Module-2

Dynamic Programming

BCSC0012

# Designing Techniques of Algorithms

Design and Analysis of Algorithms

Dr. Gaurav Kumar
Asst. Prof, CEA, GLAU, Mathura

# Designing Techniques of Algorithms

01

Divide and Conquer Strategy

02

Greedy Technique

03

Dynamic Programming

04

Branch and Bound

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# 03

# Dynamic Programming

# Dynamic Programming

01

Dynamic Programming (DP) is similar to Divide and Conquer Strategy 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 process of storing the results of sub-problems is known as **Memoization**.

02

DP is used to **solve overlapping sub-problems** and problems having optimal substructures properties.

- Here, optimal substructure means that the solution of optimization problems can be obtained by simply combining the optimal solution of all the sub-problems.

03

Whenever we have recursive function calls with the same result, instead of calling them again we try to store the result in a data structure in the form of a table and retrieve the results from the table.

# Dynamic Programming

04

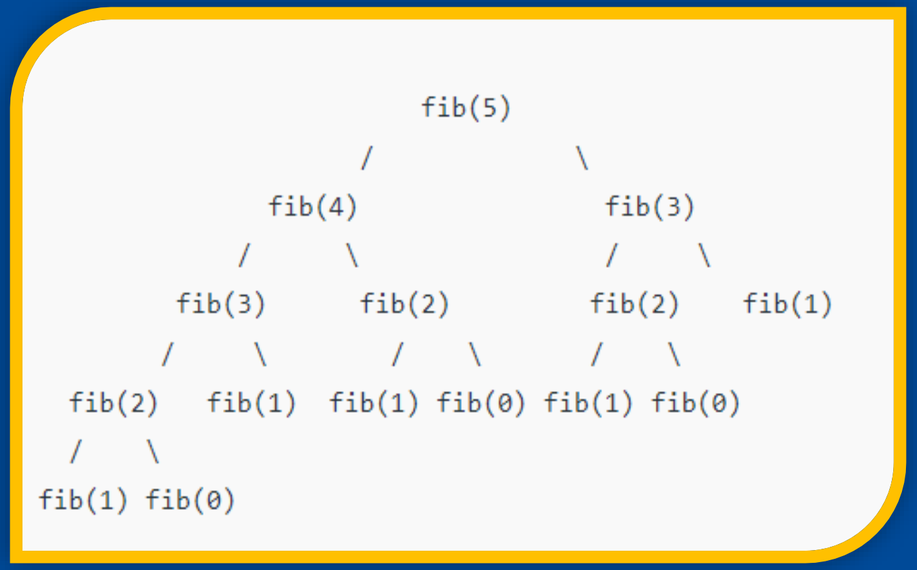
- The main use of dynamic programming is to solve optimization problems.
    - Optimization problems mean that when we are trying to find out the minimum or the maximum solution of a problem.
    - The dynamic programming guarantees to find the optimal solution of a problem if the solution exists.

05

“Dynamic” means we dynamically decide, whether to call a function or retrieve values from the table.

# Example of Dynamic Programming

Recursion tree for execution of fib(5)




We can see that the function fib(3) is being called 2 times and fib(2) is 3 times. If we would have stored the value of fib(3) and fib(2), then instead of computing it again, we could have reused the old stored value.

There are following two different ways to store the values so that these values can be reused:

**a) Memoization (Top Down)**

**b) Tabulation (Bottom Up)**

# Understanding Top-down & Bottom-up

Below is an interesting analogy

**Top-down** - First you say I will take over the world. How will you do that? You say I will take over Asia first. How will you do that? I will take over India first. I will become the Chief Minister of Delhi, etc. etc.

**Bottom-up** - You say I will become the CM of Delhi. Then will take over India, then all other countries in Asia and finally I will take over the world.

# Example of Dynamic Programming

There are following two different ways to store the values so that these values can be reused:

## a) Memoization (Top Down)

- The memoized program for a problem is similar to the recursive version with a small modification that looks into a **lookup table** before computing solutions.
- We initialize a lookup array with all initial values as NIL.
- Whenever we need the solution to a sub-problem, we first look into the lookup table. If the precomputed value is there then we return that value, otherwise, we calculate the value and put the result in the lookup table so that it can be reused later.
- Memoized version, table is filled on demand

# Example of Dynamic Programming

There are following two different ways to store the values so that these values can be reused:

