Algorithms and Data Structures II

Lecture 10:

Algorithm Design Techniques: Dynamic Programming

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

Dynamic programming

- Dynamic programming solves a given problem by combining the solutions to subproblems. Programming here means a tabular method, not writing a computer code.
- Dynamic programming is different from divide-and-conquer which partitions the problem into independent subproblems, solves the subproblems recursively, and then combines the solutions to solve the original problem.

Dynamic programming

- Dynamic programming is applicable when the subproblems are not independent but dependent, i.e., subproblems share subproblems. In this context, a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subproblems.

Dynamic programming

- A dynamic programming algorithm solves every subproblem only once and then saves its answer in a table to avoid recomputing the answer every time the subproblem is encountered.

Warshall's and Floyd's algorithms

- Warshall's algorithm to find the transitive closure of a graph and Floyd's algorithm to find the shortest paths between every pair of nodes of a weighted graph are examples of dynamic programming algorithms.

Warshall's Algorithm

- For every pair of nodes i and j, we find the path from i to j which may pass through nodes $1, 2, \ldots, k$, using the path from i to j, the paths from i to k and from k to j, that may pass through nodes $1, 2, \ldots, k-1$ (the result of the subproblems which is kept in the table $C^{k-1}[i, j]$).

$$C^k[i, j] = C^{k-1}[i, j] \vee (C^{k-1}[i, k] \wedge C^{k-1}[k, j])$$

matrix chain product

- Another example of dynamic programming algorithm for matrix chain product problem.
- Given an $\ell \times m$ matrix A and an $m \times q$ matrix B, the product

  $$\begin{array}{ccccc} C & = & A & \times & B \\ (\ell, q) & & (\ell, m) & & (m, q) \end{array}$$

  is an $\ell \times q$ matrix, for $1 \leq i \leq \ell$ and $1 \leq j \leq q$,

  $$C[i, j] = \sum_{k=1}^{m} A[i, k] \times B[k, j]$$

  If we use the standard method to compute C, $\ell \times m \times q$ multiplications are used.

matrix chain product

- Assume n matrices are to be multiplied together:

$$\begin{array}{cccccc} M_1 & M_2 & M_3 & \cdots & M_{n-1} & M_n, \\ (r_1, r_2) & (r_2, r_3) & (r_3, r_4) & \cdots & (r_{n-1}, r_n) & (r_n, r_{n+1}) \end{array}$$

  where the matrices satisfy the constraint that M_i has r_i rows and r_{i+1} columns for $1 \leq i \leq n$.

matrix chain product

- The product can be computed in many orders, e.g., in the left-to-right order $(\cdots((M_1M_2)M_3)\cdots M_{n-1})M_n$, or in the right-to-left order $M_1(M_2\cdots(M_{n-2}(M_{n-1}M_n))\cdots)$. Many other orders are also possible.
- The total number of multiplications used in different orders are different.

Example

- Given the matrix chain A_1, A_2, A_3, A_4. Let A_1 be 30×1, A_2 be 1×40, A_3 be 40×10, A_4 be 10×25.

Order of Multiplications	Number ofscalar multiplications
$A_1 \times (A_2 \times (A_3 \times A_4))$	$40 \times 10 \times 25 + 1 \times 40 \times 25 + 30 \times 1 \times 25 = 11,750$
$A_1 \times ((A_2 \times A_3) \times A_4)$	$1 \times 40 \times 10 + 1 \times 10 \times 25 + 30 \times 1 \times 25 = 1,400$
$(A_1 \times A_2) \times (A_3 \times A_4)$	$30 \times 1 \times 40 + 40 \times 10 \times 25 + 30 \times 40 \times 25 = 41,200$
$(A_1 \times (A_2 \times A_3)) \times A_4$	$1 \times 40 \times 10 + 30 \times 1 \times 10 + 30 \times 10 \times 25 = 8,200$
$((A_1 \times A_2) \times A_3) \times A_4$	$30 \times 1 \times 40 + 30 \times 40 \times 10 + 30 \times 10 \times 25 = 20,700$

matrix chain product problem

- The matrix chain product problem is to find the order of multiplying the matrices that minimizes the total number of multiplications used.

matrix chain product problem

- Now, we use dynamic programming approach to find a solution for the matrix chain product problem:
  - First, there is only one way to compute $(M_1 M_2)$ which takes $r_1 \times r_2 \times r_3$ multiplications, $(M_2 M_3)$ which takes $r_2 \times r_3 \times r_4$ multiplications, ..., and $(M_{n-1} M_n)$ which takes $r_{n-1} \times r_n \times r_{n+1}$ multiplications.
  - We record those costs in a table.

matrix chain product problem[Next Step]

- Next, we find the best way to multiply successive triples, $(M_1M_2M_3), (M_2M_3M_4), \ldots, (M_{n-2}M_{n-1}M_n)$. The minimum cost of $(M_1M_2M_3)$ is the smaller of the cost of $((M_1M_2)M_3) = r_1 \times r_2 \times r_3 + r_1 \times r_3 \times r_4$ and the cost of $(M_1(M_2M_3)) = r_2 \times r_3 \times r_4 + r_1 \times r_2 \times r_4$.

