Application: Image Registration 3D

- Two almost identical images (3D). For example a cat scan of an individual before and after a treatment.
- The problem is to transform one image into the other by translation, rotation, and shrinkage or expansion.
- For example by having both images almost identical we want to measure by how much a tumor has increased or decreased in size.
- Image registration is done by performing for each voxel \((256 \times 256 \times 256)\) we execute 100 iterations of the expression

\[
T = \sum A(x, y, z) * B(x, y, z) * C(x, y, z)
\]
• The \( A \) matrix is \( 12 \times 3 \), \( B \) is \( 3 \times 3 \) and \( C \) is \( 3 \times 1 \).

• The trees below specify the multiplication order \( (A \times B) \times C \) that requires 144 multiplications. But the multiplication order \( A \times (B \times C) \) requires only 45 multiplications.
Matrix Product Chains

Given the matrix product chain

\[ M_1 \times M_2 \times \ldots M_n, \]

\[ r_1 \text{ by } r_2 \times r_2 \text{ by } r_3 \times r_n \text{ by } r_{n+1} \]

where \( M_i \) is of size \( r_i \times r_{i+1} \), find an ordering for multiplying the matrices by multiplying two adjacent matrices using the classical algorithm.

\[ M_{p,q} \times M_{q,r} \rightarrow M_{p,r} \quad p \times q \times r \text{ operations} \]

The number of operations is \( p \times q \times r \)

125,000 mults

2,200 mults
Example:

\[ M_{10,20} \times M_{20,50} \times M_{50,1} \times M_{1,100} \]

\[ M_{50,1} \times M_{1,100} \rightarrow M_{50,100} \quad 5000 \text{ operations} \]

\[ M_{10,20} \times M_{20,50} \times M_{50,100} \]

\[ M_{20,50} \times M_{50,100} \rightarrow M_{20,100} \quad 100000 \text{ operations} \]

\[ M_{10,20} \times M_{20,100} \]

\[ M_{10,20} \times M_{20,100} \rightarrow M_{10,100} \quad 20000 \text{ operations} \]

Total operations: 125000

\[ M_{10,20} \times M_{20,50} \times M_{50,1} \times M_{1,100} \]

\[ M_{20,50} \times M_{50,1} \rightarrow M_{20,1} \quad 1000 \text{ operations} \]

\[ M_{10,20} \times M_{20,1} \times M_{1,100} \]

\[ M_{10,20} \times M_{20,1} \rightarrow M_{10,1} \quad 200 \text{ operations} \]

\[ M_{10,1} \times M_{1,100} \]

\[ M_{10,1} \times M_{1,100} \rightarrow M_{10,100} \quad 1000 \text{ operations} \]

Total operations: 2200
Greedy Method A

The greedy method: perform a cheapest (least expensive) multiplication first. This method does not generate an optimal solution all of the time. This greedy policy generates the tree on the left which is not optimal. The tree on the right hand side is an optimal one.

Cheapest First

```
12
  2 x 6
  6
   6 x 1
   1 x 1
```

18 multiplications

Optimal Solution

```
2
  2 x 1
  12
   2 x 6
   6 x 1
```

14 multiplications
Greedy Method B

The greedy method: The last multiplication should be a cheapest (least expensive) one. This method does not generate optimal solution all the time. This greedy policy generates the left hand side tree which is not optimal. The tree on the right hand side is an optimal one.

Cheapest First

Optimal Solution

40 Multiplications

37 Multiplications
Dynamic Prog.

Let $C_{i,j}$ be the minimum cost of computing

$$M_i \times M_{i+1} \times \ldots M_j$$

$$C_{i,j} = \begin{cases} 
0, & \text{if } i = j \\
\min_{i \leq k < j} (C_{i,k} + C_{k+1,j} + r_ir_{k+1}r_{j+1}), & \text{if } i < j 
\end{cases}$$

Let $kay_{i,j}$ be the value of $k$ that minimizes the computation of $C_{i,j}$.

