Algorithms and Data Structures II

Lecture 9:

Algorithm Design Techniques: Divide-and-Conquer

https://elms.u-aizu.ac.jp

Divide-and-Conquer

- Recursive algorithms solve a given problem by calling themselves recursively. They follow a divide-and-conquer approach:
    1. break the problem into several subproblems that are similar to the original problem but smaller in size,
    2. solve the subproblems recursively,
    3. combine these solutions to create a solution to the original problem.

Divide-and-Conquer

- The divide-and-conquer paradigm has three steps at each level of the recursion:
  1. Divide the problem into several smaller subproblems.
  2. Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, then solve the subproblem straightforwardly.
  3. Combine the solutions to the subproblems into the solution for the original problem.

Example: merge sort

- An example of divide-and-conquer algorithms: merge sort:
  1. Divide: divide the n sequence to be sorted into two subsequences of $n/2$ elements each.
  2. Conquer: sort the two subsequences recursively using merge sort.
  3. Combine: Merge the two sorted subsequences to get the answer.

Recall: mergesort

```
mergesort(int A[],int left,int right){
  int i,j,k,mid;
  if (right>left) {
    mid=(right+left)/2; /*DIVIDE*/
    mergesort(A,left,mid); /*CONQUER*/
    mergesort(A,mid+1,right);
    for (i=left;i<=mid;i++) B[i]=A[i];
    for (i=mid+1,j=right;i<=right;i++,j--)B[i]=A[j];
    i=left;j=right; /*COMBINE*/
    for (k = left; k <= right; k++)
    if (B[i]<B[j]) A[k]=B[i++]; else A[k]=B[j--];
  }
}
```

Master Theorem

Let $T(n)$ be a monotonically increasing function that satisfies

$$\begin{aligned} T(n) &= aT(\frac{n}{b}) + f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1, b \geq 2, c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if} a < b^d \\ \Theta(n^d \log n) & \text{if} a = b^d \\ \Theta(n^{\log_b a}) & \text{if} a > b^d \end{cases}$$

Strassen's algorithm

- Now we give another example of divide-and-conquer algorithm, Strassen's algorithm for matrix multiplication.

Matrix Multiplication

- Let A and B be two $n \times n$ matrices. The product of A and B is defined as $C = AB$, where for $1 \leq i, j \leq n$,
$$C[i, j] = \sum_{k=1}^{n} A[i, k] \times B[k, j]$$

Matrix Multiplication(contd)

- Let n be a power of 2, we can partition each of A and B into four $(n/2) \times (n/2)$ matrices and express the product of A and B in terms of these $(n/2) \times (n/2)$ matrices as:

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

Matrix Multiplication(contd 2)

- If we treat A and B as 2×2 matrices, each of whose elements are $(n/2) \times (n/2)$ matrices, then the product of A and B can be expressed in terms of sums and products of $(n/2) \times (n/2)$ matrices as given in the next slides.

Matrix Multiplication(contd 3)

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \tag{1}$$

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned} \tag{2}$$

Matrix Multiplication(contd 4)

- Assume C_{11}, C_{12}, C_{21}, and C_{22} can be computed by m multiplications and a additions (or subtractions) of the $(n/2) \times (n/2)$ matrices. By applying the algorithm recursively, we can compute the product of two $n \times n$ matrices in time $T(n)$, where for n a power of 2, $T(n)$ satisfies

$$T(n) \leq mT(\frac{n}{2}) + \frac{an^2}{4} \tag{3}$$

Matrix Multiplication(contd 5)

- The first term on the right of (3) is the cost of multiplying m pairs of $(n/2) \times (n/2)$ matrices, and the second is the cost of a additions, assuming $n^2/4$ time is required for each addition or subtraction of two $(n/2) \times (n/2)$ matrices.

$$T(n) \leq mT(\frac{n}{2}) + \frac{an^2}{4} \tag{3}$$

Matrix Multiplication(contd 6)

- For $m > 4$, it can be shown that

  $$T(n) = O(n^{\log m}),$$

  where the logarithm is based on 2.
  If we compute C_{ij} as shown in (2), $m = 8$ and

  $$T(n) = O(n^3).$$

Strassen's Matrix Multiplication Algorithm

- V. Strassen discovered a clever method of multiplying two 2×2 matrices with elements from an arbitrary ring using only seven multiplications ($m = 7$). By using the method recursively, he was able to multiply two $n \times n$ matrices in time

$$O(n^{\log 7})$$

which is of order approximately $n^{2.81}$.

