CMSC 351 - Introduction to Algorithms
Spring 2012
Lecture 8

Instructor: MohammadTaghi Hajiaghayi
Scribe: Rajesh Chitnis

1 Introduction

In this lecture we will look at Binary Search and some applications.

2 Algorithms involving sequences and sets

Usually the input for our algorithms is a finite set or a sequence. The differences between sets and sequences are as follows:

1. In sequences the order of the elements is important whereas in sets the order is not important.
2. In sets we assume an element does not appear more than once (not in multisets though) but there is no such assumption for sequences.

The input is always a sequence (though if the order is not important then we call it a set). In this chapter the input is an array of known size n. Also the elements of the array are of the same type and can be compared with each other.

We will now look at Binary Search and Sorting which are very important and universally applicable algorithms.

3 Binary Search

Basic Idea: Cut the search space in half (or approximately so) by asking only one question.

The Problem: Let $x_1, x_2, \ldots, x_n$ be a sequence of real numbers such that $x_1 \leq x_2 \leq \ldots x_n$. Given a real number z, we want to find whether z appears in the sequence and if it does, then find an index i such that $x_i = z$.

Approach: Say there is only one index i such that $x_i = z$. In general, it might not be the case and we want to find all or the smallest and largest such index. We compare z with $x_{\lceil \frac{n}{2} \rceil}$. If $z < x_{\lceil \frac{n}{2} \rceil}$ then z is clearly in the first half of the sequence. Otherwise z is in the second half of the sequence. Finding z in either half is a problem of size $\frac{n}{2}$, which can be solved by induction. We handle the base case of $n = 1$ by directly comparing z to the element.

Algorithm 1 BINARYSEARCH(z, L, R, x)

Input: An array x and integers z, L, R
Output: An index i such that $x[i] = z$ or -1 otherwise.

```
1:  if L = R then
2:    if x[L] = z then
3:      Find = L;
4:    else
5:      Find = -1;
6:    end if
7:  else
8:    M := ⌈(L+R)/2⌉;
9:    if z = x[M] then
10:     Find = M;
11:   else
12:     if z < x[M] then
13:       Find = BinarySearch(z, L, M − 1, x);
14:     else
15:       Find = BinarySearch(z, M, R, x);
16:     end if
17:   end if
18: end if
    return Find;
```

Time Complexity: Since each time a comparison is made, the range is cut by one half. So the number of comparisons is $O(\log n)$. For small values of n, binary search might not be as efficient as linear search.

4 Binary Search in a Cyclic Sequence

Definition: A sequence $x_1, x_2, \ldots, x_n$ is said to be cyclically sorted if the smallest number in the sequence is x_i for some unknown i and the sequence $x_i, x_{i+1}, \ldots, x_n, x_1, \ldots, x_{i-1}$ is sorted in increasing order, i.e., $x_i \leq x_{i+1} \leq \ldots \leq x_n \leq x_1 \leq x_2 \leq \ldots \leq x_{i-1}$.

The Problem: Given a cyclically sorted list, find the position of the minimal element in the list.

Approach: We use the idea of eliminating half of the sequence by just one comparison. Take any two numbers x_k and x_m, such that $k < m$. If $x_k < x_m$,

then i cannot be in the range $k < i \leq m$ since that would imply $x_k \geq x_m$, a contradiction. On the other hand if $x_k > x_m$ then i must be in the range $k < i \leq m$, since the order is switched somewhere in that range. Thus with one comparison we can eliminate half the elements and we can find i in $O(\log n)$ comparisons.

Algorithm 2 CyclicFind(L, R, x)

```
1: if L = R then
2:    return L;
3: else
4:    M := ⌊(L+R)/2⌋;
5:    if x[M] < x[R] then
6:       CyclicFind(L, M, x);
7:    else
8:       CyclicFind(M + 1, R, x);
9:    end if
10: end if
```

5 Binary Search for a special (fixed) index

The Problem: Given a sorted sequence of distinct integers $a_1, a_2, \ldots, a_n$ determine if there is an index i such that $a_i = i$.

