Design and Analysis of Algorithms February 6, 2015
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Erik Demaine, Srini Devadas and Nancy Lynch Recitation 1

Matrix Multiplication and the Master Theorem

1 Weighted interval scheduling

Consider requests 1,. . . ,n. For request i, $s(i)$ is the start time and $f(i)$ is the finish time, $s(i) < f(i)$. Two requests i and j are compatible if they don't overlap, i.e., $f(i) \leq s(j)$ or $f(j) \leq s(i)$. Each requet i has a weight $w(i)$. **Goal**: schedule the subset of compatible requests with maximum weight.

1.1 The $\mathrm{n} \log \mathrm{n}$ dynamic programming solution

Sort requests in earliest finish time order.

$$f(1) \leq f(2) \leq \cdots \leq f(n)$$

Definition $p(j)$ for interval j is the largest index $i < j$ such that request i and j are compatible.

Array $M[0 \ldots n]$ holds the optimal solution's values. $M[k]$ is the maximum weight if requests from 1 to k are considered.

```
M[0] = 0
for  j = 1 to n
    M[j] = max(w(j) + M[p(j)], M[j - 1])
```

Once we have M, the optimal solution can be derived by tracing it back in $O(n)$ time. Sorting requests in earliest finish time takes $O(n \log n)$ time. And the whole algorithm takes $O(n \log n)$ time.

2 Strassen

2.1 Matrix Multiplication

Take matrices A, B, multiply row i of A by column j of B to fill in entry i,j of resulting matrix, C. Running time is $\Theta(n^3)$ on square matrices, where n is the dimension of each matrix.

2.2 The Strassen Algorithm

- powerful early application of Divide and Conquer
- not the fastest matrix multiplication (though it was at time of discovery)
  - Don Coppersmith, Shmuel Winograd, Andrew Stothers, and Vassilevska Williams contributed to the current fastest method. See http://en.wikipedia.org/wiki/Coppersmith-Winograd_algorithm for details.

2.2.1 Steps

- Make A, B each 2^k x 2^k by filling remaining rows/columns with 0:
  - Why can you do this?
    - Each dimension increases by less than a factor of 2
    - Even with traditional $\Theta(n^3)$ matrix multiplication, this makes the running time always increase by a factor of less than 8, not dependent on the magnitude of N, and constant factors are always ignored when discussing complexity.
- Partition A, B, and C (elements unknown for C, but same dimensions) into 4 matrices of dimension 2^{k-1} each
- We can see that the 4 submatrices of C can be found by standard matrix multiplications of A and B, using the submatrices as "elements"

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \qquad \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$

$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$

$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$

$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

- Optimization is derived from the fact that matrix addition is much, much simpler than multiplication ($\Theta(n^2)$ instead of $\Theta(n^3)$)

$$\begin{aligned}
\text{Define } M_1 &= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\
M_2 &= (A_{2,1} + A_{2,2})B_{1,1} \\
M_3 &= A_{1,1}(B_{1,2} - B_{2,2}) \\
M_4 &= A_{2,2}(B_{2,1} - B_{1,1}) \\
M_5 &= (A_{1,1} + A_{1,2})B_{2,2} \\
M_6 &= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\
M_7 &= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}) \\
\text{Thus, } C_{1,1} = M_1 + M_4 - M_5 + M_7 &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
C_{1,2} = M_3 + M_5 &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
C_{2,1} = M_2 + M_4 &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
C_{2,2} = M_1 - M_2 + M_3 + M_6 &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{aligned}$$

Proof of correctness follows from arithmetic.

We can recursively calculate each of the above submatrices using equally-sized submatrices of $A_{1,1}$, etc., which is why we needed dimensions of 2^n instead of merely even dimensions.

When you have C, strip out rows/columns of 0s that correspond to the same parts of A and B.

- Each recursive step takes 7 multiplications and 18 additions, instead of 8 multiplications
- We can see that this would be less efficient than 8 multiplications for small matrices. For a 2-element matrix being broken into 4 1-element matrices, it's over triple the work!

Running time: T(n) = $\Theta(n^{\log_2(7)}) \approx \Theta(n^{2.8074})$
How do we get this value? (next up)

3 Master Theorem

3.1 General use

General form of a recurrence:
$T(n) = aT(n/b) + f(n)$

- $f(n)$ polynomially less than $n^{\log_b(a)}$: $T(n) = \Theta(n^{\log_b(a)})$
- $f(n)$ is $\Theta(n^{\log_b(a)} \log^k(n))$, where $k \geq 0$: $T(n) = \Theta(f(n)\log(n)) = \Theta(n^{\log_b(a)} \log^{k+1}(n))$
- $n^{\log_b(a)}$ polynomially less than $f(n)$, and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n: $T(n) = \Theta(f(n))$

If $n^{\log_b(a)}$ is greater, but not polynomially greater, than $f(n)$, the Master Theorem cannot be used to determine a precise bound.

(e.g. $T(n) = 2T(n/2) + \Theta(n/\log(n))$)

3.2 Strassen Runtime

Now, think about Strassen's algorithm. It performs 7 multiplications and 18 additions/subtractions each iteration. The addition is performed directly; the multiplications are done recursively using the Strassen Algorithm.

In each recursive step, we divide the matrix into 4 parts; however, remember that we consider the running time in terms of the dimension of the matrix, not the total number of elements.

Thus, the recurrence becomes

$$T(n) = 7T(n/2) + 18\Theta(n^2) = 7T(n/2) + \Theta(n^2)$$

We can then examine the Master Theorem:
$n^{\log_2(7))}$ is polynomially greater than n^2
Thus, $\Theta(n^{\log_2(7)})$ is the solution to the recurrence.

3.3 Median Finding

Prove that $T(n) = T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + \Theta(n)$ solves for $T(n) = \Theta(n)$.

Proof: We use the *substitution method* (details can be found in the CLRS textbook) to solve the recurrence. We first guess the form of the answer to be $O(n)$, and try to prove that $T(n) \leq dn$ for some value of d. We first assume that the bound holds for all positive $m < n$, and thus it holds for $T(\lceil \frac{n}{5} \rceil)$ and $T(\frac{7n}{10} + 6)$ if n is large enough. Substituting into the recurrence yields

$$\begin{aligned} T(n) &= T(\left\lceil \frac{n}{5} \right\rceil) + T(\frac{7n}{10} + 6) + cn && (1) \\ &\leq d(\frac{n}{5} + 1) + d(\frac{7n}{10} + 6) + cn && (2) \\ &= \frac{9}{10}dn + 7d + cn && (3) \\ &\leq dn && (4) \end{aligned}$$

The last inequality holds if $d > 10c$ when n is large enough.

3.4 Extra details

Drawing a recursion tree using the recurrence $T(n) = 4T(n/2) + \Theta(n^2)$ will show why the $\log$ factor is used if f(n) is not polynomially greater than $n^{\log_b(a)}$. Think of the total amount of work that must be done.

Feel free to examine $T(n) = 4T(n/2) + \Theta(n^2 \log(n))$ to see why the solution must be $\Theta(n^2 \log^2(n))$ instead of just $\Theta(n^2 \log(n))$.