The $kays$ can be used to find the actual ordering of the matrices (rather than just the cost) in an optimal solution.

We present three programs (from [Sa]: (1) Recursive (exponential time); (2) Recursive ($O(n^3)$ time); Iterative ($O(n^3)$ time).
int C(int i, int j) //Exponential time
{
    // Ret c(i,j) & compute kay(i,j) = kay[i][j].
    if (i == j) return 0; // one matrix
    if (i == j - 1) { // two matrices
        kay[i][i+1] = i;
        return r[i]*r[i+1]*r[i+2];}
    // more than two matrices
    // set u to min term for k = i
    int u = C(i,i) + C(i+1,j) + r[i]*r[i+1]*r[j+1];
    kay[i][j] = i;

    // compute remaining min terms and update u
    for (int k = i+1; k < j; k++) {
        int t = C(i,k) + C(k+1,j) + r[i]*r[k+1]*r[j+1];
        if (t < u) { // smaller min term
            u = t;
            kay[i][j] = k;}
    }
    return u;
}
void Traceback(int i, int j, int **kay)
// Output the multiplication ordering
{// Output best way to compute Mij.
  if (i == j) return;
  Traceback(i, kay[i][j], kay);
  Traceback(kay[i][j]+1, j, kay);
  cout << "Multiply M " << i << ", ", "
  << kay[i][j];
  cout << " and M " << (kay[i][j]+1)
  " , " << j << endl;
}
Initialize $c[i][j]$ to infinity

```c
int C(int i, int j) // Polynomial time $O(n^3)$
{ // Ret $c(i,j)$ & comp $kay(i, j) = kay[i][j]$. 
  // Avoid recomputations.

  // check if already computed
  if (c[i][j] is finite) return c[i][j];

  // $c[i][j]$ not computed before, compute now
  if (i == j) return 0; // one matrix
  if (i == j - 1) { // two matrices
    kay[i][i+1] = i;
    c[i][j] = r[i]*r[i+1]*r[i+2];
    return c[i][j];
  }

  // more than two matrices
  // set $u$ to min term for $k = i$
  int u = C(i,i) + C(i+1,j) + r[i]*r[i+1]*r[j+1];
  kay[i][j] = i;
```
// compute remaining min terms and update u
for (int k = i+1; k < j; k++) {
    int t = C(i,k) + C(k+1,j) + r[i]*r[k+1]*r[j+1];
    if (t < u) {// smaller min term
        u = t;
        kay[i][j] = k;
    }
}

c[i][j] = u;
return u;
Order the $C$’s with respect to the number of matrices in $C(i, j)$. Compute them in the following order: All the $C$’s with 1 matrix, then all the $C$’s with two matrices, etc.

\[
\begin{align*}
C_{1,6} \\
C_{1,5}, C_{2,6} \\
C_{1,4}, C_{2,5}, C_{3,6} \\
C_{1,3}, C_{2,4}, C_{3,5}, C_{4,6} \\
C_{1,2}, C_{2,3}, C_{3,4}, C_{4,5}, C_{5,6} \\
C_{1,1}, C_{2,2}, C_{3,3}, C_{4,4}, C_{5,5}, C_{6,6} \\
0 , 0 , 0 , 0 , 0 , 0 , 0
\end{align*}
\]

$O(n^2)$ $C$’s need to be computed. Each one takes at most $n$ time. Therefore the total time complexity is $O(n^3)$. 
Example

$n = 5$ and $r = (10, 5, 1, 10, 2, 10)$. I.e., $M_1$ is $10 \times 5$, $M_2$ is $5 \times 1$, $M_3$ is $1 \times 10$, $M_4$ is $10 \times 2$, and $M_5$ is $2 \times 10$.

