Search Algorithms

2501ICT/7421ICTNathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2012

Outline

Search Algorithms

Search Algorithms

Search Algorithms Overview

- Many Search Methods Exist
  - choose one that meets time/space requirements and problem size!
- An Example
  - Array filled with values
- Three simple search methods
  - Linear search: if array items are *unordered*
  - Binary search: for *sorted* arrays
  - Hash lookup: *value* determines position

Linear Search

→ Linear Search is needed if items are in a *random order*

Linear Search Algorithm

1. start with the first position within the array (index 0)
2. compare target value with value at that position
   → if equal: finish by `return`ing item's position within array
3. otherwise repeat (loop) until end of array
4. if no item was found in the loop, return "not found", e.g. -1

Linear Search in Objective-C

Example (prints: 'Two' is at position 1)

```
#import <Foundation/Foundation.h>

int linearSearchForObjectInArray(id object, NSArray *array)
{
        for (int i = 0; i < [array count]; i++)                    // for each pos
                if ([[array objectAtIndex:  i] isEqual:  object]) // is equal?
                        return i;                                  // return position
        return -1;                                                 // not found
}


int main(int argc, char *argv[])
{
    @autoreleasepool
    {
        NSArray *array = [NSArray arrayWithObjects:  @"One", @"Two", @"Three", nil];

        id object = @"Two";      // an object to search for
        printf("'%s' is at position %d\n", [object UTF8String],
                linearSearchForObjectInArray(object, array));
    }

    return EXIT_SUCCESS;
}
```

Linear Search in C++

Example (prints: 'Two' is at position 0)

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int linearSearchForStringInVector(string &object, vector<string> &array)
{
        for (int i = 0; i < (int) array.size(); i++)            // for each pos
                if (array[i] == object)                         // is equal?
                        return i;                               // return position
        return -1;                                              // not found
}


int main(int argc, char *argv[])
{
        string object = "Two";                              // an object to search for
        vector<string> array(5, object);                     // an array of strings

        cout << object << " is at position ";
        cout << linearSearchForStringInVector(object, array) << endl;

        return EXIT_SUCCESS;
}
```

Linear Search using Templates in C++

Example (replacing `string` with a template)

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

template <class T> int linearSearchForObjectInVector(T &object, vector<T> &array)
{
        for (int i = 0; i < (int) array.size(); i++)            // for each pos
                if (array[i] == object)                         // is equal?
                        return i;                               // return position
        return -1;                                              // not found
}


int main(int argc, char *argv[])
{
        string object = "Two";                          // an object to search for
        vector<string> array(5, object);                 // an array of strings

        cout << object << " is at position ";
        cout << linearSearchForObjectInVector<string>(object, array) << endl;

        return EXIT_SUCCESS;
}
```

Linear Search Complexity

- Best Case
  - item found at the beginning: $O(1)$
- Worst Case
  - item found at the end: $O(n)$
- Average Case
  - Arithmetic average: $O(\frac{n+1}{2}) = O(n)$

