CS171 Introduction to Computer Science II

Recursion

Li Xiong
Announcement

• Hw3 is extended to Monday
• Quiz 1 will be distributed Tuesday
Recursion

• Recursion concept
• Examples
  – Factorial
  – Fibonacci
  – GCD
  – Recursive graph Htree
• More examples
  – Binary search
  – Tower of Hanoi
• Cost analysis of recursive algorithms
• Divide and conquer
Recursive Method

• A method that calls itself (direct recursion)
• Every recursive method must have a base case that is not recursive

```java
void recursiveMethod() {
    ...
    if (base case) {
        ...
    }
    else {
        ...
        recursiveMethod();
        ...
    }
}
```
Divide and Conquer

- A big problem is divided into two or more problems with smaller sizes (sub-problems).
- To solve a sub-problem, you again divide it into even smaller problems.
- The process continues until you get to the base case, which can be solved trivially.
Recursion as a problem solving

- Solve one or more problems that are identical to the original problem but with smaller size
- Solve original problem by solutions of smaller problems
Binary Search in Ordered Array

• Compares the middle element with search key and reduces the search range by half in each iteration
public int find(long searchKey) {
    return recFind(searchKey, 0, nElems-1);
}

private int recFind(long searchKey, int lowerBound, int upperBound) {
    int curIn;
    curIn = (lowerBound + upperBound) / 2;
    if (a[curIn] == searchKey) // found it
        return curIn;
    else if (lowerBound > upperBound) // can’t find it
        return nElems;
    else if (a[curIn] < searchKey) // upper half
        return recFind(searchKey, curIn+1, upperBound);
    else // lower half
        return recFind(searchKey, lowerBound, curIn-1);
}
Recursion vs. Iteration

- Recursion: calls itself one or more times until a condition is met
- Iteration: repeating for a specified number of times or until a condition is met
- Every recursive function can be transformed to an iterative function using a stack and vice versa
- Recursion is an elegant way to solve many practical problems but usually sacrifices memory and computational efficiency (what is the overhead?)
The Towers of Hanoi
The Towers of Hanoi

Move all the discs from the leftmost peg to the rightmost one.
  - Only one disc may be moved at a time.
  - A disc can be placed either on empty peg or on top of a larger disc.

Is world going to end (according to legend)?
  - 40 golden discs on 3 diamond pegs.
  - World ends when certain group of monks accomplish task.

Edouard Lucas (1883)
The Towers of Hanoi

• How do we move the disks to achieve the goal?

• We also want to know in general, how many steps it takes to move N disks.

• Let’s play with it to get some intuition:
  – N=1, N=2, N=3, N=4

http://www.mazeworks.com/hanoi/
The Towers of Hanoi

- Let’s call the initial pyramid-shaped arrangements of disks on column A a **tree**.
- We call a smaller set of the disks a **subtree**.
- It turns out that the intermediate steps in the solution involves moving a subtree.
The Towers of Hanoi

- Idea:
  - Assume, for now, that you have a (magical) way of moving a subtree from A to B via C;
  - Then you move A to C;
  - Finally move the subtree from B to C via A.
  - This is similar to the 2-disk case, except the top disk is now a subtree.

- How can we move the subtree?
- What’s the base case?
a) Move subtree to B

b) Move bottom disk to C

c) Move subtree to C

Puzzle is solved!
The Towers of Hanoi

• Suppose you want to move n disks from a source tower S to a destination tower D, via an intermediate tower I.

• Initially
  – Source tower is A
  – Destination tower is C
  – Intermediate tower is B
The Towers of Hanoi

1. Move the subtree consisting of the top $n-1$ disks from $S$ to $I$;
2. Move the remaining (largest) disk from $S$ to $D$;
3. Move the subtree from $I$ to $D$. 

public class Hanoi {

    static void hanoi(int n, char from, char temp, char to) {
        if (n == 0) return;
        hanoi(n-1, from, to, temp);
        System.out.println("Move disc " + n);
        System.out.println(" from " + from + " to " + to);
        hanoi(n-1, temp, from, to);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        hanoi(N, 'A', 'B', 'C');
    }
}
General Approach: Recursion Tree
The Tower of Hanoi

• How many steps does the solution take to solve a N-disk problem?
  • N=1:
  • N=2:
  • N=3:
  • N=4:

• When is the world going to end (N=64)?
The Tower of Hanoi

• How many steps does the solution take to solve a N-disk problem?
  • N=1: 1 step
  • N=2: 3 steps
  • N=3: 7 steps
  • N=4: 15 steps

• When is the world going to end (N=64)?
Using Recurrence Relation for Cost Analysis

• Define the cost function $T(N)$ using recurrence relation and define base cases
• Solve the recurrence relation
• Derive Big O function
Defining **Recurrence Relation**

- A recurrence relation is an equation that recursively defines a sequence: each term of the sequence is defined as a function of the preceding terms.
- Examples:
  - \( T(n) = T(n-1) + 1 \)
Solving Recurrence Relations

- Rewrite $T(N)$, $T(N-1)$, $T(N-2)$, ..., with the recurrence formula
- Discover the patterns and find an expression
- Check the correctness
  - Substitute solution in initial conditions
  - Substitute solutions in the recurrence relation
Example

• Loop
  
  ```java
  for (i=0; i<N; i++) {
      do_something();
  }
  ```

• Recurrence relation
  
  • $T(n) = T(n-1) + 1$
  
  • $T(1) = 1$
Example (cont’d)

• $T(n) = T(n-1) + 1$
• $T(1) = 1$

• Expansion
  – $T(n) = T(n-1) + 1 = T(n-2) + 1 + 1 = T(n-3) + 1 + 1 + 1 = \ldots$

• Discover pattern and solution
  – $T(n) = T(1) + n - 1 = n$

• Verification
  – $T(1) = 1$
  – $T(n) = T(n-1) + 1$
Binary Search Example

- Binary Search Cost Function
- \( T(n) = T(n/2) + 1 \)
- \( T(1) = 1 \)
Binary Search Example: Solution

• Binary Search Cost Function
• \( T(n) = T(n/2) + 1 \)
• \( T(1) = 1 \)

• \( N=2^k \)
• \( T(2^k) = T(2^{k-1}) + 1 = T(2^{k-2}) + 1 + 1 + ... \)
• \( T(2^k) = T(2^0) + k = 1 + k \)
• \( T(N) = 1 + \lg N \)
The Tower of Hanoi

• How many steps does the solution take to solve a N-disk problem?
• Use recursive formula
  \[ T(N) = T(N-1) + 1 + T(N-1) = 2*T(N-1) + 1 \]
• So how to solve this?

\[ T(N) = 2^{N-1} + 2^{N-2} + 2^{N-3} + ... + 1 = 2^N - 1 \]

• So it takes an exponential number of steps!
When is the world going to end?

• Takes 585 billion years for N = 64 (at rate of 1 disc per second).
Summary

• Recursion: powerful tool allows for elegant solutions
  – But, computational overhead is high

• Can analyze runtime:
  – Intuition: draw recursive call tree
  – Analysis: Recurrence relations