matrix chain product problem[Next Step] (contd)

- In finding the costs of $((M_1 M_2)M_3)$ and $(M_1(M_2 M_3))$, we do not recompute the costs of $(M_1 M_2)$ and $(M_2 M_3)$ but simply find them from the table.
- The minimum costs of $(M_1 M_2 M_3), (M_2 M_3 M_4), \ldots, (M_{n-2} M_{n-1} M_n)$ are kept in the table.

matrix chain product problem[in general]

- We find the best way to compute $(M_i M_{i+1} \cdots M_{i+j})$ by finding the minimum cost of computing $(M_i M_{i+1} \cdots M_{k-1})(M_k \cdots M_{i+j})$ for $i < k \leq i + j$.
- The cost of $(M_i M_{i+1} \cdots M_{k-1})(M_k \cdots M_{i+j})$ is the sum of the cost of $(M_i M_{i+1} \cdots M_{k-1})$, the cost of $(M_k \cdots M_{i+j})$, and $r_i \times r_k \times r_{i+j+1}$.
- The cost of $(M_i M_{i+1} \cdots M_{k-1})$ and the cost of $(M_k \cdots M_{i+j})$ are found from the table.
- The minimum cost of $(M_i M_{i+1} \cdots M_{i+j})$ is kept in the table.

$cost[\ ][\ ]$

- Let $cost[\ ][\ ]$ be a 2D array such that $cost[i][i+j]$ keeps the minimum cost of computing the product $M_i M_{i+1} \cdots M_{i+j}$. For example,
  - $cost[1][2]$ keeps the cost of $M_1 M_2$,
  - $cost[1][3]$ keeps the minimum cost of $M_1 M_2 M_3$,
  - $cost[3][5]$ keeps the minimum cost of $M_3 M_4 M_5$,

    and
  - $cost[1][n]$ keeps the minimum cost of $M_1 M_2 M_3 \cdots M_{n-1} M_n$.
  - $cost[i][i], i = 1, 2, \ldots, n$ are 0.
  - $cost[i][j], i > j$ are NOT used.

$best[\,][\,]$

- Another array $best[\,][\,]$ is used to derive a way of multiplication which has the minimum cost. $best[i][i+j]$ keeps a value k which indicates that the product of $(M_i M_{i+1} \cdots M_{i+j})$ should be done by $(M_i M_{i+1} \cdots M_{k-1})(M_k \cdots M_{i+j})$.
- For example, if $best[1][3]$ has the value 2 then $(M_1 M_2 M_3)$ should be computed as $(M_1(M_2 M_3))$.

program with $cost[\,][\,]$ and $best[\,][\,]$

- The following is a program to compute $cost[\,][\,]$ and $best[\,][\,]$.
- In the program, array $r[i]$ and $r[i+1]$ have the numbers of rows and columns of matrix M_i, respectively.

program with $cost[\ ][\ ]$ and $best[\ ][\ ]$

```
for (i=1;i<=n;i++)
  for(j=i+1;j<=n;j++)cost[i][j]=infty;
for (i=1;i<=n;i++) cost[i][i]=0;
for (j=1;j<n;j++)
  for (i=1;i<=n-j;i++)
    for (k=i+1;k<=i+j;k++){
      t=cost[i][k-1]+cost[k][i+j]
          +r[i]*r[k]*r[i+j+1];
      if (t<cost[i][i+j])
        {cost[i][i+j]=t;best[i][i+j]=k;}
    }
```

understanding of the program

- In the the program $cost[i][i+j]$, $1 \le j \le n-1$ and $1 \le i \le n-j$, are computed by

$$cost[i][i+j] = \min_{k=i+1}^{i+j}\{cost[i][k-1] + cost[k][i+j] \\ + r[i]*r[k]*r[i+j+1]\}$$

Note that $cost[i][k-1]$ and $cost[k][i+j]$ do not need to be recomputed. The following is a procedure that finds the best order of multiplying $M_i \cdots M_j$.

understanding of the process

- For $j = 1$, the costs of $M_1M_2, M_2M_3, M_3M_4, \ldots, M_{n-1}M_n$ are computed.
- For $j = 2$, the minimum costs of $M_1M_2M_3, M_2M_3M_4, \ldots, M_{n-2}M_{n-1}M_n$ are computed.
- For $j = n - 1$, the minimum cost of $M_1M_2M_3 \cdots M_{n-1}M_n$ is computed.

for output

```
/*Find the best multiplication order for n
matrices;uses the best[][] from previous code*/
order(int i,int j){
  if (i==j) printf("M%d",i);
  else {
    printf("(");
    order(i,best[i][j]-1);
    order(best[i][j],j);
    printf(")");
  }
}
```

Note for square matrices:

Any ordering can not save multiplications since the costs for multiplications are always the same in any order.

In other words, no need to consider the ordering.