CS 4/56101

Design and Analysis of Algorithms

Kent State University

Dept. of Math & Computer Science

LECT-16

Dynamic Programming

2

Dynamic Programming

- Dynamic Programming, like the divide-and-conquer method, solves problems by combining the solutions to sub-problems.

- Pure divide-and-conquer:
  - divides problems into independent sub-problems,
  - solves the sub-problem recursively, and then,
  - combines their solutions to solve the original sub-problem.

- Dynamic programming in contrast is used when the sub-problems are not independent, that is sub-problems share sub-problems.
- It is typically applied to optimization problems.

Example:
Matrix Chain
Multiplication

Matrix Multiplication

```
MATRIX-MULTIPLY(A, B)
if columns[A] ≠ rows[B]
   then error "incompatible dimensions"
   else for i ← 1 to rows[A]
            do for j ← 1 to columns[B]
                   do C[i, j] ← 0
                      for k ← 1 to columns[A]
                          do C[i, j] ← C[i, j] + A[i, k] · B[k, j]
        return C
```

- Cost of multiplying A[p][q] x B[q][r] is p.q.r
- What is the cost of multiplying three matrices A, B, and C of sizes 10x100, 100x5, and 5x50?
- How to find the best way of multiplying?

DESIGN & ALALYSIS OF ALGORITHM

Matrix Chain Multiplication

- Given a chain A_1, A_2, A_3, .. A_n of n matrices, such than A_i has dimension p_{i-1}x p_i, find the sequence of multiplication that will result in minimum number of scalar multiplication.
  - Recursive Cost Function Catalan numbers:

$$p(n) = \begin{cases} 1 \; if \quad n = 1 \\ \sum_{k=1}^{n-1} p(k).P(n-k), if \quad n > 1 \end{cases}$$

$$P(n+1) = \Omega\left(\frac{4^n}{n^{\frac{3}{2}}}\right)$$

$$(A_1.(A_2.(A_3.A_4)))$$
$$((A_1.A_2).(A_3.A_4))$$
$$((A_1.(A_2.A_3)).A_4)$$

DESIGN & ALALYSIS OF ALGORITHM

Observations

- Existence of Optimal Substructure: In an optimum sequence of decisions, each subsequence must also be optimum.
- $(A_1\ A_2\ A_3).(A_4.\ A_5.\ A_6)$
  - Total cost is C(1..3) + C(4..6) + cost of multiplying the two final matrices.
- Recursive Solution Possible: If m[i,j] is the optimum cost of multiplying all matrices between i^{th} and j^{th} matrices ….($A_i\ A_{i+1}$ …. A_j)..
  - if i==j then m[i,j]= 0
  - otherwise,

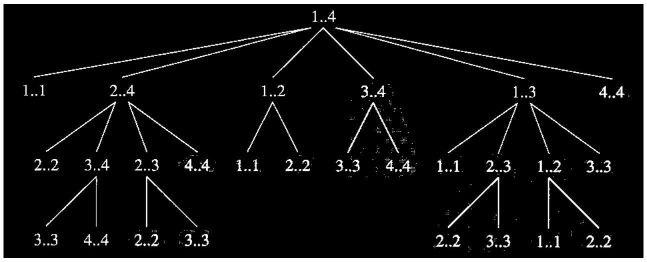
$$m[i,\ j] = \min_{i \le k \le j}\{m[i,k] + m[k+1,\ j] + p_{i-1}.p_k.p_j\}$$

Recursive Solution


```
RECURSIVE-MATRIX-CHAIN(p, i, j)
if i = j
    then return 0
m[i, j] ← ∞
for k ← i to j − 1
    do q ← RECURSIVE-MATRIX-CHAIN(p, i, k)
              + RECURSIVE-MATRIX-CHAIN(p, k + 1, j) + p_{i−1} p_k p_j
       if q < m[i, j]
          then m[i, j] ← q
return m[i, j]
```

- Running time is exponential $O(2^n)$!

Observation-2



- Existence of Overlapping Sub-problem: the same sub-sequence is part of many super sequences.
- For a string of limited size, the actual number of subproblems are quite small. $O(n^2)$ only!

DESIGN & ALALYSIS OF ALGORITHM

Dynamic Programming Solution: A Bottom up approach

- Compute the optimum cost for multiplying all matrix chains of size 2.
- Store them in a matrix m[i,j], when i-j spans two matrices.
- Use the above values to compute optimum cost for multiplying all matrix chains of size 3.
- Then size 4 .. Up to size n.

