# Dynamic Programming

Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again. The subproblems are optimized to optimize the overall solution is known as optimal substructure property. The main use of dynamic programming is to solve optimization problems. Here, optimization problems mean that when we are trying to find out the minimum or the maximum solution of a problem. The dynamic programming guarantees to find the optimal solution of a problem if the solution exists.

The definition of dynamic programming says that it is a technique for solving a complex problem by first breaking into a collection of simpler subproblems, solving each subproblem just once, and then storing their solutions to avoid repetitive computations.

**Let's understand this approach through an example.**

**0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ,…**

The numbers in the above series are not randomly calculated. Mathematically, we could write each of the terms using the below formula:

**F(n) = F(n-1) + F(n-2),**

With the base values F(0) = 0, and F(1) = 1. To calculate the other numbers, we follow the above relationship. For example, F(2) is the sum **f(0)** and **f(1),** which is equal to 1.

### How does the dynamic programming approach work?

The following are the steps that the dynamic programming follows:

* It breaks down the complex problem into simpler subproblems.
* It finds the optimal solution to these sub-problems.
* It stores the results of subproblems (memoization). The process of storing the results of subproblems is known as memorization.
* It reuses them so that same sub-problem is calculated more than once.
* Finally, calculate the result of the complex problem.

The above five steps are the basic steps for dynamic programming. The dynamic programming is applicable that are having properties such as:

Those problems that are having overlapping subproblems and optimal substructures. Here, optimal substructure means that the solution of optimization problems can be obtained by simply combining the optimal solution of all the subproblems.

In the case of dynamic programming, the space complexity would be increased as we are storing the intermediate results, but the time complexity would be decreased.

## Approaches of dynamic programming

There are two approaches to dynamic programming:

* Top-down approach
* Bottom-up approach

### Top-down approach

The top-down approach follows the memorization technique, while bottom-up approach follows the tabulation method. Here memorization is equal to the sum of recursion and caching. Recursion means calling the function itself, while caching means storing the intermediate results.

**Advantages**

* It is very easy to understand and implement.
* It solves the subproblems only when it is required.
* It is easy to debug.

**Disadvantages**

It uses the recursion technique that occupies more memory in the call stack. Sometimes when the recursion is too deep, the stack overflow condition will occur.

It occupies more memory that degrades the overall performance.

### Bottom-Up approach

The bottom-up approach is also one of the techniques which can be used to implement the dynamic programming. It uses the tabulation technique to implement the dynamic programming approach. It solves the same kind of problems but it removes the recursion. If we remove the recursion, there is no stack overflow issue and no overhead of the recursive functions. In this tabulation technique, we solve the problems and store the results in a matrix.

There are two ways of applying dynamic programming:

* **Top-Down**
* **Bottom-Up**

The bottom-up is the approach used to avoid the recursion, thus saving the memory space. The bottom-up is an algorithm that starts from the beginning, whereas the recursive algorithm starts from the end and works backward. In the bottom-up approach, we start from the base case to find the answer for the end. As we know, the base cases in the Fibonacci series are 0 and 1. Since the bottom approach starts from the base cases, so we will start from 0 and 1.

**Key points**

* We solve all the smaller sub-problems that will be needed to solve the larger sub-problems then move to the larger problems using smaller sub-problems.
* We use for loop to iterate over the sub-problems.
* The bottom-up approach is also known as the tabulation or table filling method.

Divide and Conquer Method vs Dynamic Programming

|  |  |
| --- | --- |
| **Divide and Conquer Method** | **Dynamic Programming** |
| **1.** It deals (involves) three steps at each level of recursion: **Divide** the problem into a number of subproblems. **Conquer** the subproblems by solving them recursively. **Combine** the solution to the subproblems into the solution for original subproblems. | **1.**It involves the sequence of four steps:   * Characterize the structure of optimal solutions. * Recursively defines the values of optimal solutions. * Compute the value of optimal solutions in a Bottom-up minimum. * Construct an Optimal Solution from computed information. |
| **2.** It is Recursive. | **2.** It is non Recursive. |
| **3.** It does more work on subproblems and hence has more time consumption. | **3.** It solves subproblems only once and then stores in the table. |
| **4.** It is a top-down approach. | **4.** It is a Bottom-up approach. |
| **5.** In this subproblems are independent of each other. | **5.** In this subproblems are interdependent. |
| **6. For example:** Merge Sort & Binary Search etc. | **6. For example:** Matrix Multiplication. |

# Matrix Chain Multiplication

It is a Method under Dynamic Programming in which previous output is taken as input for next.

Here, Chain means one matrix's column is equal to the second matrix's row [always].

In general:

If A = ⌊aij⌋ is a p x q matrix

   B = ⌊bij⌋ is a q x r matrix

   C = ⌊cij⌋ is a p x r matrix

Then

Matrix Chain Multiplication

Given following matrices {A1,A2,A3,...An} and we have to perform the matrix multiplication, which can be accomplished by a series of matrix multiplications

A1 xA2 x,A3 x.....x An

Matrix Multiplication operation is **associative** in nature rather commutative. By this, we mean that we have to follow the above matrix order for multiplication but we are free to **parenthesize** the above multiplication depending upon our need.

In general, for 1≤ i≤ p and 1≤ j ≤ r

Matrix Chain Multiplication

It can be observed that the total entries in matrix 'C' is 'pr' as the matrix is of dimension p x r Also each entry takes O (q) times to compute, thus the total time to compute all possible entries for the matrix 'C' which is a multiplication of 'A' and 'B' is proportional to the product of the dimension p q r.

It is also noticed that we can save the number of operations by reordering the parenthesis.

**Example1:** Let us have 3 matrices, A1,A2,A3 of order (10 x 100), (100 x 5) and (5 x 50) respectively.

Three Matrices can be multiplied in two ways:

1. **A1,(A2,A3):** First multiplying(A2 and A3) then multiplying and resultant withA1.
2. **(A1,A2),A3:** First multiplying(A1 and A2) then multiplying and resultant withA3.

No of Scalar multiplication in Case 1 will be:

1. (100 x 5 x 50) + (10 x 100 x 50) = 25000 + 50000 = 75000

No of Scalar multiplication in Case 2 will be:

1. (100 x 10 x 5) + (10 x 5 x 50) = 5000 + 2500 = 7500

To find the best possible way to calculate the product, we could simply parenthesis the expression in every possible fashion and count each time how many scalar multiplication are required.

Matrix Chain Multiplication Problem can be stated as "find the optimal parenthesization of a chain of matrices to be multiplied such that the number of scalar multiplication is minimized".

**Number of ways for parenthesizing the matrices:**

There are very large numbers of ways of parenthesizing these matrices. If there are n items, there are (n-1) ways in which the outer most pair of parenthesis can place.

(A1) (A2,A3,A4,................An)

Or (A1,A2) (A3,A4 .................An)

Or (A1,A2,A3) (A4 ...............An)

........................

Or(A1,A2,A3.............An-1) (An)

It can be observed that after splitting the kth matrices, we are left with two parenthesized sequence of matrices: one consist 'k' matrices and another consist 'n-k' matrices.

Now there are 'L' ways of parenthesizing the left sublist and 'R' ways of parenthesizing the right sublist then the Total will be L.R:

DAA Matrix Chain Multiplication

Also p (n) = c (n-1) where c (n) is the nth **Catalon number**

c (n) = DAA Matrix Chain Multiplication

On applying Stirling's formula we have

c (n) = Ω DAA Matrix Chain Multiplication

Which shows that 4n grows faster, as it is an exponential function, then n1.5.

## Development of Dynamic Programming Algorithm

1. Characterize the structure of an optimal solution.
2. Define the value of an optimal solution recursively.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct the optimal solution from the computed information.

## Dynamic Programming Approach

Let Ai,j be the result of multiplying matrices i through j. It can be seen that the dimension of Ai,j is pi-1 x pj matrix.

Dynamic Programming solution involves breaking up the problems into subproblems whose solution can be combined to solve the global problem.

At the greatest level of parenthesization, we multiply two matrices

A1.....n=A1....k x Ak+1....n)

Thus we are left with two questions:

* How to split the sequence of matrices?
* How to parenthesize the subsequence A1.....k andAk+1......n?

One possible answer to the first question for finding the best value of 'k' is to check all possible choices of 'k' and consider the best among them. But that it can be observed that checking all possibilities will lead to an exponential number of total possibilities. It can also be noticed that there exists only O (n2 ) different sequence of matrices, in this way do not reach the exponential growth.

**Step1: Structure of an optimal parenthesization:** Our first step in the dynamic paradigm is to find the optimal substructure and then use it to construct an optimal solution to the problem from an optimal solution to subproblems.

Let Ai....j where i≤ j denotes the matrix that results from evaluating the product

Ai Ai+1....Aj.