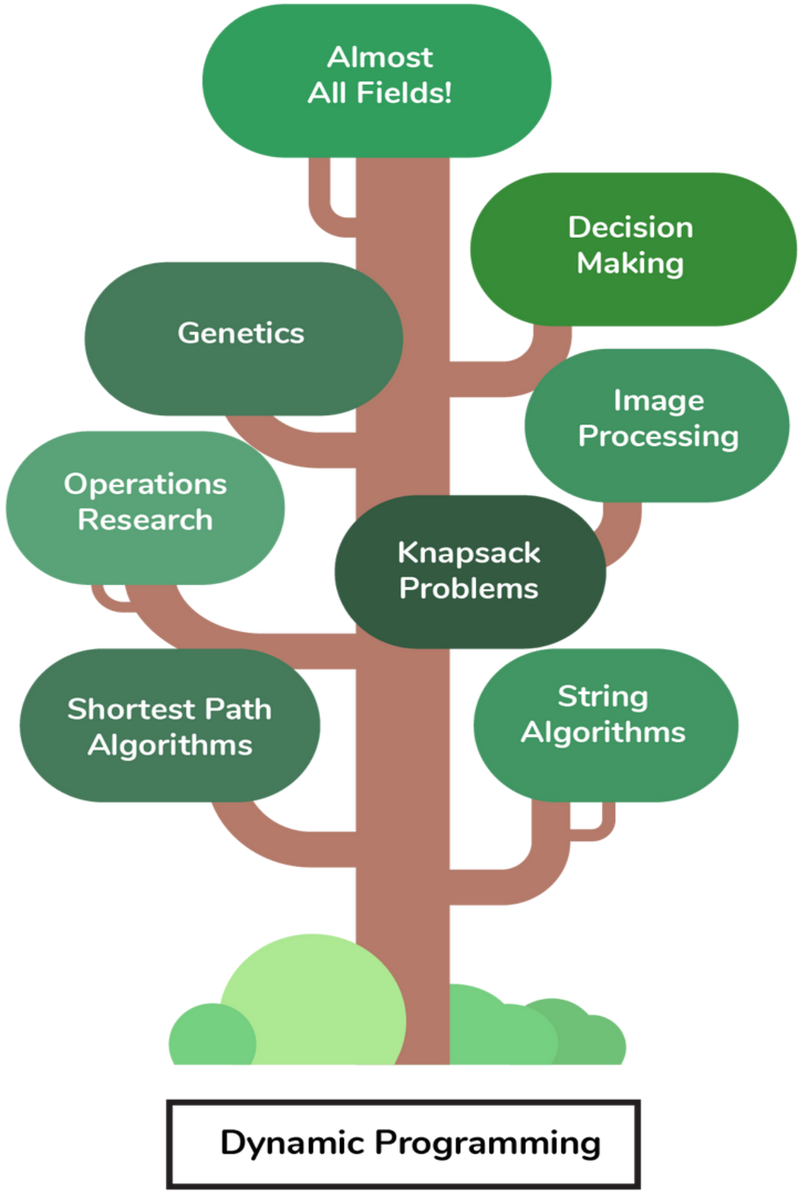
## b) Tabulation (Bottom Up)

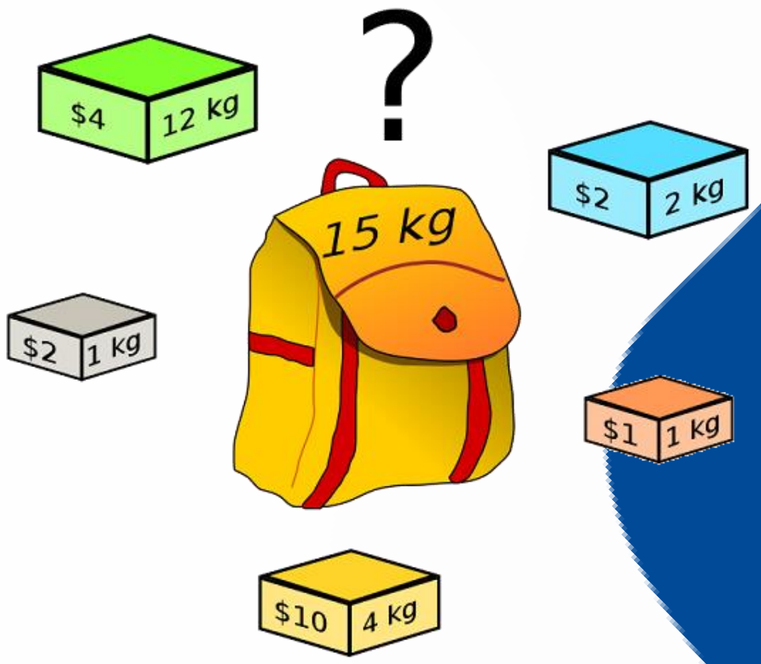
- The tabulated program for a given problem builds a table in bottom-up fashion and returns the last entry from the table.
- For example, for the same Fibonacci number, we first calculate fib(0) then fib(1) then fib(2) then fib(3), and so on. So literally, we are building the solutions of sub-problems bottom-up.
- In the Tabulated version, starting from the first entry, all entries are filled one by one.

