

Objectives

- Divide and Conquer
 - Integer multiplication
 - Matrix multiplication
- Dynamic Programming
 - Fibonacci Sequence
 - Weighted Interval Scheduling

Mar 8, 2010

CSCI211 - Sprenkle

1

Integer Arithmetic

- **Add.** Given two n -digit integers a and b , compute $a + b$.

➤ Algorithm?

► Runtime?

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

$O(n)$ operations

Mar 8, 2010

CSCI211 - Sprengle

Integer Arithmetic

- **Multiply.** Given two n -digit integers a and b , compute $a \times b$.
 - Brute force solution: $\Theta(n^2)$ bit operations

Goal: Faster algorithm

Diagram illustrating a shift register multiplier circuit. The multiplicand is 0110100000 and the multiplier is 11010101. The resulting product is 011010100110101010.

Mar 8, 2010

3

Divide-and-Conquer Multiplication: Warmup

- To multiply two n -digit integers:
 - Multiply four $\frac{1}{2}n$ -digit integers
 - Add two $\frac{1}{2}n$ -digit integers and shift to obtain result

Higher order bits Lower order bits

Shift

$$\begin{aligned}
 x &= 2^{n/2} \cdot x_1 + x_0 \\
 y &= 2^{n/2} \cdot y_1 + y_0 \\
 xy &= (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0
 \end{aligned}$$

What is the recurrence relation?

- How many subproblems?
- What is merge cost?
- What is its runtime?

Mar 8, 2010

CSCI211 - Sprenkle

Divide-and-Conquer Multiplication: Warmup

- To multiply two n -digit integers:
 - Multiply four $\frac{1}{2} n$ -digit integers
 - Add two $\frac{1}{2} n$ -digit integers and shift to obtain result

Higher order bits Lower order bits

Shift

$$\begin{aligned}x &= 2^{n/2} \cdot x_1 + x_0 \\y &= 2^{n/2} \cdot y_1 + y_0 \\xy &= (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0\end{aligned}$$

A B C D

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

- assumes n is a power of 2

Not an improvement over brute force

Mar 8, 2010

CSCI211 - Sprengle

5

Karatsuba Multiplication

- To multiply two n -digit integers:
 - Add two $\frac{1}{2}n$ digit integers
 - Multiply 3 $\frac{1}{2}n$ -digit integers
 - Add, subtract, and shift $\frac{1}{2}n$ -digit integers to obtain result

$$\begin{aligned} x &= 2^{n/2} \cdot x_1 + x_0 \\ y &= 2^{n/2} \cdot y_1 + y_0 \\ xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\ &= 2^n \cdot \underbrace{x_1 y_1}_A + 2^{n/2} \cdot (\underbrace{(x_1 + x_0)}_B (\underbrace{y_1 + y_0}_C) - \underbrace{x_1 y_1}_A - \underbrace{x_0 y_0}_C) + x_0 y_0 \end{aligned}$$

What is the recurrence relation? Runtime?

Mar 8, 2010

CSCI211 - Sprenkle

Karatsuba Multiplication

- Theorem.** [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in $O(n^{1.585})$ bit operations

$$\begin{aligned} x &= 2^{n/2} \cdot x_1 + x_0 \\ y &= 2^{n/2} \cdot y_1 + y_0 \\ xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\ &= 2^n \cdot \underbrace{x_1 y_1}_A + 2^{n/2} \cdot (\underbrace{x_1 y_0}_B + \underbrace{x_0 y_1}_C) + \underbrace{x_0 y_0}_D \end{aligned}$$

$$\begin{aligned} T(n) &\leq T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + T(\lfloor 1 + \lfloor n/2 \rfloor \rfloor) + \Theta(n) \\ &\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585}) \end{aligned}$$

Mar 8, 2010

CSCI211 - Sprenkle

7

MATRIX MULTIPLICATION

Mar 8, 2010

CSCI211 - Sprenkle

8

Matrix Multiplication

- Given two n-by-n matrices A and B, compute $C = AB$

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}$$

Ex: $c_{12} = a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} + \dots + a_{1n} b_{n2}$

- Brute force.** $\Theta(n^3)$ arithmetic operations
- Fundamental question:** Can we improve upon brute force?

