Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №4 по теме: «Сравнительный анализ алгоритмов поиска»

Выполнил:  
студент группы ПВ-21  
Адаменко И. И.

Проверил:  
профессор  
Синюк В. Г.

Белгород  
2013

Цель работы: изучение алгоритмов поиска элемента в массиве и закрепление навыков в проведении сравнительного анализа алгоритмов.

## Задания

1. Изучить алгоритмы поиска:
   1. в неупорядоченном массиве:
      1. линейный;
      2. быстрый линейный;
   2. в упорядоченном массиве:
      1. быстрый линейный;
      2. бинарный;
      3. блочный.
2. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов поиска.
3. Провести эксперименты по определению временных характеристик алгоритмов поиска. Результаты экспериментов представить в виде двух таблиц. Клетки первой содержат максимальное количество операций сравнения при выполнении алгоритма поиска, второй — среднее число операций сравнения.
4. Построить графики зависимости количества операций сравнения от количества элементов в массиве.
5. Определить аналитическое выражение функции зависимости количества операций сравнения от количества элементов в массиве.
6. Определить порядок функции временной сложности алгоритмов поиска

## Листинг программы

### searches.h

|  |  |
| --- | --- |
| 1 | */\** |
| 2 | *Header for all functions of search* |
| 3 | *\*/* |
| 4 |  |
| 5 | **int** linearSearch(**int** arr[], **int** left, **int** right, **int** element, **int** \*counter); |
| 6 | **int** quickLinearSearch(**int** arr[], **int** left, **int** right, **int** element,   **int** \*counter); |
| 7 | **int** quickLinearSearchSorted(**int** arr[], **int** left, **int** right, **int** element,  **int** \*counter); |
| 8 | **int** binarySearch(**int** arr[], **int** left, **int** right, **int** element, **int** \*counter); |
| 9 | **int** mBlockSearch(**int** arr[], **int** left, **int** right, **int** element, **int** \*counter); |

### searches.cpp

|  |  |
| --- | --- |
| 001 | */\** |
| 002 | *Functions of search* |
| 003 | *\*/* |
| 004 |  |
| 005 | **#include** <stdlib.h> |
| 006 | **#include** <math.h> |
| 007 | **#include** <string.h> |
| 008 | **#include** <iostream> |
| 009 |  |
| 010 | **int** linearSearch(**int** arr[], **int** left, **int** right, **int** element, **int** \*counter) { |
| 011 |  |
| 012 | **int** i = left; |
| 013 |  |
| 014 | **while** ((i < right) && arr[i++] != element) (\*counter) += 2; |
| 015 | i--; |
| 016 | (\*counter)++; |
| 017 |  |
| 018 | **return** (arr[i] == element) ? i : -1; |
| 019 | } |
| 020 |  |
| 021 | **int** quickLinearSearch(**int** arr[], **int** left, **int** right, **int** element,  **int** \*counter) { |
| 022 |  |
| 023 | **int** i = left; |
| 024 |  |
| 025 | arr[right++] = element; |
| 026 | **while** (arr[i++] != element) (\*counter)++; |
| 027 | (\*counter)++; |
| 028 |  |
| 029 | **return** (i == right) ? -1 : i - 1; |
| 030 | } |
| 031 |  |
| 032 | **int** quickLinearSearchSorted(**int** arr[], **int** left, **int** right, **int** element,  **int** \*counter) { |
| 033 |  |
| 034 | **int** i = left; |
| 035 |  |
| 036 | **if** (arr[left] > element) |
| 037 | **return** -1; |
| 038 |  |
| 039 | arr[right++] = element; |
| 040 | **while** (arr[i++] < element) (\*counter)++; |
| 041 | (\*counter)++; |
| 042 |  |
| 043 | **return** (i == right) ? -1 : i - 1; |
| 044 | } |
| 045 |  |
| 046 | **int** binarySearch(**int** arr[], **int** left, **int** right, **int** element,  **int** \*counter) { |
| 047 |  |
| 048 | **int** mid = 0; |
| 049 |  |
| 050 | **while** (left < right) { |
| 051 | (\*counter)++; |
| 052 | mid = left + (right - left) / 2; |
| 053 |  |
| 054 | (\*counter)++; |
| 055 | **if** (arr[left] == element) |
| 056 | **return** left; |
| 057 |  |
| 058 | (\*counter)++; |
| 059 | **if** (arr[mid] == element) { |
| 060 | (\*counter)++; |
| 061 | **if** (mid == left + 1) |
| 062 | **return** mid; |
| 063 | else |
| 064 | right = mid + 1; |
| 065 | } |
| 066 | **else** **if** (arr[mid] > element) { |
| 067 | (\*counter)++; |
| 068 | right = mid; |
| 069 | } |
| 070 | **else** { |
| 071 | (\*counter)++; |
| 072 | left = mid + 1; |
| 073 | } |
| 074 | } |
| 075 |  |
| 076 | **return** -1; |
| 077 | } |
| 078 |  |
| 079 | **int** chooseSearch(**int** arr[], **int** left, **int** right, **int** element, **int** number,  **int** \*counter) { |
| 080 | (\*counter)++; |
| 081 | **if** (number > 15) |
| 082 | **return** binarySearch(arr, left, right, element, counter); |
| 083 | **else** |
| 084 | **return** linearSearch(arr, left, right, element, counter); |
| 085 | } |
| 086 |  |
| 087 | **int** mBlockSearch(**int** arr[], **int** left, **int** right, **int** element,  **int** \*counter) { |
| 088 |  |
| 089 | **int** N = right - left; |
| 090 | **int** sqrtN = (**int**)floor(sqrt((double)N)); |
| 091 | **int** i = sqrtN; |
| 092 | **int** j = sqrtN - 1; |
| 093 |  |
| 094 | (\*counter)++; |
| 095 | **if** (arr[left] > element) |
| 096 | **return** -1; |
| 097 |  |
| 098 | **while** (j-- && (arr[i] < element)) { |
| 099 | (\*counter)++; |
| 100 | i += sqrtN; |
| 101 | } |
| 102 |  |
| 103 | (\*counter)++; |
| 104 | **if** (j) |
| 105 | **return** chooseSearch(arr, i - sqrtN + 1, i + 1, element, sqrtN,  counter); |
| 106 | **else** { |
| 107 | left = i + 1; |
| 108 | **return** chooseSearch(arr, left, right, element, right - left,  counter); |
| 109 | } |
| 110 | } |

