**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Радиотехнических систем**

**отчет**

**по лабораторной работе №1**

**по дисциплине «Информатика»**

**Тема: Бинарный поиск**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3182 |  | Ефремов Я.И. |
| Преподаватель |  | Рамаданов В.С. |

Санкт-Петербург

2024

2. Цель работы: реализовать алгоритм бинарного поиска на языке C++, проверить работоспособность, составить его схему.

3. Краткие теоретические сведения:

Бинарный поиск – алгоритм по поиску некоторого значения в отсортированном массиве со сложностью алгоритма O(). Он быстрее линейного поиска, но требует, чтобы массив был отсортированным.

Принцип алгоритма простой: изначально берется 2 границы (верхняя и нижняя) и середина между ними сравнивается с искомым значением. Если искомое значение больше, то нижняя граница сдвигается вверх, благодаря чему и середина между границами сдвигается вверх. В случае, когда искомое значение меньше, чем середина, то верхняя граница сдвигается вниз, из-за чего и середина становится меньше. Эти действия продолжаются вплоть до того момента, когда значение среднего элемента (середины) совпадет с искомым значением.

Сложность реализации заключается в правильном построении условия цикла поиска, выхода из цикла при найденном значении. Необходимо учитывать случаи, когда искомого значения нет в массиве, когда оно находится на начальных границах (первый или последний по счету элементы), когда искомое значение совпадает с первой определенной серединой, когда расстояние между верхней и нижней границей составляет один элемент ("впритык", к примеру 8 и 9). Также стоит учесть, что в C++ результатом деления целого числа на целое тоже является целое (5/2 = 2).

4. Код программы:

В ТЗ сигнатура функции выглядела так: bool binary\_search(int\* arr, int N, int target);

Разработка велась в Qt Creator 4.7.2 (Enterprise), версия языка C++11, компилятор MinGW 32 bit.

Сам код программы:

#include <chrono>

#include <iostream>

using namespace std;

bool binary\_search(const int \*arr, int N, int target){

bool type = arr[0] < arr[N-1];

if (type && (target > arr[N-1] || target < arr[0] )) return 0;

else if(!type && (target > arr[0] || target < arr[N-1] )) return 0;

if(arr[N-1] == target) return 1;

if(arr[0] == target) return 1;

int upN = N-1, downN = 0;

N /= 2;

while(downN+1 < upN){

if(arr[N] == target) return 1;

if(arr[N] > target){

if(type){

upN = N;

N = downN + (N-downN)/2;

}

else{

downN = N;

N = downN + (upN - N) / 2;

}

}

else if (arr[N] < target){

if(type){

downN = N;

N = downN + (upN - N) / 2;

}

else{

upN = N;

N = downN + (N-downN)/2;

}

}

}

return 0;

}

int main(){

int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

int a1[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

cout<<"For -3 in array a: " <<binary\_search(a, 10, -3)<<endl;

cout<<"For 8 in array a1: " <<binary\_search(a1, 10, 8)<<endl;

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

cout<<"All elements searched in array a:" << binary\_search(a, 10, i) <<endl;

}

cout<<endl;

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

cout<<"All elements searched in array a1:" << binary\_search(a1, 10, i) <<endl;

}

cout<<endl;

int \*b = new int[100000];

for(int i = 0; i < 100000; ++i){

b[i] = i;

}

auto start = chrono::steady\_clock::now();

for(int i = 0; i < 10000; ++i){

if(!binary\_search(b, 10000, i)) cout <<"Error in calculations";

}

auto end = chrono::steady\_clock::now();

cout<<"Time:" << (end - start).count()/1000000000.0f << " seconds"<<endl;

start = chrono::steady\_clock::now();

for(int i = 0; i < 10000; ++i){

for(int j = 0; j < 10000; ++j){

if(b[j] == b[i]) break;

}

}

end = chrono::steady\_clock::now();

cout<<"Time:" << (end - start).count()/1000000000.0f << " seconds" <<endl;

start = chrono::steady\_clock::now();

for(int i = -1000000; i <0; ++i){

if(binary\_search(b, 10000, i)) cout <<"Error in calculations";

}

end = chrono::steady\_clock::now();

cout<<"Time:" << (end - start).count()/1000000000.0f << " seconds" <<endl;

return 0;

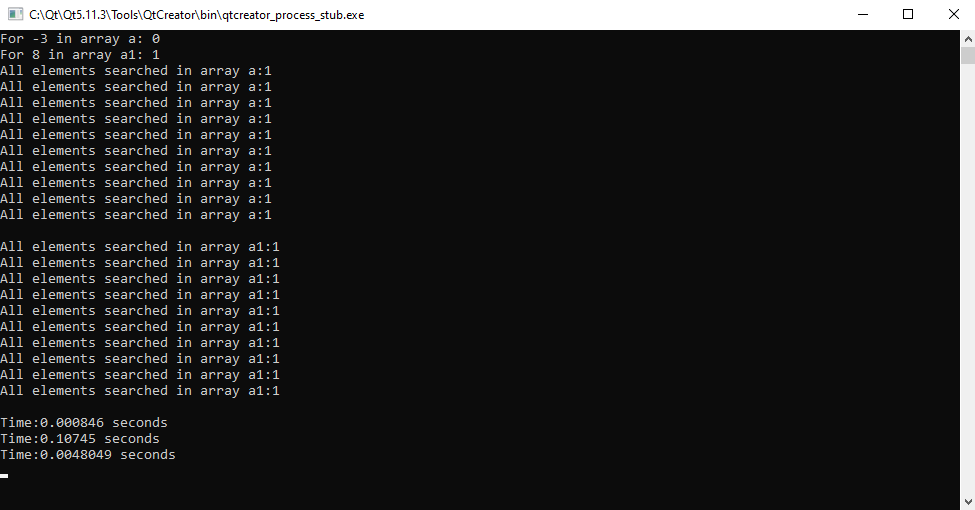
}

Примечания к коду: для замера времени использовалась библиотека std::chrono.