Binary Search

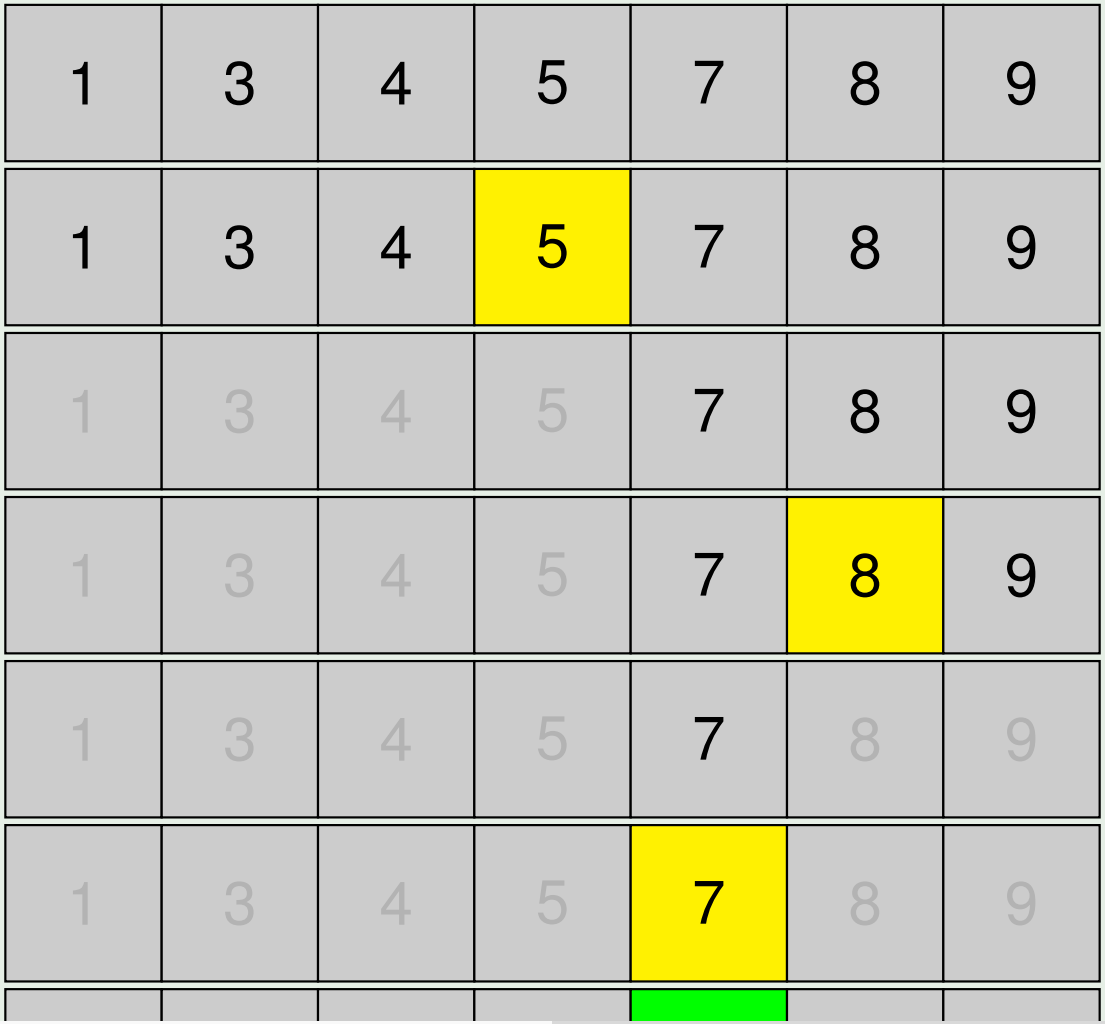
→ Items in the array *must be sorted*!

Binary Search Algorithm

1. Start with the *middle* position
2. Compare target value with value at that position
   → if equal: `return` item's position – we are done!
3. Partition array into two halves
   → one half before and one half after the current position
4. Repeat loop in lower half, is target is less than item, otherwise repeat in upper half
5. Return not found (e.g. -1) if the size of the half is zero (nothing left to search for)

Binary Search Example

Example (Searching for the value 7)



Binary Search in Objective-C

Example (prints: `3 is at position 2`)

```
#import <Foundation/Foundation.h>

int binarySearchForObjectInArray(id object, NSArray *array)
{
        int lo = 0, hi = [array count] - 1;                     // boundaries

        while (lo <= hi) {                                       // anything left?
                int i = (lo + hi) / 2;                          // mid point
                NSComparisonResult r = [[array objectAtIndex:  i] compare:  object];
                if (r == NSOrderedSame) return i;               // found?
                if (r == NSOrderedAscending) lo = i + 1;        // search upper half
                else hi = i - 1;                                // search lower half
        }
        return -1;                                              // not found
}

int main(int argc, char *argv[])
{
    @autoreleasepool
    {
        NSArray *array = [NSArray arrayWithObjects:  @"1", @"2", @"3", @"4", nil];
        printf("3 is at position %d\n", binarySearchForObjectInArray(@"3", array));
    }

    return EXIT_SUCCESS;
}
```