# Difference Between Tabulation & Memoization

| | Tabulation | Memoization |
|---|---|---|
| **State** | State Transition relation is difficult to think | State transition relation is easy to think |
| **Code** | Code gets complicated when lot of conditions are required | Code is easy and less complicated |
| **Speed** | Fast, as we directly access previous states from the table | Slow due to lot of recursive calls and return statements |
| **Subproblem solving** | If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor | If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required |
| **Table Entries** | In Tabulated version, starting from the first entry, all entries are filled one by one | Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. The table is filled on demand. |

# Application of DP




# 03.a

# 0/1 Knapsack Problem

# 0/1 Knapsack Problem

The 0/1 knapsack problem means that the items are either completely or no items are filled in a knapsack.

n=4

m=8

| | | | | |
|---|---|---|---|---|
| $P_i$ | 1 | 2 | 5 | 6 |
| $Wt_i$ | 2 | 3 | 4 | 5 |
| | Mirror | Silver nugget | Painting | Vase |

n is number of items

m is the capacity of knapsack

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# 0/1 Knapsack Problem

n=4

m=8

| $P_i$ | 1 | 2 | 5 | 6 |
|---|---|---|---|---|
| $Wt_i$ | 2 | 3 | 4 | 5 |

$x_i$ = {1, 0, 0, 1}
{0, 0, 0, 1}
{0, 1, 0, 1}
{1, 0, 1, 0}
.
.
.
.
.

- 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: **$2^4 = 16$;** 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.
- If there are n items so possible combinations will be: **$2^n$,** So. There are $2^n$ possible combinations that can be made by using the above problem.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# 0/1 Knapsack Problem

n=4

m=8

| $P_i$ | 1 | 2 | 5 | 6 |
|---|---|---|---|---|
| $Wt_i$ | 2 | 3 | 4 | 5 |

$x_i$ = {1, 0, 0, 1}
{0, 0, 0, 1}
{0, 1, 0, 1}
{1, 0, 1, 0}
.
.
.
.
.

- We can solve such type of problem using **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 large complex problem

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Algorithm of 0/1 Knapsack Problem

n=4

m=8

| $P_i$ | 1 | 2 | 5 | 6 |
|---|---|---|---|---|
| $W_i$ | 2 | 3 | 4 | 5 |

**Step- 1: We create a matrix/Table as shown as below (Table is named as T)**

| $P_i$ | $w_i$ | i \ W | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | | | | | | | | | |
| 1 | 2 | 1 | | | | | | | | | |
| 2 | 3 | 2 | | | | | | | | | |
| 5 | 4 | 3 | | | | | | | | | |
| 6 | 5 | 4 | | | | | | | | | |

- In this 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 as shown.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Algorithm of 0/1 Knapsack Problem

| | | | | | |
|---|---|---|---|---|---|
| n=4 | $P_i$ | 1 | 2 | 5 | 6 |
| m=8 | $W_i$ | 2 | 3 | 4 | 5 |

Here, T(i , w) = maximum value of the selected items

**Step-02:** Start filling the table row wise top to bottom from left to right.

Use the following formula-

**$T(i, w) = \max \{ T[i-1][w], P_i + T[i-1][w - wt_i] \}$ // Value of T(i, w) is max profit**

- This step leads to completely filling the table.

- **Then, value of the last box represents the maximum possible profit that can be put into the knapsack.**

| $P_i$ | $w_i$ | $W_i$ → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | | | | | | | | |
| 1 | 2 | 1 | | | | | | | | |
| 2 | 3 | 2 | | | | | | | | |
| 5 | 4 | 3 | | | | | | | | |
| 6 | 5 | 4 | | | | | | | | |

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Algorithm of 0/1 Knapsack Problem

n=4

m=8

| $P_i$ | 1 | 2 | 5 | 6 |
|---|---|---|---|---|
| $W_i$ | 2 | 3 | 4 | 5 |

**Step-03:**

To identify the items that must be put into the knapsack to obtain that maximum profit, Consider the last column of the table.

1) Start scanning the entries from bottom to top.

2) On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.

3) After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Exercise of 0/1 Knapsack Problem

n=4

m=8

| $P_i$ | 1 | 2 | 5 | 6 |
|---|---|---|---|---|
| $Wt_i$ | 2 | 3 | 4 | 5 |

**Step-02:** Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, w) = \max \{ T[i-1][w], P_i + T[i-1][w - wt_i] \}$$

| $P_i$ | $w_i$ |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 5 | 4 |
| 6 | 5 |

| i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | |
| 1 | | | | | | | | | |
| 2 | | | | | | | | | |
| 3 | | | | | | | | | |
| 4 | | | | | | | | | |

## Or Pseudo Code

```
Knapsack(p,wt,m,n)
 for i=0 to n
    for w=0 to m
       if i==0 || w==0
          T[i][w]=0
       else
          if wt[i]<=w
             T[i][w]=max(p[i]+T[i-1][w-wt[i]], T[i-1][w])
       else
             T[i][w]=T[i-1][w]
```

