МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное

образовательное учреждение

высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

**по дисциплине**

**“ Алгоритмы построение и анализ”**

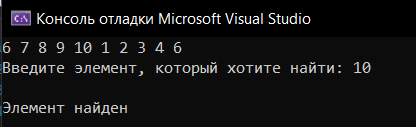
Выполнил: студент гр. ФИб-3302-51-00 ё \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ПМиИ Разова Е. В. \_\_\_\_\_\_\_\_\_\_\_

Киров 2020

***Задача 1. Линейный поиск с барьером***

Реализовать линейный поиск с барьером. Выполнить анализ временной сложности.



#include <iostream>

#include <clocale>

#include <cmath>

using namespace std;

#define e 0.005

double func(double x)

{

return pow(x, 2) - 2 \* x + 12;

}

void fisrt(double a, double b) {

double c;

int i = 0;

while (b - a > e)

{

i++;

c = (a + b) / 2;

if (func(b) \* func(c) <= 0)

a = c;

else

b = c;

}

cout << "Количество итераций = "<< i << endl;

cout << "x = " << ((a + b) / 2) << endl;

}

void second(double a, double b) {

double c;

int i = 0;

int aa = abs(func(a));

int bb = abs(func(b));

int s = 0;

bool once = true;

while (aa <= e || bb <= e || once==true)

{

once == false;

i++;

c = (a + b) / 2;

if (func(b) \* func(c) <= 0)

a = c;

else

b = c;

if (i > (bb+aa)/2) break;

}

cout << "Количество итераций = " << i << endl;

cout << "x = " << ((a + b) / 2) << endl;

}

void third(double a, double b) {

double c;

int i = 0;

while (b - a > 0.000000001)

{

i++;

c = (a + b) / 2;

if (func(b) \* func(c) <= 0)

a = c;

else

b = c;

}

cout << "Количество итераций = " << i << endl;

cout << "x = " << ((a + b) / 2) << endl;

}

int main()

{

setlocale(0, "");

double a = 1.3;

double b = 4;

int i = 0;

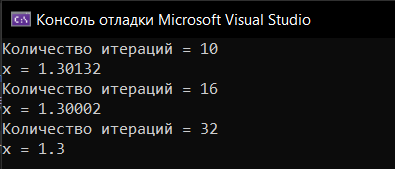
fisrt(a, b); //|a(k)-b(k)|≤e

second(a, b); //| a(k) | ≤e или | b(k) | ≤e

third(a, b); //|a(k)-b(k)|≤1e-12

}

***Задача 2. Метод половинного деления уточнения корня уравнения f(x)=0 на промежутке [a; b]***



#include <iostream>

#include <clocale>

#include <cmath>

using namespace std;

#define e 0.005

double func(double x)

{

return pow(x, 2) - 2 \* x + 12;

}

void fisrt(double a, double b) {

double c;

int i = 0;

while (b - a > e)

{

i++;

c = (a + b) / 2;

if (func(b) \* func(c) <= 0)

a = c;

else

b = c;

}

cout << "Количество итераций = "<< i << endl;

cout << "x = " << ((a + b) / 2) << endl;

}

void second(double a, double b) {

double c;

int i = 0;

int aa = abs(func(a));

int bb = abs(func(b));

int s = 0;

bool once = true;

while (aa <= e || bb <= e || once==true)

{

once == false;

i++;

c = (a + b) / 2;

if (func(b) \* func(c) <= 0)

a = c;

else

b = c;

if (i > (bb+aa)/2) break;

}

cout << "Количество итераций = " << i << endl;

cout << "x = " << ((a + b) / 2) << endl;

}

void third(double a, double b) {

double c;

int i = 0;

while (b - a > 0.000000001)

{

i++;

c = (a + b) / 2;

if (func(b) \* func(c) <= 0)

a = c;

else

b = c;

}

cout << "Количество итераций = " << i << endl;

cout << "x = " << ((a + b) / 2) << endl;

}

int main()

{

setlocale(0, "");

double a = 1.3;

double b = 4;

int i = 0;

fisrt(a, b); //|a(k)-b(k)|≤e

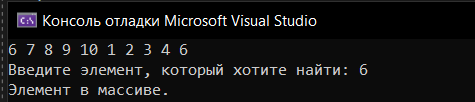
second(a, b); //| a(k) | ≤e или | b(k) | ≤e

third(a, b); //|a(k)-b(k)|≤1e-12

}

***Задача 3. Бинарный поиск в частично упорядоченном массиве***

Дан частично упорядоченный массив, который был получен из упорядоченного по не убыванию элементов массива путем перестановки двух его частей. Например, массив *M*[9, 10, 13, 2, 2, 5, 7, 8] был получен из массива *Х*[2, 2, 5, 7, 8, 9, 10, 13]. Реализовать алгоритм поиска элемента *А* в массиве *M*, имеющий оценку временной сложности О(2log2*n*).