Mar 8, 2010

CSCI211 - Sprenkle

9

Matrix Multiplication: Warmup

- Divide:** partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks
- Conquer:** multiply 8 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ recursively
- Combine:** add appropriate products using 4 matrix additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

Recurrence relation? Runtime?

Mar 8, 2010

CSCI211 - Sprenkle

10

Matrix Multiplication: Warmup

- Divide:** partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks
- Conquer:** multiply 8 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ recursively
- Combine:** add appropriate products using 4 matrix additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$$

Mar 8, 2010

CSCI211 - Sprenkle

11

Matrix Multiplication: Key Idea

- Multiply 2-by-2 block matrices with only **7** multiplications and **15** additions

Trading expensive multiplication for less expensive addition/subtraction

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_3 + P_4 - P_5 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_2 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Mar 8, 2010

CSCI211 - Sprenkle

12

Fast Matrix Multiplication [Strassen, 1969]

- **Divide**: partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks
- **Compute**: 14 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices via 10 matrix additions
- **Conquer**: multiply 7 $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices recursively
- **Combine**: 7 products into 4 terms using 8 matrix additions

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

- **Analysis.**
 - Assume n is a power of 2.
 - $T(n)$ = # arithmetic operations.

Mar 8, 2010

CSCI211 - Sprenkle

13

Fast Matrix Multiplication in Practice

- Implementation issues: problems with putting theory into practice
 - Sparsity
 - Caching effects
 - Numerical stability
 - Theoretically correct but possible problems with round off errors, etc
 - Odd matrix dimensions
 - Crossover to classical algorithm around $n = 128$

Mar 8, 2010

CSCI211 - Sprenkle

14

Fast Matrix Multiplication in Practice

- Common misperception: "Strassen is only a theoretical curiosity."
 - Advanced Computation Group at Apple Computer reports **8x** speedup on G4 Velocity Engine when $n \sim 2,500$
 - Range of instances where it's useful is a subject of controversy
- Can "Strassenize" $Ax=b$, determinant, eigenvalues, and other matrix ops

Mar 8, 2010

CSCI211 - Sprenkle

15

Fast Matrix Multiplication in Theory

- Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
A. **Yes!** [Strassen, 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$
- Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. **Impossible** [Hopcroft and Kerr, 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$
- Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. **Also impossible** $\Theta(n^{\log_3 21}) = O(n^{2.77})$
- Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
A. **Yes!** [Pan, 1980] $\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$
- **Decimal wars.**
 - December, 1979: $O(n^{2.521813})$
 - January, 1980: $O(n^{2.521801})$

Mar 8, 2010

CSCI211 - Sprenkle

16

Fast Matrix Multiplication in Theory

- **Best known.** $O(n^{2.376})$ [Coppersmith-Winograd, 1987.]
 - But *really* large constant
- **Conjecture.** $O(n^{2+\epsilon})$ for any $\epsilon > 0$.
- **Caveat.** Theoretical improvements to Strassen are progressively less practical.

Mar 8, 2010

CSCI211 - Sprenkle

17

Algorithmic Paradigms

- **Greedy.** Build up a solution incrementally, myopically optimizing some local criterion
- **Divide-and-conquer.** Break up a problem into sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem
- **Dynamic programming.** Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems

Mar 8, 2010

CSCI211 - Sprenkle

18

Dynamic Programming History

- Richard Bellman pioneered systematic study of dynamic programming in 1950s
- Etymology
 - Dynamic programming = planning over time
 - Not our typical use of "programming"
 - Secretary of Defense was hostile to mathematical research
 - Bellman sought an impressive name to avoid confrontation
 - "it's impossible to use dynamic in a pejorative sense"
 - "something not even a Congressman could object to"

Mar 8, 2010 Reference: Bellman, R. E. *Eye of the Hurricane, An Autobiography*. 19

WARMUP: FIBONACCI SEQUENCE

Mar 8, 2010

CSCI211 - Sprenkle

20

How Would You Solve the Fibonacci Sequence?

- Input: the number of Fibonacci numbers, x
- Output: display the list of the first x Fibonacci numbers

Sequence:

- $F_0 = F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

Mar 8, 2010

CSCI211 - Sprenkle

21

Soln 1: Using a List

- Typical Solution:

```
fibs = []           # create an empty list
fibs.append(1)      # append the first two Fib numbers
fibs.append(1)
print fibs[0], fibs[1],
for x in xrange(2, N):
    newfib = fibs[x-1]+fibs[x-2]
    print newfib,
    fibs.append(newfib)
print fibs          # print out the list
```

Building up solution

Running time? Space cost?

