Multiplication of Two Matrices

A. 作業目標

分別以 *School-Book Method*、*Method 1*、*Method 2* 計算兩方陣相乘結果並輸出於指定文字檔，分析並比較三種方式的 performance；此外，試結合三種方式達到最高效率。

B. Method Introduction

(1) Psuedo Code

程式整體可以簡單分成三部分：

- **主程式：line 36 ~ 39 的 main func.**
- **Recursive 終止區：line 2 ~ 8**

  處理 size ≤ 158 的狀況，直接利用 *School-Book Method* 即可達到較高效率。
- **Recursive 區塊：line 12 ~ 32**

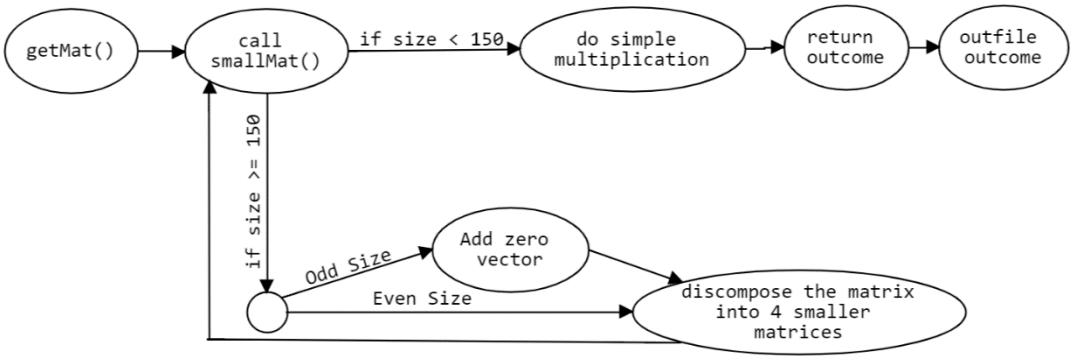
  處理 size > 158 的狀況，處理 size 可能為奇數的問題後，參考 *Method 1* 將 mat1、mat2 按照下圖分成 4 個小方陣後按照 *Method 2* 的優化算法得出 P_1 ~ P_7，進而相加減得到相乘矩陣。

```
        [mat1A | mat1B]
mat1 = [------+------]
        [mat1C | mat1D];

        [mat2A | mat2B]
mat2 = [------+------]
        [mat2C | mat2D];
```

```
1  smallMat(size, mat1, mat2)
2      if size <= 158
3          ////SCHOOL-BOOK METHOD
4          for i = 0 --> size, j = 0 --> size
5              mat[i][j] = 0;
6          for i = 0 --> size, j = 0 --> size
7              for k = 0 --> size
8                  mat[i][j] += mat1[i][k] * mat2[k][j];
9
10     else
11         ////METHOD 1 & 2
12         if size % 2 == 1
13             在column與row增加一行zero vector;
14
15         (令pi = mati1 * mati2)
16         mat11 = mat1A + mat1D; mat12 = mat2A + mat2D;
17         mat21 = mat1C + mat1D; mat22 = mat2A;
18         ...
19         mat71 = mat1B - mat1D; mat72 = mat2C + mat2D;
20
21         for i = 1 --> 7
22             pi = smallMat(newSize, mati1, mati2);
23
24         (加總)
25         for i = 0 --> newSize - 1, j = 0 --> newSize - 1
26             mat = p1 + p4 - p5 - p7;
27         for i = 0 --> newSize - 1, j = newSize --> size
28             mat = p3 + p5;
29         for i = newSize --> size, j = 0 --> newSize - 1
30             mat = p2 + p4;
31         for i = newSize --> size, j = newSize --> size
32             mat = p1 + p3 - p2 + p6;
33
34     return mat;
35
36 main function
37     [matSiz, mat1, mat2] = getMat(); (讀入方陣大小與兩方陣)
38     mat = smallMat(matSiz, mat1, mat2);
39     outfile(mat); (輸出計算結果)
```

(2) Flowchart



(3) 遭遇困難：**MatSiz 為奇數時如何拆矩陣？**

解決方法：

為了使拆矩陣的行為次數降至最低，小方陣的 size 必須為大方陣的一半，對於 size 為奇數的矩陣，拆開之前先額外加上一個 column 與一個 row 的全 zero vector，使 size 由原本的 2k+1 轉為 2k+2，進而拆成 size = k + 1 之小方陣。

