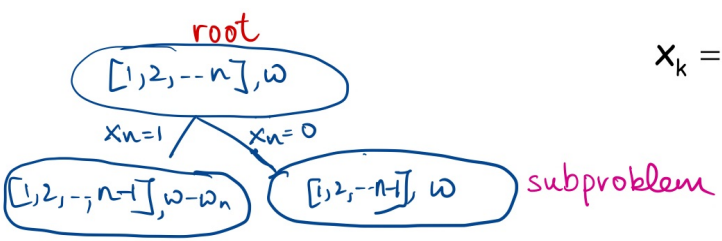
SEE PPT for previous topics

0-1 Knapsack Problem

Given a set of n unbreakable unique items, each with a weight w_k and a value v_k, determine the subset of items such that the total weight is less or equal to a given knapsack capacity W and the total value is as large as possible.

- Fractional knapsack – greedy approach
- 0-1 knapsack, brute force – $O(2^n)$

Decision Tree



$$x_k = \begin{cases} 1, \text{item k selected} \\ 0, \text{item k not selected} \end{cases}$$

OPT[K, X]

K = # of items available, $0 \le K \le n$

X = is a capacity, $0 \le X \le W$

Subproblems

Let $OPT[k, X]$ be the maximum value achievable using a knapsack of capacity X with k items.

Our choices:

① $x_k = 1$: $OPT[k, X] = v_k + OPT[k-1, X-w_k]$

② $x_k = 0$: $OPT[k, X] = OPT[k-1, X]$

Recurrence Formula

table

$$OPT[k, X] = MAX\left[v_k + \underbrace{OPT[k-1, X-w_k]}_{O(1)},\ \underbrace{OPT[k-1, X]}_{O(1)\text{ table lookup}} \right]$$

$k-1$? $X-w_k$

Base cases

$OPT[0, X] = 0$

$OPT[k, 0] = 0$

$OPT[k, X] = OPT[k-1, X]$, if $w_k > X$

Tracing the Algorithm

n = 4, W = 5

(weight, value) = (2,3), (3,4), (4,5), (5,6)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
1,2	0	0	3	4	4	3+4
1,2,3	0	0	3	4	5	7
1,2,3,4	0	0	3	4	5	7

knapsack capacity

items

OPT[n, w]

Pseudo-code

Input

```
int knapsack(int W, int w[], int v[], int n) {
  int Opt[n+1][W+1];
  for (k = 0; k <= n; k++) {
    for (x = 0; x <= W; x++) {
      if (k==0 || x==0) Opt[k][x] = 0;
      if (w[x] > x) Opt[k][x] = Opt[k-1][x];
      else
        Opt[k][x] = max( v[k] + Opt[k-1][x - w[k]],
                         Opt[k-1][x] );
    }
  }
  return Opt[n][W];
}
```

Pseudo-Polynomial Runtime

exponential

Definition. A numeric algorithm runs in pseudo-polynomial time if its running time is polynomial in the numeric value of the input but is exponential in the length of the input.

$V = \max(v_1, v_2, \ldots v_n)$

0-1 Knapsack is pseudo-polynomial algorithm, $T(n) = \Theta(n \cdot W)$

Runtime complexity must be a function of input size

Input size = $O(\log W + n \cdot \log W + n \cdot \log V + \log N)$

Runtime = $O(n \cdot W)$

exp

Input Size = $O(n \cdot \log W)$

Actual Runtime : $O(n \cdot 2^{\text{input size of } W})$

How would you find the actual items?

The table built in the algorithm does not show the optimal items, but only the maximum value. How do we find which items give us that optimal value?

n = 4, W = 5
(weight, value) = (2, 3), (3, 4), (4, 5), (5, 6)

→ If arr[i][j] is got from arr[i-1][j] it is not picked, continue to go up.

→ Else: move the pointer from current to [i-1][j-w_k]

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

DP approach

1. Define in plain English subproblem to be solved.
2. Write the recurrence relation for subproblems
3. Write pseudo code to compute the optimal value
4. Compute the runtime of the above DP algorithm in terms of i/p size.

Discussion Problem 1

You are to compute the minimum number of coins needed to make change for a given amount m. Assume that we have an unlimited supply of coins. All denominations d_k are sorted in ascending order:

$$1 = d_1 < d_2 < \ldots < d_n$$

Step 1:

Let OPT[k, x] be the minimum number of coins to represent x ($0 \le x \le m$) using first k ($1 \le k \le n$) denominations.