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving 0/1 Knapsack Problem

**Step-02:** Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, w) = \max \{ T[i-1][w], P_i + T[i-1][w - wt_i] \}$$

**For i=0**

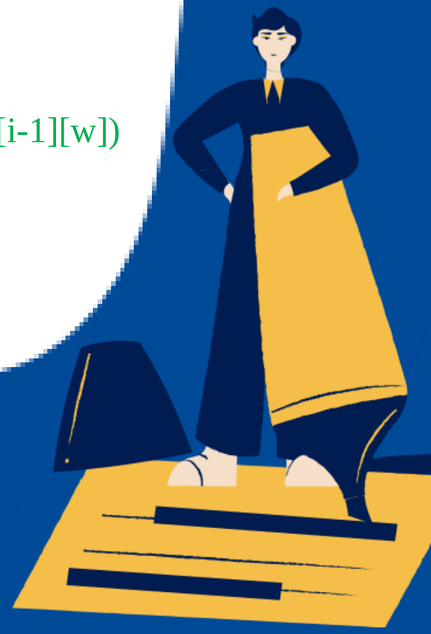
w= 0 w= 1 w= 2 ..... w= 8

T[0][0]=0 T[0][1]=0 T[0][2]=0 ..... T[0][8]=0

| | | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_i$ | $w_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | | | | | | | | | |
| 2 | 3 | 2 | | | | | | | | | |
| 5 | 4 | 3 | | | | | | | | | |
| 6 | 5 | 4 | | | | | | | | | |

**Knapsack(p,wt,m,n)**

```
for i=0 to n
    for w=0 to m
        if i==0 || w==0
            T[i][w]=0
        else if wt[i]<=w
            T[i][w]=max(p[i]+T[i-1][w-wt[i]], T[i-1][w])
        else
            T[i][w]=T[i-1][w]
```



# Solving 0/1 Knapsack Problem

**Step-02:** Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, w) = \max \{ T[i-1][w], P_i + T[i-1][w - wt_i] \}$$

**For i=1**

w= 0

T[1][0]=0

w= 1

if 2 < 1

else

T[1][1] = T[1][1]= 0

w= 2

if 2 =2

T[1][2] = max (1+ T[0][0], T[0][2])

= max( 1 + 0 , 0)

= 1

w= 3

if 2 <3

T[1][3] = max (1+ T[0][1], T[0][3])

= max( 1 + 0 , 0)

= 1

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | | | | | | | | | |
| 5 | 4 | 3 | | | | | | | | | |
| 6 | 5 | 4 | | | | | | | | | |

**Or**

**Pseudo Code**

```
Knapsack(p,wt,m,n)
  for i=0 to n
    for w=0 to m
      if i==0 || w==0
        T[i][w]=0
      else if wt[i]<=w
        T[i][w]=max(p[i]+T[i-1][w-wt[i]], T[i-1][w])
      else
        T[i][w]=T[i-1][w]
```

w= 4

if 2 <4

T[1][4] = max (1+ T[0][2], T[0][4])

= max( 1 + 0 , 0)

= 1

Continue……..

# Solving 0/1 Knapsack Problem

```
Knapsack(p,wt,m,n)
  for i=0 to n
    for w=0 to m
      if i==0 || w==0
        T[i][w]=0
      else if wt[i]<=w
        T[i][w]=max(p[i]+T[i-1][w-wt[i]], T[i-1][w])
       else
        T[i][w]=T[i-1][w]
```

**Step-02:** Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, w) = \max \{ T[i-1][w], P_i + T[i-1][w - wt_i] \}$$

**For i=2**

w= 0

T[2][0]=0

w= 1

if 3 < 1

else

T[2][1] = T[1][1]= 0

w= 2

if 3 < 2

else

T[2][2] = T[1][2]= 1

w= 3

if 3 =3

T[2][3] = max (2+ T[1][0], T[1][3])
= max( 2 + 0 , 1)
= 2

i=2, w= 4

if 3 <4

T[2][4] = max (2+ T[1][1], T[1][4])
= max( 2 + 0 , 1)
= 2

i=2, w= 5

if 3 <5

T[2][5] = max (2+ T[1][2], T[1][5])
= max( 2 + 1 , 1)
= 3

i=2, w= 6

if 3 <6

T[2][6] = max (2+ T[1][3], T[1][6])
= max( 2 + 1 , 1)
= 3

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | | | | | | | | | |
| 6 | 5 | 4 | | | | | | | | | |

Continue……..

# Solving 0/1 Knapsack Problem

```
Knapsack(p,wt,m,n)
  for i=0 to n
    for w=0 to m
      if i==0 || w==0
        T[i][w]=0
      else if wt[i]<=w
        T[i][w]=max(p[i]+T[i-1][w-wt[i]], T[i-1][w])
       else
      T[i][w]=T[i-1][w]
```

**Step-02:** Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, w) = \max \{ T[i-1][w], P_i + T[i-1][w - wt_i] \}$$

**For i=3**

w= 0

w= 1
if 4 < 1
T[3][0]=0
else
T[3][1] = T[2][1]= 0

w= 2
if 4 < 2
else
T[3][2] = T[2][2]= 1

w= 3
if 4 < 3
else
T[3][3] = T[2][3]= 2

w= 4
if 4 =4
T[3][4] = max (5+ T[2][0], T[2][4])
= max( 5 + 0 , 2)
= 5

i=3, w= 5
if 4 <5
T[3][5] = max (5+ T[2][1], T[2][5])
= max( 5 + 0 , 3)
= 5

i=3, w= 6
if 4 <6
T[3][6] = max (5+ T[2][2], T[2][6])
= max( 5 + 1 , 3)
= 6

i=3, w= 7
if 4 <7
T[3][7] = max (5+ T[2][3], T[2][7])
= max( 5 + 2 , 3)
= 7

Continue……..

