**Лабораторная работа №5**

**Тема**: Алгоритмы поиска элементов в массивах

**Цель**: Научиться реализовывать алгоритмы последовательного и бинарного поиска элементов в массивах

**последовательный поиск**

Пусть данный массив из N элементов. Необходимо определить номер элемента, имеет определенные свойства (чаще всего речь идет просто об элементе с определенным значением, равным k), либо установить что такого элемента в массиве нет.

Если массив является неупорядоченным, то единственный алгоритм поиска, можно применить в данном случае - последовательный. Суть метода - последовательно перебираются все элементы массива, если на каком-то этапе цикла оказывается, что массив закончился или оказывается искомый элемент, то цикл заканчивается.

***Один из вариантов псевдокода последовательного поиска:***

Алгоритм Последовательный поиск (массив а [1..N], искомый элемент k)

начало

введение а [1..N], k

i = 1

пока (i <= N) и (a [i] <> k) выполнять inc (i);

{После окончания цикла i равно номеру искомого элемента, а если элемент не найден, то i = N + 1}

конец

Использование цикла while вместо for позволяет сократить количество сравнений, если нужен элемент находится близко к началу массива. В этом случае, как только элемент будет найден, цикл прекратит свою работу.

**бинарный поиск**

Если массив упорядочен по росту или убыванию, то можно применить более быстрые методы поиска, такие как, например, бинарный (методом деления пополам).

Рассмотрим пример. Пусть данный массив из 9 элементов, упорядоченный по возрастанию их значений.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 8 | 9 | 16 | 22 | 28 | 36 |

Пусть требуется найти номер элемента, значение которого равно 1. Сравним это значение с первым элементом массива 1 <2. Из условия того, что массив упорядочен по росту следует, что элементы с большими номерами будут иметь большие значения, то есть другие числа в массиве гарантированно больше чем 2. Отсюда можно сделать вывод, что числа 1 в массиве нет.

Пусть требуется найти элемент со значением 55. Сравним это число с первым элементом. 55> 2. Следовательно, можно предположить, что в массиве число 55 встречается. Теперь сравним это число с последним элементом массива (из условия упорядоченности массива, его последний элемент является самым большим, а все остальные элементы соответственно меньше его). 55> 36. Итак, в массиве числа 55 быть не может.

Пусть требуется найти элемент со значением 22. Сравним это число с крайними элементами массива, и убедимся, что такое число может присутствовать среди его элементов.

Возьмем элемент, находящийся в середине массива (если число элементов четное, то в какую сторону будет выполняться округления - неважно).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 8 | 9 | 16 | 22 | 28 | 36 |

Элемент, находящийся в середине массива не тот, который нам нужен. (9 <> 22). Однако, этот элемент делит массив на две части: одну в которой искомого числа точно быть не может (в данном примере это левая половина массива, с условия упорядоченности все элементы, расположенные слева от 9 будут меньше этого числа) и вторую, в которой можно продолжать поиск.

Рассмотрим теперь правую часть массива. Найдем элемент, находящийся в ее середине.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 8 | 9 | 16 | 22 | 28 | 36 |

Это значение равно искомому, процесс поиска закончен.

Пусть требуется найти элемент со значением 7. Сравним это число с крайними элементами массива и убедимся что поиск имеет смысл.

Рассмотрим элемент, находящийся в середине массива.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 8 | 9 | 16 | 22 | 28 | 36 |

Этот элемент не равен искомому, следовательно, поиск необходимо продолжить. Потому что 9> 7, следовательно, правую половину массива можно исключить из рассмотрения, и продолжить поиск только в левой. Рассмотрим элемент, находящийся в середине левой половины:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 8 | 9 | 16 | 22 | 28 | 36 |

Этот элемент снова не равна 7, следовательно поиск надо продолжить. Из условия упорядоченности массива, поиск будем продолжать в правой части нашего «укороченного» массива.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 8 | 9 | 16 | 22 | 28 | 36 |

Рассмотрим элемент, находящийся в середине выделенной части массива.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 4 | 5 | 8 | 9 | 16 | 22 | 28 | 36 |

Этот элемент не равен искомому. Однако, в результате постоянного уменьшения область поиска в массиве уменьшилась до трех элементов, и каждый из них был рассмотрен на каком-то шаге. Итак, процесс поиска закончен, результат - сообщение о том, что элемент с таким значением в массиве отсутствует.

На этих примерах видно, что с каждым шагом размер области поиска в массиве уменьшается в два раза, что приводит к сокращению числа операций. Например, за 10 шагов размер области поиска будет уменьшен в 1024 раза, а для массива из 1000 000 элементов понадобится всего 20 шагов. Количество операций в данном методе равна O (log n).