Step 2:

$$OPT[k, x] = \underset{O(1)}{MIN}\left[\overbrace{OPT[k-1, x], OPT[k, x-d_k]+1}^{O(1)}\right]$$

$$OPT[1, x] = x$$
$$OPT[k, 0] = 0$$

Step 3: DIY

Step 4: Runtime $O(n \cdot m)$

is it polynomial? NO

∴ Actual Runtime: $O(n \cdot 2^m)$

P.T.O

Longest Common Subsequence

a b c d e
b d a e

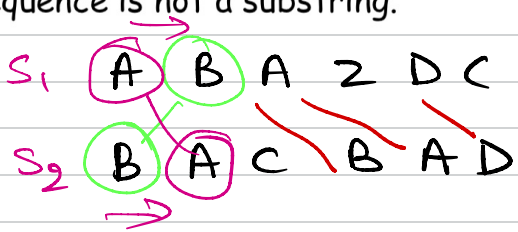
We are given string S_1 of length n, and string S_2 of length m.

Our goal is to produce their longest common subsequence.

A subsequence is a subset of elements in the sequence taken in order (with strictly increasing indexes.) Or you may think as removing some characters from one string to get another.

Note, a subsequence is not a substring.

Intuition:



① prefix $S_1[0, 1, \ldots i] \rightarrow$ A, AB, ABA, ... (array)

② suffix $S_1[i, \ldots, n] \rightarrow$ C, DC, ZDC, ... (array ∵ one side is fixed)

③ $S_1[i, \ldots, j]$ table (∵ there are 2 choices)

p :(- 0

Subproblems

Let $LCS[i,j]$ be the max length of the LCS $s_1[0..i]$ and $s_2[0...j]$

We have made use of the first choice from the previous PAGE.

Choices

① $s_1[i] = s_2[j]$

$$LCS[i,j] = 1 + LCS[i-1, j-1]$$

② $s_1[i] \neq s_2[j]$

$$LCS[i,j] = MAX\begin{bmatrix} LCS[i-1,j] \\ LCS[i,j-1] \end{bmatrix}$$

$$LCS[i,j] = \begin{cases} 1 + LCS[i-1, j-1], & \text{if } s_1[i] = s_1[j] \\ MAX\begin{bmatrix} LCS[i-1,j] \\ LCS[i,j-1] \end{bmatrix} \end{cases}$$

Base cases:

Empty string

$$LCS[0,j] = LCS[i,0] = 0$$

Runtime : $O(n \cdot m)$

Is it polynomial? YESS!!

Pseudo-code

```
int LCS(char[] S1, int n, char[] S2, int m)
{
int table[n+1, m+1];
table[0…n, 0] = table[0, 0…m] = 0; //init
for(i = 1; i <= n; i++)
  for(j = 1; j <= m; j++)
      if (S1[i] == S2[j])  table[i, j] = 1 + table[i-1, j-1]
      else
      table[i, j] = max(table[i, j-1], table[i-1, j]);

return table[n, m];
}
```

Discussion Problem 2

A subsequence is palindromic if it reads the same left and right. Devise a DP algorithm that takes a string and returns the length of the longest palindromic subsequence (not necessarily contiguous) .

For example, the string

QRAECCETCAURP

has several palindromic subsequences, RACECAR is one of them.

Given a string of size n

① How many substrings? $O(n^2)$

② How many subsequences $O(2^n)$

Step 1:

Let OPT[i, j] be the longest palindrome

$S_0, S_1, \ldots S_i, \ldots S_j \ldots S_n$

Step 2:

Case 1: $S[i] = S[j]$

$$OPT[i, j] = OPT[i+1, j-1] + 2$$

Case 2: $S[i] \neq S[j]$

$$OPT[i, j] = MAX[OPT[i+1, j], OPT[i, j-1]]$$

Base cases

$OPT[i, i] = 1$

$OPT[i, j] = 2$, if $S[i] = S[j]$ & $j = i+1$

↳ example:

"AA"

i=0, j=1

i=1, j=0

P.T.O

X

Base cases

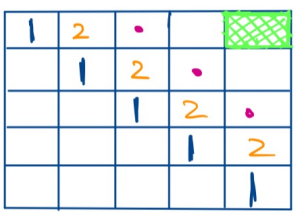
$OPT[i, i] = 1$

$OPT[i, j] = 2$, if $s[i] = s[j]$ and $j = i+1$

$\Rightarrow$ example
"AA"
$i=0, j=1$
$i=1, j=0$

Runtime : $O(n^2)$

Is it polynomial ? YES



Can we use LCS to solve this problem ?