If i < j then any parenthesization of the product Ai Ai+1 ......Aj must split that the product between Ak and Ak+1 for some integer k in the range i ≤ k ≤ j. That is for some value of k, we first compute the matrices Ai.....k & Ak+1....j and then multiply them together to produce the final product Ai....j. The cost of computing Ai....k plus the cost of computing Ak+1....j plus the cost of multiplying them together is the cost of parenthesization.

**Step 2: A Recursive Solution:** Let m [i, j] be the minimum number of scalar multiplication needed to compute the matrixAi....j.

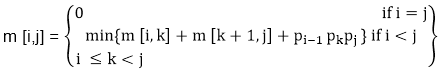
If i=j the chain consist of just one matrix Ai....i=Ai so no scalar multiplication are necessary to compute the product. Thus m [i, j] = 0 for i= 1, 2, 3....n.

If i<j we assume that to optimally parenthesize the product we split it between Ak and Ak+1 where i≤ k ≤j. Then m [i,j] equals the minimum cost for computing the subproducts Ai....k and Ak+1....j+ cost of multiplying them together. We know Ai has dimension pi-1 x pi, so computing the product Ai....k and Ak+1....jtakes pi-1 pk pj scalar multiplication, we obtain

m [i,j] = m [i, k] + m [k + 1, j] + pi-1 pk pj

There are only (j-1) possible values for 'k' namely k = i, i+1.....j-1. Since the optimal parenthesization must use one of these values for 'k' we need only check them all to find the best.

So the minimum cost of parenthesizing the product Ai Ai+1......Aj becomes

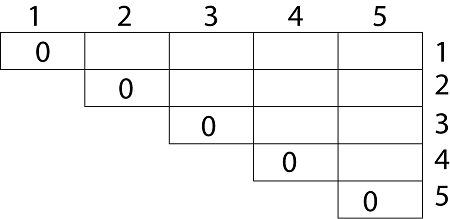


To construct an optimal solution, let us define s [i,j] to be the value of 'k' at which we can split the product Ai Ai+1 .....Aj To obtain an optimal parenthesization i.e. s [i, j] = k such that

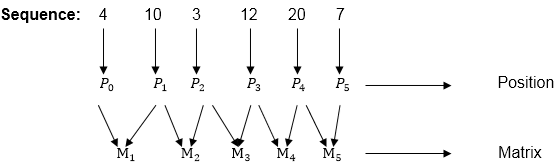
m [i,j] = m [i, k] + m [k + 1, j] + pi-1 pk pj

# Example of Matrix Chain Multiplication

**Example:** We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.



Let us proceed with working away from the diagonal. We compute the optimal solution for the product of 2 matrices.



Here P0 to P5 are Position and M1 to M5 are matrix of size (pi to pi-1)

On the basis of sequence, we make a formula

Example of Matrix Chain Multiplication

In Dynamic Programming, initialization of every method done by '0'.So we initialize it by '0'.It will sort out diagonally.

We have to sort out all the combination but the minimum output combination is taken into consideration.

**Calculation of Product of 2 matrices:**

1. m (1,2) = m1 x m2

= 4 x 10 x 10 x 3

= 4 x 10 x 3 = 120

2. m (2, 3) = m2 x m3

= 10 x 3 x 3 x 12

= 10 x 3 x 12 = 360

3. m (3, 4) = m3 x m4

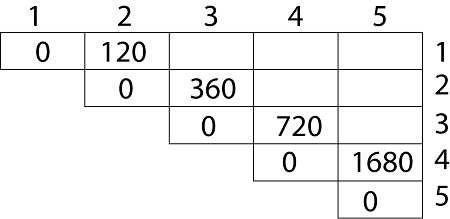
= 3 x 12 x 12 x 20

= 3 x 12 x 20 = 720

4. m (4,5) = m4 x m5

= 12 x 20 x 20 x 7

= 12 x 20 x 7 = 1680



* We initialize the diagonal element with equal i,j value with '0'.
* After that second diagonal is sorted out and we get all the values corresponded to it

Now the third diagonal will be solved out in the same way.

**Now product of 3 matrices:**

M [1, 3] = M1 M2 M3

1. There are two cases by which we can solve this multiplication: ( M1 x M2) + M3, M1+ (M2x M3)
2. After solving both cases we choose the case in which minimum output is there.

Example of Matrix Chain Multiplication

**M [1, 3] =264**

As Comparing both output **264** is minimum in both cases so we insert **264** in table and ( M1 x M2) + M3 this combination is chosen for the output making.

M [2, 4] = M2 M3 M4

1. There are two cases by which we can solve this multiplication: (M2x M3)+M4, M2+(M3 x M4)
2. After solving both cases we choose the case in which minimum output is there.

DAA Example of Matrix Chain Multiplication

**M [2, 4] = 1320**

As Comparing both output **1320** is minimum in both cases so we insert **1320** in table and M2+(M3 x M4) this combination is chosen for the output making.

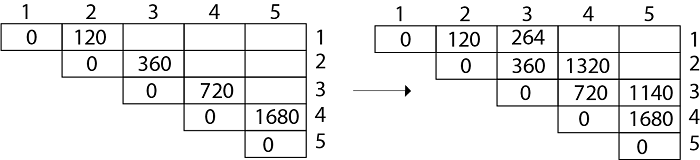
M [3, 5] = M3 M4 M5

1. There are two cases by which we can solve this multiplication: ( M3 x M4) + M5, M3+ ( M4xM5)
2. After solving both cases we choose the case in which minimum output is there.

Example of Matrix Chain Multiplication

M [3, 5] = 1140

As Comparing both output **1140** is minimum in both cases so we insert **1140** in table and ( M3 x M4) + M5this combination is chosen for the output making.



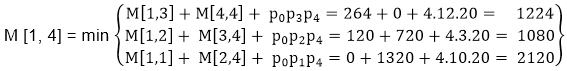
Now Product of 4 matrices:

M [1, 4] = M1 M2 M3 M4

There are three cases by which we can solve this multiplication:

1. ( M1 x M2 x M3) M4
2. M1 x(M2 x M3 x M4)
3. (M1 xM2) x ( M3 x M4)

After solving these cases we choose the case in which minimum output is there



**M [1, 4] =1080**

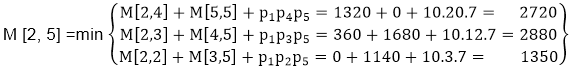
As comparing the output of different cases then '**1080**' is minimum output, so we insert 1080 in the table and (M1 xM2) x (M3 x M4) combination is taken out in output making,

M [2, 5] = M2 M3 M4 M5

There are three cases by which we can solve this multiplication:

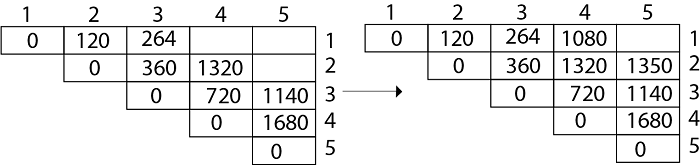
1. (M2 x M3 x M4)x M5
2. M2 x( M3 x M4 x M5)
3. (M2 x M3)x ( M4 x M5)

After solving these cases we choose the case in which minimum output is there



M [2, 5] = 1350

As comparing the output of different cases then '**1350**' is minimum output, so we insert 1350 in the table and M2 x( M3 x M4 xM5)combination is taken out in output making.



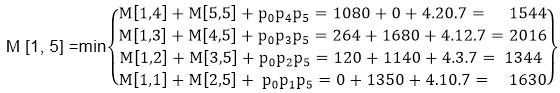
**Now Product of 5 matrices:**

M [1, 5] = M1 M2 M3 M4 M5

There are five cases by which we can solve this multiplication:

1. (M1 x M2 xM3 x M4 )x M5
2. M1 x( M2 xM3 x M4 xM5)
3. (M1 x M2 xM3)x M4 xM5
4. M1 x M2x(M3 x M4 xM5)

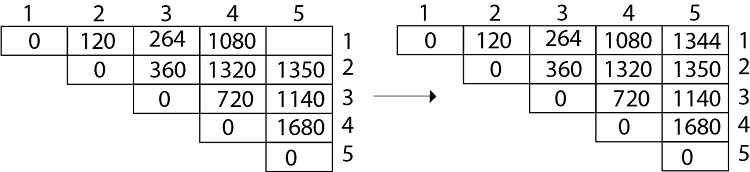
After solving these cases we choose the case in which minimum output is there



M [1, 5] = 1344

As comparing the output of different cases then '**1344**' is minimum output, so we insert 1344 in the table and M1 x M2 x(M3 x M4 x M5)combination is taken out in output making.