DESIGN & ALALYSIS OF ALGORITHM

Algorithm


```
MATRIX-CHAIN-ORDER(p)
n ← length[p] − 1
for i ← 1 to n
      do m[i, i] ← 0
for l ← 2 to n
      do for i ← 1 to n − l + 1
             do j ← i + l − 1
                m[i, j] ← ∞
                for k ← i to j − 1
                    do 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
return m and s
```

Best k for optimum division of the sequence i-j

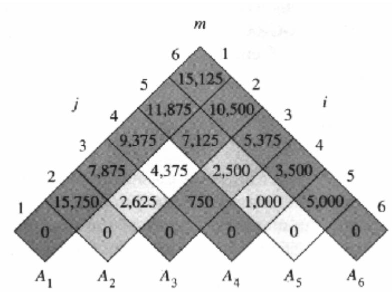
Example

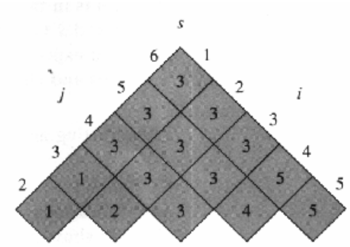
DESIGN &
ALALYSIS OF
ALGORITHM

matrix	dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

Example

matrix	dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25





DESIGN & ALALYSIS OF ALGORITHM

LECT-16, S-13

ALG00S, javed@kent.edu

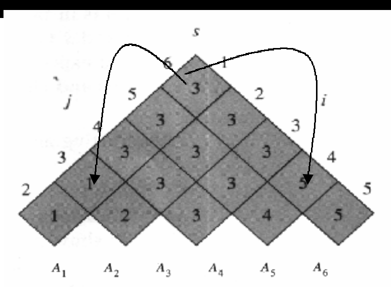
Javed I. Khan@1999

Constructing the Optimal Solution

```
MATRIX-CHAIN-MULTIPLY(A, s, i, j)
if j > i
   then X ← MATRIX-CHAIN-MULTIPLY(A, s, i, s[i, j])
        Y ← MATRIX-CHAIN-MULTIPLY(A, s, s[i, j] + 1, j)
        return MATRIX-MULTIPLY(X, Y)
   else return A_i
```

In the example of Figure 16.1, the call MATRIX-CHAIN-MULTIPLY($A, s,$ $1, 6$) computes the matrix-chain product according to the parenthesization

$$((A_1(A_2A_3))((A_4A_5)A_6)) \ . \tag{16.3}$$



DESIGN & ALALYSIS OF ALGORITHM

LECT-16, S-14

ALG00S, javed@kent.edu

Javed I. Khan@1999

Complexity of Algorithm

```
MATRIX-CHAIN-ORDER(p)
n ← length[p] − 1
for i ← 1 to n
     do m[i, i] ← 0
for l ← 2 to n
     do for i ← 1 to n − l + 1
            do j ← i + l − 1
               m[i, j] ← ∞
               for k ← i to j − 1
                   do 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
return m and s
```

Best k for optimum division of the sequence i-j

- Running time?
- Space?

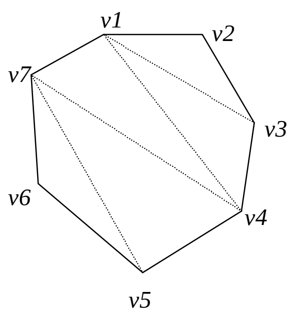
DESIGN & ALALYSIS OF ALGORITHM

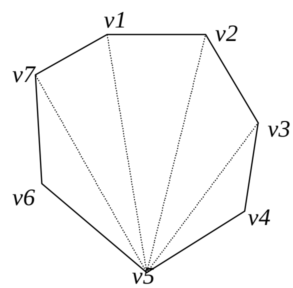
ALG00S, javed@kent.edu
Javed I. Khan@1999

Example: Optimal Polygon Triangulation

Polygon Triangulation

- We are given a convex polygon $P=<v_0,v_1,\ldots v_{n-1}>$ and a weight function w defined on triangles formed by sides and chords of P. The problem is to find a triangulation that minimizes the sum of the weights of the triangles in the triangulation.