В данном случае функция binary\_seacrh определяет как отсортирован массив сверху вниз или снизу вверх и сохраняет этот тип в переменной type. В соответствии с этим идут две ветки, в каждой из которых прописан алгоритм для работы с данным типом массива (под типом подразумевается вид отсортированного массива – сверху вниз или снизу вверх).

В функции main созлаются несколько пробных массивов и на них тестируется работа функции main.

5. Вывод программы:



В 1 строке – результат работы функции binary\_search при вводе числа -3, которого нет в массиве

Во 2 строке – результат работы функции binary\_search при вводе числа 8, которое есть в массиве

Следующие 10 строк – результат работы функции при поиске каждого элемента из массива a[] (от 0 до 9). Если бы бинарный поиск что-то не нашел, то в одной или нескольких строк были бы нули.

Потом идет пустая строка.

Следующие 10 строк - Следующие 10 строк – результат работы функции при поиске каждого элемента из массива a1[] (от 9 до 0). В отличие от прошлого этот массив идет сверху вниз. Таким образом, мы проверили работу бинарного поиска на обоих видах массивов.

Дальше идет пустая строка.

Следующие 3 строки – время:

1 – суммарное время за которое бинарный поиск находит каждый элемент в массиве со значениями на промежутке [0;10000). Количество элементов массива: 10000

2 – суммарное время за которое линейный поиск находит каждый элемент в массиве, описанном выше.

3 – сколько требуется времени бинарному поиску, чтобы убедиться, что данных значений нет в массиве (все значения, при передаче в функцию как аргумент target, лежат по левую сторону от массива).

6. Схема алгоритма бинарного поиска:

В данном случае, не особого смысла рассматривать действия происходящие в функции main, ведь они являются тестовыми/отладочными. Если рассматривать все действия происходящие в функции binary\_search, то схема алгоритма получится слишком громоздкой и в большинстве своём неинформативной. Поэтому наиболее продуктивно написать схему алгоритма для бинарного поиска в массиве, отсортированном от меньшего к большему. Для начала выделим код, на основании которого будет написан алгоритм(должно быть наоборот):

bool binary\_search(const int \*arr, int N, int target){

bool type = arr[0] < arr[N-1];

if(target > arr[N-1] || target < arr[0]) return 0;

if(arr[0] == target) return 1;

if(arr[N-1] == target) return 1;

int upN = N-1, downN = 0;

N /= 2;

while(downN+1 < upN){

if(arr[N] == target) return 1;

if(arr[N] > target){

upN = N;

N = downN + (N-downN)/2;

}

else if (arr[N] < target){

downN = N;

N = downN + (upN - N) / 2;

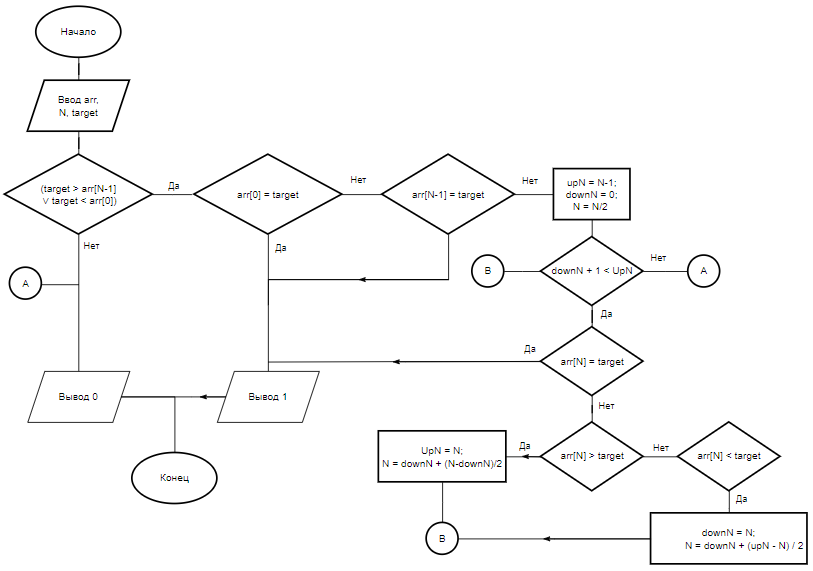
}

}

return 0;

}

Схема алгоритма:



7. Вывод: в ходе лабораторной работы я написал программу, в которой реализовал алгоритм бинарного поиска. Работоспособность программы была доказана тестами. В конце была составлена схема алгоритма бинарного поиска для отсортированного снизу вверх массива.