CSF 364
Design & Analysis of Algorithms

Algorithm Design Techniques

Matrix-Chain Multiplication:

- Optimal Substructure Property
- Recurrence Relation
- Dynamic Programming Algorithm
  - Space and Time Complexity

Example – MCM – Optimal Sub-Structure

- Let $M_{i..j}$ denote the result of the product M_i * M_{i+1} * … M_j
- An optimal parenthesization splits the chain between M_k and M_{k+1} for some k, where 1<=k<n.
  - The resulting parenthesizations for the subchains must be optimal for the respective subchains.
    - Why?
- i.e. optimal substructure property holds for MCM.
  - Hence MCM is a candidate for Dynamic Programming.

EXAMPLE – MCM - RECURRENCE

- Let m[i,j] be the minimum number of scalar multiplications required for computing $M_{i..j}$
  - Then m[1,n] is the required value (to be computed).
- m[i,j] can be defined recursively as follows:
  - $m[i,j] = 0$ if $i=j$,
  - $m[i,j] = \min_{i <= k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j \}$ if $i<j$

Example – McM – DP Solution - Outline

Recurrence: (for j-i > 0)
$m[i,j] = \min_{i <= k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j \}$

```
DP_MCM(p,n)   // p[i-i]*p[i] is the size of matrix Mi, 0<i<=n
{
   for (i=1; i<n; i++) m[i,i] =0;
   for (len=2; len<=n; l++)   // len is length of the sequence i..j
   …
   return m ;
}
```

/* Use induction on the start of the chain (i.e. i)
as well as the length of the chain (i.e. j-i+1) */

EXAMPLE – MCM – DP SOLUTION

Recurrence: (for j-i > 0)

$m[i,j] = \min_{i <= k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j \}$

```
DP_MCM(p,n)   // p[i-i]*p[i] is the size of matrix Mi, 0<i<=n
{
   for (i=1; i<n; i++) m[i,i] =0;
   for (len=2; len<=n; len++)   // len is length of the sequence i..j
        for (i=1; i<=n-len+1; i++)  {
           j = i+len-1;
           ...
           for (k = i; k<j; k++) { // compute min. over all k
                q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
                ...
            }
        }
  return m;
}
```

EXAMPLE – MCM – DP SOLUTION

Recurrence: (for j-i > 0)

$m[i,j] = \min_{i <= k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j \}$

```
DP_MCM(p,n)    // p[i-i]*p[i] is the size of matrix Mi, 0<i<=n
{
   for (i=1; i<n; i++) m[i,i] =0;
   for (len=2; len<=n; len++)   // len is length of the sequence i..j
        for (i=1; i<=n-len+1; i++)  {
            j = i+len-1;
            m[i,j] = MAX_INT; // identity for minimum
            for (k = i; k<j; k++) { // compute min. over all k
                 q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
                 if (q < m[i,j]) then m[i,j] = q;  // update min.
             }
        }
  return m;
}
```

Example – McM – DP Solution

```
DP_MCM(p,n)
{
   for (i=1; i<n; i++) m[i,i] =0;
   for (len=2; len<=n; len++)
       for (i=1; i<=n-len+1; i++)  {
          j = i+len-1;
          m[i,j] = MAX_INT;
          for (k = i; k<j; k++) {
              q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
              if (q < m[i,j]) then m[i,j] = q;
          }
       }
   return m;
}
```

This procedure computes the minimal number of scalar multiplications required.

- How do we get the parenthesization that results in the minimal number of scalar multiplications?

Example – McM – DP Solution

```
DP_MCM(p,n) {
for (i=1; i<n; i++) m[i,i] =0;
  for (len=2; len<=n; len++)
    for (i=1; i<=n-len+1; i++)  {
      j = i+len-1;  m[i,j] = MAX_INT;
      for (k = i; k<j; k++) {
          q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
          if (q < m[i,j]) then {m[i,j] = q; s[i,j] = k; /*the point of split */ }
      }
    }
  return (m, s) ;
}
```

k is the (current) optimal point of split for the chain **i..j**

Only **m[1,n]** is needed as the (final) measure.

But what about **s**? Do we need to return all of **s**?

Example – McM – DP Solution

```
DP_MCM(P,n) {
for (i=1; i<n; i++) m[i,i] =0;
   for (len=2; len<=n; len++)
      for (i=1; i<=n-len+1; i++)  {
         j = i+len-1;  m[i,j] = MAX_INT;
         for (k = i; k<j; k++) {
               q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
               if (q < m[i,j]) then {m[i,j] = q; s[i,j] = k; /*the point of split */ }
         }
      }
  return (m, s) ;
}
```

Time Complexity?

Space Complexity?

- Can the space be pruned? or
- How much space can be pruned under what conditions?