# Отчет №7. Алгоритмы поиска

100% программистов, во время обучения, рано или поздно столкнутся с необходимостью проверить наличие в массиве определённого значения.

Существует несколько общеизвестных алгоритмов поиска в языках программирования.

# Линейный поиск

### В ниже приведенном примере объявим массив на 50 элементов и заполним его используя [генератор случайных чисел rand()](https://purecodecpp.com/archives/1012). Предложим пользователю ввести искомое значение с клавиатуры и реализуем проверку на наличие этого

значения в нашем массиве. Если значение будет найдено в каком-либо

### элементе массива – выведем на экран индекс этого элемента. Это классический пример.

**Код:**

1. #include <iostream>
2. #include <iomanip>
3. #include <ctime>
4. using namespace std;

#### 5.

1. //прототипы функций
2. int linSearch(int arr[], int requiredKey, int size); // линейный поиск
3. void showArr(int arr[], int size); // показ массива

9.

#### 10. int main()

11. {

1. setlocale(LC\_ALL, "rus");
2. const int arrSize = 50;
3. int arr[arrSize];

#### int requiredKey = 0; // искомое значение (ключ)

1. int nElement = 0; // номер элемента массива
2. srand(time(NULL)); 18.
3. //запись случ. чисел в массив от 1 до 50

#### for (int i = 0; i < arrSize; i++)

21. {

22. arr[i] = 1 + rand() % 50;

23. } 24.

#### 25. showArr(arr, arrSize);

26.

1. cout << "Какое число необходимо искать? ";
2. cin >> requiredKey; // ввод искомого числа

#### //поиск искомого числа и запись номера элемента

1. nElement = linSearch(arr, requiredKey, arrSize); 32.

33. if (nElement != -1)

34. {

#### //если в массиве найдено искомое число - выводим индекс элемента на экран

1. cout << "Значение " << requiredKey << " находится в ячейке с индексом: " << nElement << endl;

37. }

38. else

39. {

#### //если в массиве не найдено искомое число

1. cout << "В массиве нет такого значения" << endl;

42. }

43. return 0;

44. }

#### 45.

1. //вывод массива на экран
2. void showArr(int arr[], int arrSize)

48. {

49. for (int i = 0; i < arrSize; i++)

#### 50. {

51. cout << setw(4) << arr[i]; 52. if ((i + 1) % 10 == 0)

53. {

54. cout << endl;

#### 55. }

56.

57. }

58. cout << endl << endl;

59. }

#### 60.

1. //линейный поиск
2. int linSearch(int arr[], int requiredKey, int arrSize)

63. {

64. for (int i = 0; i < arrSize; i++)

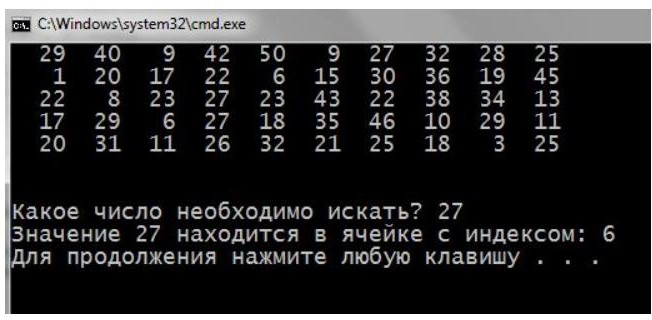
#### 65. {

1. if (arr[i] == requiredKey)
2. return i;

68. }

69. return -1;

**70. }.**



# Пример из лабораторной работы

template<typename T>

int search\_sequential(std::vector<T>& arr, T key) ///Последовательный поиск по ключу

{

for (int i = 0; i < arr.size(); i++)

{

if (arr[i] == key) //Просто сверяем с ключом

{

return i;

}

}

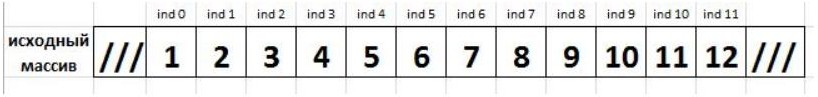
throw std::invalid\_argument("Не удалось найти элемент!\n");

# }

# Бинарный поиск

Двоичный (бинарный) поиск является более эффективным решением в случае, если массив заранее отсортирован.