Do we need a whole list?

Mar 8, 2010

CSCI211 - Sprenkle

22

Soln 2: Using Three Variables

- Only need the solutions to the last two problems ($F[k-1]$, $F[k-2]$)

```
lastNum = 1
twoAgo = 1
print twoAgo, lastNum,
for n in xrange(2, N):
    nthNum = twoAgo + lastNum
    print nthNum,
    twoAgo = lastNum
    lastNum = nthNum
```

Write as a recurrence

Mar 8, 2010

CSCI211 - Sprenkle

23

Soln 3: Recursion

```
def fibonacci(n):
    return fibonacci(n-1) + fibonacci(n-2)
```

- What is the running time of this algorithm?

Mar 8, 2010

CSCI211 - Sprenkle

24

Dynamic Programming Memoization Process

- Create a table with the possible inputs
- If the value is in the table, return it (without recomputing it)
- Otherwise, call function recursively
 - Add value to table for future reference

How does this template map to our Fibonacci problem?

Mar 8, 2010

CSCI211 - Sprenkle

25

Memoization Example: Fibonacci

```
memoized_fibonacci(n):
    for j = 1 to n:
        results[j] = -1    # -1 means undefined

    return memoized_fib_recurs(results, n)

memoized_fib_recurs(results, n):
    if results[n] != -1:    # value is defined
        return results[n]
    if n == 1:
        val = 1
    elif n == 2:
        val = 1
    else:
        val = memoized_fib_recurs(results, n-2)
        val = val + memoized_fib_recurs(results, n-1)
        results[n] = val
    return val
```

Runtime?

Mar 8, 2010

CSCI211 - Sprenkle

26

Memoization Example: Fibonacci

Alternative version...

```
memoized_fibonacci(n):
    for j = 1 to n:
        results[j] = -1    # -1 means undefined
        results[1] = 1
        results[2] = 1

    return memoized_fib_recurs(results, n)

memoized_fib_recurs(results, n):
    if results[n] != -1:    # value is defined
        return results[n]

    val = memoized_fib_recurs(results, n-2)
    val = val + memoized_fib_recurs(results, n-1)
    results[n] = val
    return val
```

Mar 8, 2010

CSCI211 - Sprenkle

27

WEIGHTED INTERVAL SCHEDULING

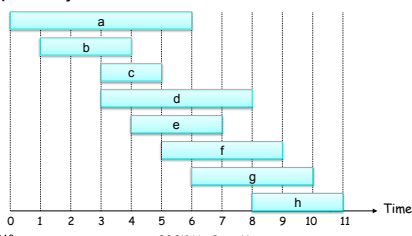
Mar 8, 2010

CSCI211 - Sprenkle

28

Weighted Interval Scheduling

- Job j starts at s_j , finishes at f_j , and has weight or value v_j
- Two jobs are **compatible** if they don't overlap
- **Goal**: find maximum **weight** subset of mutually compatible jobs



Mar 8, 2010

CSCI211 - Sprenkle

29

Unweighted Interval Scheduling Review

- **Recall**. Greedy algorithm works if all weights are 1 (or equivalent).
 - Consider jobs in ascending order of finish time
 - Add job to subset if it is compatible with previously chosen jobs

What happens to Greedy algorithm if we add weights to the problem?

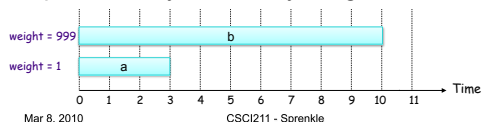
Mar 8, 2010

CSCI211 - Sprenkle

30

Limitation of Greedy Algorithm

- **Recall.** Greedy algorithm works if all weights are 1.
 - Consider jobs in ascending order of finish time
 - Add job to subset if it is compatible with previously chosen jobs
- **Observation.** Greedy algorithm can fail spectacularly if arbitrary weights are allowed



