Lecture 20

Interpolation Search
Exponential Search

October 28, 2021
Thursday

Interpolation Search

Interpolation Search

- Binary search always starts with the middle element.
- However, if we have some idea about the distribution of values in data. We don’t have to start in middle!
  - Consider a person looking for a word in dictionary.
  - They usually don’t start from the middle.
  - A person looking for a word starting with V, generally assumes that entries beginning with ‘V’ start closer to the end of the dictionary.
  - Then the next decision will be based on the results from the first one.

Interpolation Search

- In simple words, people use some knowledge
    - About the expected distribution of the data elements
        - To “compute? where to look next.
- This form of “Computed” Binary Search is known as ***Interpolation / Dictionary Search.***

Interpolation Search

- In interpolation search
  - We search data [ ] at a position *p* that is appropriate to the value of key as follows

$$int\ \ p = low + \left\{ \left((double)\, \frac{(hi - low)}{data\,[\,hi\,] - data\,[\,low\,]} \right) \times (key - data\,[\,low\,] \right\}$$

```
int InterpolationSearch ( int data [ ], int key, int n) {

        int low = 0, high = n - 1;
        while (low <= high && key >= data [ low ] && key <= data [ high ]) {
                if ( low == high) {
                        if (data [ low ] == key) return low;
                        return -1;
                }

                int p = low + ( ( (double) (high - low) / (data [ high ] - data [ low ] ) ) * ( key - data [ low ] ) );

                if ( data [ p ]  == x )
                        return p;

                if ( data [ p ] < key )
                        low = p + 1;
                else
                        high = p - 1;


                return - 1;
        }

        return -1;
}
```

TIME COMPLEXITY

- Time complexity
  - Best Case: *O (1)*
  - Average Case: *O (log (log n))*
  - *Worst Case: O* (n)
    - If the elements of data [ ], are uniformly distributed and the values increase exponentially
    - 1, 2, 3, 4, 5, 6, 7, 8, 9, 100000000.
  - Space Complexity *O (1)*

Exponential Search

EXPONENTIAL SEARCH

- Name is misleading.
- Exponential Search solves the problem of unbounded array.
  - When we don’t know the last element index.
- Requires sorted data.
- Requires two steps
  - Find the range where data will be present
  - Perform binary search on that range.

EXPONENTIAL IMPLEMENTATION

```
int ExponentialSearch (data [ ], int low, int high, int value)

        if ( data [ 0 ] == x )

                return 0;
        int i = 1;
        while ( i < n && data [ i ] <= value )
                i *= 2;

        return BinarySearch (data, i / 2, min (i, n - 1), x);
}
```

BINARY IMPLEMENTATION

```
BinarySearch (data [ ], int low, int high, int value)
    while ( low <= high) {
        int mid = low + ( high - low ) / 2;
        if (data [ mid ] == value )
            return mid;
        if (data [ mid ] < value )
            low = mid + 1;
        if (data [ mid ] > value )
            high = mid - 1;
    }
    return -1;
}
```

Time Complexity

- Worst Case: *O (log i).*
    - *i* is the index of the key element in the array.
- Average Case is same as worst case: *O (log i).*
- In best case scenario the first center item is the target and we only make one comparison, *O (1).*
- It will outperform binary search when the key is near the beginning.
- Space complexity
    - With iterative binary search: *O (1)*
    - With recursive binary search: *O (log n)*