**Final Output is:**



**Step 3: Computing Optimal Costs:** let us assume that matrix Ai has dimension pi-1x pi for i=1, 2, 3....n. The input is a sequence (p0,p1,......pn) where length [p] = n+1. The procedure uses an auxiliary table m [1....n, 1.....n] for storing m [i, j] costs an auxiliary table s [1.....n, 1.....n] that record which index of k achieved the optimal costs in computing m [i, j].

The algorithm first computes m [i, j] ← 0 for i=1, 2, 3.....n, the minimum costs for the chain of length 1.

Algorithm of Matrix Chain Multiplication

**MATRIX-CHAIN-ORDER (p)**

1. n length[p]-1

2. for i ← 1 to n

3. do m [i, i] ← 0

4. for l ← 2 to n // l is the chain length

5. do for i ← 1 to n-l + 1

6. do j ← i+ l -1

7. m[i,j] ← ∞

8. for k ← i to j-1

9. do q ← m [i, k] + m [k + 1, j] + pi-1 pk pj

10. If q < m [i,j]

11. then m [i,j] ← q

12. s [i,j] ← k

13. return m and s.

We will use table s to construct an optimal solution.

**Step 1: Constructing an Optimal Solution:**

**PRINT-OPTIMAL-PARENS (s, i, j)**

1. if i=j

2. then print "A"

3. else print "("

4. PRINT-OPTIMAL-PARENS (s, i, s [i, j])

5. PRINT-OPTIMAL-PARENS (s, s [i, j] + 1, j)

6. print ")"

**Analysis:** There are three nested loops. Each loop executes a maximum n times.

1. l, length, O (n) iterations.
2. i, start, O (n) iterations.
3. k, split point, O (n) iterations

Body of loop constant complexity

**Total Complexity is: O (n3)**

## Algorithm with Explained Example

**Question: P [7, 1, 5, 4, 2}**

**Solution:** Here, P is the array of a dimension of matrices.

So here we will have 4 matrices:

A17x1 A21x5 A35x4 A44x2

i.e.

First Matrix A1 have dimension 7 x 1

Second Matrix A2 have dimension 1 x 5

Third Matrix A3 have dimension 5 x 4

Fourth Matrix A4 have dimension 4 x 2

Let say,

From P = {7, 1, 5, 4, 2} - (Given)

And P is the Position

p0 = 7, p1 =1, p2 = 5, p3 = 4, p4=2.

Length of array P = number of elements in P

∴length (p)= 5

From step 3

Follow the steps in Algorithm in Sequence

According to Step 1 of Algorithm Matrix-Chain-Order

**Step 1:**

n ← length [p]-1

Where n is the total number of elements

And length [p] = 5

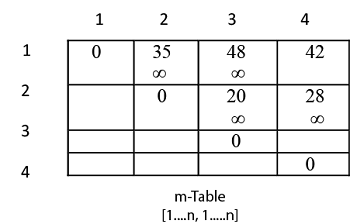
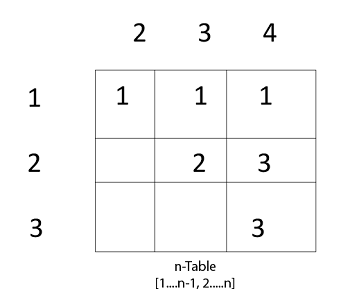
∴ n = 5 - 1 = 4

**n = 4**

Now we construct two tables m and s.

Table m has dimension [1.....n, 1.......n]

Table s has dimension [1.....n-1, 2.......n]

Now, according to step 2 of Algorithm

1. **for** i ← 1 to n
2. **this** means: **for** i ← 1 to 4 (because n =4)
3. **for**  i=1
4. m [i, i]=0
5. m [1, 1]=0
6. Similarly **for** i = 2, 3, 4
7. m [2, 2] = m [3,3] = m [4,4] = 0
8. i.e. fill all the diagonal entries "0" in the table m
9. Now,
10. l ← 2 to n
11. l ← 2 to 4    (because n =4 )

**Case 1:**

1. When l - 2

for (i ← 1 to n - l + 1)

i ← 1 to 4 - 2 + 1

i ← 1 to 3

**When i = 1**

do j ← i + l - 1

j ← 1 + 2 - 1

j ← 2

**i.e. j = 2**

Now, m [i, j] ← ∞

i.e. m [1,2] ← ∞

Put ∞ in m [1, 2] table

for k ← i to j-1

k ← 1 to 2 - 1

k ← 1 to 1

**k = 1**

**Now q ← m [i, k] + m [k + 1, j] + pi-1 pk pj**

for l = 2

i = 1

j =2

k = 1

q ← m [1,1] + m [2,2] + p0x p1x p2

and m [1,1] = 0

for i ← 1 to 4

∴ q ← 0 + 0 + 7 x 1 x 5

**q ← 35**

We have m [i, j] = m [1, 2] = ∞

Comparing q with m [1, 2]

q < m [i, j]

i.e. **35 < m [1, 2]**

35 < ∞

True

then, **m [1, 2 ] ← 35 (∴ m [i,j] ← q)**

**s [1, 2] ← k**

and the value of **k = 1**

**s [1,2 ] ← 1**

Insert "1" at dimension s [1, 2] in table s. And 35 at m [1, 2]

2. l remains 2

L = 2

i ← 1 to n - l + 1

i ← 1 to 4 - 2 + 1

i ← 1 to 3

for i = 1 done before

Now value of i becomes 2

**i = 2**

j ← i + l - 1

j ← 2 + 2 - 1

j ← 3

j = 3

m [i , j] ← ∞

i.e. **m [2,3] ← ∞**

Initially insert ∞ at m [2, 3]

Now, for k ← i to j - 1

k ← 2 to 3 - 1

k ← 2 to 2

**i.e. k =2**

**Now, q ← m [i, k] + m [k + 1, j] + pi-1 pk pj**

For l =2

i = 2

j = 3

k = 2

q ← m [2, 2] + m [3, 3] + p1x p2 x p3

q ← 0 + 0 + 1 x 5 x 4

q ← 20

Compare q with m [i ,j ]

If q < m [i,j]

i.e. 20 < m [2, 3]

20 < ∞

True

Then m [i,j ] ← q

m [2, 3 ] ← 20

and s [2, 3] ← k

and k = 2

**s [2,3] ← 2**

3. Now i become 3

i = 3

l = 2

j ← i + l - 1

j ← 3 + 2 - 1

j ← 4

**j = 4**

Now, m [i, j ] ← ∞

m [3,4 ] ← ∞

Insert ∞ at m [3, 4]

for k ← i to j - 1

k ← 3 to 4 - 1

k ← 3 to 3

**i.e. k = 3**

**Now, q ← m [i, k] + m [k + 1, j] + pi-1 pk pj**

i = 3

l = 2

j = 4

k = 3

q ← m [3, 3] + m [4,4] + p2 x p3 x p4

q ← 0 + 0 + 5 x 2 x 4

**q 40**

Compare q with m [i, j]

If q < m [i, j]

40 < m [3, 4]

40 < ∞

True

Then, m [i,j] ← q

m [3,4] ← 40

and s [3,4] ← k

s [3,4] ← 3

**Case 2:** l becomes 3

L = 3

for i = 1 to n - l + 1

i = 1 to 4 - 3 + 1

i = 1 to 2

When i = 1

j ← i + l - 1

j ← 1 + 3 - 1

j ← 3

**j = 3**

Now, m [i,j] ← ∞

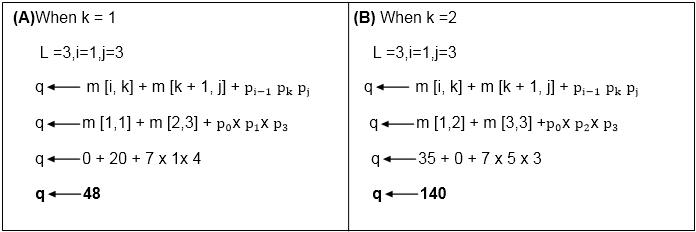
**m [1, 3] ← ∞**

for k ← i to j - 1

k ← 1 to 3 - 1

k ← 1 to 2

Now we compare the value for both k=1 and k = 2. The minimum of two will be placed in m [i,j] or s [i,j] respectively.



