теории множеств широко используются во всех областях современной *математики* и *математической логики*; они имеют принципиальное значение для вопросов обоснования математики логическими средствами.

Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений. Вместе с тем лишь в отдельных языках программирования предусмотрены встроенные средства для работы с множествами. Программная реализация множества может выполняться различными способами (в соответствии с требованиями конкретной задачи или с общих позиций) и обычно тесно связана с использованием битовых операций в выбранном языке программирования. Данная работа посвящена изучению одного из возможных подходов к хранению и обработке множеств.

# Постановка задачи

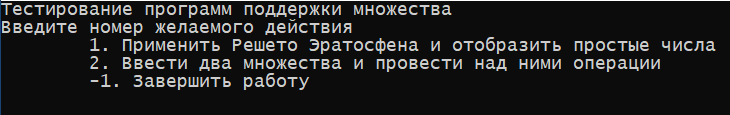
Я ставлю перед собой несколько главных учебно-практических задач:

1. Создать класс битового поля и реализовать в нём следующие функции:
   * установить бит (в единицу);
   * очистить бит (в ноль);
   * получить значение бита;
   * сравнить два битовых поля;
   * выполнить операцию “логическое или” для двух битовых полей;
   * выполнить операцию “логическое и” для двух битовых полей;
   * выполнить операцию “логическое отрицание” для битового поля.
2. Создать класс множества и реализовать в нём следующие функции:
   * включение элемента в множество;
   * исключение элемента из множества;
   * проверка наличия элемента в множестве;
   * сравнение множеств
   * сложение множеств;
   * пересечение множеств;
   * разность множеств;
   * копирование множества;
   * вычисление максимальной мощности множества.

# Руководство пользователя

Программа предназначена демонстрации работы класса множества.

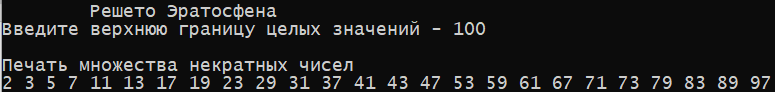
Управление осуществляется через меню. Пользователю следует ввести число 1, 2 или -1 для выбора соответствующего действия.



1. Применить Решето Эратосфена и отобразить простые числа

Пункт используется демонстрации работы функции, отображающей простые числа до заданного верхнего значения.

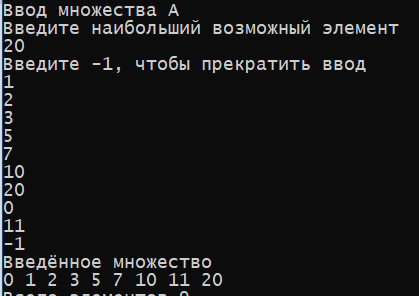
Пользователю предлагается ввести верхнюю границ целых значений. Затем после ввода, печатаются через пробел все простые числа не превосходящие заданную границу



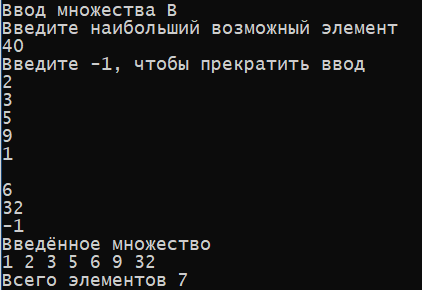
1. Ввести два множества и провести над ними операции

Пункт используется для проведения операций над двумя введёнными множествами.

Пользователю предлагается ввести множество A.



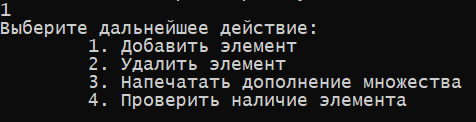
Затем – множество B.



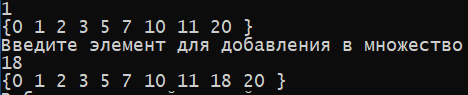
После ввода предлагается выбрать номер дальнейшего действия.

1. Операции над A

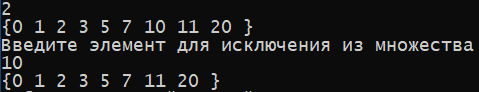
При выборе этого пункта появляется меню выбора операции.



Добавить элемент.



Удалить элемент.



2. Операции над B

3. Операции над обоими множествами

-1. Завершить работу c этими множествами

# Руководство программиста

*Описание функций и процедур*

В программе используются 4 процедуры и 6 функций:

1. void menu – меню;
2. int createArray – генерация массива;
3. void printArray – печать массива;
4. int heapSort – сортировка кучей;
5. void siftDown – формирование дерева;
6. int ShellSort –сортировка Шелла;
7. int selectionSort – сортировка выбором
8. int search – линейный поиск;
9. int binarySearch – бинарный поиск
10. void swap – перестановка двух элементов.
11. Меню

Данная процедура используется для взаимодействия с пользователем программы. По запросу пользователя может вызывать другие функции и процедуры.  
Параметры отсутствуют.