Пример псевдокода бинарного поиска может быть следующим:

Алгоритм Бинарный поиска (массив а [1..N], искомый элемент k)

начало

введение а [1..N], k

  left: = 1;

  right: = N;

  // Сразу проверим, не является искомый элемент крайним

если a [1] = k тогда вививедення «1» // 1 - номер искомого элемента

иначе если a [N] = k тогда вививедення N // N номер искомого элемента

// Сравним с крайними, решим вопрос: а стоит ли вообще искать?

иначе если k <a [left] тогда вививедення «Искомого числа нет"

иначе если k> a [right] тогда вививедення «Искомого числа нет"

// Цикл поиска: найдем номер в середине текущей области поиска

иначе

          начало

            повторять

              c = (left + right) div 2

              если a [c] <k тогда left = c

              если a [c]> k тогда right = c

            условие выхода из цикла (a [c] = k) или (right-left <= 1); // Пока элемент не найден, или пока область поиска не стала состоять всего из двух соседних элементов

            если a [c] = k тогда вывода с // в с - номер искомого элемента

            иначе «Искомого числа нет"

    конец

конец

Здесь left и rght - номера элементов массива, которые являются границами текущей области поиска. Например, если left = 3 и right = 7 это значит что поиск ведется среди элементов с номерами от 3 до 7 и именно в этой области на данном этапе будет искаться середина.

**Алгоритм прямого (последовательного) поиска подстроки**

Обозначим строка, в которой ведется поиск как массив элементов Т, искомый подстрока - массив элементов W, N – длин

а строки, М - длина подстроки.

**Идея алгоритма:**

1) i = 1;

2) сравнить и-и символ массива T с первым символом массива W;

3) совпадение → сравнить следующий символ массива Т со вторым символом массива W и так далее;

4) расхождение → и: = i + 1 и переход на пункт 2.

**Условие окончания алгоритма:**

1) подряд М сравнений удачные;

2) и + M> N, то есть слово не найдено.

Пример псевдокода прямого поиска:

Алгоритм Прямой поиск (арг. S: строка; арг. X: строка; арг. Place: целое число, рез. Res: логический тип)

// Функция возвращает результат поиска в строке S подстроки X

// Place - место первого вхождения

 начало

   Res = FALSE

   i = 1

   пока (i <= длина (S) -длина (X) +1) и (Res = FALSE) выполнять

        если S [i] = X [i] то

        начало

         ... // Цикл сравнения символов в подстроке и строке

         если все символы совпали то

        начало

         Res = TRUE

Place = i

       конец

     иначе i = i + 1

        Вывод результатах поиска // если Res = TRUE - подстрока найден и в Place индекс первого символа пидрадка в строке, если Res = FALSE - подстрока не найдена

  конец

Задача 1:

1. Реализовать алгоритм последовательного поиска элемента в массиве.

2. Реализовать алгоритм бинарного поиска элемента в массиве.

3. Сравнить их быстродействие.

Задача 2:

1. Реализовать алгоритм последовательного поиска подстроки.

Задача 3 (дополнительное): Реализовать одно из следующих задач:

1. Есть информация о наличии мест в вагоне поезда (использовать массив, например, с 0 и 1). Вывести на экран номера свободных нижних мест (нижние места должны нечетные номера). И выяснить, можно ли приобрести билеты на два рядом расположены места (размещение в одной куче учитывать не обязательно).

2. "Лотерея". В лотерейном билете есть 15 чисел от 1 до 90. Они упорядочены по росту. Ведущий достает слой с некоторым случайным числом. Определить, является ли это число в билете?

3. Вводиться последовательность из n натуральных чисел. Необходимо определить наименьшее натуральное число, отсутствует в последовательности.

**Лабораторна робота №5**

**Тема: Алгоритми пошуку елементів у масивах**

**Мета: Навчитися реалізовувати алгоритми послідовного та бінарного пошуку елементів в масивах**

**Завдання 1:**

1. Реалізувати алгоритм послідовного пошуку елемента в масиві.
2. Реалізувати алгоритм бінарного пошуку елемента в масиві.
3. Порівняти їх швидкодію.

**Завдання 2:**

1. Реалізувати алгоритм послідовного пошуку підрядка.

**Завдання 3 (додаткове):** Реалізувати одне з наведених нижче завдань:

1. Є інформація про наявність місць у вагоні потяга (використати масив, наприклад, з 0 і 1). Вивести на екран номери вільних нижніх місць (нижні місця мають непарні номери). Та з'ясувати, чи можна придбати квитки на два поруч розташовані місця (розміщення в одній купі враховувати не обов'язково).

