Design and Analysis of Algorithm

Lecture-7:

Contents

Randomized Quick Sort

Partition around a random element.

- Select pivot element randomly from the given array.
- Swap last element and selected pivot element.
- Now apply normal quicksort

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + cn$$

Selection Problem

Input: An Array of n elements and an integer k

Output: k^{th} smallest element in the array

The Partition algorithm of Quicksort can also be used to obtain an efficient solution for the selection problem

If the partitioning element v is positioned at $a[j]$, then $j-1$ elements are less than or equal to $a[j]$ and $n-j$ elements are greater than or equal to $a[j]$.

Algorithm

```
Algorithm Select1(a, n, k)
// Selects the kth-smallest element in a[1 : n] and places it
// in the kth position of a[ ]. The remaining elements are
// rearranged such that a[m] ≤ a[k] for 1 ≤ m < k, and
// a[m] ≥ a[k] for k < m ≤ n.
{
    low := 1; up := n + 1;
    a[n + 1] := ∞; // a[n + 1] is set to infinity.
    repeat
    {
        // Each time the loop is entered,
        // 1 ≤ low ≤ k ≤ up ≤ n + 1.
        j := Partition(a, low, up);
                // j is such that a[j] is the jth-smallest value in a[ ].
        if (k = j) then return;
        else if (k < j) then up := j; // j is the new upper limit.
                else low := j + 1; // j + 1 is the new lower limit.
    } until (false);
}
```

Complexity

$$T(n) = T\left(\frac{n}{2}\right) + n$$

$$T(n) = O(n)$$

Other approaches

Input: An Array of n elements and an integer k
Output: k^{th} smallest element in the array

(a) Sort the given array and return the k^{th} element $\quad T(n) = O(nlog_2n)$

(a) Find 1^{st} minimum and delete $\quad T(n) = O(kn)$
Find 2^{nd} minimum and delete
.
.
.
Find k^{th} minimum and return

Simple Approach

```
Algorithm  MatrixMultiply (A,B)
// Assume dimension of A is (n x n)  and dimension of B is (n x n)
 {
   else
   {
    define C matrix as (n×n)
    for i=1:n
    {
      for j=1:n
      {
       C_ij = 0
          for k=1:n
          {
            C_ij =  C_ij + a_ik . b_kj
          }
    }
}
```

$C_{ij}= 0$

$C_{ij}= C_{ij} + a_{ik}.b_{kj}$

Approach Illustration

$$\text{Matrix 1}\begin{Bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{Bmatrix} \qquad \text{Matrix 2}\begin{Bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{Bmatrix}$$

$$\begin{array}{c}\text{Matrix 1}\\ * \\ \text{Matrix 2}\end{array}\begin{Bmatrix} 1*1+1*2+1*3 & 1*1+1*2+1*3 & 1*1+1*2+1*3 \\ 2*1+2*2+2*3 & 2*1+2*2+2*3 & 2*1+2*2+2*3 \\ 3*1+3*2+3*3 & 3*1+3*2+3*3 & 3*1+3*2+3*3 \end{Bmatrix}$$

$$\begin{array}{c}\text{Matrix 1}\\ * \\ \text{Matrix 2}\end{array}\begin{Bmatrix} 6 & 6 & 6 \\ 12 & 12 & 12 \\ 18 & 18 & 18 \end{Bmatrix}$$

Time complexity of this approach is O(n^3)

Divide and conquer approach

1) Divide matrices A and B in 4 sub-matrices of size $\frac{N}{2} \times \frac{N}{2}$
2) Calculate product AB by recursively computing $C_{11}, C_{12}\ C_{21}\ and\ C_{22}$

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

then

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

Divide and conquer approach

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Complexity

$$T(n) = f(x) = \begin{cases} c \;, & n \le 2 \\ 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2, & otherwise \end{cases}$$

$$T(n) = f(x) = \begin{cases} c \;, & n \le 2 \\ 8T\left(\frac{n}{2}\right) + n^2, & otherwise \end{cases}$$

$$Complexity = O(n^3)$$

Hence no improvement over the conventional method has been made. Since matrix multiplications are more expensive than matrix additions $O(n^3)$versus $O(n^2)$

Based on this logic $\boldsymbol{Volker\ Strassen}$ defined the approach for matrix multiplication faster than $O(n^3)$ in 1969

Selection Problem

Input: Two square matrix of size $n \times n$
Output: Multiplication of two matrix

Divide the input matrices A and B into $\frac{n}{2} \times \frac{n}{2}$ sub-matrices, which takes $\theta(1)$ time

* Create 10 metrics $S_1, S_2, S_3, S_4, \ldots \ldots \ldots S_{10}$

$$
\begin{aligned}
S_1 &= B_{12} - B_{22}, & S_6 &= B_{11} + B_{22}, \\
S_2 &= A_{11} + A_{12}, & S_7 &= A_{12} - A_{22}, \\
S_3 &= A_{21} + A_{22}, & S_8 &= B_{21} + B_{22}, \\
S_4 &= B_{21} - B_{11}, & S_9 &= A_{11} - A_{21}, \\
S_5 &= A_{11} + A_{22}, & S_{10} &= B_{11} + B_{12}.
\end{aligned}
$$

Algorithm

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \,, \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \,, \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \,, \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \,, \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \,, \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \,, \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} \,.
\end{aligned}
$$

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

Complexity

$$T(n) = \begin{cases} b & if\ (n \le 2) \\ 7T\left(\frac{n}{2}\right) + 16n^2 & , Otherwise \end{cases}$$

Complexity: $O\left(n^{\log_2 7}\right) = O(n^{2.81})$

$$T(n) = \begin{cases} b & if\ (n \le 2) \\ 7T\left(\frac{n}{2}\right) + cn^2 & , Otherwise \end{cases}$$

Can this complexity be further reduced ?

1) **Coppersmith–Winograd algorithm**	**1990**	$\boldsymbol{O\left(n^{2.375}\right)}$
2) **Andrew Stothers algorithm**	**2010**	$\boldsymbol{O\left(n^{2.374}\right)}$
3) **Virginia Vassilevska Williams algorithm**	**2011**	$\boldsymbol{O(n^{2.372})}$

Advantages of Divide and Conquer

Easy Soliution

Difficult problems can be solved easily. D & C is a powerful strategy for solving difficult problems.

Maintains Parallelism:

D & C divides the entire problem into various sub-problems. Thus, we can process these sub-problems on different processors. Thus ensuring multiprocessing.

Memory Access:

Each problem is divided into various sub-problems. Thus, the size of each sub-problem becomes small and can be stored and processed in the cache memory.

Disadvantages of Divide and Conquer

- One major disadvantage of D & C strategy is that involves recursion which is slow in nature.

- The overhead of function calls, stack size can hinder in the overall efficiency of the code. Another problem with this strategy is that the efficiency depends on implementation of logic. For some problems, iterative solution can be efficient in comparison to a recursive solution.