Ans: Yes we can, the input will be as follows:

LCS[s_1, reverse(s_1)]

Static Optimal Binary Search Tree

↳ cannot change the given probability

Build a binary search tree which gives a minimum search cost, assuming we know the frequencies p_i with which data k_i is accessed. The tree cannot be modified after it has been constructed.

Want to build a binary search tree with minimum expected search cost:

$$\text{Expected Cost} = \sum_{i=1}^{n} p_i \, \text{depth}(k_i)$$

depth (root) = 1 ⇒ one assumption

Example

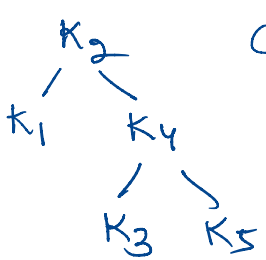
Consider 5 items

$$k_1 < k_2 < k_3 < k_4 < k_5$$

and their search probabilities

$$p_1 = 0.25,\ p_2 = 0.2,\ p_3 = 0.05,\ p_4 = 0.2,\ p_5 = 0.3.$$

Inorder of $k_1, k_2 \ldots k_5$



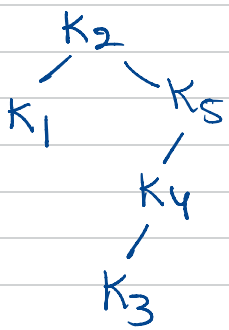
Cost = 2 × 0.25 +
1 × 0.2 +
3 × 0.05 +
2 × 0.2 +
3 × 0.3 = 2.15

Since our tree structure is not unique, let us consider another probability.

Another possibility

$k_1 < k_2 < k_3 < k_4 < k_5$

$p_1 = 0.25, p_2 = 0.2, p_3 = 0.05, p_4 = 0.2, p_5 = 0.3$



Cost = 2×0.25
1×0.2
2×0.3
3×0.2
4×0.05 = 2.1

Optimal Substructure

Since my main structure is a TREE let us take our substructure to be a SUBTREE.

Let OPT[i,j] be the minimum cost of a tree made of $K_i, K_{i+1}, \ldots, K_{j-1}, K_j$.

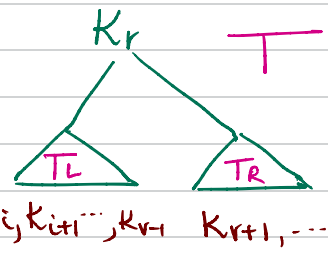
$$K_i \leq K_{i+1} \leq \ldots \leq K_j$$

NOTE : BST cannot have duplicates

Subproblem is part of our i/p & our i/p is a tree so our subproblem will be a subtree

What is the root of OPT[i,j] subtree?

$K_i \le K_r \le K_j$



r is the root

$$cost_T = OPT[i,j] = \underbrace{1 \times P_r}_{\text{root}} + \sum_{s=i}^{r-1} P_s \times depth_T(K_s) + \sum_{s=r+1}^{j} P_s \times depth_T(K_s)$$

$$depth_T(K_s) = depth_{T_L}(K_s) + 1$$

$$= depth_{T_R}(K_s) + 1$$

$$cost_T = P_r + \sum P_s (1 + depth_{T_L}(K_s)) + \sum P_s (1 + depth_{T_R}(K_s))$$

$$= P_i + P_{i+1} + \cdots + P_{r-1} + P_r + P_{r+1} + \cdots + P_j + \sum P_s \, depth_{T_L}(K_s) + \sum P_s \, depth_{T_R}(K_s)$$

$\sum P_s \, depth_{T_L}(K_s)$ → $OPT[i, r-1]$

$\sum P_s \, depth_{T_R}(K_s)$ → $OPT[r+1, j]$

P.T.O

Recurrence Relation

$$OPT[i,j] = P_i + \cdots + P_j + \min_{i \le r \le j} [OPT[i,r-1] + OPT[r+1,j]]$$