Binary Search in C++

Example (prints: `6 is at position 3`)

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

template <class T> int binarySearchForObjectInVector(T &object, vector<T> &array)
{
        int lo = 0, hi = array.size() - 1;              // boundaries

        while (lo <= hi) {                               // anything left?
                int i = (lo + hi) / 2;                  // mid point
                if (array[i] == object) return i;       // found?
                if (array[i]  < object) lo = i + 1;     // search upper half
                else hi = i - 1;                        // search lower half
        }
        return -1;                                      // not found
}

int main(int argc, char *argv[]) {
        vector<int> array(5, 0);                         // an array of ints
        for (int i = 0; i < 5; i++) array[i] = 2*i;      // initialise
        int x = 6;
        cout << x << " is at position ";
        cout << binarySearchForObjectInVector<int>(x, array) << endl;

        return EXIT_SUCCESS;
}
```

Binary Search Complexity

- Best Case
  - Item found in the middle of the array: O(1)
- Worst Case
  - Item found at very end or not at all
  - How often can you half the array?
    - $n/2/2 \cdots /2 \leq 1 \quad \rightarrow \quad \frac{n}{2^k} \leq 1 \quad \Rightarrow \; k = log_2 n \Rightarrow \quad \mathrm{O}(\log n)$
- Average = Worst Case: O(log n)

Recursive Binary Search

- → Works the same way as the iterative approach
- → Recursive function needs to know the current boundaries

Recursive Binary Search in Objective-C

Example (Recursive Binary Search Objective-C Example)

```
#import <Foundation/Foundation.h>

int doSearchForObjectInArray(id object, NSArray *array, int lo, int hi)
{
        if (lo > hi) return -1;                                  // not found

        int mid = (lo + hi) / 2;                                 // mid point
        NSComparisonResult r = [[array objectAtIndex:  mid] compare:  object];
        if (r == NSOrderedSame) return mid;                      // found?
        if (r == NSOrderedAscending) lo = mid + 1;               // search upper half
        else hi = mid - 1;                                       // search lower half

        return doSearchForObjectInArray(object, array, lo, hi); // recursive call
}


int recursiveBinarySearchForObjectInArray(id object, NSArray *array)
{
        int lo = 0, hi = [array count] - 1;                      // boundaries

        return doSearchForObjectInArray(object, array, lo, hi); // recursive function
}
```

Recursive Binary Search in C++

Example (Recursive Binary Search C++ Example)

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

template <class T> int doSearchForObjectInVector(T &object, vector<T> &array, int
lo, int hi)
{
        if (lo > hi) return -1;                                  // not found

        int mid = (lo + hi) / 2;                                 // mid point
        if (array[mid] == object) return mid;                    // found?
        if (array[mid]  < object) lo = mid + 1;                  // upper half
        else hi = mid - 1;                                       // lower half

        return doSearchForObjectInVector(object, array, lo, hi); // recurse
}


template <class T> int recursiveBinarySearchForObjectInVector(T &object, vector<T>
&array)
{
        int lo = 0, hi = array.size() - 1;                       // boundaries

        return doSearchForObjectInVector(object, array, lo, hi); // recursive func
}
```

Hash Tables

- Is there a faster way to find elements than binary search?
  - yes, but elements need to be ordered in a specific way
- O(1) requires that index is found in a single step
  - → independent of the number of elements in the array
- Hash Table
  - calculate table index from an object's value
  - → `hash` method in Objective-C
  - → internal `hash` function in C++

A simple Hash Function

Example (prints: 'Test' has a hash value of 1099)

```
#include <stdio.h>
#include <string.h>

unsigned simple_hash(const char *string)                    // simple hash function
{
        unsigned n = strlen(string);                        // string length
        unsigned hash = n;                                  // initial hash value
        for (unsigned i = 0; i < n; i++)                     // for each character
                hash += (i+1) * string[i];                  // multiply by pos and add
        return hash;
}


int main(int argc, char *argv[])
{
        char *string = "Test";                              // a simple string

        unsigned hash = simple_hash(string);                // calculate hash

        printf("'%s' has a hash value of %u\n", string, hash);

        return 0;
}
```

