Searching

Searching For a Good Subtitle

CSCI 3700 — Data Structures and Objects

Department of Computer Science and Information Systems
Youngstown State University

Robert W. Kramer

Outline

Motivation
Humans vs. Computers

Here's a list
3 1 6 8 4 5 2 7

- Is 4 in the list?
  - How do you know?
- Is 9 in the list?
  - How do you know?

You're Cheating!
- Humans can process lists with 7 ± 2 items innately
- Computers can't do that
- Need a methodical way to do searching

Why Search?

And why study searching?

- Searching is one of the most basic operations performed by programs
- A good repertoire of search algorithms is essential

The General Idea

- You have a list of data L with n elements in it
- Elements are in positions $0 - (n-1)$
- You are searching for a key k that may or may not be in L
- There may be assumptions about L in various algorithms

A Dramatization

With less drama. . . than an Oxford comma

Is this what you're looking for? No.
Is this what you're looking for? No.
Is this what you're looking for? No.
. . .*time passes*. . .
Is this what you're looking for? No.
Is this what you're looking for? Yes.

Note

This is the essence of *Sequential Search*

Sequential Search

A simple, robust search algorithm

The basic idea:

- Start at the beginning of the list
- Step through L, item by item, until either:
    - The key k is found, or
    - We run out of data to examine

Note

We can step through elements of L in any order.
Without a reason to do otherwise, start at the beginning and work toward the end.

Sequential Search

The algorithm

procedure $\textsc{SequentialSearch}(L, n, k)$
    $i \leftarrow 0$
    while $i < n$ **and** $L[i] \neq k$ **do** ▷ Short-circuit evaluation
        $i \leftarrow i + 1$
    end while

    if $i < n$ **then**
        return success
    else
        return failure
    end if
end procedure

Sequential Search

Three questions about the algorithm

- What if the first list position is 1, not 0?
- Can we always compare keys?
- How do we indicate success and failure?

First Question

1-based lists

Yes, this is easy. If the first position is j, replace all occurrences of $L[i]$ with $L[i + j]$.

Second Question

Can we always compare keys?

Yes.

- Lists for searching must always contain *comparable* data types
- Numeric keys use standard operators
- Strings can use either `strcmp()` or C++ string operators
- More complex keys may require more work

Third Question

Success and failure

Common methods:

- Return valid index for success, invalid index for failure
- Return pointer to data if successful, `NULL` if not
- Either of the above if successful, throw an exception if not

We will use either the first or last method

Sequential Search Performance

- Are there loops in the algorithm? Yes.
- Does the loop depend upon the amount of data? Yes.
- Linear or logarithmic loop? Linear.
- Big-O value: $O(n)$

Can We Do Better?

Remember, the list is unordered

- In theory, no.
- In practice, maybe.

Improving Sequential Search

Again, maybe

Start by looking at the loop.

- Three actions in the loop:
  - Compare $i < n$
  - Compare $L[i] \neq k$
  - Increment i

Can we eliminate one of these?

Reducing Work
Making one action do the work of two

Remove the comparison $i < n$

- Won't that run forever if k isn't in L? Yes, but...
- What if we add k to the end of the list?
    - Algorithm now guaranteed to find k
    - $i < n$? k was in the original list
    - $i = n$? k was not in the original list, we found the copy we added

This is *Sentinel Search*

Sentinel Search

The algorithm

procedure $\textsc{SentinelSearch}(L, n, k)$
    $L[n] \leftarrow k$ ▷ Add the key
    $i \leftarrow 0$
    while $L[i] \neq k$ **do**
        $i \leftarrow i + 1$
    end while
    Remove k from list if necessary
    if $i < n$ **then**
        return success
    else
        return failure
    end if
end procedure

Sentinel Search Restrictions

Without restrictions, we'd have no Sequential Search

Two restrictions on Sentinel Search:

Restriction 1

You must be able to add k to the end of the list and remove it if necessary
Must use Sequential Search otherwise

Restriction 2

Adding and removing must be faster than $O(n)$ time
No faster than Sequential Search otherwise

Sentinel Search Performance

- Are there loops in the algorithm? Yes.
- Does the loop depend upon the amount of data? Yes.
- Linear or logarithmic loop? Linear.
- Big-O value: $O(n)$
- This is no better than Sequential Search!
- In practice, slightly faster

Another Optimization

Look at the data in L

- One big assumption in Sequential Search analysis
  - All items equally likely to be search targets
- What if the probability of being a target was not equal?
- Place more likely targets at the front of the list!