Mar 8, 2010

CSCI211 - Sprenkle

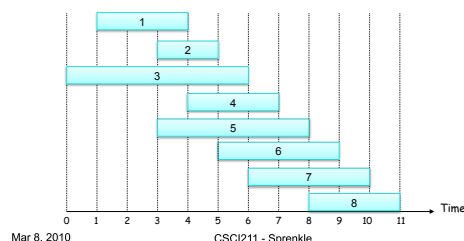
31

Weighted Interval Scheduling

Notation. Label jobs by finishing time: $f_1 \leq f_2 \leq \dots \leq f_n$

Def. $p(j)$ = largest index $i < j$ such that job i is compatible with j

Ex: $p(8) = 5$, $p(7) = 3$, $p(2) = 0$



Mar 8, 2010

CSCI211 - Sprenkle

32

Dynamic Programming

- Assume we have an optimal solution
- **Notation.** $OPT(j)$ = **value** of optimal solution to the *problem* consisting of job requests 1, 2, ..., j
 - What is something *obvious* we can say about the optimal solution with respect to job j ?

Mar 8, 2010

CSCI211 - Sprenkle

33

Dynamic Programming: Binary Choice

- **Notation.** $OPT(j)$ = **value** of optimal solution to the *problem* consisting of job requests 1, 2, ..., j
 - Case 1: OPT selects job j
 - Case 2: OPT does not select job j

Explore both of these cases...
 • What jobs are in OPT ? Which are not?
 Keep in mind our definition of p

Mar 8, 2010

CSCI211 - Sprenkle

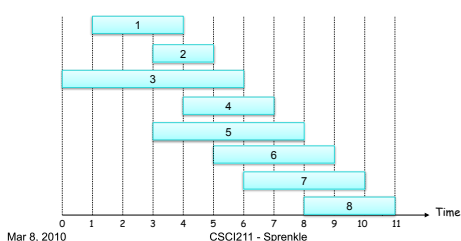
34

Weighted Interval Scheduling

Notation. Label jobs by finishing time: $f_1 \leq f_2 \leq \dots \leq f_n$

Def. $p(j)$ = largest index $i < j$ such that job i is compatible with j

Ex: $p(8) = 5$, $p(7) = 3$, $p(2) = 0$



Mar 8, 2010

CSCI211 - Sprenkle

35

Dynamic Programming: Binary Choice

- **Notation.** $OPT(j)$ = **value** of optimal solution to the *problem* consisting of job requests 1, 2, ..., j
 - Case 1: OPT selects job j
 - can't use incompatible jobs $\{ p(j) + 1, p(j) + 2, \dots, j - 1 \}$
 - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., $p(j)$
 - Case 2: OPT does **not** select job j
 - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., $j-1$

Formulate $OPT(j)$ as a recurrence relation

Mar 8, 2010

CSCI211 - Sprenkle

36

Dynamic Programming: Binary Choice

- **Notation.** $OPT(j)$ = **value** of optimal solution to the *problem* consisting of job requests 1, 2, ..., j
 - **Case 1: OPT selects job j**
 - can't use incompatible jobs $\{ p(j) + 1, p(j) + 2, \dots, j - 1 \}$
 - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., $p(j)$
 - **Case 2: OPT does not select job j**
 - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., $j-1$

Two options: $Opt(j) = v_j + OPT(p(j))$
 $Opt(j) = Opt(j-1)$

Formulate $OPT(j)$ in terms of smaller subproblems
Which should we choose?

Mar 8, 2010

CSCI211 - Sprenkle

37

Dynamic Programming: Binary Choice

- **Notation.** OPT = **value** of optimal solution to the problem consisting of job requests 1, 2, ..., j
 - **Case 1: OPT selects job j**
 - can't use incompatible jobs $\{ p(j) + 1, p(j) + 2, \dots, j - 1 \}$
 - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., $p(j)$
 - **Case 2: OPT does not select job j**
 - must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., $j-1$

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max \{ v_j + OPT(p(j)), OPT(j-1) \} & \text{otherwise} \end{cases}$$

Choose the better of the two solutions

Mar 8, 2010

CSCI211 - Sprenkle

38

Assignments

- Finish reading Chapter 5, start Chapter 6
 - 5.2-5.5
 - 6 – front matter, 6.1
- PS6 due Friday

Mar 8, 2010

CSCI211 - Sprenkle

39