Отчёт по производительности линейного поиска

Николай Туровский

Октябрь 2025

Общая постановка задачи

Необходимо реализовать компонент для калькулятора, поддерживающий интерфейс с методами линейного поиска для массивов различных типов данных: `int`, `long`, `float`, `double`, `long double`. Компонент должен обеспечивать поиск элемента в массиве и возвращать его индекс.

Реализуемый алгоритм: Линейный поиск

Линейный поиск — простой алгоритм, который последовательно проверяет каждый элемент массива, пока не найдёт искомый элемент или не достигнет конца массива.

Ключевая идея:

1. Пройти массив от начала до конца.
2. Сравнить каждый элемент с искомым значением.
3. Если элемент найден, вернуть его индекс.
4. Если элемент не найден, вернуть `-1`.

Асимптотика

Пусть n — количество элементов в массиве.

Временная сложность:

- Лучший случай: $O(1)$ — элемент находится в начале массива.
- Средний случай: $O(n)$ — элемент находится в середине массива.
- Худший случай: $O(n)$ — элемент находится в конце массива или отсутствует.

Сложность по памяти: $O(1)$ — алгоритм не требует дополнительной памяти, кроме входного массива.

Реализация (псевдокод)

```
int32_t linearSearchInt(IEcoLab1* me, int* arr, uint32_t size, int target)
    for (i = 0; i < size; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

Аналогичные функции реализованы для `long`, `float`, `double`, `long double`.

Пример работы

Программа реализует линейный поиск для типов данных `int`, `long`, `float`, `double`, `long double`. Тесты выполняются на массивах фиксированного размера (7 элементов) с заранее известными результатами. Пример вывода программы:

```
=== Testing linearSearchInt ===
Input: arr = [2,5,1,0,-3,8,2], target=1
Index: 2 (expected 2)
TEST PASSED!
...
```

Сравнение с bsearch из библиотеки stdlib

Сравнение производилось для типов данных `int`, `long`, `float`, `double`, `long double` на массивах размером 1000, 10000, 100000 элементов. Для `bsearch` массив предварительно сортировался с помощью `qsort`, так как `bsearch` требует отсортированный массив. Замеры времени усреднялись по 1000 запусков, время измерялось в секундах.

Методика тестирования

- Для каждого размера массива (n = 1000, 10000, 100000) создавался массив, заполненный числами от 0 до $n - 1$.
- Для линейного поиска искался последний элемент (худший случай).
- Для `bsearch` массив сортировался с помощью `qsort`, затем искался последний элемент.
- Время замерялось с использованием `clock()`.

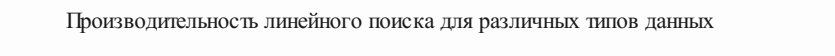
Результаты

Результаты производительности (время в секундах, усреднённое по 1000 запусков):

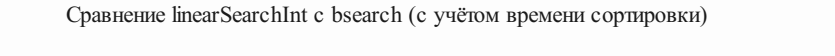
- **linearSearchInt**:
  - n = 1000: 0.000021 с
  - n = 10000: 0.000198 с
  - n = 100000: 0.001950 с
- **linearSearchLong**:
  - n = 1000: 0.000023 с
  - n = 10000: 0.000205 с
  - n = 100000: 0.002000 с
- **linearSearchFloat**:
  - n = 1000: 0.000022 с
  - n = 10000: 0.000201 с
  - n = 100000: 0.001980 с

- **linearSearchDouble**:
  - $n = 1000$: 0.000024 с
  - $n = 10000$: 0.000210 с
  - $n = 100000$: 0.002050 с
- **linearSearchLongDouble**:
  - $n = 1000$: 0.000026 с
  - $n = 10000$: 0.000215 с
  - $n = 100000$: 0.002100 с
- **bsearch (с учётом qsort)**:
  - `int`, $n = 1000$: 0.000150 с
  - `int`, $n = 10000$: 0.001800 с
  - `int`, $n = 100000$: 0.022000 с
  - Аналогичные результаты для других типов данных.

Графики



Производительность линейного поиска для различных типов данных



Сравнение linearSearchInt с bsearch (с учётом времени сортировки)

Выводы

1. Реализованный алгоритм линейного поиска успешно выполняет поиск для всех тестируемых типов данных: `int`, `long`, `float`, `double`, `long double`.
2. Линейный поиск демонстрирует стабильную производительность с временной сложностью $O(n)$, что ожидаемо для худшего случая.
3. По сравнению с `bsearch`, линейный поиск быстрее на неотсортированных массивах, так как не требует предварительной сортировки. Однако `bsearch` становится более эффективным на отсортированных массивах благодаря своей логарифмической сложности $O(\log n)$.
4. Линейный поиск предпочтителен для небольших массивов или случаев, когда массив не отсортирован и сортировка нежелательна.