Now from above

Value of q become minimum for **k=1**

∴ m [i,j] ← q

**m [1,3] ← 48**

Also m [i,j] > q

**i.e. 48 < ∞**

∴ m [i , j] ← q

m [i, j] ← 48

and s [i,j] ← k

**i.e. m [1,3] ← 48**

**s [1, 3] ← 1**

**Now i become 2**

i = 2

l = 3

then j ← i + l -1

j ← 2 + 3 - 1

j ← 4

**j = 4**

so m [i,j] ← ∞

m [2,4] ← ∞

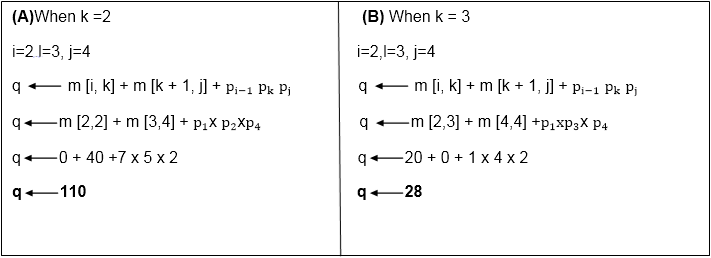
Insert initially ∞ at m [2, 4]

for k ← i to j-1

k ← 2 to 4 - 1

k ← 2 to 3

Here, also find the minimum value of m [i,j] for two values of k = 2 and k =3



1. But 28 <     ∞
2. So m [i,j] ← q
3. And q  ← 28
4. m [2, 4]  ← 28
5. and   s [2, 4]  ← 3
6. e. It means in s table at s [2,4] insert 3 and at m [2,4] insert 28.

**Case 3:** l becomes 4

L = 4

For i ← 1 to n-l + 1

i ← 1 to 4 - 4 + 1

i ← 1

**i = 1**

do j ← i + l - 1

j ← 1 + 4 - 1

j ← 4

**j = 4**

Now m [i,j] ← ∞

m [1,4] ← ∞

for k ← i to j -1

k ← 1 to 4 - 1

k ← 1 to 3

**When k = 1**

q ← m [i, k] + m [k + 1, j] + pi-1 pk pj

q ← m [1,1] + m [2,4] + p0xp4x p1

q ← 0 + 28 + 7 x 2 x 1

**q ← 42**

Compare q and m [i, j]

m [i,j] was ∞

i.e. m [1,4]

if q < m [1,4]

42< ∞

True

Then m [i,j] ← q

**m [1,4] ← 42**

**and s [1,4] 1 ? k =1**

**When k = 2**

L = 4, i=1, j = 4

q ← m [i, k] + m [k + 1, j] + pi-1 pk pj

q ← m [1, 2] + m [3,4] + p0 xp2 xp4

q ← 35 + 40 + 7 x 5 x 2

**q ← 145**

Compare q and m [i,j]

Now m [i, j]

i.e. m [1,4] contains 42.

So if q < m [1, 4]

But 145 less than or not equal to m [1, 4]

**So 145 less than or not equal to 42.**

So no change occurs.

When k = 3

l = 4

i = 1

j = 4

q ← m [i, k] + m [k + 1, j] + pi-1 pk pj

q ← m [1, 3] + m [4,4] + p0 xp3 x p4

q ← 48 + 0 + 7 x 4 x 2

q ← 114

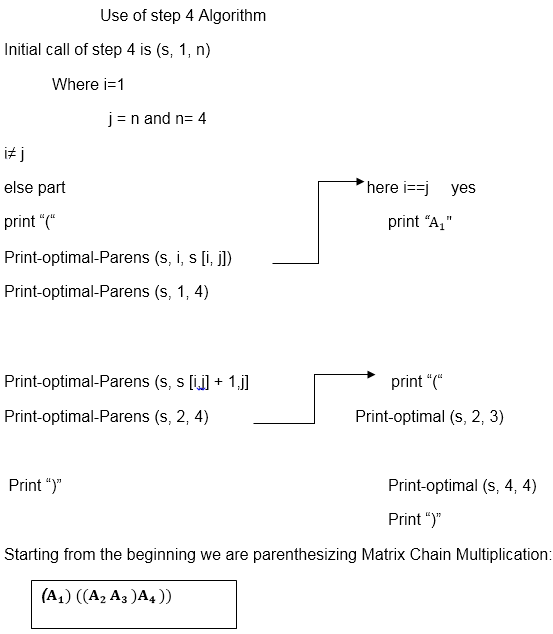
Again q less than or not equal to m [i, j]

i.e. 114 less than or not equal to m [1, 4]

**114 less than or not equal to 42**

So no change occurs. So the value of m [1, 4] remains 42. And value of s [1, 4] = 1

Now we will make use of only s table to get an optimal solution.



# Longest Common Sequence (LCS)

A subsequence of a given sequence is just the given sequence with some elements left out.

Given two sequences X and Y, we say that the sequence Z is a common sequence of X and Y if Z is a subsequence of both X and Y.

In the longest common subsequence problem, we are given two sequences X = (x1 x2....xm) and Y = (y1 y2 yn) and wish to find a maximum length common subsequence of X and Y. LCS Problem can be solved using dynamic programming.

## Characteristics of Longest Common Sequence

A brute-force approach we find all the subsequences of X and check each subsequence to see if it is also a subsequence of Y, this approach requires exponential time making it impractical for the long sequence.

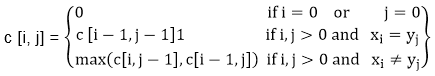
Given a sequence X = (x1 x2.....xm) we define the ith prefix of X for i=0, 1, and 2...m as Xi= (x1 x2.....xi). For example: if X = (A, B, C, B, C, A, B, C) then X4= (A, B, C, B)

**Optimal Substructure of an LCS:** Let X = (x1 x2....xm) and Y = (y1 y2.....) yn) be the sequences and let Z = (z1 z2......zk) be any LCS of X and Y.

* If xm = yn, then zk=x\_m=yn and Zk-1 is an LCS of Xm-1and Yn-1
* If xm ≠ yn, then zk≠ xm implies that Z is an LCS of Xm-1and Y.
* If xm ≠ yn, then zk≠yn implies that Z is an LCS of X and Yn-1

**Step 2: Recursive Solution:** LCS has overlapping subproblems property because to find LCS of X and Y, we may need to find the LCS of Xm-1 and Yn-1. If xm ≠ yn, then we must solve two subproblems finding an LCS of X and Yn-1.Whenever of these LCS's longer is an LCS of x and y. But each of these subproblems has the subproblems of finding the LCS of Xm-1 and Yn-1.

Let c [i,j] be the length of LCS of the sequence Xiand Yj.If either i=0 and j =0, one of the sequences has length 0, so the LCS has length 0. The optimal substructure of the LCS problem given the recurrence formula



**Step3: Computing the length of an LCS:** let two sequences X = (x1 x2.....xm) and Y = (y1 y2..... yn) as inputs. It stores the c [i,j] values in the table c [0......m,0..........n].Table b [1..........m, 1..........n] is maintained which help us to construct an optimal solution. c [m, n] contains the length of an LCS of X,Y.

# Algorithm of Longest Common Sequence

**LCS-LENGTH (X, Y)**

1. m ← length [X]

2. n ← length [Y]

3. for i ← 1 to m

4. do c [i,0] ← 0

5. for j ← 0 to m

6. do c [0,j] ← 0

7. for i ← 1 to m

8. do for j ← 1 to n

9. do if xi= yj

10. then c [i,j] ← c [i-1,j-1] + 1

11. b [i,j] ← "↖"

12. else if c[i-1,j] ≥ c[i,j-1]

13. then c [i,j] ← c [i-1,j]

14. b [i,j] ← "↑"

15. else c [i,j] ← c [i,j-1]

16. b [i,j] ← "← "

17. return c and b.

## Example of Longest Common Sequence

**Example:** Given two sequences X [1...m] and Y [1.....n]. Find the longest common subsequences to both.

Example of Longest Common Sequence

here X = (A,B,C,B,D,A,B) and Y = (B,D,C,A,B,A)

m = length [X] and n = length [Y]

m = 7 and n = 6

Here x1= x [1] = A y1= y [1] = B

x2= B y2= D

x3= C y3= C

x4= B y4= A

x5= D y5= B

x6= A y6= A

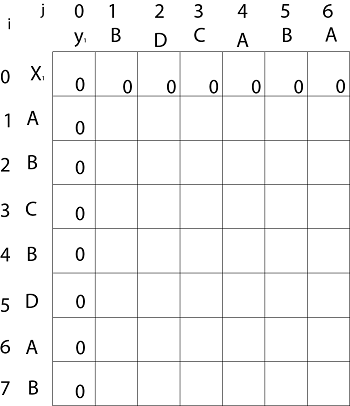
x7= B

Now fill the values of c [i, j] in m x n table

Initially, for i=1 to 7 c [i, 0] = 0

For j = 0 to 6 c [0, j] = 0

That is:



**Now for i=1 and j = 1**

x1 and y1 we get x1 ≠ y1 i.e. A ≠ B

And c [i-1,j] = c [0, 1] = 0

c [i, j-1] = c [1,0 ] = 0

That is, c [i-1,j]= c [i, j-1] so c [1, 1] = 0 and b [1, 1] = ' ↑ '