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | | | | | | | | | |

# Solving 0/1 Knapsack Problem

**Step-02:** Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, w) = \max \{ T[i-1][w], P_i + T[i-1][w - wt_i] \}$$

```
Knapsack(p,wt,m,n)
  for i=0 to n
    for w=0 to m
      if i==0 || w==0
        T[i][w]=0
      else if wt[i]<=w
        T[i][w]=max(p[i]+T[i-1][w-wt[i]], T[i-1][w])
       else
      T[i][w]=T[i-1][w]
```

**For i=4**

w= 0
T[4][0]=0

w= 1
if 5 < 1
else
T[4][1] = T[3][1]= 0

w= 2
if 5 < 2
else
T[4][2] = T[3][2]= 1

w= 3
if 5 < 3
else
T[4][3] = T[3][3]= 2

w= 4
if 5 < 4
else
T[4][4] = T[3][4]= 5

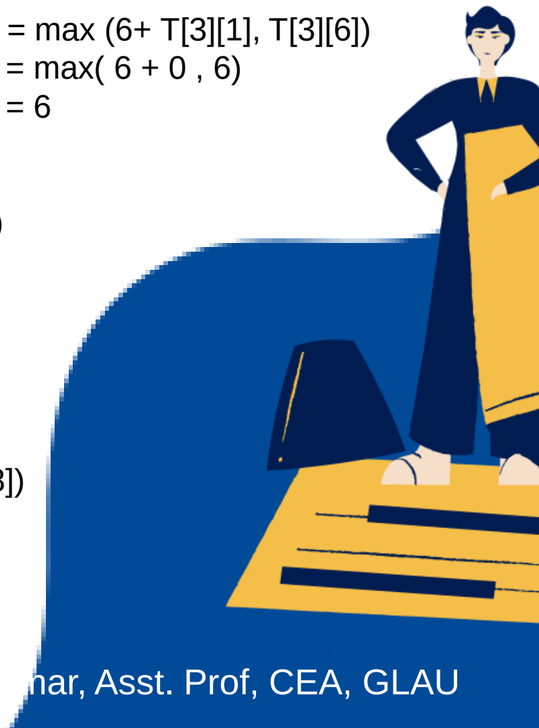
w= 5
if 5 =5
T[4][5] = max (6+ T[3][0], T[3][5])
= max( 6 + 0 , 5)
= 6

i=4, w= 6
if 5 <6
T[4][6] = max (6+ T[3][1], T[3][6])
= max( 6 + 0 , 6)
= 6

i=4, w= 7
if 5 <7
T[4][7] = max (6+ T[3][2], T[3][7])
= max( 6 + 1 , 7)
= 7

i=4, w= 8
if 5 <8
T[4][8] = max (6+ T[3][3], T[3][8])
= max( 6 + 2 , 7)
= max (8, 7)
= 8

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |



# Solving 0/1 Knapsack Problem

Value of T(i, w) is max profit , here T[4][8]= 8 is the maximum profit of Knapsack

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

**Step-03:** To identify the items that must be put into the knapsack to obtain that maximum profit,

Consider the last column of the table.

1) Start scanning the entries from bottom to top.
2) On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
3) After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving 0/1 Knapsack Problem

Value of T(i, w) is max profit , here T[4][8]= 8 is the maximum profit

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

Step-03:

To identify the items that must be put into the knapsack to obtain that maximum profit,

Consider the last column of the table.

1) Start scanning the entries from bottom to top.
2) On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
3) After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

Step-03:

```
while (i > 0 && w>0)
   if (T[i][w] == T[i-1][w]
       Not select the current item
       i--
   else
       select the item i
       i--;
       w= w-wt[i]
```

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving 0/1 Knapsack Problem

Value of T(i, w) is max profit , here T[4][8]= 8 is the maximum profit

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

Step-03:

**To identify the items that must be put into the knapsack to obtain that maximum profit,**

**Consider the last column of the table.**

1) **Start scanning the entries from bottom to top.**
2) **On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.**
3) **After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.**

Step-03:

```
while (i > 0 && w>0)
   if (T[i][w] == T[i-1][w]
       Not select the current item
       i--
   else
       select the item i
       i--;
       w= w-wt[i]
```

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving 0/1 Knapsack Problem

Value of T(i, w) is max profit , here T[4][8]= 8 is the maximum profit

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

Step-03:

**To identify the items that must be put into the knapsack to obtain that maximum profit,**

**Consider the last column of the table.**

1) **Start scanning the entries from bottom to top.**
2) **On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.**
3) **After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.**

Step-03:

```
while (i > 0 && w>0)
   if (T[i][w] == T[i-1][w]
       Not select the current item
       i--
   else
       select the item i
       i--;
       w= w-wt[i]
```

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving 0/1 Knapsack Problem

Value of T(i, w) is max profit , here T[4][8]= 8 is the maximum profit

| $P_i$ | $w_i$ | i \ w | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

**Item Selected = { 0, 1, 0, 1}**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Time Complexity of 0/1 Knapsack Problem

