CSOR4231

Analysis of Algorithms

Algorithms are Everywhere

Examples

- Maps
- Fedex
- Biology
- Physics
- Computer Operating Systems
- Self-Driving Cars
- Determining if you should get a job/loan/school admission
- Regulating your heart
- Space Shuttle
- Machine Learning
- ...

Why is this the right time to study algorithms?

- **Mathematical understanding**
- **fast computers**
- **ability to get algorithm implementations to users**
- **good interfaces**

What do we study in this class

- **Given a problem, we find the right algorithm**
- **We use math**
- **We prove that our work is right**
- **We keep an eye on practice/implementation, but our goal is to solve the clean well-defined problem.**

What are the skills most people need

- Given a new problem, how do we design an algorithm
- Knowing what is efficient and what is not, to help you
  - model problems
  - use existing algorithms
  - decide which algorithms to extend
  - realize when a problem is too hard to solve quickly

First problem to consider: Matrix Multiplication

$$C = A \cdot B$$

$$\begin{bmatrix} 3 & 1 & 1 \\ 2 & 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 6 \\ 2 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 6 & 20 \\ 5 & 18 \end{bmatrix}$$

Algorithm for Matrix Multiplication

$$C = A \cdot B$$

$$\begin{bmatrix} 3 & 1 & 1 \\ 2 & 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 6 \\ 2 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 6 & 20 \\ 5 & 18 \end{bmatrix}$$

Write pseudocode

```
// input: A, an n × m matrix and B, an m × p matrix
// output: C, an n × p matrix
for i = 1 to n
    for j = 1 to p
        C[i, j] = 0
        for k = 1 to m
            C[i, j] += A[i, k] · B[k, j]
```

Analysis

```
// input: A, an n × m matrix and B, an m × p matrix
// output: C, an n× matrix
for i = 1 to n
    for j = 1 to p
        C[i, j] = 0
        for k = 1 to m
            C[i, j]+ = A[i, k] · B[k, j]
```

Running time

- **3 nested loops**
- $O(nmp)$ **time**
- **if** $n = m = p$ **, then** $O(n^3) time$
- **Lower bound of** $\Omega(n^2)$

Can we do better?

- **We are implementing the standard definition efficiently, what else could we do?**
- **You have to do n^3 operatation, each of n^2 entries of C , involves adding up the result of n multipications.**

Can we do better?

- **We are implementing the standard definition efficiently, what else could we do?**
- **You have to do n^3 operatation, each of n^2 entries of C, involves adding up the result of n multipications.**

Maybe divide and conquer can help

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

$$r = ae + bf \quad (1)$$

$$s = ag + bh \quad (2)$$

$$t = ce + df \quad (3)$$

$$u = cg + dh \quad (4)$$

Maybe divide and conquer can help

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

$$r = ae + bf \quad (5)$$
$$s = ag + bh \quad (6)$$
$$t = ce + df \quad (7)$$
$$u = cg + dh \quad (8)$$

Multiply 2 $n \times n$ matrices takes

- **8 multiplications of $n/2 \times n/2$ matrices**
- **4 additions of $n/2 \times n/2$ matrices**

- **Adding two $n \times n$ matrices takes $O(n^2)$ time**
- **Adding matrices seems easier than multiplying them**

Let's Analyze

Let $T(n)$ **be the time to multiply 2** n **by** n **matrices**

$$T(n) = \begin{cases} 8T(n/2) + 4(n/2)^2 & \textbf{if } n > 1 \\ 1 & \textbf{if } n = 1 \end{cases}$$

Let's Analyze

Let $T(n)$ **be the time to multiply 2** n **by** n **matrices**

$$T(n) = \begin{cases} 8T(n/2) + 4(n/2)^2 & \textbf{if } n > 1 \\ 1 & \textbf{if } n = 1 \end{cases}$$

As we will learn, this solves to $O(n^3)$ **.**

But consider the following recurrence

$$T(n) = \begin{cases} 7T(n/2) + 18(n/2)^2 & \textbf{if } n > 1 \\ 1 & \textbf{if } n = 1 \end{cases}$$

As we will learn, this solves to $O(n^{\log_2 7}) = O(n^{2.81..})$ **.**

But can we multiply 2 $n \times n$ **matrices by doing** 7 **multiplications of** $n/2 \times n/2$ **matrices and** 18 **additions of** $n/2 \times n/2$ **matrices.**

Strassen's Algorithm

To Compute

$$r = ae + bf \tag{9}$$
$$s = ag + bh \tag{10}$$
$$t = ce + df \tag{11}$$
$$u = cg + dh \tag{12}$$

Calculations

$$P_1 = a(g - h) = ag - ah$$
$$P_2 = (a + b)h = ah + bh$$

$$s = P_1 + P_2$$

$$P_3 = (c + d)e = ce + de$$
$$P_4 = d(f - e) = df - de$$

$$t = P_3 + P_4$$

$$P_5 = (a + d)(e + h) = ae + ah + de + dh$$
$$P_6 = (b - d)(h + f) = -dh - df + bh + bf$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$P_7 = (a - c)(e + g) = ae + ag - ce - cg$$

$$u = P_5 + P_1 - P_3 - P_7$$

Course Logistics

Another Problem

Investing for someone who knows the Future: You are given the prices of a stock for each of the next n days. You can buy once and sell once and you want to maximize your profit.

Example

Day	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	70	90	40	27	69	80	13	50	35	75	51	53	56	10	15	41

Questions:

- How long does the naive algorithm take?
- Can we improve this with divide and conquer?