Предположим, что массив из 12-ти элементов отсортирован по возрастанию:



Пользователь задает искомое значение (ключ поиска). Допустим 4. На

первой итерации [массив](https://purecodecpp.com/archives/1) делится на две части (ищем средний элемент – midd): (0 + 11) / 2 = 5 (0.5 отбрасываются). Сначала, проверяется значение среднего элемента массива. Если оно совпадает с ключом – алгоритм

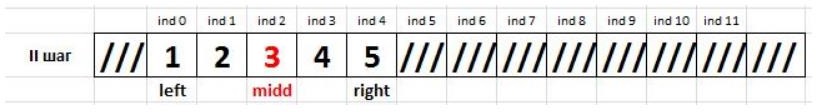
прекратит работу и программа выведет сообщение, что значение найдено. В нашем случае, ключ не совпадает со значением среднего элемента.



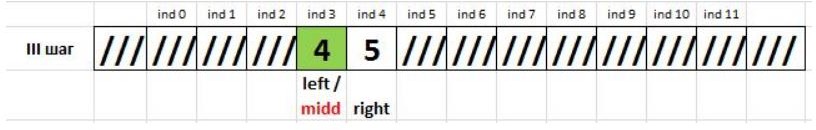
Если ключ меньше значения среднего элемента, алгоритм не будет

проводить поиск в той половине массива, которая содержит значения

больше ключа (т.е. от среднего элемента до конца массива). Правая граница поиска сместится (midd – 1). Далее снова деление массива на 2.



Ключ снова не равен среднему элементу. Он больше него. Теперь левая граница поиска сместится (midd + 1).



На третьей итерации средний элемент – это ячейка с индексом 3: (3 + 4) / 2 =

1. Он равен ключу. Алгоритм завершает работу.

**Код:**

1. C++

* 1. #include <iostream>
  2. using namespace std; 4.

#### // функция с алгоритмом двоичного поиска

1. int Search\_Binary (int arr[], int left, int right, int key) 7. {
2. int midd = 0;
3. while (1)

#### 10. {

11. midd = (left + right) / 2; 12.

1. if (key < arr[midd]) // если искомое меньше значения в ячейке
2. right = midd - 1; // смещаем правую границу поиска

#### else if (key > arr[midd]) // если искомое больше значения в ячейке

1. left = midd + 1; // смещаем левую границу поиска
2. else // иначе (значения равны)
3. return midd; // функция возвращает индекс ячейки

#### if (left > right) // если границы сомкнулись

1. return -1;

22. }

23. }

24.

#### 25. int main()

|  |  |  |
| --- | --- | --- |
| 26. | { |  |
| 27. |  | setlocale (LC\_ALL, "rus"); |
| 28. |  |  |
| 29. |  | const int SIZE = 12; |
| **30.** |  | **int array[SIZE] = {};** |
| 31. |  | int key = 0; |

32. int index = 0; // индекс ячейки с искомым значением

33.

34. for (int i = 0; i < SIZE; i++) // заполняем и показываем массив

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **35.** | **{** |  |  | |
| 36. |  | array[i] = i + 1; |
| 37. |  | cout << array[i] << " | "; |
| 38. | } |  |
| 39. |  |  |
| **40.** | **cout** | **<< "\n\nВведите любое число:** | **";** | |
| 41. | cin >> key; | | |  |
| 42. |  | | |  |
| 43. | index = Search\_Binary (array, 0, SIZE, key); | | |  |
| 44. |  | | |  |
| **45.** | **if (index >= 0)** | | |  |
| 46. | cout << "Указанное число находится в ячейке | | | с |
| индексом:  47. | " << index << "**\n\n**";  else | | |  |
| 48. | cout << "В массиве нет такого числа!**\n\n**"; | | |  |
| 49. |  | | |  |
| **50.** | **return 0;** | | |  |

# Пример из лабораторной работы

template<typename T>

int search\_binary(std::vector<T>& arr, T key) ///Бинарный поиск

{

int left = 0, right = arr.size() - 1;

int mid;

while (left < right)