**Finding Maximum Profit**

```
Knapsack(p,wt,m,n)
    for i=0 to n
        for w=0 to m
            if i==0 || w==0
                T[i][w]=0
            else if wt[i]<=w
                T[i][w]=max(p[i]+T[i-1][w-wt[i]], T[i-1][w])
            else
                T[i][w]=T[i-1][w]
```

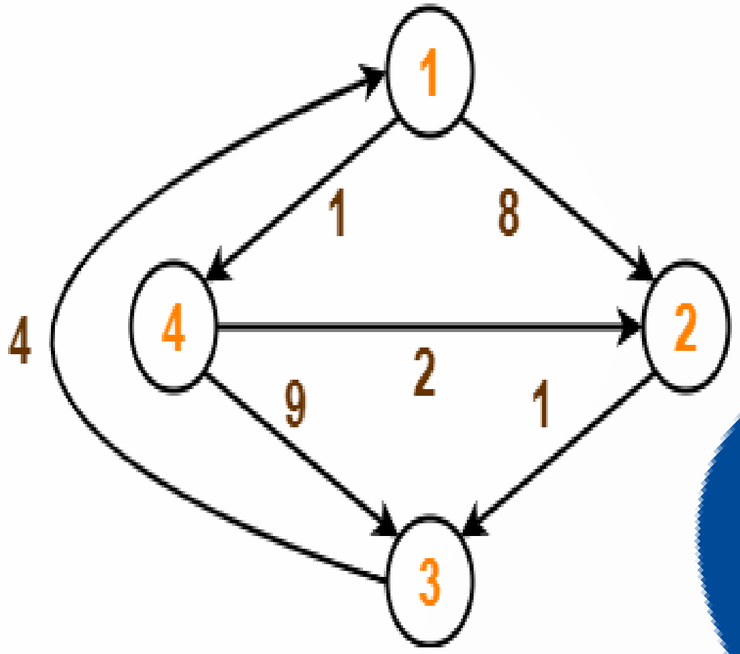
n*m

**Finding List of Items**

```
while (i > 0 && w>0)
    if (T[i][w] == T[i-1][w]
        Not select the current item
        i--
    else
        select the item i
        i--;
        w= w-wt[i]
```

n

Total Time= n*m + n
= **O(nm)**

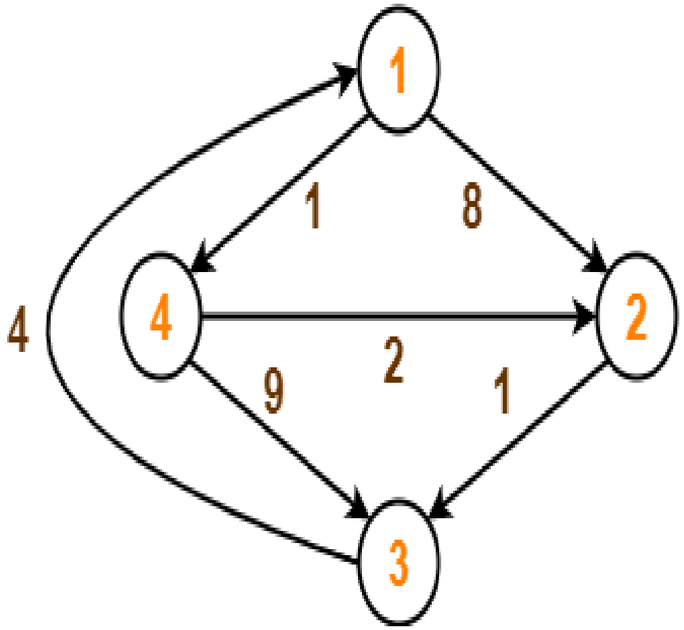



# 03.b

# All Pair Shortest Path Problem

# All Pair Shortest Path Problem

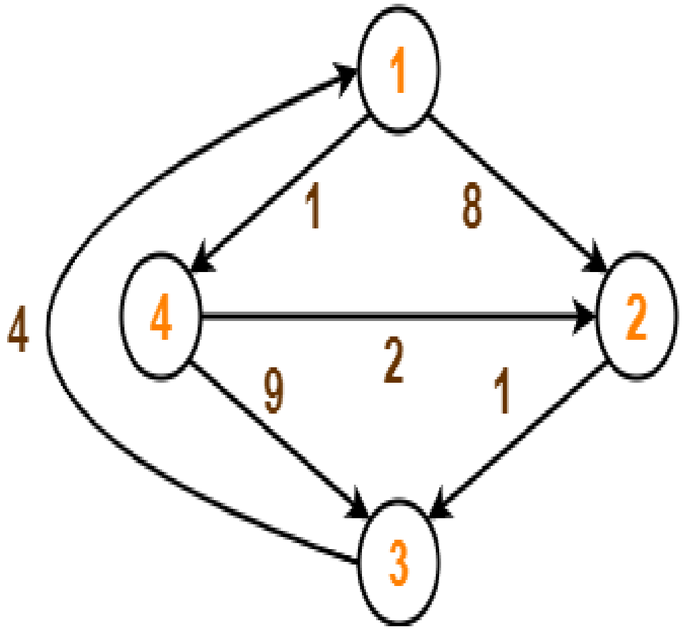
- The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.




