CSE 417: Algorithms and Computational Complexity

Winter 2001

Lecture 6

Instructor: Paul Beame

TA: Gidon Shavit

Algorithm Design Techniques

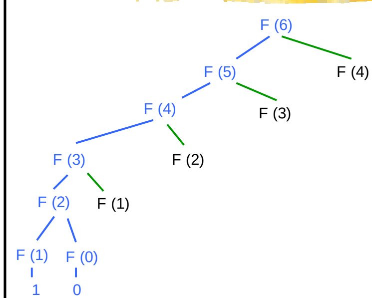
- Dynamic Programming
  - Given a solution of a problem using smaller sub-problems, e.g. a recursive solution
  - Useful when the same sub-problems show up again and again in the solution

A simple case: Computing Fibonacci Numbers

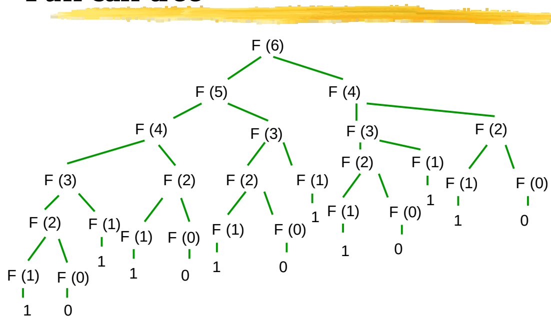
- Recall $F_n=F_{n-1}+F_{n-2}$ and $F_0=0$, $F_1=1$
- Recursive algorithm:
  - Fibo(n)

```
if n=0 then
   return(0)
else if n=1 then
   return(1)
else
   return(Fibo(n-1)+Fibo(n-2))
```


Call tree - start



Full call tree



Memo-ization

- 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

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-1]
  endfor
  return(F[n])

7

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 with knowing the internal details of how the sub-problems are solved
    - principle of optimality

8

List partition problem

- Given: a sequence of n positive integers $s_1,\dots,s_n$ and a positive integer k
- Find: a partition of the list into up to k blocks:
  $s_1,\dots,s_{i_1}|s_{i_1+1}\dots s_{i_2}|s_{i_2+1}\dots s_{i_{k-1}}|s_{i_{k-1}+1}\dots s_n$
  so that the sum of the numbers in the largest block is as small as possible.
  i.e. find spots for up to $k-1$ dividers

9

Greedy approach

- Ideal size would be $P=\sum_{i=1}^{n} s_i/k$
- Greedy: walk along until what you have so far adds up to P then insert a divider
- Problem: it may not exact (or correct)
  - 100 200 400 500 900 | 700 600 | 700 600
  - sum is 4800 so size must be at least 1600.

10

Recursive solution

- Try all possible values for the position of the last divider
- For each position of this last divider
  - there are $k-2$ other dividers that must divide the list of numbers prior to the last divider as evenly as possible
    - $s_1,\dots,s_{i_1}|s_{i_1+1}\dots s_{i_2}|s_{i_2+1}\dots s_{i_{k-1}}|s_{i_{k-1}+1}\dots s_n$
  - recursive sub-problem of the same type

11

Recursive idea

- Let $M[n,k]$ the smallest cost (size of largest block) of any partition of the n into k pieces.
- If between the i^{th} and $i+1^{st}$ is the best position for the last divider then
  $M[n,k]=\max(\underbrace{M[i,k-1]}_{\text{max cost of 1st k-1 blocks}}, \overbrace{\sum_{j=i+1}^{n} s_j}^{\text{cost of last block}})$
- In general
  - $M[n,k]=\min_{i<n} \max(M[i,k-1], \sum_{j=i+1}^{n} s_j)$

12

Time-saving - prefix sums

- Computing the costs of the blocks may be expensive and involved repeated work
- Idea: Pre-compute prefix sums
  - $p[1]=s_1$ $p[2]=s_1+s_2$ $p[3]=s_1+s_2+s_3$ … $p[n]=s_1+s_2+...+s_n$
  - cost: n additions, space n
  - Length of block $s_{i+1}+... + s_j$ is just $p[j]-p[i]$

Linear Partition Algorithm

- Partition(S,k):

p[0]←0; **for** i=1 **to** n **do** p[i] ←p[i-1]+s_i
for i=1 **to** n **do** M[i,1] ←p[i]
for j=1 **to** k **do** M[1,j] ← s_1
for i=2 **to** n **do**
    for j=2 **to** k **do**
        M[i,j]←$\min_{pos<i}$\{max(M[pos,j-1], p[i]-p[pos])\}
        D[i,j] ←value of pos where min is achieved

Linear Partition Algorithm

- Partition(S,k):

p[0]←0; **for** i=1 **to** n **do** p[i] ←p[i-1]+s_i
for i=1 **to** n **do** M[i,1] ←p[i]
for j=1 **to** k **do** M[1,j] ← s_1
for i=2 **to** n **do**
    for j=2 **to** k **do**
        M[i,j]←∞
        for pos=1 **to** i-1 **do**
            s←max(M[pos,j-1], p[i]-p[pos])
            if M[i,j]>s **then**
                M[i,j] ←s ; D[i,j] ←pos