**2. "Лотерея". У лотерейному квитку є 15 чисел від 1 до 90. Вони впорядковані по зростанню. Ведучий дістає шар з деяким випадковим числом. Визначити, чи є це число у квитку?**

3. Вводитися послідовність із n натуральних чисел. Необхідно визначити найменше натуральне число, відсутнє в послідовності.

Задача № 1\_1 Реализовать алгоритм последовательного поиска элемента в массиве.

// последовательный поиск не отсортированного массива

int Linear\_Searh(int \_array[], int \_array\_size, int \_user\_value)

{

*for*(int i = 0; i < \_array\_size; i++) {

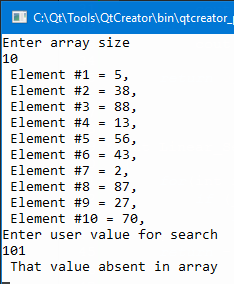
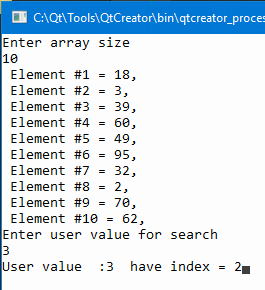
*if* (\_array[i] == \_user\_value)

*return* i;

}

*return* -1;

}



Задача № 1\_2 Реализовать алгоритм бинарного поиска элемента в массиве.

//Бинарный поиск отсортированного массива

int Binary\_Search(int \_array[], int \_array\_size, int \_user\_value)

{

int low = 0; *//* *начальное* *число*

int high = \_array\_size - 1; *//* *конечное* *число*

int middle = 0; *//* *среднее* *значение*

*while* (low <= high) *//* *пока* *число* *начальное* *меньше* *или* *равно* *конечному*

{ middle = (low + high) / 2; *//* *нахождение* *середины*

*if*(\_user\_value == \_array[middle]) *//* *если* *польз.* *число* *равно* *значение* *середины*

*return* middle; *//* *вернуть* *средний* *индекс*

*else* *if* (\_user\_value > \_array[middle]) *//* *если* *польз.* *число* *больше* *значения* *середины*

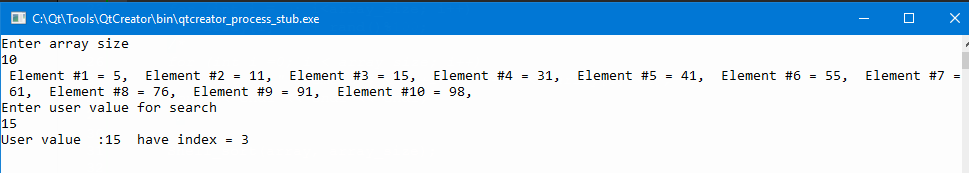
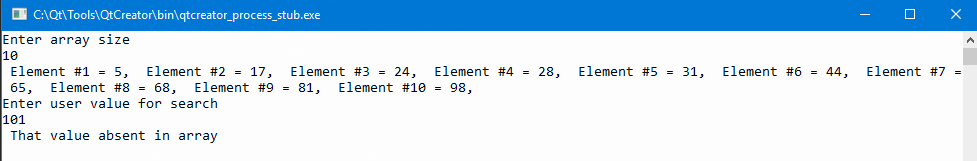
low = middle + 1; *//* *младшему* *присвоить*

*else*

high = middle - 1;

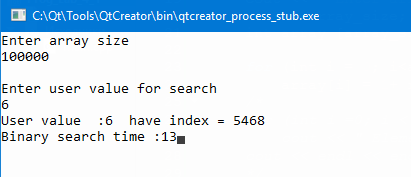
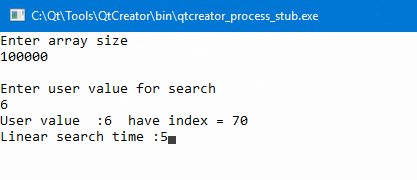
}

*return* -1;

}

Задача № 1\_3 Сравнить их быстродействие.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Array size | Linear search  time | Binary search  time |  |
| 1 | 1000 | 8 | 4 |  |
| 2 | 10000 | 5 | 4 |  |
| 3 | 30000 | 6 | 5 |  |
| 4 | 50000 | 6 | 4 |  |
| 5 | 100000 | 8 | 9 |  |
| average value | | 6.6 | 5.2 |  |



// Программа сортировки и последовательного поиска

// main.cpp

#include <iostream>

#include <ctime>

using namespace std;

int Binary\_Search(int \_array[], int \_array\_size, int \_user\_value);

int Linear\_Searh(int \_array[], int \_array\_size, int \_user\_value);

void Shell\_sort(int \_array[], int \_array\_size);

time\_t \_time = 0;

int main()

