Lecture 18: Friday February 14, 2003

today

- dynamic programming [CLRS Ch. 15.1-3]
  - chained matrix multiplication

soon

- union find [CLRS Ch. 21]
- graph algorithms [CLRS Ch. 22,23]

recall: dynamic programming

- an algorithm design technique
- DP: avoiding recomputation of repeated subproblems by storing subproblem answers in tables/arrays

memoization: a DP improvement on recursion

- idea: keep recursion, avoid recomputation ...
  - store f( ) values in array F[ ]
  - if F[j] not yet initialized, compute it
  - if F[j] is initialized, access it
  - F[j] computed **only once**
- memoization: recursion with DP

a d.p. example: computing Fibonacci numbers

memoization: a DP improvement on recursion

- idea: keep recursion, avoid recomputation …
  - store f( ) values in array F[ ]
  - if F[j] not yet initialized, compute it
  - if F[j] is initialized, access it
  - F[j] computed **only once**
- memoization: recursion with DP

- 2nd implementation: memoization (recursion with DP)

```
proc dpfib(n)                                    f5
  proc dpf(n)                                   /  \
    if F[n]=? then                            f4    f3
      F[n]<-dpf(n-1)+dpf(n-2)                /  \
    return F[n]                            f3    f2
                                          /  \
                                        f2    f1
  for j<-1 to n do                     /  \
    F[j]<-?                          f1    f0
  F[0]<-0; F[1]<-1
  dpf(n)
```

- ☺ fewer repeated function calls
- for each k, dpf(k) called $\leq$ twice ($F[k]$ known after 1st call)
- time $T_n \in \Theta(n)$ ☺

- 3rd implementation: full dynamic programming (with iteration)

```
proc dpf(n)
  F[0]<-0; F[1]<-1
  for j<-2 to n do
    F[j] <- F[j-1] + F[j-2]
  return F[n]
```

- time $T_n \in \Theta(n)$ ☺

a d.p. example: order for chained matrix multiplication

- input: matrices $A_1, \ldots, A_n$ with dimensions $d_0 \times d_1$, $\ldots d_{n-1} \times d_n$
- ouput: order in which matrices should be multiplied so that $A_1 \times A_2 \times \ldots \times A_n$ is computed using minimum number of scalar multiplications

```
A_1  A_2     A_3   A_4        d_0  d_1  d_2  d_3  d_4
 XX  XXXXXX  XXXX  XXX         5    2    6    4    3
 XX  XXXXXX  XXXX  XXX
 XX          XXXX  XXX
 XX          XXXX  XXX
 XX          XXXX
             XXXX
```

matrix mult'n order	scalar multiplications
$((A_1A_2)A_3)A_4$	$5 \times 2 \times 6 + 5 \times 6 \times 4 + 5 \times 4 \times 3 = 240$
$(A_1(A_2A_3))A_4$	$5 \times 2 \times 4 + 2 \times 6 \times 4 + 5 \times 4 \times 3 = 148$
$(A_1A_2)(A_3A_4)$	$5 \times 2 \times 6 + 5 \times 6 \times 3 + 6 \times 4 \times 3 = 222$
$A_1((A_2A_3)A_4)$	$5 \times 2 \times 3 + 2 \times 6 \times 4 + 2 \times 4 \times 3 = 102$
$A_1(A_2(A_3A_4))$	$5 \times 2 \times 3 + 2 \times 6 \times 3 + 6 \times 4 \times 3 = 138$

- 1st approach: 'brute force' (a.k.a. exhaustive enumeration)
  let M_n be number of mult'n orders ... how big is M_n?

n	1	2	3	4	5	6	...
M_n	1	1	2	5	14	42	...

- let C_n be the number of binary trees with
  - $n + 1$ leaves, n non-leaves
  - each non-leaf has two children

```
      /\        /\        /\        /\        /\
     /\        /\        /\/\        /\        /\
    /\          /\                  /\          /\
```

n	0	1	2	3	4	5	...
C_n	1	1	2	5	14	42	...

- these binary trees can be constructed recursively

root:		1 non-leaf
left subtree:	$j + 1$ leaves	j non-leaves
right subtree:	$n - j$ leaves	$n - j - 1$ non-leaves

- numbers C_n: Catalan numbers [1838]
- $C_n = \begin{cases} 1 & \text{if } \ n = 0, 1 \\ \sum_{j=1}^{n-1} C_j \, C_{n-j-1} & \text{if } \ n \geq 2 \end{cases}$
- $M_{n+1} = C_n = \dfrac{\binom{2n}{n}}{n+1} \approx \dfrac{4^n}{n\sqrt{\pi n}}$
- see "Concrete Mathematics" by Graham/Knuth/Patashnik
- brute force approach: time in $\Omega((4-\epsilon)^n)$ ☹

- 2nd approach: recursion
- let $M(x,y)$ be the minimum number of scalar multiplications needed to perform $A_x \times A_{x+1} \times \ldots \times A_y$
- $M(x,y) = \begin{cases} 0 & \text{if } \; x = y \\ \min_{x \le t < y}\{M(x,t) + M(t+1,y) + d_{x-1}d_t d_y\} & \text{if } \; x < y \end{cases}$
- e.g. $M(1,4) = \min \left\{ \begin{array}{l} M(1,1) + M(2,4) + d_0 d_1 d_4, \\ M(1,2) + M(3,4) + d_0 d_2 d_4, \\ M(1,3) + M(4,4) + d_0 d_3 d_4 \end{array} \right\}$
- implementation: recursion

```
proc M(x,y)
  if x=y then return 0
  else
    cost <- infty
    for t <- x to y-1 do
      new <- M(x,t)+ M(t+1,y)+ d[x-1]d[t]d[y]
      if new < cost then
        cost <- new
    return cost
```

```
                         14

11       24          12    34          13       44

   22 34    23 44 11 22  33 44 11 23    12 33

     33 44 22 33              22 33 11 22
```