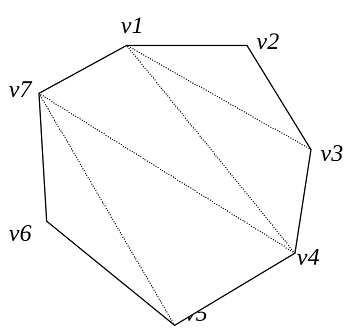


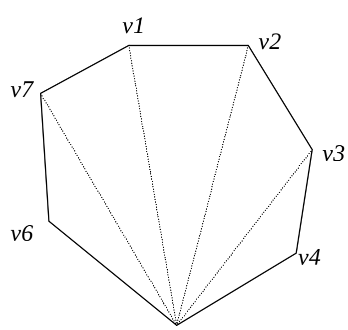


$$w(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$$

Observations

- Optimum substructure:
- Overlapping subproblems:





Dynamic programming Solution

- For all degenerated polygon of size 2, $<v_{i-1}, v_i>$ cost = zero.
- For all polygons of size 3 the cost is

$$w(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$$

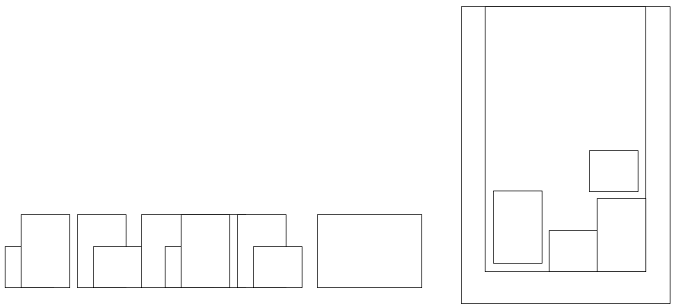
- For all polygons of size 4 or more try all division point k and pick the best:

$$t[i, j] = \min_{i \le k \le j-1} \{t[i,k] + t[k+1, j] + w(\Delta v_{i-1} v_k v_j)\}$$

Example:
Knap/ Sack Problem

1/0 Knapsack Problem

- You have a shopping bag (knapsack) with capacity C lb. There are n items in a super market. Each item has a value p_i and weight w_i. Which of the n items will you pick to maximize your profit?

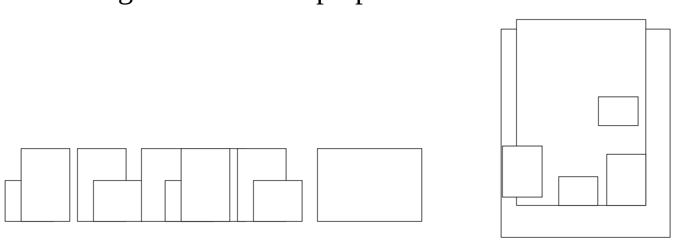


DESIGN & ALALYSIS OF ALGORITHM

LECT-16, S-21
ALG00S, javed@kent.edu
Javed I. Khan@1999

Observations

- Optimum substructure: If a solution is optimum for a large profit P with weight W items, each of the smaller subsolutions with profit P-c_i and weight W-w_i are also optimum.
- Overlapping Subproblems: An optimum solution with smaller subset of objects in a bag can be part of a large number of superproblems.



DESIGN & ALALYSIS OF ALGORITHM

LECT-16, S-22
ALG00S, javed@kent.edu
Javed I. Khan@1999

Problem Formulation

- Objective is to maximize

$$\sum_{i=1}^{n} p_i x_i$$

- subject to constraints:

$$\sum_{i=1}^{n} w_i x_i \le c$$

- Select the 0/1 values for x's.

Solution

- Let f(i,y) denote the profit value of an optimal solution with remaining capacity y and remaining objects i,i+1, ….n.
- When, there is only the last object (terminating condition):

$$f(n,y) = \begin{cases} p_n & if \;\; y \ge w_n \\ 0 & 0 \le y < w_n \end{cases}$$

- Otherwise (recursion):

$$f(i,y) = \begin{cases} \max\{f(i+1,y), f(i+y, y-w_i)+p_i\} & if \;\; y \ge w_i \\ f(i+1,y) & 0 \le y < w_i \end{cases}$$

Recursive Solution


```
F(i,y)
{   if(i=n) return (y<w[n]?
    0:p[n];
    if(y<w[i]) return f(i+1,y);
    return
    max(F(i+1,y),F(i+1,y-
    w[i])+p[i]);
}
```

Complexity?

Dynamic Prog. Solution

DESIGN &
ALALYSIS OF
ALGORITHM

- Matrix f[i][y], will store the best profit value for all remaining capacity y smaller than C, for remaining objects.
- A bottom up approach, first computes f(n,*). For all y less that w_n it is zero. For all y between w_n and C it is W_n. (terminating condition)
- Now compute f(i,*) in the order i=n-1, n-2….2. (use the recursion condition).
- Complexity: O(nc).