(對照 Pseudo Code 中 line 12 ~ 13 的部分)

C. Analyzation

(1) *School-Book Method*

簡單分析 Pseodo Code 中 line 4 ~ 8，第二個 for 迴圈 i、j、k 都要從 0 增加至 size – 1，故當 size = n，需要 n^3 次 iterations，可得到 **Order of Growth = $\Theta(n^3)$**。

(2) *Method 1*

將 *Method 1* 簡單分析如下：

步驟	**次數 * 時間**
1 * 1 mat	T(1) → Θ(1)
補入 zero vector (size 為奇數時才需要補)	Θ(1) or Θ(0) → Θ(1)
呼叫 `smallMat()`	8 * T(⌈(n+1) / 2⌉)
矩陣相加	Θ((⌈(n+1) / 2⌉)^2) → Θ(n^2)

$$T(n) = \begin{cases} \theta(1), & n = 1 \\ 8 * T\left(\left\lceil \frac{n+1}{2} \right\rceil\right) + \theta(n^2+1), & n > 1 \end{cases} \longrightarrow \boldsymbol{T(n) = \theta\left(8^{log_2 n}\right) = \theta(n^3)}$$

(3) *Method 2*

與 *Method 1* 相似，唯一不同處為呼叫 `smallMat()`之次數減為 7。

略為調整 T(n)的計算過程如下：

$$T(n) = \begin{cases} \theta(1), & n = 1 \\ \mathbf{7} * T\left(\left\lceil \frac{n+1}{2} \right\rceil\right) + \theta(n^2+1), & n > 1 \end{cases} \longrightarrow \boldsymbol{T(n) = \theta\left(7^{\log_2 n}\right) = \theta(n^{2.807})}$$

D. Experiment Results

(1) **Size = 500** (tests on provided data: *case1.txt*)

```
[algo08@eng02 ~]$ ./mul1 case1.txt out1_1.txt
Time: 0.3s
[algo08@eng02 ~]$ ./mul2 case1.txt out1_2.txt
Time: 6.75s
[algo08@eng02 ~]$ ./mul3 case1.txt out1_3.txt
Time: 4.38s
[algo08@eng02 ~]$ ./best case1.txt out1_B.txt
Time: 0.25s
```

右圖中，*mul1*、*mul2*、*mul3* 分別為使用 *School-Book Method*、*Method 1*、*Method 2* 產生的執行檔；*best* 為依照不同 size 大小選用不同 method 的程式(如 Pseudo Code 所描述)。由執行時間可以觀察出以下幾個特性：

i. *School-Book Method* 的效率較 *Method 1*、*Method 2* 高的原因為 *Method 1* 相較於 *School-Book Method* 不僅沒有減少計算量，更需花額外時間呼叫記憶體；*Method 2* 雖然少了 1/8 次的計算量，但在拆至 size 較小時，額外呼叫記憶體的時間大於減少的計算時間，因此效率依舊較 *School-Book Method* 低。

ii. *Method 2* 效率高於 *Method 1* 顯示每八次乘法減少一次可以有效增加程式效率。

iii. 優化程式(*best*)的效率明顯高於 *Method 1*、*Method 2*，但與 *School-Book Method* 相較並沒有加快非常多，因為 size 還相當小，在拆成原本的 1/4 後即直接使用 *School-Book Method* 了，因此在執行時間上並沒有明顯差異。

(2) **Size = 1400** (自行產生的 data file)

```
[algo08@eng02 ~]$ ./mul1 case1400.txt out1400_1.txt
Time: 23.27s
[algo08@eng02 ~]$ ./best case1400.txt out1400_B.txt
Time: 3.95s
```

在提高 Matrices 的大小後，執行時間明顯的減少。與 *School-Book Method* 相較，size 較大時，呼叫記憶體的時間小於減少一次乘法所節省的時間，使程式速度加快許多；且 size 較小時，改採用 *School-Book Method* 以避免呼叫記憶體耗去過多時間，直接算出矩陣乘積以達最高效率。