# Floyd Warshall Algorithm

Solving All Pair Shortest Path Problem

- The Floyd Warshall Algorithm is used for solving the All Pairs Shortest Path problem.




~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Floyd Warshall Algorithm

Solving All Pair Shortest Path Problem

Step -1 : Remove all the self loops and parallel edges

Step- 2:

a. Write the initial distance matrix ($d_{ij}$). It represents the distance between every pair of vertices in the form of given weights.

b. For diagonal elements (representing self-loops), distance value = 0.

c. For vertices having a direct edge between them, distance value = weight of that edge.

d. For vertices having no direct edge between them, distance value = ∞.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Floyd Warshall Algorithm

Solving All Pair Shortest Path Problem

Step -3 :Then we update the distance matrix by considering all vertices as an intermediate vertex.

(The idea is to one by one pick all vertices and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path)

- if k is an intermediate vertex in shortest path from i to j then we update the value of

  **dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]**
- If k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Floyd Warshall Algorithm

Solving All Pair Shortest Path Problem

Let $d_{ij}^{(k)}$ be the weight of the shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2.....k}.

$$d_{ij}^{(0)} \leftarrow w_{ij} \quad \text{if } k=0$$

$$d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) \text{ if } k>=1$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Pseudo Code of Floyd Warshall Algorithm

Solving All Pair Shortest Path Problem

**FLOYD - WARSHALL (W)**

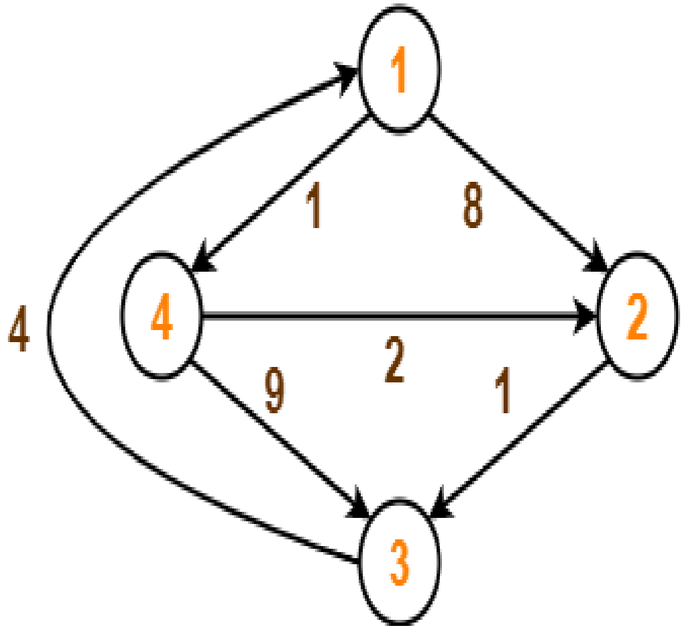
1. n ← rows [W].
2. $D^0 \leftarrow W$
3. for k ← 1 to n
4. do for i ← 1 to n
5. do for j ← 1 to n
6. do $d_{ij}^{(k)} \leftarrow \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. return $D^{(n)}$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving Problem Using Floyd Warshall Algorithm

Step -1 : Remove all the self loops and parallel edges (There is no self loop and parallel edge in this graph)

Step- 2: Create a initial distance matrix (Direct Path)



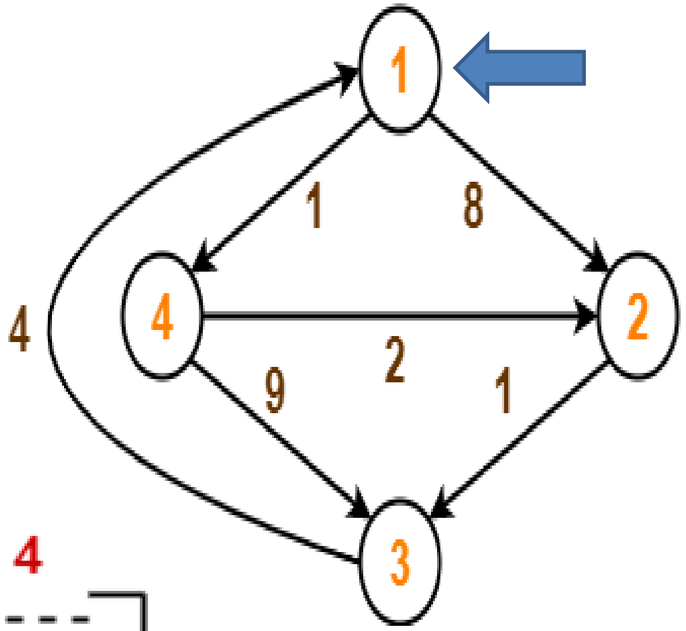

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving Problem Using Floyd Warshall Algorithm

Step -3 :Then we update the distance matrix by considering all vertices as an intermediate vertex.