1. Генерация массива

Данная функция генерирует динамический массив заданного размера, состоящий из целых чисел.

Функция возвращает адрес начала массива.

Параметры (int size): параметром является размер массива.

1. Печать массива

Данная процедура выводит построчно каждый элемент массива в консоль.

Параметры (int \*a, int size): первым параметром является указатель на начало массива, вторым – размер массива.

1. Сортировка кучей

Данная функция выполняет пирамидальную сортировку массива.

Функция возвращает 1, когда массив отсортирован.

Параметры (int \*numbers, int array\_size): первым параметром является указатель на начало массива, вторым – размер массива.

1. Просеивание

Данная процедура формирует сортирующее дерево. Является частью сортировки кучей.

Параметры (int \*numbers, int root, int bottom): первым параметром является указатель на начало массива, вторым –"корень", третьим– "низ" дерева.

1. Сортировка Шелла

Данная функция выполняет сортировку Шелла.

Функция возвращает 1, когда массив отсортирован.

Параметры (int \*a, int size): первым параметром является указатель на начало массива, вторым – размер массива.

1. Сортировка выбором

Данная функция выполняет сортировку выбором.

Функция возвращает 1, когда массив отсортирован.

Параметры (int \*a, int size): первым параметром является указатель на начало массива, вторым – размер массива.

1. Линейный поиск

Данная функция выполняет линейный поиск числа в массиве.

Возвращает номер числа в массиве.

Параметры (int \*a, int size, int element): первым параметром является указатель на начало массива, вторым – размер массива, третьим – число для поиска.

1. Бинарный поиск

Данная функция выполняет бинарный поиск числа в массиве.

Возвращает номер числа в массиве.

Параметры (int \*a, int size, int element): первым параметром является указатель на начало массива, вторым – размер массива, третьим – число для поиска.

1. Перестановка двух элементов

Данная процедура меняет местами значения двух элементов массива.

Параметры (int \*x, int \*y):

*Описание использованных алгоритмов*

Ниже представлены алгоритмы некоторых функций и процедур.

* Генерация массива

Описание

Функция выделяет память для динамического массива выбранного размера и наполняет его случайными целыми числами.

Алгоритм

a = выделение памяти для size целочисленных элементов

Если память не выделена, то

Печать "Память не выделена. Попробуйте выбрать меньший размер массива"

вернуть 0

Всё если

Для i=0 до size

a[i] = случайное число

Всё для i

Вернуть a;

* Печать массива

Описание

Цикл по каждому элементу массива и его вывод на экран.

Алгоритм

Для i=0 до size

Печать a[i]

Всё для i

* Линейный поиск

Описание

Цикл по каждому элементу массива сравнение с искомым значением.

Алгоритм

Для i=0 до size

Если a[i] == element то вернуть i+1

Всё для i

вернуть -1 /\*Означает что при проходе циклом число так и не было найдено\*/

* Бинарный поиск

Описание

Отсортированный массив многократно делится на половины для поиска числа только правее или левее средних значений.

Алгоритм

low = 0

high = size - 1

Пока low <= high

middle = (low + high) / 2;

Если element < a[middle]

high = middle - 1

Иначе Если element > a[middle]

low = middle + 1

Иначе

Вернуть middle+1

Всё пока

Вернуть -1

# Заключение

В заключение, хочется отметить, что на данный момент существует много методов сортировки. Все они заслуживают внимания и исследования. В той или иной ситуации могут применяться не только разные алгоритмы, но и модификации одного и того же алгоритма сортировки.

Если память жёстко ограничена, можно использовать сортировки с квадратичной сложностью, как например сортировка выбором.

Если требуется выполнить сортировку максимально быстро любыми средствами, подойдет пирамидальная сортировка.

Или если код программы должен не слишком сильно усложняться, но при этом сортировка должна быть достаточной быстрой, лучше всего использовать сортировку Шелла и ей подобные.

В ходе работы были решены и достигнуты все поставленные задачи и цели.

# Литература

1. Аналитический портал «Гуманитарные технологии». Теория множеств – Режим доступа: https://gtmarket.ru/concepts/7073– Загл. с экрана.

# Приложение

#include <stdlib.h>

#include <stdio.h>

#include <locale.h>

void menu();

int createArray(int);

void printArray(int \*a, int size);

int heapSort(int \*, int);

void siftDown(int \*, int, int);

int ShellSort(int \*, int);

int selectionSort(int \*, int);

int search(int \*, int, int);

int binarySearch(int \*, int, int);

void swap(int \*x, int \*y);

int main()

{

setlocale(LC\_ALL, "Rus");

menu();

}

void menu()

{

int \*array;

int size;

int enable = 1;

int item, method, element, start, stop;

int sorted = 0;

int created = 0;

int index = -1;

while (enable)

{

printf("Введите номер действия:**\n**");

printf("**\t**1. Генерация нового массива**\n**");

printf("**\t**2. Печать всех элементов**\n**");

printf("**\t**3. Сортировка**\n**");

printf("**\t**4. Поиск элемента**\n**");

printf("**\t**5. Завершить работу**\n\n**");

scanf("%i", &item);

printf("**\n**");

if(created || (item == 1) || (item == 5))

