

CSE1301 Computer Programming Lecture 31: List Processing (Search)

1

Topics

- An array as a list
- Searching
 - Linear search
 - Binary search (sorted list)
- Efficiency of an algorithm

2

Arrays as Lists

- An array
 - stores several elements of the same type
 - can be thought of as a list of elements:

```

13
5
19
10
7
27
17
1
    int a[8]
    [13 5 19 10 7 27 17 1]
    
```

3

Linear Search

- Problem: determine if an element is present in an array
- Method:
 - start at one end
 - look at each array element until the sought element is found
- Also called *sequential search*

4

Linear Search: Algorithm and Code

<pre> isPresent (array, val, arraySize) { set count to 0 while (not yet processed all array elements) { if (current array element is val) { return true } increment count } return false } </pre>	<pre> int isPresent (int *arr, int val, int N) { int count; for (count=0; count<N; count++) { if (arr[count]==val) { return 1; } } return 0; } </pre>
---	---

5

Linear Search -- Exercise

- How would you modify the program so that it returns the position of the sought item (i.e., **findPosition** rather than **isPresent**)?
- How would you indicate “not found”?

6

What does Efficiency Mean?

- Algorithm: a set of instructions describing how to do a task
- Program: an implementation of an algorithm
- **Complexity theory** describes the time and space used by an algorithm

The time and space requirements of an algorithm enable us to measure how efficient it is

7

Types of Computer Resources

- **Time**: elapsed period from start to finish of the execution of an algorithm
- **Space (memory)**: amount of storage required by an algorithm
- **Hardware**: physical mechanisms required for executing an algorithm

8

How to Measure Efficiency?

- Use your watch? Use the computer clock?
- Not a good idea, because:
 - What if you run your program on different computers?
 - Your program may also wait for I/O or other resources
 - While running a program, a computer performs many other computations
 - Depends on programming/coding skill

9

Abstract Notion of Efficiency

- We are interested in the number of steps executed by an algorithm
 - step \approx execution of an instruction
- The running time of an algorithm is proportional to the number of steps executed by the algorithm
- Running time is given as a function of the size of the input data: "**Big-O Notation**"

10

Big-O Notation

- Big-O notation is a function of the size of the input
- Example:
 - Input: N integers
 - Algorithm complexity:
 - Constant $O(1)$
 - Logarithmic $O(\log N)$
 - Linear $O(N)$
 - $n \log(n)$ $O(N \log N)$
 - Quadratic $O(N^2)$
 - Cubic $O(N^3)$
 - Exponential $O(2^N)$

11

Calculating Complexity with the Big-O Notation

- Simplify and choose the highest term
- Examples:
 - $2 + 3N + 10N + 3N^2 + 100$
 $= 3N^2 + 13N + 102 \approx O(N^2)$
 - $40N + N^3 \approx O(N^3)$
 - $25 \approx O(1)$

12

Binary Search -- Example 3

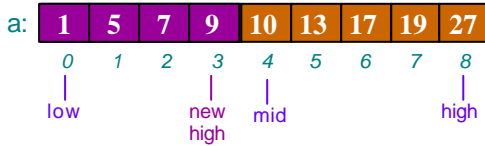
Case 3: $val < a[mid]$

$val = 7$

$low = 0, high = 8$

$mid = (0 + 8) / 2 = 4$

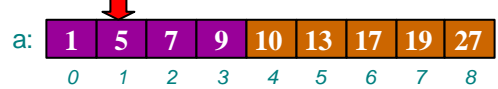
$new\ high = mid - 1 = 3$



19

Binary Search -- Example 3 (cont)

$val = 7$



20

Binary Search -- Algorithm and Code

<pre>isPresent(array, val, arraySize) { set low to first array position set high to last array position while (low <= high) { set mid to half of low + high if (array element in mid is val) { return true } else if (middle value < val) { set low to mid + 1 } else { set high to mid - 1 } } return false }</pre>	<pre>int isPresent(int *arr, int val, int N) { int low = 0; int high = N - 1; int mid; while (low <= high) { mid = (low + high)/2; if (arr[mid]==val) { return 1; } else if (arr[mid] < val) { low = mid + 1; } else { high = mid - 1; } } return 0; }</pre>
---	--

21

Binary Search: Exercise

- What happens if the sought value is not in the list?
- How would you modify the code so that it returns the position of the sought item (i.e., `findPosition` rather than `isPresent`)?

22

Binary Search Efficiency

- What is the size of the input data?
 - The size of the array being searched is N
- What is the *time complexity* of this algorithm?
 - Each time through the loop we perform
 - 3 comparisons
 - 3 arithmetic operations
 - 2 assignments
 - Total: 8 operations

23

Binary Search Efficiency (cont)

- Best case?
 - item is in the middle
 - 5 operations $\approx O(1)$
- Worst case?
 - item is not found
 - $8 \times \log_2 N$ operations $\approx O(\log_2 N)$
- Average case?
 - $O(\log_2 N)$

24

Calculating the Worst Case Complexity

- After 1 bisection $N/2$ items
- After 2 bisections $N/4 = N/2^2$ items
- . . .
- After i bisections $N/2^i = 1$ item

$$i = \log_2 N$$

25

Exercise

Problem: How would you implement linear search or binary search over an array of structs?

Method: The array must be sorted by ID, name or mark, depending on the search key

26

Exercise (cont)

```

struct studentRec
{
    int      IDNumber;
    char     name[NAMELEN];
    float    mark;
};
typedef struct studentRec Student;

struct classRec
{
    int      count;
    Student  student[MAX_STUDENTS];
};
typedef struct classRec ClassType;

ClassType  class;
Student    findStudent(ClassType *class, int IDNum)
{
    ...
}

```

27

Notes on Searching

- Linear search can be done on any (sorted or unsorted) list, but it is inefficient
- Binary search
 - requires a list to be sorted
 - is more efficient
- Sorting a list: [Lecture 31](#)

28

Reading

- Forouzan and Gilberg, Section 8.6
- Knuth, *Sorting and Searching*, Section 6.2.1 (ed. 2)

29