122COM: Searching algorithms

David Croft
david.croft@coventry.ac.uk

February 5, 2016

1 Introduction

This week we are looking at the problem of locating elements in a data structure. Searching is a classic problem in computer science and there are a range of different approaches available. From this week onwards the majority of labs will be taught in a combination of C++ and Python.

BIT, Media and 110 resit students
Unless stated otherwise, BIT, Media and 110 resit students can solve the lab problems using Python. If you want to use C++ you can but it is not a requirement.

Everyone else (e.g. Computer Science, Computing, Ethical Hacking, Games)
You are expected to use C++. If you want, you can solve the problems in Python first and then adapt your code to C++ but the final version should be in C++.

This weeks work is being kept relatively light as the majority of students will still be getting used to programming in C++. If you are struggling with the move to C++ or with programming in general then you should be attending programming support sessions.

The source code files for this lab and the lecture are available at:

```
https://github.com/dscroft/122COM_searching_algorithms.git

git@github.com:dscroft/122COM_searching_algorithms.git
```

2 Linear search

Linear search is the simplest method for finding a specific value in a sequence.

Starting at the beginning of the sequence of elements, step through and compare each element to the value being search for until either the end of the sequence is reached or a matching element is found.

Pre-lab work:

1. Complete the pre_linear_search program.
   - The function `linear_search` should return true if `value` is in `sequence` and false otherwise.
   - There are starter programs in C++ and Python already written for you.
   - You cannot use any built in find functions such as `in` or `find`.
     E.g. anything like the example below is clearly not acceptable.

```
def linear_search( sequence, value ):
    if value in sequence:
        return True
    else:
        return False
```

2. Complete pre_string_search program.
   - 110 resit students can skip this task and come back to it later, it is more important for you to complete the binary search question.
   - The function `search_string` should return true if `substr` is in `text` and false otherwise.
   - Use the naive string search method shown in the lecture slides where the substring is tested in every possible position.
   - Notice that the provided code comes with a series of tests to check that your function is working correctly.

3 Binary search

Binary search is a faster method of searching a sequence than linear search. Binary search is a divide and conquer algorithms, it is, however, limited to searching sorted sequences.

A full demonstration of the binary search algorithm is in the lecture slides but as a reminder:

1. Find middle value of sequence.
2. If search value == middle value then success.
3. If search value is < middle value then forget about the top half of the sequence.
4. If search value is > middle value then forget about the bottom half of the sequence.
5. Repeat from step 1 until `len(sequence)==0`.

Lab work:

3. Write a program that will binary search an ordered, linear, indexed sequence.
    - You can use either a recursive or an iterative solution.
    - lab_binary_search_recursive and lab_binary_search_iterative have been provided to get you started.
    - Note that the supplied code comes with a series of tests to check that your code works correctly.
4. Write a program that reads in and stores integer values provided by a user.
    - The program should keep reading until the user enters a value for a second time.
    - Some starter code can be found in lab_duplicates.
    - Write some tests to confirm that your code works correctly.

Extended work:

5. Modify your duplicates code so that when it encounters a duplicate, it prints out all the values the user entered in reverse order.
6. Try and implement the Booyer-Moore algorithm as explained in the lecture.
    - If researching elsewhere you may find Booyer-Moore described in terms of two rules, the ”bad character rule” and the ”good suffix rule”. Ignore the ”good” rule and focus just on the ”bad” rule.