This is *Probability Search*

Probability Search

It probably works better

Probability Search comes in two flavors:

- Static
  - Boring...
  - Arrange data *before* searching
  - Use Sequential or Sentinel Search
- Dynamic
  - Start with Sequential or Sentinel Search
  - If k is found, move it forward in the list

Dynamic Probability Search

The algorithm

procedure $\textsc{ProbabilitySearch}(L, n, k)$
    $i \leftarrow 0$
    while $i < n$ **and** $L[i] \neq k$ **do**
        $i \leftarrow i + 1$
    end while
    if $i < n$ **then**
        $i \leftarrow \textsc{Promote}(L, i)$
        return success
    else
        return failure
    end if
end procedure

The PROMOTE Function

Moving the key forward in the list

```
procedure PROMOTE(L, i)
    temp ← L[i]                          ▷ Key being promoted
    j ← i − Δ                            ▷ Promotion destination
    if j < 0 then
        j ← 0                            ▷ Can't be negative
    end if
    while i > j do                       ▷ Slide elements back
        L[i] ← L[i − 1]
        i ← i − 1
    end while
    L[i] ← temp                          ▷ Key's new home
    return i
end procedure
```

Probability Search Performance

- Are there loops in the algorithm? Yes.
- Does the loop depend upon the amount of data? Yes.
- Linear or logarithmic loop? Linear.
- Big-O value: $O(n)$
- This is still no better than Sequential Search!
- In practice, faster if probability distribution not even

Probability Search Caveat

Only one restriction

Must be able to rearrange data without affecting other parts of the program

Ordered Search

Taking advantage of information

Suppose the list is sorted in ascending order

- May not have to search entire list
- Can stop when list element is larger than key

This gives us *ordered search*

Ordered Search

The algorithm

```
procedure OrderedSearch(L, n, k)
    i ← 0
    while i < n and L[i] < k do          ▷ Short-circuit evaluation
        i ← i + 1
    end while

    if i < n and L[i] = k then
        return success
    else
        return failure
    end if
end procedure
```

Ordered Search Performance

- Are there loops in the algorithm? Yes.
- Does the loop depend upon the amount of data? Yes.
- Linear or logarithmic loop? Linear.
- Big-O value: $O(n)$
- This is still no better than Sequential Search!
- In practice, faster unless large values frequently sought
- Can be combined with Sentinel Search

We Interrupt This Program. . .

Next, some games

Can We Do Better?

Why do I keep asking this question?

The first game models a sequential search

- Examine item
- Yes or no response
- Make another guess

The second game provides more information

- Now get too high / too low information
- Able to find item more quickly

This gives us *binary search*

Binary Search

The general idea

The general idea is this:

1. Pick the item in the middle of the list
2. If that's the key, stop
3. Otherwise, throw out half of the list and go back to step 1

Note

We don't really throw out the list. We use two indexes to mark the first and last position that could contain the key

Binary Search Algorithm

procedure BinarySearch(L, n, k)
    $low \leftarrow 0$
    $high \leftarrow n - 1$
    while $low \leq high$ **do**
        $mid \leftarrow \frac{low+high}{2}$
        if $L[mid] = k$ **then**
            return mid
        else
            if $L[mid] < k$ **then**
                $low \leftarrow mid + 1$
            else
                $high \leftarrow mid - 1$
            end if
        end if
    end while
    return -1
end procedure

Binary Search Performance

- Are there loops in the algorithm? Yes.
- Does the loop depend upon the amount of data? Yes.
- Linear or logarithmic loop? Logarithmic.
- Big-O value: $O(\lg n)$
- This is much better than Sequential Search!
- Don't forget, the list must be sorted!

Can We Do Better?

You just *knew* this would come up again

Can we do better?

- Binary search performs two key comparisons inside the loop
- Can move one comparison outside of the loop

This is the *forgetful binary search*

Forgetful Binary Search Algorithm

procedure FORGETFULBINARYSEARCH(L, n, k)
 $low \leftarrow 0$
 $high \leftarrow n - 1$
 while $low < high$ **do**   ▷ This is different
  $mid \leftarrow \frac{low+high}{2}$
  if $L[mid] < k$ **then**   ▷ No check for $=$
   $low \leftarrow mid + 1$
  else
   $high \leftarrow mid$ ▷ mid not $mid - 1$
  end if
 end while
 if $L[low] = k$ **then**   ▷ Now we check for $=$
  return low
 else
  return -1
 end if
end procedure

Forgetful Binary Search

Differences from the regular version

Regular	Forgetful
Two comparisons per loop	One comparison per loop
$<$, $=$ or $>$ information	$<$, $\geq$ information
Stops when key found	Continues until list is singleton
Stops when list is empty	See above $\uparrow$

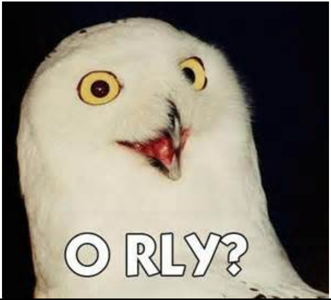
Forgetful Binary Search Performance

- Are there loops in the algorithm? Yes.
- Does the loop depend upon the amount of data? Yes.
- Linear or logarithmic loop? Logarithmic.
- Big-O value: $O(\lg n)$
- This is the same as regular binary search

Forgetful Binary Search Performance

Forgetting is good. . . usually

- Forgetful binary search always cuts the list down to a single element
- Regular binary search stops as soon as possible
- However, forgetful binary search is usually faster



Can We Do Better?

Last time in this slide show, I promise!

With relative (higher/lower) information...

- On average, yes — *interpolation search*
- In the worst case, no

However, what if we had *absolute* location information?

Summary

- Sequential search is basis of many $O(n)$ search algorithms
- Many variations work better in practice, but have restrictions
- Binary search is faster, but requires a sorted list
- Forgetful binary search is usually even faster