SearchAlgorithms

August 8, 2020

1 Search Algorithms

1.1 Table of Contents

1.2 header includes required for this notebook

[1]:
```
#include <iostream>
#include <vector>
#include <random>
#include <iterator>
#include <ctime>
#include <cstdlib>
#include <algorithm>

using namespace std;
```

[2]:
```
// operator<< overloaded to print a vector
template<class T>
ostream& operator<<(ostream& out, const vector<T>& v) {
    char comma[3] = {'\0', ' ', '\0'};
    out << '[';
    for (auto& e: v) {
        out << comma << e;
        comma[0] = ',';
    }
    out << "]";
    return out;
}
```

1.3 Sequential Search

- find a key in a sequence container

- input is unsorted vector
- output is the index if key found, -1 if key not found
- Algorithm:
    1. start from the first index
    2. if the key matches with the element at the index, return index
    3. otherwise move to the next element (index)
    4. repeat from step 2
    5. if key doesn't match with any of the element, return -1

[3]:
```
template<class T>
int sequentialSearch(const vector<T> & v, T key) {
    int index = 0;
    while (index < v.size()) {
        if (v[index] == key) // found our element; key comparison that controls␣
↪the loop
            return index;
        else
            index ++;
    }
    return -1;
}
```

[4]:
```
void generateRandomNumbers(vector<int> &rands, int count, int start, int end) {
    // fill the vectors with random numbers
    random_device rd;
    //https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine
    // generates high quality random unsigned ints
    mt19937 mt(rd());
    uniform_int_distribution<> dis(start, end); // numbers between start and␣
↪end inclusive
    generate(rands.begin(), rands.end(), bind(dis, ref(mt)));
}
```

[5]:
```
vector<int> nums(20);
```

[6]:
```
generateRandomNumbers(nums, 20, 0, 20);
cout << nums << endl;
```

```
[14, 0, 4, 6, 3, 1, 18, 3, 3, 5, 12, 18, 0, 11, 19, 16, 18, 15, 0, 4]
```

[7]:
```
int key;
int searchIndex;
```

[8]:
```
// generate a random number and search in nums vector...
srand(time(NULL));
key = rand()%20;
```

[9]:
```
searchIndex = sequentialSearch<int>(nums, key);
if (searchIndex >= 0)
    cout << key << " found at index " << searchIndex << endl;
else
    cout << key << " not found!" << endl;
```

```
0 found at index 1
```

1.4 Sequential Search Asymptotic Analysis

- look for key comparison/operation
- Best case: 1 comparison, $O(1)$
- Average case: $n/2$ comparison, $O(n)$
- Worst case: n comparison, $O(n)$

1.5 Binary Search

- input is a sequence sorted in increasing order
- imagine searching for a word in a dictionary or someone's name in a phone directory
- uses divide and conquer technique
  - in each iteration, the search space is reduced by half
  - if key is found at the middle, return the index
  - repeat the search in lower or upper half of the sequence until sequence is exhausted
- visualize binary search: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/AnalProgram.html

[10]:
```
template <class T>
int binarySearch(const vector<T> &v, T key) {
    int low = 0;
    int high = v.size()-1;
    while (low <= high) { // stop when low and high cross
        int mid = (low+high)/2; // check middle of sequence
        if (v[mid] == key) // found it
            return mid; // return the index
        else if (v[mid] > key) // check in left half
            high = mid - 1;
        else // check in right half
            low = mid + 1;
    }
    return -1;
}
```

[11]:
```
vector<int> nums1(20);
```

[12]:
```
generateRandomNumbers(nums1, 20, 0, 20);
cout << nums1 << endl;
```

```
[1, 13, 13, 7, 18, 1, 3, 15, 17, 2, 18, 15, 7, 3, 3, 8, 6, 19, 1, 5]
```

[13]:
```
// for binary search to work, sequence must be sorted
sort(nums1.begin(), nums1.end());
cout << nums1 << endl;
```

```
[1, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7, 8, 13, 13, 15, 15, 17, 18, 18, 19]
```

[14]:
```
// generate a random number and search in nums1 vector...
srand(time(NULL));
key = rand()%20;
cout << " key to search = " << key << endl;
```

```
 key to search = 8
```

[15]:
```
searchIndex = binarySearch<int>(nums1, key);
if (searchIndex >= 0)
    cout << key << " found at index " << searchIndex << endl;
else
    cout << key << " not found!" << endl;
```

```
8 found at index 11
```

1.6 Binary Search Asymptotic Analysis

- Best case: 1 comparison $O(1)$
- Average and Worst cases: $(O(logn))$
- binary search analysis visualization: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/AnalProgran
- each loop of binarySearch cuts the size of the sequence (problem size) approximately in half and for each problem size, we do $O(1)$ comparison for a total of $\sum_{i=0}^{logn} 1$

1.6.1 as n grows, the $O(n)$ running time for sequential search in the average and worst cases quickly becomes much larger than the $O(logn)$ of binary search

1.7 Empirical Analysis: Linear Search Vs Binary Search

[16]:
```
// function to time sequentialSearch and binarySearch
double timeit(const vector<int> &v, int key, int (*searchFunc)(const␣
↪vector<int> &, int)) {
    clock_t begin = clock();
    int i = (*searchFunc)(v, key);
    clock_t end = clock();
    double elapsed_secs = double(end - begin) / CLOCKS_PER_SEC;
    return elapsed_secs;
}
```

[31]:
```
void compareSearchAlgos(int N) {
    vector<int> nums(N);
    generateRandomNumbers(nums, N, 0, N);
    // make a copy of nums
```

```
    vector<int> sortedNums = nums;
    sort(sortedNums.begin(), sortedNums.end());
    // generate a random number and search in nums1 vector...
    srand(time(NULL));
    int key = rand()%N+1;
    cout << "key to search = " << key << endl;
    cout << "Sequential Search time: " << timeit(nums, key, sequentialSearch)␣
↪<< " seconds." << endl;
    cout << "Binary Search time: " << timeit(sortedNums, key, binarySearch) <<␣
↪" seconds." << endl;
}
```

1.7.1 Sequential and Binary Search Comparison with 100 K integers

[32]:
```
compareSearchAlgos(100000);
```

```
 key to search = 23182
Sequential Search time: 0.002065
Binary Search time: 1.8e-05
```

1.7.2 Sequential and Binary Search Comparison with 1 M integers

[33]:
```
compareSearchAlgos(1000000);
```

```
 key to search = 251777
Sequential Search time: 0.000768
Binary Search time: 1.4e-05
```

1.7.3 Sequential and Binary Search Comparison with 1 B integers

[34]:
```
compareSearchAlgos(1000000000);
/*
// sorting took much longer!
key to search = 505016941
Sequential Search time: 2.88439 seconds
Binary Search time: 4e-05 seconds
*/
```

```
 key to search = 505016941
Sequential Search time: 2.88439
Binary Search time: 4e-05
```

[ ]:
```

```