**Now for i=1 and j = 2**

x1 and y2 we get x1 ≠ y2 i.e. A ≠ D

c [i-1,j] = c [0, 2] = 0

c [i, j-1] = c [1,1 ] = 0

That is, c [i-1,j]= c [i, j-1] and c [1, 2] = 0 b [1, 2] = ' ↑ '

**Now for i=1 and j = 3**

x1 and y3 we get x1 ≠ y3 i.e. A ≠ C

c [i-1,j] = c [0, 3] = 0

c [i, j-1] = c [1,2 ] = 0

so c [1,3] = 0 b [1,3] = ' ↑ '

**Now for i=1 and j = 4**

x1 and y4 we get. x1=y4 i.e A = A

c [1,4] = c [1-1,4-1] + 1

= c [0, 3] + 1

= 0 + 1 = 1

c [1,4] = 1

b [1,4] = ' ↖ '

**Now for i=1 and j = 5**

x1 and y5 we get x1 ≠ y5

c [i-1,j] = c [0, 5] = 0

c [i, j-1] = c [1,4 ] = 1

Thus c [i, j-1] > c [i-1,j] i.e. c [1, 5] = c [i, j-1] = 1. So b [1, 5] = '←'

**Now for i=1 and j = 6**

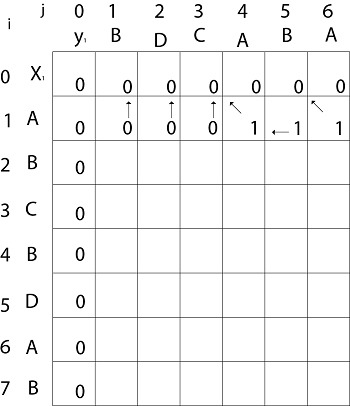
x1 and y6 we get x1=y6

c [1, 6] = c [1-1,6-1] + 1

= c [0, 5] + 1 = 0 + 1 = 1

c [1,6] = 1

b [1,6] = ' ↖ '



**Now for i=2 and j = 1**

We get x2 and y1 B = B i.e. x2= y1

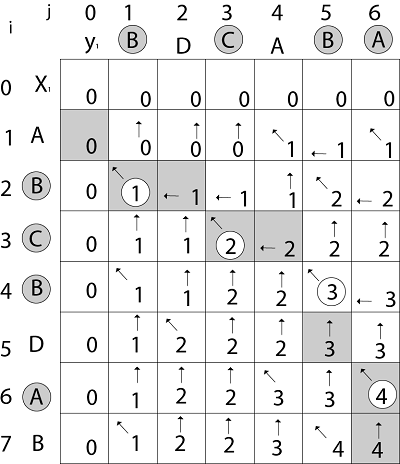
c [2,1] = c [2-1,1-1] + 1

= c [1, 0] + 1

= 0 + 1 = 1

c [2, 1] = 1 and b [2, 1] = ' ↖ '

Similarly, we fill the all values of c [i, j] and we get



**Step 4: Constructing an LCS:** The initial call is PRINT-LCS (b, X, X.length, Y.length)

**PRINT-LCS (b, x, i, j)**

1. if i=0 or j=0

2. then return

3. if b [i,j] = ' ↖ '

4. then PRINT-LCS (b,x,i-1,j-1)

5. print x\_i

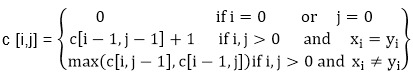
6. else if b [i,j] = ' ↑ '

7. then PRINT-LCS (b,X,i-1,j)

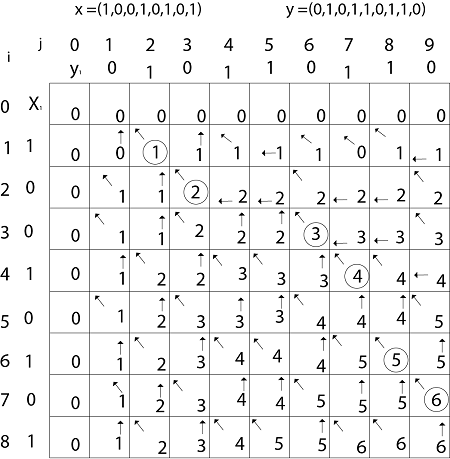
8. else PRINT-LCS (b,X,i,j-1)

**Example:** Determine the LCS of (1,0,0,1,0,1,0,1) and (0,1,0,1,1,0,1,1,0).

**Solution:** let X = (1,0,0,1,0,1,0,1) and Y = (0,1,0,1,1,0,1,1,0).



We are looking for c [8, 9]. The following table is built.



From the table we can deduct that LCS = 6. There are several such sequences, for instance (1,0,0,1,1,0) (0,1,0,1,0,1) and (0,0,1,1,0,1)

0/1 Knapsack problem

Here knapsack is like a container or a bag. Suppose we have given some items which have some weights or profits. We have to put some items in the knapsack in such a way total value produces a maximum profit.

For example, the weight of the container is 20 kg. We have to select the items in such a way that the sum of the weight of items should be either smaller than or equal to the weight of the container, and the profit should be maximum.

There are two types of knapsack problems:

* 0/1 knapsack problem
* Fractional knapsack problem

We will discuss both the problems one by one. First, we will learn about the 0/1 knapsack problem.

### What is the 0/1 knapsack problem?

The 0/1 knapsack problem means that the items are either completely or no items are filled in a knapsack. For example, we have two items having weights 2kg and 3kg, respectively. If we pick the 2kg item then we cannot pick 1kg item from the 2kg item (item is not divisible); we have to pick the 2kg item completely. This is a 0/1 knapsack problem in which either we pick the item completely or we will pick that item. The 0/1 knapsack problem is solved by the dynamic programming.

### What is the fractional knapsack problem?

The fractional knapsack problem means that we can divide the item. For example, we have an item of 3 kg then we can pick the item of 2 kg and leave the item of 1 kg. The fractional knapsack problem is solved by the Greedy approach.

### Example of 0/1 knapsack problem.

Consider the problem having weights and profits are:

Weights: {3, 4, 6, 5}

Profits: {2, 3, 1, 4}

The weight of the knapsack is 8 kg

The number of items is 4

The above problem can be solved by using the following method:

xi = {1, 0, 0, 1}

= {0, 0, 0, 1}

= {0, 1, 0, 1}

The above are the possible combinations. 1 denotes that the item is completely picked and 0 means that no item is picked. Since there are 4 items so possible combinations will be:

**24 = 16;** So. There are 16 possible combinations that can be made by using the above problem. Once all the combinations are made, we have to select the combination that provides the maximum profit.

Another approach to solve the problem is dynamic programming approach. In dynamic programming approach, the complicated problem is divided into sub-problems, then we find the solution of a sub-problem and the solution of the sub-problem will be used to find the solution of a complex problem.

### How this problem can be solved by using the Dynamic programming approach?

First,

we create a matrix shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |

In the above matrix, columns represent the weight, i.e., 8. The rows represent the profits and weights of items. Here we have not taken the weight 8 directly, problem is divided into sub-problems, i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8. The solution of the sub-problems would be saved in the cells and answer to the problem would be stored in the final cell. First, we write the weights in the ascending order and profits according to their weights shown as below:

**wi = {3, 4, 5, 6}**

**pi = {2, 3, 4, 1}**

**The first row and the first column would be 0 as there is no item for w=0**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 |  |  |  |  |  |  |  |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i=1, W=1**

w1 = 3; Since we have only one item in the set having weight 3, but the capacity of the knapsack is 1. We cannot fill the item of 3kg in the knapsack of capacity 1 kg so add 0 at M[1][1] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 |  |  |  |  |  |  |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 1, W = 2**

w1 = 3; Since we have only one item in the set having weight 3, but the capacity of the knapsack is 2. We cannot fill the item of 3kg in the knapsack of capacity 2 kg so add 0 at M[1][2] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |  |  |  |  |  |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i=1, W=3**

w1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is also 3; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][3] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 |  |  |  |  |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i=1, W = 4**

W1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 4; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][4] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 |  |  |  |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i=1, W = 5**

W1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 5; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][5] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 |  |  |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i =1, W=6**

W1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 6; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][6] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 |  |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i=1, W = 7**

W1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 7; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][7] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 |  |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i =1, W =8**

W1 = 3; Since we have only one item in the set having weight equal to 3, and weight of the knapsack is 8; therefore, we can fill the knapsack with an item of weight equal to 3. We put profit corresponding to the weight 3, i.e., 2 at M[1][8] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 |  |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**Now the value of 'i' gets incremented, and becomes 2.**