#include <iostream>

#include <clocale>

using namespace std;

//Для начала ищем момент, после которого массив перестает быть упорядоченным, затем в зависимости от значения

//элемента выбираем правую или левую половину и проводим там бинарный поиск.

bool binarysearch(int arr[], int left, int m, int value);

int search1(int arr[], int left, int m);

void FindElement(int value, int arr[]);

bool binarysearch(int arr[], int left, int m, int value)

{

if (left > m)

return false;

else

{

int middle = (left + m) / 2;

if (arr[middle] == value) return true;

else if (arr[middle] > value) return binarysearch(arr, left, middle - 1, value);

else return binarysearch(arr, middle + 1, m, value);

}

}

int search1(int arr[], int left, int m)

{

if (left > m)

return -1;

else if (left == m)

return left;

else

{

int middle = (left + m) / 2 + (left + m) % 2;

if (arr[middle] >= arr[left])

return search1(arr, middle, m);

else

if (arr[middle] < arr[left])

return search1(arr, left, middle - 1);

else

return -1;

}

}

void FindElement(int value, int arr[])

{

int size = 10;

int border = search1(arr, 0, size - 1);

bool isSearch = false;

if (value > arr[size - 1])

isSearch = binarysearch(arr, 0, border, value);

else

isSearch = binarysearch(arr, border + 1, size - 1, value);

if (isSearch == true)

cout << "Элемент в массиве.";

else

cout << "Такого элемента нет";

}

int main()

{

setlocale(LC\_ALL, "Russian");

int \*arr = new int[10] { 6, 7, 8, 9, 10, 1, 2, 3, 4, 6 };

for (int i = 0; i < 10; i++)

cout << arr[i] << " ";

cout << "\nВведите элемент, который хотите найти: ";

int a;

cin >> a;

FindElement(a, arr);

}

***Задача 4. Поиск в двумерном массиве***

Дан двумерный массив *А* размерности *n*\**m*, каждая строка и каждый столбец которого упорядочены по не убыванию, и число *Х*. Определить, содержится ли число *Х* в массиве *А*

1. за время O(*n*2), используя последовательный просмотр всех элементов;
2. за время O(min(*n*, *m*)\*log2(max(*n*, *m*))), используя бинарный поиск;
3. за время O(*n*+*m*), используя линейный поиск.



#include <iostream>

using namespace std;

#define M 3

#define N 3

// бинарный для строки

bool binarySearch1D(int arr[], int K)

{

int low = 0;

int high = N - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == K)

return true;

if (arr[mid] < K)

low = mid + 1;

else

high = mid - 1;

}

return false;

}

bool binaryseach(int matrix[M][N], int K)

{

int low = 0;

int high = M - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

// если элемент находится в диапазоне этой строки, тогда вызовите 1-D двоичный поиск в этой строке

if (K >= matrix[mid][0]

&& K <= matrix[mid][N - 1])

return binarySearch1D(matrix[mid], K);

// если элемент меньше, чем начальный элемент этой строки, тогда поиск в верхних строках иначе поиск

// в нижних строках

if (K < matrix[mid][0])

high = mid - 1;

else

low = mid + 1;

}

return false;

}

bool bruteforce(int matrix[M][N], int K)

{

bool a1 = false;

for (int i = 0; i < M && !a1; i++)

for (int j = 0; j < N && !a1; j++)

if (matrix[i][j] == K)

a1 = true;

if (a1 == true)

return true;

else

return false;

}

//простая идея - удалять строку или столбец в каждом сравнении, пока не будет найден элемент.

//начинать поиск с правого верхнего угла матрицы.

bool linearsearch(int matrix[M][N], int x)

{

int smallest = matrix[0][0], largest = matrix[M - 1][N - 1];

if (x < smallest || x > largest)

return false;

// установить индексы для верхнего правого элемента

int i = 0, j = N - 1;

while (i < N && j >= 0)

{

if (matrix[i][j] == x)

{

return true;

}

if (matrix[i][j] > x)

j--;

// Проверьте, если mat [i] [j] <x

else

i++;

}

return 0;

}

int main()

{

int matrix[M][N] = { { 1, 3, 5},

{ 10, 11, 16},

{ 23, 30, 34 } };

int K = 3;//искомый элемент

//if (bruteforce(matrix, K)) //полный перебор

//if (linearsearch(matrix, K)) //линейный

if (binaryseach(matrix, K)) //бинарный

cout << "Found" << endl;

else

cout << "Not found" << endl;

}