Hash Function Complexity

- Best Case
  - String has only one character: O(1)
- Average and Worst Case
  - String has k characters: O(k)

  $\rightarrow$ k does not depend on the number of strings n $\Rightarrow$ O(1)
- `simple_hash()` is a low quality hash function

  $\rightarrow$ different strings are likely to result in same hash value

  $\Rightarrow$ better to use high quality hash functions (e.g. Objective-C `hash` method)
- Cryptographic hash methods
  - focus on security rather than performance
  - statistically uniform hash methods
  - high bit count (160 bits or more)

  $\Rightarrow$ goal: impossible to engineer an object with a given hash value

Using a Hash Method

Example (output depends on hash and capacitydepends on hash method and capacity)

```
#import <Foundation/Foundation.h>

#define CAPACITY        10                                  // must be known upfront
#define INDEX(object)   ([object hash] % CAPACITY)          // calculate array index
#define ADD(table,obj)  (table[INDEX(obj)] = (obj))         // store object

int main(int argc, char *argv[])
{
    @autoreleasepool
    {
        id table[CAPACITY];                                 // fixed size hash table

        ADD(table, @"One");                                 // add some objects
        ADD(table, @"Two");
        ADD(table, @"Three");

        id object = @"Three";                               // an object to search for

        printf("'%s' is at position %lu\n", [object UTF8String], INDEX(object));
    }

    return EXIT_SUCCESS;
}
```

Hash Table Caveats

- Table capacity cannot be changed
  - index depends on hash function and table size
- Difficult to write a high quality hash function
  - → different object values may result in same hash value
- Collisions are possible
  - two different objects with the same hash value!
  - → high quality hash function makes collision less likely
  - → table fill level should not exceed appx. 60 %
  - → collisions need to be detected and handled
- No Fuzzy Matching
  - hash values are calculated from exact object values
  - → no wildcards or regular expressions can be searched for

Use of Hash Tables

- Dictionaries (Maps)
  - store objects associated with keys
  - immediately find an object for a given key
- `NSDictionary` and `NSMutableDictionary` in Objective-C
  - `- objectForKey:` and `- setObject:ForKey:`
  - `+ dictionaryWithCapacity:` fixed initial size
    - → highly efficient if final capacity is not exceeded
- `std::map` in C++
  - `insert()` and `find()` access methods
  - overloaded array operators: “associative arrays”
  - → uses binary trees instead of hash maps
    - ⇒ O(log n) complexity (same as binary search)
    - → Standard hash table classes in C++11: `unordered_map`

NSDictionary Example

Example (an English/German dictionary in Objective-C)

```
#import <Foundation/Foundation.h>

int main(int argc, char *argv[])
{
    @autoreleasepool
    {
        NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
        /*
         *      @"German:",     @"English:",
         */
                @"Eins",        @"One",
                @"Zwei",        @"Two",
                @"Drei",        @"Three",
                nil];

        id key = @"Three";                          // a key to search for
        id value = [dict objectForKey:  key];     // its corresponding value

        printf("The German translation for '%s' is '%s'\n",
                        [key UTF8String], [value UTF8String]);
    }

    return EXIT_SUCCESS;
}
```

std::map Example

Example (an English/German dictionary in C++)

```
#include <iostream>
#include <map>

using namespace std;

int main(int argc, char *argv[])
{
        map<const char*, const char *> dict;

        /*
         *   English      German
         */
        dict["one"]     = "eins";
        dict["two"]     = "zwei";
        dict["three"]   = "drei";

        const char *key = "three";              // a key to search for
        const char *value = dict[key];          // its corresponding value

        cout << "The German translation for ";
        cout << key << " is "<< value << endl;

        return EXIT_SUCCESS;
}
```