**When i =2, W = 1**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have only one item in the set having weight equal to 4, and the weight of the knapsack is 1. We cannot put the item of weight 4 in a knapsack, so we add 0 at M[2][1] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 |  |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i =2, W = 2**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have only one item in the set having weight equal to 4, and the weight of the knapsack is 2. We cannot put the item of weight 4 in a knapsack, so we add 0 at M[2][2] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i =2, W = 3**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 3. We can put the item of weight 3 in a knapsack, so we add 2 at M[2][3] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i =2, W = 4**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 4. We can put item of weight 4 in a knapsack as the profit corresponding to weight 4 is more than the item having weight 3, so we add 3 at M[2][4] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 2, W = 5**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 5. We can put item of weight 4 in a knapsack and the profit corresponding to weight is 3, so we add 3 at M[2][5] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 2, W = 6**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 6. We can put item of weight 4 in a knapsack and the profit corresponding to weight is 3, so we add 3 at M[2][6] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 2, W = 7**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 7. We can put item of weight 4 and 3 in a knapsack and the profits corresponding to weights are 2 and 3; therefore, the total profit is 5, so we add 5 at M[2][7] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 5 |  |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 2, W = 8**

The weight corresponding to the value 2 is 4, i.e., w2 = 4. Since we have two items in the set having weights 3 and 4, and the weight of the knapsack is 7. We can put item of weight 4 and 3 in a knapsack and the profits corresponding to weights are 2 and 3; therefore, the total profit is 5, so we add 5 at M[2][7] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**Now the value of 'i' gets incremented, and becomes 3.**

**When i = 3, W = 1**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set having weights 3, 4, and 5, and the weight of the knapsack is 1. We cannot put neither of the items in a knapsack, so we add 0 at M[3][1] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 3, W = 2**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set having weight 3, 4, and 5, and the weight of the knapsack is 1. We cannot put neither of the items in a knapsack, so we add 0 at M[3][2] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 3, W = 3**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set of weight 3, 4, and 5 respectively and weight of the knapsack is 3. The item with a weight 3 can be put in the knapsack and the profit corresponding to the item is 2, so we add 2 at M[3][3] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 2 |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 3, W = 4**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 4. We can keep the item of either weight 3 or 4; the profit (3) corresponding to the weight 4 is more than the profit corresponding to the weight 3 so we add 3 at M[3][4] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 3, W = 5**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 5. We can keep the item of either weight 3, 4 or 5; the profit (3) corresponding to the weight 4 is more than the profits corresponding to the weight 3 and 5 so we add 3 at M[3][5] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i =3, W = 6**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 6. We can keep the item of either weight 3, 4 or 5; the profit (3) corresponding to the weight 4 is more than the profits corresponding to the weight 3 and 5 so we add 3 at M[3][6] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i =3, W = 7**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set of weight 3, 4, and 5 respectively, and weight of the knapsack is 7. In this case, we can keep both the items of weight 3 and 4, the sum of the profit would be equal to (2 + 3), i.e., 5, so we add 5 at M[3][7] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 |  |
| 4 | 0 |  |  |  |  |  |  |  |  |

**When i = 3, W = 8**

The weight corresponding to the value 3 is 5, i.e., w3 = 5. Since we have three items in the set of weight 3, 4, and 5 respectively, and the weight of the knapsack is 8. In this case, we can keep both the items of weight 3 and 4, the sum of the profit would be equal to (2 + 3), i.e., 5, so we add 5 at M[3][8] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 |  |  |  |  |  |  |  |  |

**Now the value of 'i' gets incremented and becomes 4.**

**When i = 4, W = 1**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 1. The weight of all the items is more than the weight of the knapsack, so we cannot add any item in the knapsack; Therefore, we add 0 at M[4][1] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 |  |  |  |  |  |  |  |

**When i = 4, W = 2**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 2. The weight of all the items is more than the weight of the knapsack, so we cannot add any item in the knapsack; Therefore, we add 0 at M[4][2] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 |  |  |  |  |  |  |

**When i = 4, W = 3**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 3. The item with a weight 3 can be put in the knapsack and the profit corresponding to the weight 4 is 2, so we will add 2 at M[4][3] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 |  |  |  |  |  |

**When i = 4, W = 4**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 4. The item with a weight 4 can be put in the knapsack and the profit corresponding to the weight 4 is 3, so we will add 3 at M[4][4] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 |  |  |  |  |

**When i = 4, W = 5**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 5. The item with a weight 4 can be put in the knapsack and the profit corresponding to the weight 4 is 3, so we will add 3 at M[4][5] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 3 |  |  |  |

**When i = 4, W = 6**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 6. In this case, we can put the items in the knapsack either of weight 3, 4, 5 or 6 but the profit, i.e., 4 corresponding to the weight 6 is highest among all the items; therefore, we add 4 at M[4][6] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 1 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 3 | 4 |  |  |

**When i = 4, W = 7**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 7. Here, if we add two items of weights 3 and 4 then it will produce the maximum profit, i.e., (2 + 3) equals to 5, so we add 5 at M[4][7] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 3 | 4 | 5 |  |

**When i = 4, W = 8**

The weight corresponding to the value 4 is 6, i.e., w4 = 6. Since we have four items in the set of weights 3, 4, 5, and 6 respectively, and the weight of the knapsack is 8. Here, if we add two items of weights 3 and 4 then it will produce the maximum profit, i.e., (2 + 3) equals to 5, so we add 5 at M[4][8] shown as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 3 | 4 | 5 | 5 |

As we can observe in the above table that 5 is the maximum profit among all the entries. The pointer points to the last row and the last column having 5 value. Now we will compare 5 value with the previous row; if the previous row, i.e., i = 3 contains the same value 5 then the pointer will shift upwards. Since the previous row contains the value 5 so the pointer will be shifted upwards as shown in the below table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 3 | 4 | 5 | 5 |

Again, we will compare the value 5 from the above row, i.e., i = 2. Since the above row contains the value 5 so the pointer will again be shifted upwards as shown in the below table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 3 | 4 | 5 | 5 |

Again, we will compare the value 5 from the above row, i.e., i = 1. Since the above row does not contain the same value so we will consider the row i=1, and the weight corresponding to the row is 4. Therefore, we have selected the weight 4 and we have rejected the weights 5 and 6 shown below:

**x = { 1, 0, 0}**

The profit corresponding to the weight is 3. Therefore, the remaining profit is (5 - 3) equals to 2. Now we will compare this value 2 with the row i = 2. Since the row (i = 1) contains the value 2; therefore, the pointer shifted upwards shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 3 | 0 | 0 | 0 | 2 | 3 | 3 | 3 | 5 | 5 |
| 4 | 0 | 0 | 0 | 2 | 3 | 3 | 4 | 5 | 5 |

Again we compare the value 2 with a above row, i.e., i = 1. Since the row i =0 does not contain the value 2, so row i = 1 will be selected and the weight corresponding to the i = 1 is 3 shown below:

**X = {1, 1, 0, 0}**

The profit corresponding to the weight is 2. Therefore, the remaining profit is 0. We compare 0 value with the above row. Since the above row contains a 0 value but the profit corresponding to this row is 0. In this problem, two weights are selected, i.e., 3 and 4 to maximize the profit.

# Longest Palindromic Subsequence

Before knowing about the longest palindromic subsequence, we should about "what is subsequence". A subsequence is a sequence that is achieved from the sequence by removing some of the elements from the sequence without changing the order of the remaining elements.

**Here we have to find the longest Palindromic Subsequence in the given string. Let's understand through some examples.**

**Example 1:**

**Input string: "a d b b c a"**

The longest palindromic subsequence is "a b b a". The length of the subsequence is 4.

**Example 2:**

**Input string: "p q r d r p d"**

The longest palindromic subsequence is "p r d r p". The length of the subsequence is 5.

**Example 3:**

**Input string: "p q r r q p"**

The longest palindromic subsequence is "p q r r q p". The length of the subsequence is 6.

In order to solve this problem, we will use the dynamic programming. Here we use the memorization matrix.