{

mid = (left + right) / 2; //находим индекс центрального значения

//И смотрим какую границу двигать, в зависимости от того с какой стороны лежит искомое значение

if (arr[mid] >= key)

{

right = mid;

}

else

{

left = mid + 1;

}

}

if (arr[left] != key)

{

throw std::invalid\_argument("Не удалось найти нужный элемент, возможно массив был не отсортирован!\n");

}

return left;

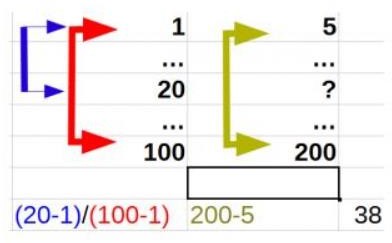
# }

# Интерполирующий поиск

**Интерполяция** – это определение области поиска, путем вычисления подобия расстояний между искомым значением и всей областью.

Вычисляется длина области поиска, и длина от начала области до некоего числа (скажем до центрального элемента в массиве). Вычисление это

проводится как с номерами элемента, так и с их значениями, после чего полученная длина области умножается на длину между значениями, и результат прибавленный к значению из начала области дает искомое.



Формула достаточно проста – вычисляется длина между номерами первого элемента и искомого (задаваемого точнее). Такая же длина считается между первым и последним номерами. Длины между собой делятся, как раз и

получая вычисление подобия. То же самое происходит со значениями

элементов – так же вычисляется расстояние между граничными значениями в массиве. Специально выделяю цветом понятия и связанные с ними части формулы.

Полученная длина номеров элементов массива умножается на длину значений в этих (граничащих) элементах и прибавляется значение в первой ячейке массива.

Получается: 1 + (20-1)/(100-1) \* (200-5) = 38 с копейками.

Результат и есть то самое искомое. Т.е. при таких значениях как на картинке в ячейке №20 будет стоять 38. Вот и весь смысл интерполяции – составления

подобия между номерами элементов и между значениями элементов.

**Код:**

1. терполирующий поиск в С++C++
2. #include <iostream>
3. using namespace std; 4.

#### 5. int main()

6. {

7. //Массив значений в котором пойдет поиск

8. int MyArray[] { 1, 2, 4, 6, 7, 89, 123, 231, 1000, 1235 };

9.

#### int x = 0; //Текущая позиция массива, с которым сравнивается искомое

1. int a = 0; //Левая граница области, где ведется поиск
2. int b = 9; //Правая граница области, где ведется поиск
3. int WhatFind = 123; //Значение, которое нужно найти

#### bool found; //Переменка-флаг, принимающая True если искомое найдено

1. */\*\*\*\*\*\*\*\*\*\*\*\* Начало интерполяции*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

1. //Цикл поиска по массиву, пока искомое не найдено

#### //или пределы поиска еще существуют

1. for (found = false; (MyArray[a] < WhatFind) && (MyArray[b] > WhatFind) && !found; )

22. {

1. //Вычисление интерполяцией следующего элемента, который будет сравниваться с искомым
2. x = a + ((WhatFind - MyArray[a]) \* (b - a)) / (MyArray[b] - MyArray[a]);

#### //Получение новых границ области, если искомое не найдено

|  |  |  |
| --- | --- | --- |
| 26. |  | if (MyArray[x] < WhatFind) |
| 27. |  | a = x + 1; |
| 28. |  | else if (MyArray[x] > WhatFind) |
| 29. |  | b = x - 1; |
| **30.** |  | **else** |
| 31. |  | found = true; |
| 32. | } |  |
| 33. |  |  |

34. */\*\*\*\*\*\*\*\*\*\*\*\*\*\* Конец интерполяции*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

#### 35.

1. //Если искомое найдено на границах области поиска, показать на какой границе оно
2. if (MyArray[a] == WhatFind)
3. cout << WhatFind << " founded in element " << a

<< endl;

1. else if (MyArray[b] == WhatFind)

#### cout << WhatFind << " founded in element " << b

**<< endl;**

1. else
2. cout << "Sorry. Not found" << endl; 43.

44. return 0;

**45. }**

## Источники:

**-** [**https://purecodecpp.com/algoritmy-v-c**](https://purecodecpp.com/algoritmy-v-c) (сайт работает только с VPN)