{

switch (item)

{

case 1:

printf("Введите число элементов**\n**");

scanf("%i", &size);

free(array);

sorted = 0;

array = createArray(size);

if(array)

{

created = 1;

printf("Массив успешно создан**\n**");

}

**break**;

case 2:

printArray(array, size);

**break**;

case 3:

printf("Выберите метод сортировки:**\n**");

printf("**\t**1. Сортировка выбором**\n**");

printf("**\t**2. Сортировка Шелла**\n**");

printf("**\t**3. Пирамидальная сортировка**\n\n**");

scanf("%i", &method);

switch (method)

{

case 1:

start = clock();

sorted = selectionSort(array, size);

stop = clock();

printf("**\n**Затрачено: %i мс**\n**", stop - start);

**break**;

case 2:

start = clock();

sorted = ShellSort(array, size);

stop = clock();

printf("**\n**Затрачено: %i мс**\n**", stop - start);

**break**;

case 3:

start = clock();

sorted = heapSort(array, size);

stop = clock();

printf("**\n**Затрачено: %i мс**\n**", stop - start);

**break**;

default:

printf("**\n**Неправильный ввод.**\n\n**");

}

printf("**\n**");

**break**;

case 4:

printf("Введите значение для поиска: ");

scanf("%i", &element);

printf("**\n**");

if (sorted)

{

printf("Выберите метод поиска:**\n**");

printf("**\t**1. Линейный**\n**");

printf("**\t**2. Бинарный**\n\n**");

scanf("%i", &method);

printf("**\n**");

}

else method = 1;

switch (method)

{

case 1:

start = clock();

index = search(array, size, element);

stop = clock();

printf("**\n**Затрачено: %i мс**\n**", stop - start);

**break**;

case 2:

start = clock();

index = binarySearch(array, size, element);

stop = clock();

printf("**\n**Затрачено: %i мс**\n**", (stop - start));

**break**;

default:

printf("**\n**Неправильный ввод.**\n**");

}

if (index != -1)

{

printf("Искомый элемент находится под номером %i", index);

printf("**\n\n**");

}

else printf("Элемент не найден**\n**");

**break**;

case 5:

enable = 0;

printf("**\n**Программа завершила работу**\n**");

**break**;

default:

printf("**\n**Неправильный ввод.**\n\n**");

}

}

else printf("Неправильный ввод.**\n\n**");

}

free(array);

}

int createArray(int size)

{

int \*a = (int\*) malloc(size \* sizeof(int));

if (a == 0)

{

printf("Память не выделена. Попробуйте выбрать меньший размер массива**\n\n**");

return 0;

}

int i;

srand (time(NULL));

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

{

a[i] = rand();

}

return a;

}

void printArray(int \*a, int size)

{

int i;

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

{

printf("%i**\n**", a[i]);

}

}

void swap(int \*x, int \*y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

int selectionSort(int \*a, int size)

{

int i, j, min;

for(i = 0; i < size-1; i++)

{

min = i;

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

{

if(a[j] < a[min])

min = j;

}

swap(&a[i], &a[min]);

}

return 1;

}

int ShellSort(int \*a, int size)

{

int t, i, j, k;

for(k = size/2; k > 0; k /=2)

{

for(i = k; i < size; i++)

{

t = a[i];

for(j = i; j>=k; j-=k)

{

if(t < a[j-k])

a[j] = a[j-k];

else

**break**;

}

a[j] = t;

}

}

return 1;

}

void siftDown(int \*numbers, int root, int bottom)

{

int maxChild;

int done = 0;

while ((root \* 2 <= bottom) && (!done))

{

if (root \* 2 == bottom)

maxChild = root \* 2;

else if (numbers[root \* 2] > numbers[root \* 2 + 1])

maxChild = root \* 2;

else

maxChild = root \* 2 + 1;

if (numbers[root] < numbers[maxChild])

{

int temp = numbers[root];

numbers[root] = numbers[maxChild];

numbers[maxChild] = temp;

root = maxChild;

}

else

done = 1;

}

}

int heapSort(int \*numbers, int array\_size)

{

int i;

for (i = (array\_size / 2) - 1; i >= 0; i--)

siftDown(numbers, i, array\_size);

for (i = array\_size - 1; i >= 1; i--)

{

swap(&numbers[0], &numbers[i]);

siftDown(numbers, 0, i - 1);

}

return 1;

}

int search(int \*a, int size, int element)

{

int i;

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

{

if(a[i] == element) return i+1;

}

return -1;

}

int binarySearch(int \*a, int size, int element)

{

int low, high, middle;

low = 0;

high = size - 1;

while (low <= high)

{

middle = (low + high) / 2;

if (element < a[middle])

high = middle - 1;

else if (element > a[middle])

low = middle + 1;

else

return middle+1;

}

return -1;

}