**Suppose the string which is given below:**

**Input string: "a g b d b a".** Here, we consider the matrix of same length as that of the string. The length of the string is 6 so we consider the matrix as 6\*6. The indices of characters 'a', 'g', 'b', 'd', 'b', 'a' are 0, 1, 2, 3, 4, 5 respectively.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |

First, we consider the length equal to 1, i.e., l = 1. It means that we consider the single character at a time. If we are considering the single character then the length of the longest palindromic subsequence would also be equal to 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 |  |  |  |  |  |
| 1 |  | 1 |  |  |  |  |
| 2 |  |  | 1 |  |  |  |
| 3 |  |  |  | 1 |  |  |
| 4 |  |  |  |  | 1 |  |
| 5 |  |  |  |  |  | 1 |

Now we consider the length of string as 2. It means that we consider two characters at a time. First, we consider "a g" as a string. Since both the characters are different so the length of the longest palindromic subsequence would be 1 shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 |  |  |  |  |
| 1 |  | 1 |  |  |  |  |
| 2 |  |  | 1 |  |  |  |
| 3 |  |  |  | 1 |  |  |
| 4 |  |  |  |  | 1 |  |
| 5 |  |  |  |  |  | 1 |

Now we consider another string as "gb". It means that we consider two characters at a time. Since both the characters are different so the length of the longest palindromic subsequence would be 1 shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 |  |  |  |  |
| 1 |  | 1 | 1 |  |  |  |
| 2 |  |  | 1 |  |  |  |
| 3 |  |  |  | 1 |  |  |
| 4 |  |  |  |  | 1 |  |
| 5 |  |  |  |  |  | 1 |

Now we consider another string as "bd". It means that we consider two characters at a time. Since both the characters are different so the length of the longest palindromic subsequence would be 1 shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 |  |  |  |  |
| 1 |  | 1 | 1 |  |  |  |
| 2 |  |  | 1 | 1 |  |  |
| 3 |  |  |  | 1 |  |  |
| 4 |  |  |  |  | 1 |  |
| 5 |  |  |  |  |  | 1 |

Now we consider another string as "db". It means that we consider two characters at a time. Since both the characters are different so the length of the longest palindromic subsequence would be 1 shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 |  |  |  |  |
| 1 |  | 1 | 1 |  |  |  |
| 2 |  |  | 1 | 1 |  |  |
| 3 |  |  |  | 1 | 1 |  |
| 4 |  |  |  |  | 1 |  |
| 5 |  |  |  |  |  | 1 |

Now we consider another string as "ba". It means that we consider two characters at a time. Since both the characters are different so the length of the longest palindromic subsequence would be 1 shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 |  |  |  |  |
| 1 |  | 1 | 1 |  |  |  |
| 2 |  |  | 1 | 1 |  |  |
| 3 |  |  |  | 1 | 1 |  |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Consider the length of the string as 3, i.e., "a g b". It means that we consider three characters at a time. Since the first and the last character of a string is different so we consider either "ag" or "gb". The length of "ag" and "gb" is 1 so we put 1 at S[1][2] shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 |  |  |  |
| 1 |  | 1 | 1 |  |  |  |
| 2 |  |  | 1 | 1 |  |  |
| 3 |  |  |  | 1 | 1 |  |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Consider the string as "g b d". It means that we consider three characters at a time. Since the first and the last character of a string is different so we consider either "gb" or "bd". The length of "gb" and "bd" is 1 so we put 1 at S[1][3] shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 |  |  |  |
| 1 |  | 1 | 1 | 1 |  |  |
| 2 |  |  | 1 | 1 |  |  |
| 3 |  |  |  | 1 | 1 |  |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Consider the string as "b d b". It means that we consider three characters at a time. Since the first and the last character of a string is same so the maximum palindromic subsequence would be equal to (2 + 1), i.e., 3. We put 3 at S[1][4] shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 |  |  |  |
| 1 |  | 1 | 1 | 1 |  |  |
| 2 |  |  | 1 | 1 | 3 |  |
| 3 |  |  |  | 1 | 1 |  |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Consider the string as "d b a". It means that we consider three characters at a time. Since the first and the last character of a string is different so we consider either "db" or "ba". The length of "db" and "ba" is 1 so we put 1 at S[1][5] shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 |  |  |  |
| 1 |  | 1 | 1 | 1 |  |  |
| 2 |  |  | 1 | 1 | 3 |  |
| 3 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Now we have length l=4. We consider the indices from 0 to 3. Consider the string as "a g b d". It means that we consider four characters at a time. Since the first and the last character of a string is different so we take three characters string either "a g b" or "g b d". The length of both "a g b" and "g b d" is same, i.e., 1 so we put 1 at S[0][3].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 | 1 |  |  |
| 1 |  | 1 | 1 | 1 |  |  |
| 2 |  |  | 1 | 1 | 3 |  |
| 3 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Consider the indices from 1 to 4. Consider the string as "g b d b". It means that we consider four characters at a time. Since the first and the last character of a string is different so we take three characters string at a time either "g b d" or "b d b". The length of palindromic subsequence of string "b d b" is maximum, i.e., 3, so we add 3 at S[1][4].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 | 1 |  |  |
| 1 |  | 1 | 1 | 1 | 3 |  |
| 2 |  |  | 1 | 1 | 3 |  |
| 3 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Consider the indices from 2 to 5. Consider the string as "b d b a". It means that we consider four characters at a time. Since the first and the last character of a string is different so consider the three-character string either "b d b" or "d b a". The length of the palindromic subsequence of string "b d b" is maximum, i.e., 3, so we add 3 at **S[2][5]** shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 | 1 |  |  |
| 1 |  | 1 | 1 | 1 | 3 |  |
| 2 |  |  | 1 | 1 | 3 | 3 |
| 3 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Now we have length l = 5. Consider the indices from 0 to 4. Consider the string as "a g b d b". It means that we take five characters at a time. Since the first and the last character of a string is different so we consider the four-character string either as "a g b d" or "g b d b". The length of the palindromic subsequence of string "g b d b" is 3 which is maximum so we add 3 at S[0][4] shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 | 1 | 3 |  |
| 1 |  | 1 | 1 | 1 | 3 |  |
| 2 |  |  | 1 | 1 | 3 | 3 |
| 3 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

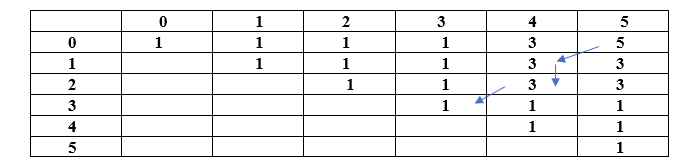
Consider the indices from 1 to 5. Consider the string as "g b d b a". It means that we take five characters at a time. Since the first and the last character of a string is different so we consider the four-character string either as "g b d b" or "b d b a". Both the strings have the same length, i.e., 3, we add 3 at S[1][5] shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 | 1 | 3 |  |
| 1 |  | 1 | 1 | 1 | 3 | 3 |
| 2 |  |  | 1 | 1 | 3 | 3 |
| 3 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

Now we have length l =6. Consider the indices from 0 to 5. Consider the string as "a g b d b a". It means that we take six characters at a time. Since the first and the last character of a string is same, so the value at S[0][5] would be equal to 2 plus value at S[1][4], i.e. (2+3) equal to 5 shown as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 1 | 1 | 3 | 5 |
| 1 |  | 1 | 1 | 1 | 3 | 3 |
| 2 |  |  | 1 | 1 | 3 | 3 |
| 3 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  |  | 1 | 1 |
| 5 |  |  |  |  |  | 1 |

As we can observe in the above table that considering string **"a g b d b a"** produces the length of the palindromic subsequence, i.e., 5. Since the length 5 comes from the string "g b d b" having length 3, the length 3 comes from the string "b d b" having length equal to 3, and the length 3 comes from the string "d" having length 1 shown as below:



Now we will find the string. Since 5 comes from 3; therefore, the character at 0 and 5 would be included in the string, i.e., 'a' shown as below:

S: a a

Since 3 comes from the length 3; therefore, the characters at 2 and 4 would be included in the string shown as below:

S: a b b a

Since the length 3 comes from 1; therefore, the character at 3, i.e., 'd' would be included in the string shown as below:

**S: a b d b a**

In this problem, the palindromic subsequence is "a b d b a" which is having the maximum palindromic subsequence (5).

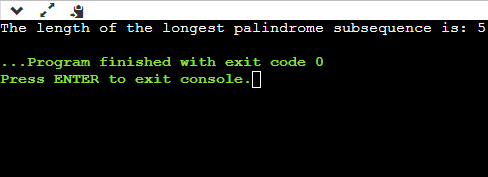
**Algorithm of palindromic subsequence**

1. if(input[i] == input[j])
2. {
3. T[i][j] = T[i+1][j-1] + 2
4. }
5. else
6. {
7. T[i][j] = max{T[i+1][j], T[i][j-1]}
8. }

**Implementation in C**

1. #include**<stdio.h>**
2. #include**<string.h>**
3. // A max function to find the maximum of two integers
4. int max (int x, int y) { return (x **>** y)? x : y; }
5. // Returns the length of the longest palindromic subsequence in sequence
6. int longest\_palindrome(char a[], int i, int j)
7. {
8. // Base Case 1: If there is only 1 character
9. if (i == j)
10. return 1;
11. // \*Base Case 2: If there are only 2
12. // characters and both are same\*
13. if (a[i] == a[j] && i + 1 == j)
14. return 2;
15. // If the first and last characters match
16. if (a[i] == a[j])
17. return longest\_palindrome(a, i+1, j-1) + 2;
18. // If the first and last characters do not match
19. return max(longest\_palindrome(a, i, j-1), longest\_palindrome(a, i+1, j) );
20. }
21. /\* Driver program to test above functions \*/
22. int main()
23. {
24. char seq[] = "ABBDCACB";
25. int n = strlen(seq);
26. printf("The length of the longest palindrome subsequence is: %d", longest\_palindrome(seq, 0, n-1));
27. return 0;
29. }

**Output**



# Longest Common Subsequence

Here longest means that the subsequence should be the biggest one. The common means that some of the characters are common between the two strings. The subsequence means that some of the characters are taken from the string that is written in increasing order to form a subsequence.

