Recursion, Search, Selection

Cheng, Wei  COMP110-001  June 12, 2014
Today

• Introduction to Recursion

• Introduction to Search & Selection

– Not a major focus of the final exam
– But you should be able to understand the code given in slides (and know how to use the code in similar problems by making slight modification).
Recursion

• Whenever an algorithm has one subtask that is a smaller version of the entire algorithm’s task, it is said to be recursive.

• **Recursion**: you write a method to solve a big task, and the method invokes itself to solve a smaller subtask

• E.g., I want to eat 5 apples now. My subtask can be eating 4 apples, eating 3 apples, eating 2 apples, et.....

• To eat 5 apples, I can do:
  – Eat 3 apples + Eat 2 apples
  – Eat 1 apple + Eat 4 apples
Recursion

- Eating 1 apple is the smallest task I can have. I cannot divide it anymore.
- This is the base case in recursion.

- Recursion is to divide a big task into smaller tasks. Smaller tasks are then divided further. Until we reach base case.
Recursion

• Let’s start with the simplest example: calculating factorial
  – Factorial(n) = n * (n-1) * (n-2) * ..... * 3 * 2 * 1

• How do you solve a task with smaller task(s)?
  – Factorial(n) = n * Factorial(n-1)
Recursion

- Translate this into Java code

```java
public static int factorial(int n) {
    if (n==1) return 1; // base case
    else return n * factorial(n-1);
}
```
Recursion

• The recursion form can be more natural in many problems (than using loops)

• Some problems can be hard to formulate using naïve looping (but such problems are beyond the scope of this course)

• Let’s see more recursion examples:

• Digits to Words from textbook
Recursion: Digits to Words

• Define a method that takes a single integer as an argument and displays the digits of that integer as words.

• For example, if the argument is the number 223, the method should display:
  
two two three

• Base case?
• Recursive rule?
Recursion: Digits to Words

• Base case: only 1 digit
  – print word for 1 digit

• Recursive rule:
  Print words for $n$ digits ->
  
  (print words for first $n-1$ digits) + (print word for last digit)
public static void displayAsWords( int number )
{
    if (number < 10) // base case
        System.out.print(getWordFromDigit(number) + " ");
    else  //number has two or more digits
    {
        displayAsWords(number / 10);
        System.out.print(getWordFromDigit(number % 10) + " ");
    }
}

You should be able to write out: getWordFromDigit(int num)
Recursion: Digits to Words

• `displayAsWords(987);`

```java
// Code for invocation of displayAsWords(987)
if (987 < 10)
    System.out.print(getWordFromDigit(987) + " ");
else // 987 has two or more digits
{
    displayAsWords(987 / 10);
    System.out.print(getWordFromDigit(987 % 10) + " ");
}

// Code for invocation of displayAsWords(98)
if (98 < 10)
    System.out.print(getWordFromDigit(98) + " ");
else // 98 has two or more digits
{
    displayAsWords(98 / 10);
    System.out.print(getWordFromDigit(98 % 10) + " ");
}

// Code for invocation of displayAsWords(9)
if (9 < 10)
    System.out.print(getWordFromDigit(9) + " ");
else // 9 has two or more digits
{
    displayAsWords(9 / 10);
    System.out.print(getWordFromDigit(9 % 10) + " ");
}
```
Search

• A simple problem

• Given a list of numbers (in an array), how do you search for a number?
  – Return Yes & location if the number is found in the array
  – Return No if the number is not found
Sequential Search

• Basic idea (from wiki)
  – For each item in the list:
    • if that item has the desired value, stop the search and return the item's location.
  – Return *Not Found*.

• Can you do better than this (by making it faster)?

• The general answer is no
  – No assumptions made on array
  – In worst case, have to examine each array element at least once
• What about sorted array? (numbers are in sorted order)

• Can you make the linear search faster?
• Let’s see an example:
Given $n$ numbers:

- In linear search, I explore one possible choice in each iteration.
  - Worst case: $n$ comparisons needed.

- With the new search algorithm (which only works on sorted array), I can eliminate half of the search space in each iteration!
  - How many comparisons do I need in the worst case?
int binary_search(int A[], int key, int imin, int imax) {
    // test if search range is empty
    if (imax < imin) {
        return KEY_NOT_FOUND; // set is empty
    } else {
        // calculate midpoint to cut set in half
        int imid = midpoint(imin, imax);
        // three-way comparison
        if (A[imid] > key) // key is in lower subset
            return binary_search(A, key, imin, imid-1);
        else if (A[imid] < key) // key is in upper subset
            return binary_search(A, key, imid+1, imax);
        else // key has been found
            return imid;
    }
}
Search

- Search algorithm is a very big topic
- We just covered two simplest cases:
  - Linear search in array of numbers
  - Binary search in sorted array
- A lot more with different data structures:
  - Search in graphs and trees (computer science concepts, not the usual graph/tree)
  - E.g., search for a move in chess game / tic-tac-toe game
  - Search for relations/patterns in social network communication graph
Selection

• Simplest Selection Problem:
  – Find the smallest / largest number in a given list (array)
  – No assumption made on the list (so it is not sorted)

• We have solved this in lab 4
  – Loop through each element, keep the largest/smallest

• Let’s relax the problem a bit
• Find the k-th smallest (or largest) element in a list of numbers

• How to solve this problem?
  – Go through each element, for each element, check its position in list
    • How many operations in the worst case
  – Sort array first. Then get the k-th element
    • How many operations in the worst case
Quickselect (quick in practice, but not in the worst case)

- To find $k$-th smallest number in $n$ numbers:
  - Randomly pick a number from the list, call it $p$
  - Partition the array into two parts:
    - Numbers that are $< p$ ($m$ numbers)
    - Numbers that are $> p$ ($n - m - 1$ numbers)
  - If $m == k - 1$, $p$ is the $k$-th smallest
  - If $m > k$, find the $k$-th smallest in the $m$ numbers
  - If $m < k$, find the $(k - m - 1)$-th smallest in the $(n-m-1)$ numbers

- On average, this requires $\sim n \times \text{constant}$ operations
- But in the worst case, it is $\sim n^2 \times \text{constant}$
There is a worst-case linear algorithm for k-th smallest element selection (Median of Medians algorithm)

Published in 1973

For search & selection, we used numbers / array for simplicity.

When working with real data (in other complicated forms), you can apply the same idea if you can identify the same abstract problem behind.