$C(1, 5) \leftarrow \min \left\{ \begin{align*}
C(1, 1) + C(2, 5) + 10 \times 5 \times 10 \\
C(1, 2) + C(3, 5) + 10 \times 1 \times 10 \\
C(1, 3) + C(4, 5) + 10 \times 10 \times 10 \\
C(1, 4) + C(5, 5) + 10 \times 2 \times 10
\end{align*} \right\}$

$C(1, 5) = 190 \leftarrow \min \left\{ \begin{align*}
0 + 90 + 500 \\
50 + 40 + 100 \\
150 + 200 + 1000 \\
90 + 0 + 200
\end{align*} \right\}$
\[ C(2, 5) \leftarrow \min \begin{cases} C(2, 2) + C(3, 5) + 5 \times 1 \times 10 \\ C(2, 3) + C(4, 5) + 5 \times 10 \times 10 \\ C(2, 4) + C(5, 5) + 5 \times 2 \times 10 \end{cases} \]

\[ C(2, 5) = 90 \leftarrow \min \begin{cases} 0 + 40 + 50 \\ 50 + 200 + 500 \\ 30 + 0 + 100 \end{cases} \]

\[ C(3, 5) \leftarrow \min \begin{cases} C(3, 3) + C(4, 5) + 1 \times 10 \times 10 \\ C(3, 4) + C(5, 5) + 1 \times 2 \times 10 \end{cases} \]

\[ C(3, 5) = 40 \leftarrow \min \begin{cases} 0 + 200 + 100 \\ 20 + 0 + 20 \end{cases} \]
\[ C(2, 4) \leftarrow \min \begin{cases} 
C(2, 2) + C(3, 4) + 5 \times 1 \times 2 \\
C(2, 3) + C(4, 4) + 5 \times 10 \times 2 
\end{cases} \]

\[ C(2, 4) = 30 \leftarrow \min \begin{cases} 
0 + 20 + 10 \\
50 + 0 + 100 
\end{cases} \]

\[ C(1, 3) \leftarrow \min \begin{cases} 
C(1, 1) + C(2, 3) + 10 \times 5 \times 10 \\
C(1, 2) + C(3, 3) + 10 \times 1 \times 10 
\end{cases} \]

\[ C(1, 3) = 150 \leftarrow \min \begin{cases} 
0 + 50 + 500 \\
50 + 0 + 100 
\end{cases} \]
\[
C(1, 4) \leftarrow \min \begin{cases}
C(1, 1) + C(2, 4) + 10 \times 5 \times 2 \\
C(1, 2) + C(3, 4) + 10 \times 1 \times 2 \\
C(1, 3) + C(4, 4) + 10 \times 10 \times 2
\end{cases}
\]

\[
C(1, 4) = 90 \leftarrow \min \begin{cases}
0 + 30 + 100 \\
50 + 20 + 20 \\
150 + 0 + 200
\end{cases}
\]
There is a simple $O(n \log n)$ greedy algorithm for the MPC problem, but with a very complex proof (Hu and Shing).

The nice property about dynamic programming is that it can be used to solve more complex programs after making minor modifications to the code. For example see the next version of the matrix product chain problem. It does not appear in any textbook.
Assume there is unbounded number of processors. Each CPU may be computing a different matrix product at the same time. If two CPUs are computing matrix products then the computation time (real time) is the maximum of the times to multiply the matrices, i.e., the max of the time required for the first product and the time required for the second product. With one processor (sequential computation) the time (real time) is the sum of the times required to multiply the matrices.
Computing Optimal MPC Time

The cost of computing the product chain:
Sequentially (210: cost given below the node) and
Distributed (160: cost given to the right of the
device).

S :: 260
D :: 170
Computing Optimal MPC Time

The cost of computing the product chain:
Sequentially (19) and Distributed (7).
Optimal MPC in Distributed System

\[ C_{i,j} = \begin{cases} 
0, & \text{i=j} \\
\min_{i \leq k < j} (\max\{C_{i,k}, C_{k+1,j}\} + r_i r_{k+1} r_{j+1}) & \text{i<j} 
\end{cases} \]

Time Complexity for computing an optimal solution is the same as in the sequential case.
Example (Distributed MPC)

