CS101 Lecture 28: Searching Algorithms

Linear Search
Binary Search

Aaron Stevens (azs@bu.edu)
17 April 2013

What You’ll Learn Today

- What is searching?
- Why does searching matter?
- How is searching accomplished?
- Why are there different searching algorithms?
- On what basis should we compare algorithms?
Searching

Searching
Attempts to find an item in a collection. It is always possible that the item might not be found.

Search key
The field (or fields) on which the search is based.

Example: Search Key

Consider this list:

<table>
<thead>
<tr>
<th>#</th>
<th>NAME</th>
<th>POSL</th>
<th>HT.</th>
<th>WT.</th>
<th>AGE</th>
<th>EXP.</th>
<th>COLLEGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>Aiken, Danny</td>
<td>LB</td>
<td>6-4</td>
<td>260</td>
<td>24</td>
<td>2</td>
<td>Virginia</td>
</tr>
<tr>
<td>24</td>
<td>Arrington, Kyle</td>
<td>CB</td>
<td>5-10</td>
<td>196</td>
<td>26</td>
<td>4</td>
<td>Hofstra</td>
</tr>
<tr>
<td>92</td>
<td>Bequette, Jake</td>
<td>DE</td>
<td>6-5</td>
<td>265</td>
<td>23</td>
<td>R</td>
<td>Arkansas</td>
</tr>
<tr>
<td>38</td>
<td>Bolden, Brandon</td>
<td>RB</td>
<td>5-11</td>
<td>220</td>
<td>22</td>
<td>R</td>
<td>Mississippi</td>
</tr>
<tr>
<td>97</td>
<td>Brace, Ron</td>
<td>DL</td>
<td>6-3</td>
<td>330</td>
<td>25</td>
<td>4</td>
<td>Boston College</td>
</tr>
<tr>
<td>12</td>
<td>Brady, Tom</td>
<td>QB</td>
<td>6-4</td>
<td>225</td>
<td>35</td>
<td>13</td>
<td>Michigan</td>
</tr>
<tr>
<td>84</td>
<td>Branch, Deion</td>
<td>WR</td>
<td>5-9</td>
<td>195</td>
<td>33</td>
<td>11</td>
<td>Louisville</td>
</tr>
<tr>
<td>81</td>
<td>Cannon, Marcus</td>
<td>OL</td>
<td>6-5</td>
<td>340</td>
<td>24</td>
<td>2</td>
<td>Texas Christian</td>
</tr>
<tr>
<td>25</td>
<td>Chung, Patrick</td>
<td>S</td>
<td>5-11</td>
<td>210</td>
<td>25</td>
<td>4</td>
<td>Oregon</td>
</tr>
<tr>
<td>23</td>
<td>Cole, Marquise</td>
<td>CB</td>
<td>5-10</td>
<td>195</td>
<td>29</td>
<td>4</td>
<td>Northwestern</td>
</tr>
<tr>
<td>63</td>
<td>Connolly, Dan</td>
<td>OL</td>
<td>6-4</td>
<td>310</td>
<td>30</td>
<td>7</td>
<td>Southeast Missouri State</td>
</tr>
<tr>
<td>96</td>
<td>Cunningham, Jermaine</td>
<td>DE</td>
<td>6-3</td>
<td>255</td>
<td>24</td>
<td>3</td>
<td>Florida</td>
</tr>
<tr>
<td>71</td>
<td>Deaderick, Brandon</td>
<td>DL</td>
<td>6-4</td>
<td>305</td>
<td>25</td>
<td>3</td>
<td>Alabama</td>
</tr>
<tr>
<td>37</td>
<td>Dennard, Alfonzo</td>
<td>CB</td>
<td>5-10</td>
<td>200</td>
<td>23</td>
<td>R</td>
<td>Nebraska</td>
</tr>
</tbody>
</table>

http://www.patriots.com/team/
Example: Search Key

Now consider this list:

<table>
<thead>
<tr>
<th>#</th>
<th>NAME</th>
<th>POS.</th>
<th>HT.</th>
<th>WT.</th>
<th>AGE</th>
<th>EXP</th>
<th>COLLEGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Gostkowski, Stephen</td>
<td>K</td>
<td>6-1</td>
<td>215</td>
<td>28</td>
<td>7</td>
<td>Memphis</td>
</tr>
<tr>
<td>11</td>
<td>Edelman, Julian</td>
<td>WR</td>
<td>5-10</td>
<td>200</td>
<td>26</td>
<td>4</td>
<td>Kent State</td>
</tr>
<tr>
<td>12</td>
<td>Brady, Tom</td>
<td>QB</td>
<td>6-4</td>
<td>225</td>
<td>35</td>
<td>13</td>
<td>Michigan</td>
</tr>
<tr>
<td>14</td>
<td>Mesko, Zoltan</td>
<td>P</td>
<td>6-5</td>
<td>230</td>
<td>26</td>
<td>3</td>
<td>Michigan</td>
</tr>
<tr>
<td>15</td>
<td>Mallett, Ryan</td>
<td>QB</td>
<td>6-6</td>
<td>245</td>
<td>24</td>
<td>2</td>
<td>Arkansas</td>
</tr>
<tr>
<td>18</td>
<td>Slater, Matthew</td>
<td>WR</td>
<td>6-0</td>
<td>210</td>
<td>27</td>
<td>5</td>
<td>UCLA</td>
</tr>
<tr>
<td>22</td>
<td>Ridley, Stevan</td>
<td>RB</td>
<td>5-11</td>
<td>220</td>
<td>23</td>
<td>2</td>
<td>LSU</td>
</tr>
<tr>
<td>23</td>
<td>Cole, Marquise</td>
<td>CB</td>
<td>5-10</td>
<td>195</td>
<td>29</td>
<td>4</td>
<td>Northwestern</td>
</tr>
<tr>
<td>24</td>
<td>Arrington, Kyle</td>
<td>CB</td>
<td>5-10</td>
<td>195</td>
<td>26</td>
<td>4</td>
<td>Hofstra</td>
</tr>
<tr>
<td>25</td>
<td>Chung, Patrick</td>
<td>S</td>
<td>5-11</td>
<td>210</td>
<td>25</td>
<td>4</td>
<td>Oregon</td>
</tr>
<tr>
<td>26</td>
<td>Martin, Derrick</td>
<td>S</td>
<td>5-10</td>
<td>203</td>
<td>27</td>
<td>7</td>
<td>Wyoming</td>
</tr>
<tr>
<td>27</td>
<td>Wilson, Tavon</td>
<td>DB</td>
<td>6-0</td>
<td>210</td>
<td>22</td>
<td>R</td>
<td>Illinois</td>
</tr>
<tr>
<td>28</td>
<td>Gregory, Steve</td>
<td>S</td>
<td>5-11</td>
<td>200</td>
<td>29</td>
<td>7</td>
<td>Syracuse</td>
</tr>
</tbody>
</table>

Example: Search Key

NFL uniform numbers follow a specific scheme to make search by number easy for TV announcers:

- Numbers 1 to 19 are worn by quarterbacks, kickers, and punters.
- Numbers 20 to 49 are worn by running backs, tight ends, cornerbacks and safeties.
- Numbers 50 to 59 are worn by linebackers and offensive linemen.
- Numbers 60 to 79 are worn by members of both the offensive line and defensive line.
- Numbers 80 to 89 are worn by wide receivers and tight ends.
- Numbers 90 to 99 are worn by linebackers and defensive linemen.

<table>
<thead>
<tr>
<th>#</th>
<th>NAME</th>
<th>POS.</th>
<th>HT.</th>
<th>WT.</th>
<th>AGE</th>
<th>EXP</th>
<th>COLLEGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Ninkovich, Rob</td>
<td>DL</td>
<td>6-2</td>
<td>260</td>
<td>28</td>
<td>7</td>
<td>Purdue</td>
</tr>
<tr>
<td>51</td>
<td>Mayo, Jerod</td>
<td>LB</td>
<td>6-1</td>
<td>250</td>
<td>26</td>
<td>5</td>
<td>Tennessee</td>
</tr>
<tr>
<td>54</td>
<td>Hightower, Don't'a</td>
<td>LB</td>
<td>6-4</td>
<td>270</td>
<td>22</td>
<td>R</td>
<td>Alabama</td>
</tr>
<tr>
<td>55</td>
<td>Spikes, Brandon</td>
<td>LB</td>
<td>6-2</td>
<td>255</td>
<td>25</td>
<td>3</td>
<td>Florida</td>
</tr>
<tr>
<td>58</td>
<td>White, Tracy</td>
<td>LB</td>
<td>6-0</td>
<td>235</td>
<td>31</td>
<td>10</td>
<td>Howard</td>
</tr>
<tr>
<td>59</td>
<td>Rivera, Mike</td>
<td>LB</td>
<td>6-2</td>
<td>255</td>
<td>26</td>
<td>1</td>
<td>Kansas</td>
</tr>
</tbody>
</table>
Why Does Searching Matter?

Given a large enough set of data, it could take a long time to find something!

Example:
Consider a collection with 10,000,000 records (e.g. a phone book).
How would you find what you’re looking for?

Searching Algorithms
Algorithms that traverse the collection in an attempt to find the desired item.

Why Does Searching Matter?

Searching is an operation which can take a lot of computing cycles:

How many cycles?
It depends:
- How many items in the collection
- Choice of searching algorithm

What does this mean to the user?
A Naïve Searching Algorithm

While there are more items in the list:
   Look at the next item.
   Is this what you were looking for?
   If yes, all done.
   If no, take the next item.
If you got here and didn’t find the item, then it must not be in the list.

We call this **Linear Search**.

Linear Search Example

Consider the game of Scrabble.

A good Scrabble player can pick up a lot of extra points by knowing 2- and 3-letter words.
Linear Search Example

Consider searching a small dictionary, to see if a word exists. In the game of Scrabble, there are 931 valid 3-letter words.

