# Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Курский государственный университет»

Кафедра программного обеспечения и администрирования информационных систем  
Направление подготовки математическое обеспечение и администрирование информационных систем  
Форма обучения очная

Отчет  
по лабораторной работе №3  
«Сравнительный анализ алгоритма поиска»

Выполнил:  
студент группы 213 Файтельсон А.А.  
  
Проверил:  
ассистент кафедры ПОиАИС Овсянников А.В.

Курск, 2024

# Цель работы

Изучение алгоритмов поиска элемента в массиве и закрепление навыков в проведении сравнительного анализа алгоритмов.

# Листинг программы

package main  
import (  
 "errors"  
 "fmt"  
 "math/rand"  
 "slices"  
 "testing"  
 "time"  
)  
func generateRandomArray(N int) []int {  
 rand.Seed(time.Now().UnixNano())  
 array := make([]int, N)  
 for i := 0; i < N; i++ {  
 array[i] = rand.Intn(10000000000)  
 }  
 return array  
}  
  
func NaiveSearch(v int, slice []int) (int, error) {  
 for i, value := range slice {  
 if value == v {  
 return i, nil  
 }  
 }  
 return 0, errors.New("element not found")  
}  
  
func FastNaiveSearch(v int, slice []int) (int, error) {  
 n := len(slice)  
 slice = append(slice, v)  
 i := 0  
 for slice[i] != v {  
 i++  
 }  
 if i == n {  
 return 0, errors.New("element not found")  
 }  
 return i, nil  
}  
  
func BlockSearch(v int, slice []int) (int, error) {  
 if len(slice) == 0 {  
 return -1, errors.New("element not found")  
 }  
 medium := len(slice) / 2  
 if slice[medium] == v {  
 return medium, nil  
 }  
 if slice[medium] > v {  
 result, err := BlockSearch(v, slice[:medium])  
 if err != nil {  
 return -1, err  
 }  
 return result, nil  
 } else {  
 result, err := BlockSearch(v, slice[medium+1:])  
 if err != nil {  
 return -1, err  
 }  
 return medium + 1 + result, nil  
 }  
}  
  
func BinarySearch(v int, slice []int) (int, error) {  
 l, r := 0, len(slice)  
 for l < r {  
 mid := (l + r) / 2  
 if slice[mid] == v {  
 return mid, nil  
 }  
 if slice[mid] > v {  
 r = mid  
 } else {  
 l = mid + 1  
 }  
 }  
 return -1, errors.New("element not found")  
}  
  
func BenchmarkNaiveSearch(b \*testing.B) {  
 for \_, t := range SizeOfCases {  
 b.Run(fmt.Sprintf("size of case: %d", t), func(b \*testing.B) {  
 b.StopTimer()  
 value := 355  
 arr := generateRandomArray(t)  
 b.StartTimer()  
 for i := 0; i < b.N; i++ {  
 NaiveSearch(value, arr)  
 }  
 })  
 }  
}

# Результаты сравнительного анализа алгоритмов

(в микросекундах)

Таблица 1 - Максимальное количество операций сравнения

Таблица 2 - Среднее количество операций сравнения

# Вывод по работе

Для решения задачи поиска элемента в отсортированном массиве нужно использовать бинарный поиск, а в остальных случаях обычный поиск работает приемлемо.