$n = 5$ and $r = (10, 5, 1, 10, 2, 10)$. I.e., $M_1$ is $10 \times 5$, $M_2$ is $5 \times 1$, $M_3$ is $1 \times 10$, $M_4$ is $10 \times 2$, and $M_5$ is $2 \times 10$.

\[
C(1, 5) \leftarrow \min \left\{ \begin{array}{l}
\max\{C(1, 1), C(2, 5)\} + 10 \times 5 \times 10 \\
\max\{C(1, 2), C(3, 5)\} + 10 \times 1 \times 10 \\
\max\{C(1, 3), C(4, 5)\} + 10 \times 10 \times 10 \\
\max\{C(1, 4), C(5, 5)\} + 10 \times 2 \times 10 \\
\end{array} \right. 
\]

\[
C(1, 5) = 150 \leftarrow \min \left\{ \begin{array}{l}
\max\{0, 90\} + 500 \\
\max\{50, 40\} + 100 \\
\max\{150, 200\} + 1000 \\
\max\{70, 0\} + 200 \\
\end{array} \right. 
\]

\[
C(1, 1) = C(5, 5) = 0.
\]

\[
C(1, 2) = 50
\]

\[
C(4, 5) = 200
\]
\[ C(2, 5) \leftarrow \min \left\{ \begin{array}{l} \max\{C(2, 2), C(3, 5)\} + 5 \times 1 \times 10 \\ \max\{C(2, 3), C(4, 5)\} + 5 \times 10 \times 10 \\ \max\{C(2, 4), C(5, 5)\} + 5 \times 2 \times 10 \end{array} \right. \]

\[ C(2, 5) = 90 \leftarrow \min \left\{ \begin{array}{l} \max\{0, 40\} + 50 \\ \max\{50, 200\} + 500 \\ \max\{30, 0\} + 100 \end{array} \right. \]

\[ C(2, 2) = C(3, 3) = 0. \]

\[ C(2, 3) = 50 \]

\[ C(4, 5) = 200 \]

\[ C(3, 5) \leftarrow \min \left\{ \begin{array}{l} \max\{C(3, 3), C(4, 5)\} + 1 \times 10 \times 10 \\ \max\{C(3, 4), C(5, 5)\} + 1 \times 2 \times 10 \end{array} \right. \]

\[ C(3, 5) = 40 \leftarrow \min \left\{ \begin{array}{l} \max\{0, 200\} + 100 \\ \max\{20, 0\} + 20 \end{array} \right. \]
\[ C(2, 4) \leftarrow \min \left\{ \begin{array}{l}
max\{C(2, 2), C(3, 4)\} + 5 \times 1 \times 2 \\
max\{C(2, 3), C(4, 4)\} + 5 \times 10 \times 2
\end{array} \right\} \]

\[ C(2, 4) = 30 \leftarrow \min \left\{ \begin{array}{l}
max\{0, 20\} + 10 \\
max\{50, 0\} + 100
\end{array} \right\} \]

\[ C(1, 3) \leftarrow \min \left\{ \begin{array}{l}
max\{C(1, 1), C(2, 3)\} + 10 \times 5 \times 10 \\
max\{C(1, 2), C(3, 3)\} + 10 \times 1 \times 10
\end{array} \right\} \]

\[ C(1, 3) = 150 \leftarrow \min \left\{ \begin{array}{l}
max\{0, 50\} + 500 \\
max\{50, 0\} + 100
\end{array} \right\} \]
\[ C(1, 4) \leftarrow \min \left\{ \begin{array}{l}
\max\{C(1, 1), C(2, 4)\} + 10 \times 5 \times 2 \\
\max\{C(1, 2), C(3, 4)\} + 10 \times 1 \times 2 \\
\max\{C(1, 3), C(4, 4)\} + 10 \times 10 \times 2
\end{array} \right\} \]

\[ C(1, 4) = 70 \leftarrow \min \left\{ \begin{array}{l}
\max\{0, 30\} + 100 \\
\max\{50, 20\} + 20 \\
\max\{150, 0\} + 200
\end{array} \right\} \]

\[ C(3, 4) = 20 \]