Let's search for some words, and see how many comparisons it takes to find them (or to find that they don't exist).

Linear Search: Python Example

Define a list of all the valid 3-letter Scrabble words:

```python
# define the list of all valid 3-letter words:
```
Linear Search: Python Example

```python
query = raw_input("which word to search for? ")
query = query.upper()
found = False  # indicate successful search
index = 0  # index into the array (list)

# repeat the following steps until the word is found
# or we reach the end of the list, whatever happens first
while (not found) and (index < len(wordlist)):
    # print out the 'current' word we're looking at
    print "comparing",query,"to",wordlist[index]

    if query == wordlist[index]:
        # we've found it, so we'll stop the loop
        found = True

        # increment the index so we will check the next word
        index = index + 1

    if found:
        print query, "is a valid word."
    else:
        print query,"is NOT a valid word; you just made that word up!"
```

Linear Search

Characteristics of the Linear Search:
- Simple to implement.
- It doesn’t matter if the collection is sorted.

How many items do you need to look at to find what you’re looking for?
- *What if it is in the first position?*
- *What if it is in the last position?*
- *What if it is not found?*
Calculating the Running Time

How many steps are required to do a linear search through a list of \(n\) items?

- If we check each one, it will take looking through \(n\) items to complete the search.
- On average, we might expect to look at \(n/2\) items on a given search.
- If the item is not found, we will not discover that until the end of the list – after traversing all \(n\) items.

We call Linear Search an \(O(n)\) algorithm.

---

Running Time Analysis

![Graph showing run time analysis](image-url)
How else might we approach this problem?
Hint: can we divide and conquer?

**Divide and Conquer**

**Binary Search** uses a divide-and-conquer strategy.
It requires that the collection be sorted.

Basic strategy:
- Divide the list in half based on some mid point, to get two shorter lists (before and after mid point).
- Compare the search key to the mid point. Does it come before or after the midpoint? Search the corresponding sub list.
- Continue splitting and searching sub lists until we get a list of length 1:
  - either this is our item, or the item is not in the collection.
**Binary Search: Python Example**

```python
query = raw_input("which word to search for? ")
query = query.upper()
start = 0 # index the start of the search domain
end = len(wordlist) # index the end of search domain

while (end-start) > 1:
    mid = (end-start)/2 + start
    print "Comparing",query,"to",wordlist[mid]
    if query < wordlist[mid]:
        # query would come before mid
        print "Your word",query,"would be before",wordlist[mid]
        end = mid
    else:
        # query would come after mid
        start = mid
        print "Your word",query,"would be after",wordlist[mid]

if wordlist[start] == query:
    print query,"is a valid word."
else:
    print query,"is NOT a valid word; you just made that word up!"
```

**Calculating the Running Time**

How do we calculate the running time for Binary Search?

- Determine the number of comparisons.
- Determine the number of times we split the collection.

Each time we want to split the list, we need to make one comparison (search item to mid point), proceed to search a sub-list.
Running Time: Binary Search

How many times do we split a list of size $n$?
We keep splitting (in half) until we reach 1. How many splits is that?

For $n = 2$, splits = 1
For $n = 4$, splits = 2
For $n = 8$, splits = 3
For $n = 16$, splits = 4
For $n = 32$, splits = 5
For $n = 64$, splits = 6

What is the pattern here?

Running Time: Binary Search

Pattern: Each time we double the length of the list ($n$), we increase the number of splits by 1.

This is the opposite of the exponential relationship.
Recall that:

$2^2 = 2 \times 2 = 4$
$2^3 = 2 \times 2 \times 2 = 8$
$2^4 = 2 \times 2 \times 2 \times 2 = 16$

Generally: $2^x = n$
Recall: Logarithms

The base-2 logarithm describes how many times we need to divide a value in half to obtain 1:

\[ \log_2(2) = 1 \]
\[ \log_2(4) = 2 \]
\[ \log_2(8) = 3 \]
\[ \log_2(16) = 4 \]
\[ \log_2(32) = 5 \]

\[ \log_2(n) = x \]

where \( x \) is the power to which we would raise 2 to obtain \( n \):

\[ 2^x = n \]

Running Time: Binary Search

For a list of size \( n \), we have \( \log_2(n) \) splits.

Thus, Binary Search is an \( O(\log_2(n) ) \) algorithm.

\[ \log_2(n) < n \]

(this is true for all \( n > 1 \))
We can barely see the line for $\log_2(n)$!

Now we see $\log_2(n)$! Note: change of scale.
Aaron’s Card Trick...

Playing cards have a natural ordering...

Suits: Clubs, Diamonds, Hearts, Spades
Cards: A, 2, 3, 4, 5, 6, 7, 8, 9, J, Q, K

What You Learned Today

- Searching is...
- Search key
- Linear search
- Binary search
Announcements and To Do

- HW 12: make appointment to show it to Zhongchen
- No labs this week