{

int array\_size; // размер массива

int userValue; // пользовательское число

int result; // результат

srand(time(0)); // автоматическая рандомизация

cout << "Enter of array size = ";

cin >> array\_size; // получение размера массива

int array[array\_size]; // массив

// заполнение случайными числами

for (int i = 0; i< array\_size; i++)

array[i] = 1+ rand()%100;

// отображение не отсортированного массива

// for (int i = 0; i< array\_size; i++)

// cout << " Array # " << i+1 << " = " << array[i] << endl;

// сортировка Шелла

Shell\_sort(array,array\_size);

// отображение отсортированного массива

// for (int i = 0; i< array\_size; i++)

// cout << " Array # " << i+1 << " = " << array[i] << endl;

// получение пользовательского числа для поиска

cout << "\nEnter an integer: " << endl;

cin >> userValue;

// вызов функции последовательного поиска

result = Binary\_Search(array, array\_size, userValue);

//result = Linear\_Searh(array, array\_size, userValue);

if(result >= 0) {

cout << "The number " << array[result] << " was found at the"

" element with index " << result+1 << endl;

cout << "Time to search" << \_time/ CLOCKS\_PER\_SEC << endl;

}

else

cout << "The number \"" << userValue << " \" was not found. " << endl;

return 0;

}

Задача 2: Реализовать алгоритм последовательного поиска подстроки.

int LinearString\_Search(char\* array\_main, char\* array\_user)

{

int i = 0; *//* *итератор* *для* *главной* *строки*

int j = 0; *//* *итератор* *для* *польз.* *строки*

int length = strlen(array\_user); *//* *длина* *строки* *для* *условия* *выхода*

*do* {

*if* (array\_main[i] == array\_user[j]) *//* *если* *равны*

{

i++; j++; *//* *то* *инкремент* *на* *следущий* *символ*

}

*else*

{

i++; j = 0; *//* *иначе* *инкремент* *главной* *строки*

}

}*while* (length != j); *//* *пока* *длина* *не* *равна* *итератору* *польз.* *строки*

*return* (i+1) - j ;

}

#include <iostream>

#include <cstring>

*using* *namespace* std;

int LinearString\_Search(char\* array\_main, char\* array\_user);

int main()

{

cout << "\*\* Enter the main string = ";

char main\_string[100];

cin >> main\_string;

cout << "\*\* Enter the user string = ";

char user\_string[100];

cin >> user\_string;

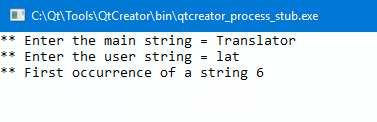
int result = 0;

result = LinearString\_Search(main\_string, user\_string);

cout << "\*\* First occurrence of a string " << result << endl; ;

*return* 0;

}



Задача 3: (дополнительное): Реализовать одно из следующих задач:

**2. "Лотерея". В лотерейном билете есть 15 чисел от 1 до 90. Они упорядочены по возрастанию. Ведущий достает некоторое случайное число. Определить, находится ли это число в билете?**

*//* *Программа* *поиска* *в* *лотерейном* *билете*

*//* *случайно* *вытянутого* *числа*

#include <iostream>

#include <ctime>

*using* *namespace* std;

void Shell\_sort(int \_array[], int \_array\_size);

int Binary\_Search(int \_array[], int \_array\_size, int \_user\_value);

int main()

{

srand(time(0)); *//* *автоматическая* *рандомизация*

int array\_size = 15;

int lottery\_ticket[array\_size]; *//* *лоттерейный* *билет*

int result; *//* *результат* *поиска*

int random\_value = 1+rand()%90; *//* *случайное* *число*

*//* *автозаполнение* *билета* *числами*

*for* (int i =0; i < array\_size; i++)

lottery\_ticket[i] = 1+ rand()%90;

*//* *алгоритм* *сортировки*

Shell\_sort(lottery\_ticket, array\_size);

*//* *Просмотр* *нашего* *билета*

*for*(int i=0; i< array\_size; i++)

cout << "The "<< i+1<< " position in our ticket are = "

<< " | " <<lottery\_ticket[i] << " | " << endl;

*//* *алгоритм* *бинарного* *поиска*

result = Binary\_Search(lottery\_ticket, array\_size, random\_value);

*//* *результат*

*if*(result >= 0)

cout << "The random value " << lottery\_ticket[result]

<< " was found at the lottery ticket on the \"" << result+1

<< "\" postion"<< endl;

*else*

cout << "The random value \"" << random\_value

<< " \" was not found in our ticket. " << endl;

*return* 0;

}

