CSE 417: Algorithms and Computational Complexity

4: Dynamic Programming, I
Fibonacci

Winter 2006
Lecture 12
W. L. Ruzzo
Some Algorithm Design Techniques, I

• General overall idea
  – Reduce solving a problem to a smaller problem or problems of the same type

• Greedy algorithms
  – Used when one needs to build something a piece at a time
  – Repeatedly make the greedy choice - the one that looks the best right away
    – e.g. closest pair in TSP search
  – Usually fast if they work (but often don't)
Some Algorithm Design Techniques, II

• Divide & Conquer
  – Reduce problem to one or more sub-problems of the same type
  – Typically, each sub-problem is at most a constant fraction of the size of the original problem
    • e.g. Mergesort, Binary Search, Strassen’s Algorithm, Quicksort (kind of)
Some Algorithm Design Techniques, III

• Dynamic Programming
  – Give a solution of a problem using smaller sub-problems, e.g. a recursive solution
  – Useful when the same sub-problems show up again and again in the solution
“Dynamic Programming”

Program — A plan or procedure for dealing with some matter

– Webster’s New World Dictionary
A simple case: Computing Fibonacci Numbers

• Recall $F_n = F_{n-1} + F_{n-2}$ and $F_0 = 0$, $F_1 = 1$

• Recursive algorithm:
  – $\text{Fibo}(n)$
    
    \begin{verbatim}
    if n=0 then return(0)
    else if n=1 then return(1)
    else return(Fibo(n-1)+Fibo(n-2))
    \end{verbatim}
Call tree - start

F (6)
  /  
F (5)  F (4)
 /  
F (4)  F (3)
 /  
F (3)  F (2)
 /  
F (2)  F (1)
 /  
F (1)  F (0)
 /  
1 1
1 0
Full call tree
Memo-ization (Caching)

• Remember all values from previous recursive calls
• Before recursive call, test to see if value has already been computed
• Dynamic Programming
  – Convert memo-ized algorithm from a recursive one to an iterative one (top-down → bottom-up)
Fibonacci - Memo-ized Version

initialize: F[i] ← undefined for all i
F[0] ← 0
F[1] ← 1
FiboMemo(n):
    if(F[n] undefined) {
        F[n] ← FiboMemo(n-2)+FiboMemo(n-1)
    }
    return(F[n])
Fibonacci - Dynamic Programming Version

FiboDP(n):
    F[0] ← 0
    F[1] ← 1
    for i=2 to n do
        F[i] ← F[i-1]+F[i-2]
    endfor
    return(F[n])
Dynamic Programming

• Useful when
  – same recursive sub-problems occur repeatedly
  – Can anticipate the parameters of these recursive calls
  – The solution to whole problem can be figured out without knowing the internal details of how the sub-problems are solved
    • principle of optimality