BASECASES :

$OPT[i,i] = P_i$

$OPT[i,i-1] = 0$

OPTIMAL SOLUTION :

OPT [top right corner]

Filling up the table

array p = [p_1, p_2, …, p_n]
set OPT[i, i-1] = 0, for $1 \le i \le n$
set OPT[i, i] = p_i, for $1 \le i \le n$

```
for(k = 1; k < n; k++)
  for(i = 1; i <= n-k, i++)
    j = i + k;
   OPT[i, j] =p_i+...+p_j+ min_r(OPT[i,r-1]+OPT[r+1,j]);
                            (i ≤ r ≤ j)
return OPT[1,n];
```

O(n) O(1) O(1)

Runtime Complexity-? $\Rightarrow O(n^3)$

Example

n = 5

(prob,value)=(0.1,5), (0.3,6), (0.9,4), (0.3,3), (0.1,8)

p_1 p_2 p_3 p_4 p_5

		0	1	2	3	4	5
(0.1, 5)	1	0	0.1				
(0.3, 6)	2		0	0.3			
(0.9, 4)	3			0	0.9		
(0.3, 3)	4				0	0.3	
(0.1, 8)	5					0	0.1

if r=1 ; OPT[1,0] OPT[2,2]

if r=2 , : OPT[1,1] OPT[3,2]

OPT[1,0] = 0

OPT[2,2] = 0.3 +

0.3

Excercise : OPT[1,2]

$$OPT[1,2] = 0.1 + 0.3 + \min_{1 \le r \le 2}[OPT[1,r-1] + OPT[r+1,2]]$$

$$= 0.1 + 0.3 + 0.1 = 0.5$$

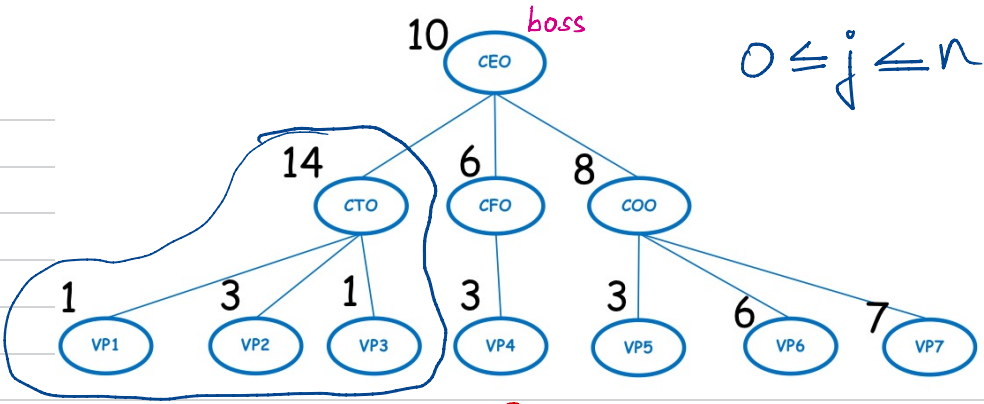
because 0.1 < 0.3 from the 2 sums we have computed.

OPT[1,1] = 0.1

OPT[3,2] = 0

Discussion Problem 1

Suppose you are organizing a company party. The corporation has a tree-like ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee j has a value v_j (a positive integer), representing how enjoyable their presence would be at the party. Our goal is to determine which employees to invite, subject to these constraints, to maximize the total value of invitees.



$0 \leq j \leq n$

Shall we invite the boss?

① No, $OPT = 36 = 14 + 6 + (3+6+7)$

② Yes, $OPT = 34 = 10 + 1 + 3 + 1 + 3 + 3 + 6 + 7$

Let $OPT[r]$ be the maximum fun value of a subtree rooted at r.

Choices:

① r is selected : $OPT[r] = V_r + \sum_g OPT[g]$

② r is not selected : $OPT[r] = \sum_c OPT[c]$

g = grandchildren , c = children

Reccurance Relation

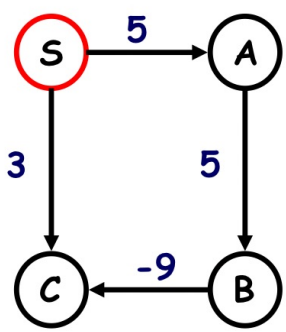
$$OPT[r] = MAX\left[V_r + \sum g \; , \sum c\right]$$

Base cases:

$OPT[NULL] = 0$

Runtime: $O(n)$ ⇒ This is so because when we start from the bottom of the tree we traverse upwards, we compute and visit each node only once.