Approach: Again we cannot use binary search here, but the principle can be applied. If $a_{\lceil \frac{n}{2} \rceil}$ is exactly $\lceil \frac{n}{2} \rceil$ then we are done; otherwise if it is less than $\lceil \frac{n}{2} \rceil$ then $a_{\lceil \frac{n}{2} \rceil - 1}$ is less than $\lceil \frac{n}{2} \rceil - 1$ and so on since all numbers are distinct. Thus no number in the first half of the sequence can satisfy the property and we can concentrate only on the second half. A similar idea holds if $a_{\lceil \frac{n}{2} \rceil}$ is greater than $\lceil \frac{n}{2} \rceil$. The algorithm is given in the book [1].

6 Binary Search in Sequences of Unknown Size

Sometimes we use a procedure like binary search to double the search rather than halve it. Consider a regular search problem with known size. We cannot halve the search range, since we do not know the boundaries. Instead we search for an element $x_i \geq z$. If we can find x_i, we can do binary search from 1 to i. First we compare z and x_1. If $z \leq x_1$, then $z = x_1$. Now by induction we know $z \geq x_i$ for $j \geq i \geq 1$. If we now compare z to x_{2j}, then we double the search space with one comparison. If $z \leq x_{2j}$, then we know $x_j \leq z \leq x_{2j}$ and we can find z with $O(\log j)$ additional comparisons. Overall if i is the smallest index such that $x_i \geq z$, then it takes $O(\log i)$ comparisons to find an x_j such that $z \leq x_j$ and another $O(\log i)$ comparisons to find i. The same algorithm can also be used when the size of the sequence is known but we suspect that

i is very small. This gives an improvement since we have $O(\log i)$ instead of $O(\log n)$. However since it is actually $2 \log i$, this gives an improvement only if $2 \log i \leq \log n \Rightarrow i \leq \sqrt{n}$ or $i = O(\sqrt{n})$.

7 Interpolation Search

Interpolation search is a combination of binary search and linear search. It is an algorithm for searching for a given key value in an indexed array that has been sorted by the values of the key [2]. Interpolation search basically describes how we would search through a telephone book for a particular name, the key value by which the book's entries are ordered. In each search step it calculates where in the remaining search space the sought item might be, based on the key values at the bounds of the search space and the value of the sought key, usually via a linear interpolation. The key value actually found at this estimated position is then compared to the key value being sought. If it is not equal, then depending on the comparison, the remaining search space is reduced to the part before or after the estimated position.

Binary search always chooses the middle of the remaining search space whereas linear search uses equality only as it compares elements one-by-one. Interpolation search tries to combine the good points of the above search algorithms. Let us consider an example: Suppose we want to open page 200 from what looks like a 800 page book. We will try to open a page which seems around one-fourth of the size of the book. If we hit 200 exactly, then that will be perfect. Suppose we open page 250. Then the entire search space is now restricted to 250 pages and we know that we need to go back by one-fifth of the search space ($\frac{250-200}{250} = \frac{1}{5}$). We can continue this process till we get get close to 200 and then perform a simple linear search by turning the pages one at a time.

We now give a formal description of Interpolation Search in Algorithm 3: The performance of Interpolation Search depends not only on the size of the sequence, but also on the input. See Example 6.4 from the book [1] for examples where even interpolation search checks every number in the sequence. However, interpolation search is very efficient for inputs which are uniformly distributed, e.g., pages of a book. It can be shown that the average number of comparisons performed by interpolation search, where the average is taken over all sequences, is $O(\log \log n)$.

See more on Interpolation search and other applications of binary search in the book [1].

References

[1] Udi Manber, *Introduction to Algorithms - A Creative Approach*

[2] Wikipedia article on Interpolation Search

Algorithm 3 INTERPOLATIONSEARCH(X, n, z)

Input: A sorted array X in the range 1 to n, and a search key z.
Output: Position (an index i such that $X[i] = z$, or 0 if no such index exists)

```
1: begin
2: if z < X[1] or z > X[n] then
3:    Position:=0 (unsuccessful search)
4: else
5:    Position:= Int-Find(z,1,n)
6: end if
7: end

8: function Int-Find(z,Left,Right):integer:
9: begin
10: if X[left]= z then
11:    Int-Find = Left
12: else
13:    if Left=Right or X[Left]=X[Right] then
14:       Int-Find = 0
15:    else
16:       NextGuess:=Left+ (z − X[Left])(Right − Left) / (X[Right] − X[Left])
17:       If z < X[NextGuess], Then Int-Find = Int-Find(z,Left,NextGuess-1)
          Else, Int-Find = Int-Find(z,NextGuess,Right)
18:    end if
19: end if
20: end
```