**Let's understand the subsequence through an example.**

Suppose we have a string 'w'.

**W1 = abcd**

**The following are the subsequences that can be created from the above string:**

* ab
* bd
* ac
* ad
* acd
* bcd

The above are the subsequences as all the characters in a sub-string are written in increasing order with respect to their position. If we write ca or da then it would be a wrong subsequence as characters are not appearing in the increasing order. The total number of subsequences that would be possible is 2n, where n is the number of characters in a string. In the above string, the value of 'n' is 4 so the total number of subsequences would be 16.

**W2= bcd**

By simply looking at both the strings w1 and w2, we can say that bcd is the longest common subsequence. If the strings are long, then it won't be possible to find the subsequence of both the string and compare them to find the longest common subsequence.

**Finding LCS using dynamic programming with the help of a table.**

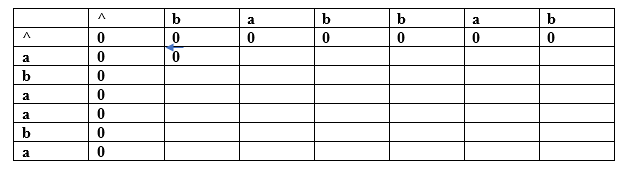
**Consider two strings:**

X= a b a a b a

Y= b a b b a b

**(a, b)**

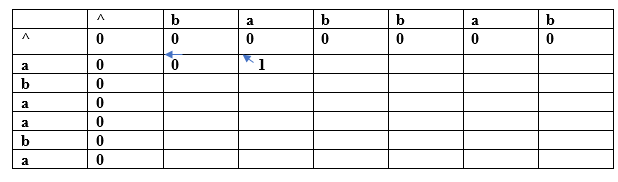
**For index i=1, j=1**



Since both the characters are different so we consider the maximum value. Both contain the same value, i.e., 0 so put 0 in (a,b). Suppose we are taking the 0 value from 'X' string, so we put arrow towards 'a' as shown in the above table.

**(a, a)**

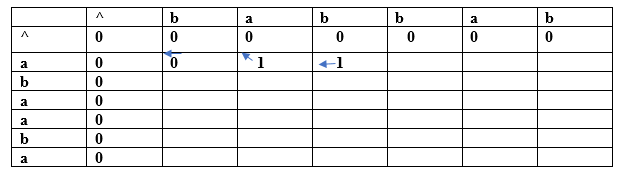
**For index i=1, j=2**



Both the characters are the same, so the value would be calculated by adding 1 and upper diagonal value. Here, upper diagonal value is 0, so the value of this entry would be (1+0) equal to 1. Here, we are considering the upper diagonal value, so the arrow will point diagonally.

**(a, b)**

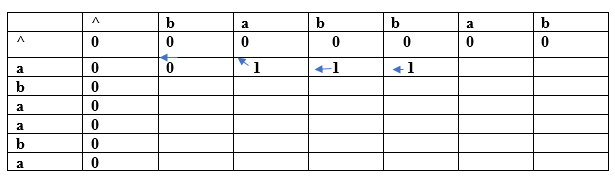
**For index i=1, j=3**



Since both the characters are different so we consider the maximum value. The character 'a' has the maximum value, i.e., 1. The new entry, i.e., (a, b) will contain the value 1 pointing to the 1 value.

**(a, b)**

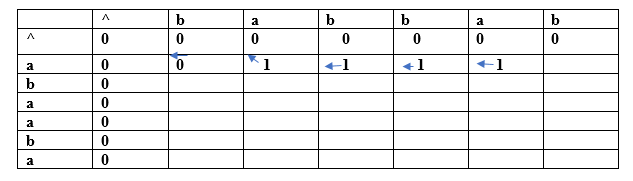
**For index i=1, j=4**



Since both the characters are different so we consider the maximum value. The character 'a' has the maximum value, i.e., 1. The new entry, i.e., (a, b) will contain the value 1 pointing to the 1 value.

**(a, a)**

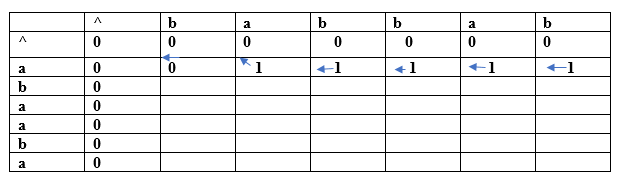
**For index i=1, j=5**



Both the characters are same so the value would be calculated by adding 1 and upper diagonal value. Here, upper diagonal value is 0 so the value of this entry would be (1+0) equal to 1. Here, we are considering the upper diagonal value so arrow will point diagonally.

**(a, b)**

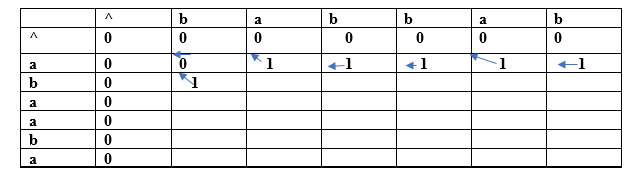
**For index i=1, j=6**



Since both the characters are different so we consider the maximum value. The character 'a' has the maximum value, i.e., 1. The new entry, i.e., (a, b) will contain the value 1 pointing to the 1 value.

**(b, b)**

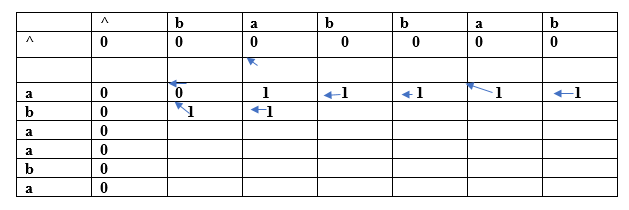
**For index i=2, j=1**



Both the characters are same so the value would be calculated by adding 1 and upper diagonal value. Here, upper diagonal value is 0 so the value of this entry would be (1+0) equal to 1. Here, we are considering the upper diagonal value so arrow will point diagonally.

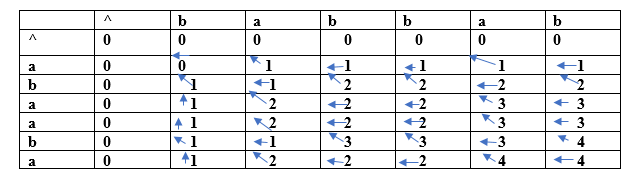
**(b, a)**

**For index i=2, j=2**



Since both the characters are different so we consider the maximum value. The character 'a' has the maximum value, i.e., 1. The new entry, i.e., (a, b) will contain the value 1 pointing to the 1 value.

In this way, we will find the complete table. The final table would be:



In the above table, we can observe that all the entries are filled. Now we are at the last cell having 4 value. This cell moves at the left which contains 4 value.; therefore, the first character of the LCS is 'a'.

The left cell moves upwards diagonally whose value is 3; therefore, the next character is 'b' and it becomes 'ba'. Now the cell has 2 value that moves on the left. The next cell also has 2 value which is moving upwards; therefore, the next character is 'a' and it becomes 'aba'.

The next cell is having a value 1 that moves upwards. Now we reach the cell (b, b) having value which is moving diagonally upwards; therefore, the next character is 'b'. The final string of longest common subsequence is 'baba'.

**Why a dynamic programming approach in solving a LCS problem is more efficient than the recursive algorithm?**

If we use the dynamic programming approach, then the number of function calls are reduced. The dynamic programming approach stores the result of each function call so that the result of function calls can be used in the future function calls without the need of calling the functions again.

In the above dynamic algorithm, the results obtained from the comparison between the elements of x and the elements of y are stored in the table so that the results can be stored for the future computations.

The time taken by the dynamic programming approach to complete a table is O(mn) and the time taken by the recursive algorithm is 2max(m, n).

**Algorithm of Longest Common Subsequence**

1. Suppose X and Y are the two given sequences
2. Initialize a table of LCS having a dimension of X.length \* Y.length
3. XX.label = X
4. YY.label = Y
5. LCS[0][] = 0
6. LCS[][0] = 0
7. Loop starts from the LCS[1][1]
8. Now we will compare X[i] and Y[j]
9. if X[i] is equal to Y[j] then
10. LCS[i][j] = 1 + LCS[i-1][j-1]
11. Point an arrow LCS[i][j]
12. Else
13. LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])