Pre-Placement Grind

Hardik Rajpal

July 26, 2023

Contents

1 Week 1

1.1 Searching Algorithms

Notes from GFG

These are algorithms to check for the existence of an element or to retrieve it from a data structure. The retrieval can also involve only returning the position (index) or a pointer to the element. There are two types:

1. Sequential search: check every element based on a pre-determined sequence (ex. linear, alternating, etc.), and return the matches.

2. Interval search: Designed for searching in **sorted** data structures. They involve **repeatedly** dividing the search space into intervals which can be excluded entirely after certain checks (ex. binary search).

Some search algorithms are discussed below:

1. **Linear Search:** Straighforward for-loop iterating over all elements in an array.
   Time: O(n)
   Space: O(1)

2. **Sentinel Linear Search:** Reduces the number of comparisons by eliminating the need to check if the index is within bounds. This is accomplished by appending the target element to the end of the array, and treating its index in the result as "not found."
   Time: O(n)
   Space: O(1)

3. **Binary Search:** It's used for sorted arrays. It involves comparing the element at the center of the interval (defined initially as the entire array), with the target element. One of the halves of the interval is picked based on this comparison. The interval shrinks until the target is found or an interval of size one is not equal to the element. It can be implemented recursively or iteratively, each involving a step similar to $m = l + \frac{(r-l)}{2}$ while $l \leq r$
   Time: .
   Space: O(log(n)) O(1)

4. **Meta Binary Search:** Seems unimportant but check it here
   Time: O(log(n))
   Space: O(1)

5. **K-ary Search:** The search space is divided into k intervals in each step and one of them is picked to proceed further by comparing the target element to the interval markers.
   Time: O(log(n)). The reduction is of a constant term: $log_k 2$
   Space: O(1)

6. **Jump Search:** The sorted array is examined in jumps of the optimal size $\sqrt{n}$,until the element being examined is greater than the target element. The interval is then shrunk to the previous interval. The shurnken interval can be examined linearly or with another jump search.
   Time: O($2\sqrt{n} = O(\sqrt{n})$), or $O(n^{1/2} + n^{1/4} + n^{1/8}...) = O(\sqrt{n})$
   Space: O(1)

7. **Interpolation Search:** It improves over binary search only if the data is uniformly distributed. It involves selecting the splitting point of the current search space by comparing the target value to the current lower and upper bounds of the space. Linear interpolation involves the following equations:
   $slope = (arr[r] - arr[l])/(r - l)$
   $m = l + slope \times (x - arr[l])$
   Time: O(log(log(n))) on average, O(n) WCS.
   Space: O(1)

8. **Exponential or Unbounded (Binary) Search:** We examine the search space from the lower end l, comparing $l + 2^k - 1$ with the target element x, where k is the number of comparisons so far, until $x < arr[l + 2^k - 1]$. Then, we examine the interval bounded by $l + 2^{k-1} - 1$ and $l + 2^k - 1$, using binary search.
Time: O(log(n)), where n is the length of the array or where the first occurrence of the target element exists in an unbounded array.
Space: O(1)

9. **Fibonacci Search:** The array must be sorted. We first find the Fibonacci number $f(m)$ that exceeds the length of the given array. We compare the target element to the element at $arr[f(m-2)]$. We pick an interval based on the outcome.
Time: O(log(n))
Space: O(1)

1.2 Sorting Algorithms