СРАВНЕНИЕ МЕТОДА ПОИСКА

Дадонов Александр Дмитриевич

студент, Липецкий государственный технический университет, РФ, г. Липецк

Гаев Леонид Витальевич

к.т.н., доцент, Липецкий государственный технический университет РФ, г. Липецк

Аннотация. Был проведён анализ и исследование двух алгоритмов поиска, блочного и бинарного. С помощью программ и замера процессорного времени подсчитана эффективность каждого.

Ключевые слова: блочный поиск, двоичный поиск, эффективность, анализ.

Поиск – процесс нахождения конкретной информации в ранее созданном множестве данных. Обычно данные представляют собой записи, каждая из которых имеет хотя бы один ключ.

Ключ поиска – это поле записи, по значению которого происходит поиск.

Ключи используются для отличия одних записей от других.

Программа с блочным поиском:	Программа с двоичным поиском:

```cpp
#include <iostream>
#include<thread>
using namespace std;

const int N = 100;
int block_searh(int V[], int number, int size) {
    int count = 0;
    int k = sqrt(size);//количество
    int resIndex = -1;
    for (int i = k; i < size; i += k) {
        if (V[i] > number) {
            for (int j = i; j > i - k; j -= 1) {
                count++;
                if (V[j] == number) {
                    resIndex = 0;
                    break;
                }
                if (V[j] < number) {
                    break;
                }
            }
            break;
        }
    }
    if (resIndex == 0) cout << "Элемент найден!" << endl;
    else cout << "Элемент не найден!" << endl;

    cout << "Среднее сравнение :" << ceil(sqrt(N)) << " " << count << endl;
    return 0;
}

int main() {

    setlocale(LC_ALL, "Russian");

    int s[N];
    for (int i = 0; i < N; i++) {
```

```cpp
 #include <iostream>
#include<thread>
using namespace std;
const int N = 10000;

int bin_searh(int V[], int number, int size) {
    int count = 0;
    int left = 1;
    int right = size;
    int resIndex = -1;
    while (right != left) {
        int mid = (right + left) / 2;
        if (number == V[mid]) {
            resIndex = mid;
            count++;
            break;
        }
        else if (number > V[mid]) {
            left = mid;
            count++;

        }
        else {
            right = mid;
            count++;
        }
    }
    if (V[right] == number) {
        resIndex = right;
        count++;
    }
    if (resIndex != -1) cout << "Элемент найден!" << endl;
    else cout << "Элемент не найден!" << endl;

    cout << "Среднее сравнение :" << count << endl;
    return 0;
}
```

```
        s[i] = i;
    }
    auto start =
chrono::high_resolution_clock::now();
    block_searh(s, 8, N);
    auto end =
chrono::high_resolution_clock::now();

    chrono::duration<float> duration = end -
start;

    cout << "Время работы программы: " <<
duration.count();
}
```

```
int main() {

    setlocale(LC_ALL, "Russian");

    int s[N];
    for (int i = 0; i < N; i++)
        s[i] = i;
    auto start =
chrono::high_resolution_clock::now();
    bin_searh(s, 88, N);
    auto end =
chrono::high_resolution_clock::now();

    chrono::duration<float> duration = end
- start;

    cout << "Время работы программы: " <<
duration.count();
}
```

Работа **блочного поиска** заключается в том, что массив делится на блоки, наилучший вариант количество блоков – $\sqrt{N}$, а наихудший – $2\sqrt{N}$.

I этап: Аргумент поиска сравнивается с последними элементами каждого из блоков, если не будет получено значение, то результат поиска будет отрицательным и программа будет приступать ко II этапу.

II этап: Осуществляется последовательный поиск в найденном блоке в направлении от последней к первой записи в блоке.

Чтобы мы могли использовать бинарный или двоичный поиск, для начала мы должны отсортировать сам массив.

Рассмотрим **двоичный(бинарный) поиск**, двоичный поиск делит массив на две части, затем аргумент поиска сравнивается со средним элементом массива, если значение аргумента поиска меньше среднего элемента, то мы рассматриваем все элементы массива от 0 до N/2, где N – это количество элементов массива, а если аргумента поиска больше текущего элемента, то рассматривается элементы массива от N/2 до N. Таким образом, осуществляется последовательное деление пополам интервала поиска до тех пор, пока не будет найден искомый элемент или длина последовательности не станет равной единице.

Работа двух программ заключается в нахождении данного элемента, в массиве с размерностью: 100, 1000, 10000 и 100000.

А также подсчёта времени с помощью библиотеки **thread** и средних количества сравнений для двоичного поиска – $\log_2(N-1)$, а для блочного поиска – $\sqrt{N}$.

Таблица 1.

Результаты со средним количество сравнений.

		Блочный поиск	Двоичный поиск
№	**Кол-во элементов**	**Среднее кол-во сравнений**	**Среднее кол-во сравне**
1	100	10	7
2	1000	32	10
3	10000	100	14
4	100000	317	17

Таблица 2.

Теперь рассмотрим опыт с фактическими сравнениями

		Блочный поиск		Двоичный поиск	
№	**Кол-во элементов**	**Фактическое кол-во сравнений**	**Время** 10^{-8}	**Фактическое кол-во сравнений**	**Вр** **10**
1	100	9	11346	6	
2	1000	31	11844	9	
3	10000	139	14377	12	
4	100000	277	14793	16	

Вывод:

По результатам эксперимента можно сделать вывод, что для поиска случайного значения в отсортированном массиве, более эффективный — это двоичный(бинарный) поиск.

Список литературы:

1. Кнут, Д. Э. Искусство программирования, том 3. Сортировка и поиск / Д. Э. Кнут. – М.: "Вильямс", 2012. – 824 с.

2. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл. – М.: Техносфера, 2004. – 368 с.

3. Ахо, А. В. Структуры данных и алгоритмы / А. В. Ахо, Дж. Хопкрофт, Дж. Д. Ульман. – М.: "Вильямс", 2010. – 400 с