### main.cpp

|  |  |
| --- | --- |
| 01 | */\** |
| 02 | *Main program* |
| 03 | *\*/* |
| 04 |  |
| 05 | **#include** <iostream> |
| 06 | **#include** "searches.h" |
| 07 | **#include** <time.h> |
| 08 |  |
| 09 | **#define** N 450 |
| 10 | **#define** TESTS 1000 |
| 11 |  |
| 12 | **void** generate(**int** arr[], **int** \*max, **int** \*mid, **bool** mode) { |
| 13 |  |
| 14 | **int** counter = 0, sum = 0; |
| 15 | **int** i = 0, j = 0; |
| 16 |  |
| 17 | \*max = 0; |
| 18 |  |
| 19 | **for** (i = 0; i < TESTS; i++) { |
| 20 | **if** (mode) { |
| 21 | **for** (j = 0; j < N; j++) |
| 22 | arr[j] = rand() % N; |
| 23 | } |
| 24 | **else** { |
| 25 | **for** (j = 0; j < N; j++) |
| 26 | arr[j] = j; |
| 27 | } |
| 28 |  |
| 29 | mBlockSearch(arr, 0, N, ((rand() % (N + N/10)) - N/10),  &counter); |
| 30 | **if** (counter > \*max) |
| 31 | \*max = counter; |
| 32 |  |
| 33 | sum += counter; |
| 34 | counter = 0; |
| 35 | } |
| 36 |  |
| 37 | \*mid = (**int**)(sum / TESTS); |
| 38 | } |
| 39 |  |
| 40 | **int** main() { |
| 41 |  |
| 42 | **int** arr[N + 1]; |
| 43 | **int** max, mid; |
| 44 |  |
| 45 | srand((**unsigned**)time(NULL)); |
| 46 |  |
| 47 | generate(arr, &max, &mid, false); |
| 48 | printf(*"Max: %d, Mid: %d\n"*, max, mid); |
| 49 |  |
| 50 | **return** 0; |
| 51 | } |

## Результаты работы программы

### Максимальное количество операций сравнения

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритмы поиска | Количество элементов в массиве | | | | | | | | |
| 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 |
| Линейный (неупорядоченный массив) | 101 | 201 | 301 | 401 | 501 | 601 | 701 | 801 | 901 |
| Быстрый линейный (неупорядоченный м.) | 51 | 101 | 151 | 201 | 251 | 301 | 351 | 401 | 451 |
| Быстрый линейный (упорядоченный м.) | 51 | 101 | 151 | 201 | 251 | 301 | 351 | 401 | 451 |
| Бинарный (упорядоченный м.) | 24 | 28 | 32 | 32 | 32 | 36 | 36 | 36 | 36 |
| Блочный (упорядоченный м.) | 23 | 30 | 39 | 45 | 48 | 38 | 43 | 41 | 42 |

### Среднее количество операций сравнения

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритмы поиска | Количество элементов в массиве | | | | | | | | |
| 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 |
| Линейный (неупорядоченный массив) | 69 | 141 | 213 | 279 | 346 | 409 | 480 | 541 | 609 |
| Быстрый линейный (неупорядоченный м.) | 33 | 67 | 101 | 136 | 171 | 202 | 231 | 271 | 299 |
| Быстрый линейный (упорядоченный м.) | 27 | 54 | 83 | 110 | 131 | 163 | 195 | 216 | 242 |
| Бинарный (упорядоченный м.) | 19 | 23 | 26 | 27 | 27 | 29 | 30 | 30 | 31 |
| Блочный (упорядоченный м.) | 13 | 16 | 19 | 21 | 24 | 22 | 24 | 24 | 25 |

### Графики функций временной сложности