The Shortest Path Problem



Dijkstra's greedy algorithm does not work on graphs with negative weights.

How can we use Dynamic Programming to find the shortest path? We need to somehow defined ordered subproblems, otherwise we may get an exponential runtime.

In Dijkstra's we consider a node only once and this cannot be followed if the graph consists a negative edge. In this case we will have to consider a node more than once (maximum V-1 times) to compute the shortest path. Hence we make use of DP.

Intuition

Consider the path (with k edges)

$$v = w_0, w_1, \ldots, w_{k-1}, w_k = u.$$

To have an optimal substructure the following path (k-1 edges)

$$v = w_0, w_1, \ldots, w_{k-1}$$

must be a shortest path to w_{k-1}.

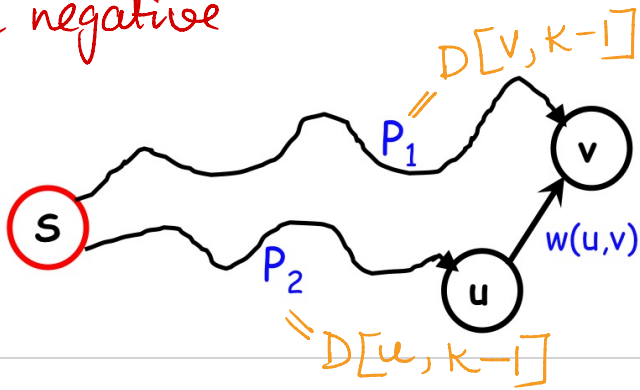
Thus, we will be counting *the number of edges* in the shortest path. This is how we order subproblems.

D[v, k] denotes the length of the shortest path from s to v that uses at most k edges.

Here our path is represented by the number of edges.

The Bellman-Ford Algorithm

w could be negative



$$D[v,K] = \begin{cases} P_1 \\ P_2 + w(u,v) \end{cases}$$

$D[v,K]$ denotes the shortest path from s to v using at most K edges.

Case 1 : Path uses at most K-1 edges.

Do not use adjacent vertex

$D[v,K] = D[v,K-1]$

$\Rightarrow$ Don't update the table

Case 2 : Otherwise, use the adjacent vertex

Let u be an adjacent vertex.

$$D[v,k] = \min_u \left[D[u,K-1] + w(u,v) \right]$$

$1 \leq K \leq V-1$

Recurrance Equation:

$$D[v, k] = \underset{O(1)}{MIN}\left[D[v, k-1], \underset{O(V)}{\min_{u}}\left(D[u, k-1] + \underset{O(1)}{w(u, v)}\right)\right]$$

(the $D[v, k-1]$ term: $O(1)$; the $D[u, k-1]$ term: $O(1)$)

Base Cases

$$D[v, 0] = \infty$$

$$D[s, k] = 0$$

Implementation

D[v,k] denotes the length of the shortest path from s to v that uses at most k edges.

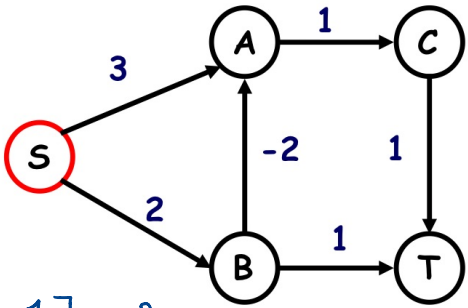
```
D[v,0] = INFINITY; v != s
D[s,k] = 0; for all k

V  for k=1 to V-1:                    V
V    for each v in V:
E      for each edge (u,v) in E:      E
         D[v, k] = min(D[v,k-1] , w(u,v) + D [u,k-1])
```

Since the total number of child edges cannot possibly exceed E, we take our worst case to be E.