Strassen's Matrix Multiplication Algorithm(contd)

- In Strassen's method, to compute (1), first compute the following products

$$
\begin{aligned}
P_1 &= (A_{12} - A_{22})(B_{21} + B_{22}), && (4)\\
P_2 &= (A_{11} + A_{22})(B_{11} + B_{22}), && (5)\\
P_3 &= (A_{11} - A_{21})(B_{11} + B_{12}), && (6)\\
P_4 &= (A_{11} + A_{12})B_{22}, && (7)\\
P_5 &= A_{11}(B_{12} - B_{22}), && (8)\\
P_6 &= A_{22}(B_{21} - B_{11}), && (9)\\
P_7 &= (A_{21} + A_{22})B_{11}. && (10)
\end{aligned}
$$

Strassen's Matrix Multiplication Algorithm(contd)

- Then compute the C_{ij}'s using the formula

$$
\begin{aligned}
C_{11} &= P_1 + P_2 - P_4 + P_6, && (11)\\
C_{12} &= P_4 + P_5, && (12)\\
C_{21} &= P_6 + P_7, && (13)\\
C_{22} &= P_2 - P_3 + P_5 - P_7. && (14)
\end{aligned}
$$

The above method takes 7 multiplications and 18 additions/substractions.

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$,

$$\begin{aligned}
C_{11} &= P_1 + P_2 - P_4 + P_6 \\
P_1 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
P_2 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_4 &= (A_{11} + A_{12})B_{22} \qquad P_6 = A_{22}(B_{21} - B_{11})
\end{aligned}$$

$$\begin{aligned}
C_{11} &= A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22} \\
&\quad + A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22} \\
&\quad - A_{11}B_{22} - A_{12}B_{22} + A_{22}B_{21} - A_{22}B_{11} \\
&= A_{11}B_{11} + A_{12}B_{21}
\end{aligned}$$

Strassen's Matrix Multiplication Algorithm(contd 2)

- Similarly, we can verify

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

Winograd's Variant of Strassen's Algorithm

- Winograd's variant of Strassens algorithm proposed a method of multiplying two matrices with 7 multiplications and 15 additions/substractions. Although this improvement does not affect the complexity of the algorithm, reducing the number of additions to fifteen from the eighteen used in Strassen's original algorithm can have practical significance.

Winograd's Variant of Strassen's Algorithm(contd)

- Winograd's variant multiplies two matrices using recursion as Strassen's original algorithm does. The temporary variables used at each level of recursion are as follows:

Winograd's Variant of Strassen's Algorithm(contd 2)

$$\begin{aligned}
S_1 &= A_{21} + A_{22}, \\
S_2 &= S_1 - A_{11}, \\
S_3 &= A_{11} - A_{21}, \\
S_4 &= A_{12} - S_2,
\end{aligned}$$

$$\begin{aligned}
T_1 &= B_{12} - B_{11}, \\
T_2 &= B_{22} - T_1, \\
T_3 &= B_{22} - B_{12}, \\
T_4 &= B_{21} - T_2,
\end{aligned}$$

Winograd's Variant of Strassen's Algorithm(contd 3)

$$\begin{aligned} P_1 &= A_{11}B_{11}, \\ P_2 &= A_{12}B_{21}, \\ P_3 &= S_1T_1, \\ P_4 &= S_2T_2, \\ P_5 &= S_3T_3, \\ P_6 &= S_4B_{22}, \\ P_7 &= A_{22}T_4, \end{aligned}$$

$$\begin{aligned} U_1 &= P_1 + P_2, \\ U_2 &= P_1 + P_4, \\ U_3 &= U_2 + P_5, \\ U_4 &= U_3 + P_7, \\ U_5 &= U_3 + P_3, \\ U_6 &= U_2 + P_3, \\ U_7 &= U_6 + P_6, \end{aligned}$$

Winograd's Variant of Strassen's Algorithm(contd 4)

- Further, the values of C_{ij} can be defined as:

$$\begin{aligned} C_{11} &= U_1, \quad C_{12} = U_7, \\ C_{21} &= U_4, \quad C_{22} = U_5. \end{aligned}$$

- It has been determined that further reduction in the number of multiplications and additions is not possible when implementing variations on Strassen's original algorithm.

Strassen's algorithm

- There are two key issues when efficiently applying Strassen's algorithm to arbitrary matrices.
  - First the constraint that the matrix size be a power of 2 must be handled.
  - The second key issue for efficiency of Strassen's algorithm is controlling the depth of recursion.

Strassen’s algorithm

- Matrix multiplication is a fundamental operation and is critical when attempting to speed up scientific computations.
- The performance of matrix multiplication is dependent on two elements:
  - the operation count and
  - the memory reference count.

  Minimizing both of these factors will produce an optimal algorithm .

Strassen's algorithm

- The standard algorithm for multiplying two $n \times n$ matrices requires n^3 scalar multiplications and $n^3 - n^2$ scalar additions, for a total arithmetic operation count of $2n^3 - n^2$.
- If one level of Strassen's algorithm is applied to 2×2 matrices whose elements are $n/2 \times n/2$ blocks and the standard algorithm is used for the seven block matrix multiplications, the total operation count is

$$7(2(n/2)^3 - (n/2)^2) + 18(n/2)^2 = (7/4)n^3 + (11/4)n^2$$

Strassen's algorithm

- The ratio of this operation count to that required by the standard algorithm alone is seen to be

$$\frac{7n^3 + 11n^2}{8n^3 - 4n^2} \Rightarrow \frac{7}{8}$$

12.5 % improvement