Runtime - ? $O(V \cdot E)$

Example



$K = 1$: $D[A,1] = 3$
$D[B,1] = 2$

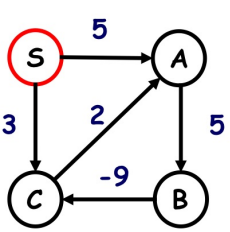
$K = 2$: $D[A,2] = \min(3, 2-2) = 0$
$D[C,2] = 4$
$D[T,2] = 3$

$K = 3$: $D[C,3] = \min(4, 2-2+1) = 1$
$D[T,3] = \min(3, 3+1+1) = 3$

$K = 4$: $D[T,4] = \min(3, 2-2+1+1) = 2$

How would you apply the Bellman-Ford algorithm to find out if a graph has a negative cycle?

If there is a positive loop then our sol^n will still be optimal as we are considering only the minimum.

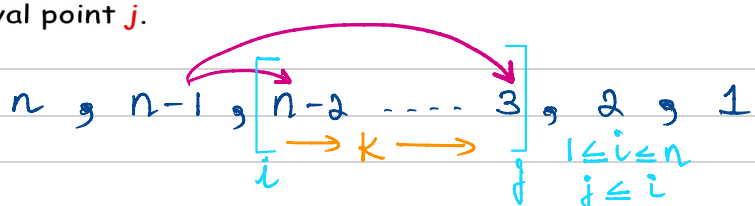


C-A-B-C
2+5+(-9) = -2

→ We add an extra loop to our traversal
→ After the last loop if the table entry changes, there is a negative cycle
→ return no solution exists.

Discussion Problem 2

There are n trading posts along a river numbered n, $n-1$..., 1. At any of the posts you can rent a canoe to be returned at any other post downstream. (It is impossible to paddle against the river). For each possible departure point i and each possible arrival point $j < i$, the cost of a rental is $C[i, j]$. However, it can happen that the cost of renting from i to j is higher than the total costs of a series of shorter rentals. In this case you can return the first canoe at some post k between i and j and continue your journey in a second (and, maybe, third, fourth . . .) canoe. There is no extra charge for changing canoes in this way. Give a dynamic programming algorithm to determine the minimum cost of a trip by canoe from each possible departure point i to each possible arrival point j.



Let $OPT[i, j]$ be the minimum rental cost going from i to j.

$$OPT[i, j] = \min_{j \le k \le i} \left[C[i, k] + OPT[k, j] \right]$$

$O(n)$

$OPT[i,j] \Rightarrow n \times n = O(n^2)$

$$OPT[i, i] = 0$$

Runtime : $O(n^2 \times n) = O(n^3)$

P.T.O

Variant B

$i = n$, $j = 1$

Use an 1 D array

Let OPT[i] be the minimum rental cost from i to 1.

$$OPT[i] = \min_{1 \le K \le i} [C[i,K] + OPT[K]]$$

$OPT[1] = 0$ ⇒ Basecase

Runtime = $O(n \times n) = O(n^2)$

CHAIN MATRIX MULTIPLICATION — Abdul Bari

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$2 \times 3 \qquad 3 \times 2$

must definitely satisfy to do CMM

Dimension = 2×2

Total no of multiplications = $2 \times 3 \times 2 = 12$

Example : (With example formula)

Let solution = C[1,3]

A_1 A_2 A_3

2×3 3×4 4×2

d_0 d_1 d_1 d_2 d_2 d_3

$(A_1 \times A_2) \times A_3$

2 3 3 4 4 2

C[1,2] = 24 0 C[3,3]

$2 \times 3 \times 4 = 24$

$\Rightarrow 2 \times 4 \quad 4 \times 2$

$2 \times 4 \times 2 = 16$

d_0 d_2 d_3

$\therefore$ Total = 24 + 0 + 16 = 40

$C[i,k] + C[k+1,j] + d_{i-1} \times d_k \times d_j$

$A_1 \times (A_2 \times A_3)$

2 3 3 4 4 2

C[1,1] 0 C[2,3] = 24

$3 \times 4 \times 2 = 24$

$\Rightarrow 2 \times 3 \quad 3 \times 2$

$2 \times 3 \times 2 = 12$

d_0 d_1 d_3

$\therefore$ Total = 0 + 24 + 12 = 36

CHEAPER

$C[i,k] + C[k+1,j] + d_0 \times d_k \times d_j$

Observe the reduced equation carefully

FORMULA: $C[i,j] = \min_{i \le k < j} \{ C[i,k] + C[k+1,j] + d_{i-1} \times d_k + d_j \}$