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$




$d_{ij}^{(k)} \leftarrow \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

$d_{23}^{(1)} \leftarrow \min (d_{23}^{(1-1)}, d_{21}^{(1-1)} + d_{12}^{(1-1)})$

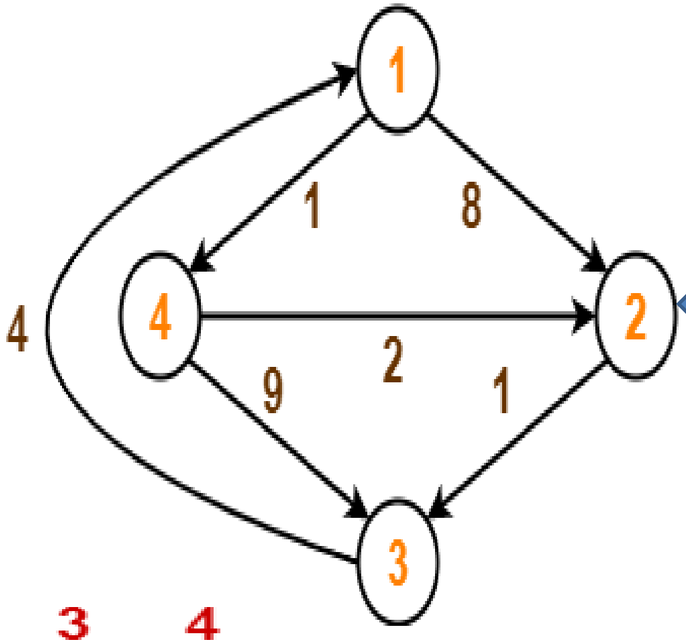
$\min(1, \infty)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving Problem Using Floyd Warshall Algorithm

Step -3 :Then we update the distance matrix by considering all vertices as an intermediate vertex.

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$



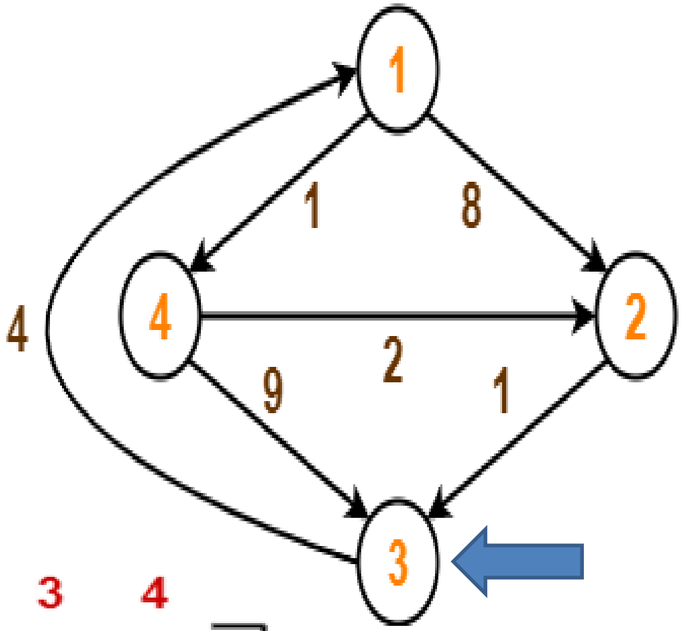

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{array}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving Problem Using Floyd Warshall Algorithm

Step -3 :Then we update the distance matrix by considering all vertices as an intermediate vertex.

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix} \end{array}$$



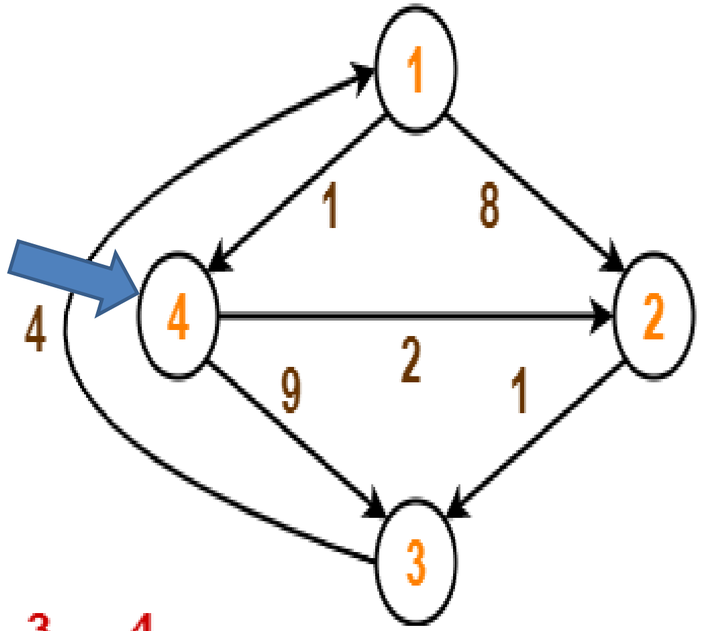

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Solving Problem Using Floyd Warshall Algorithm

Step -3 :Then we update the distance matrix by considering all vertices as an intermediate vertex.

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

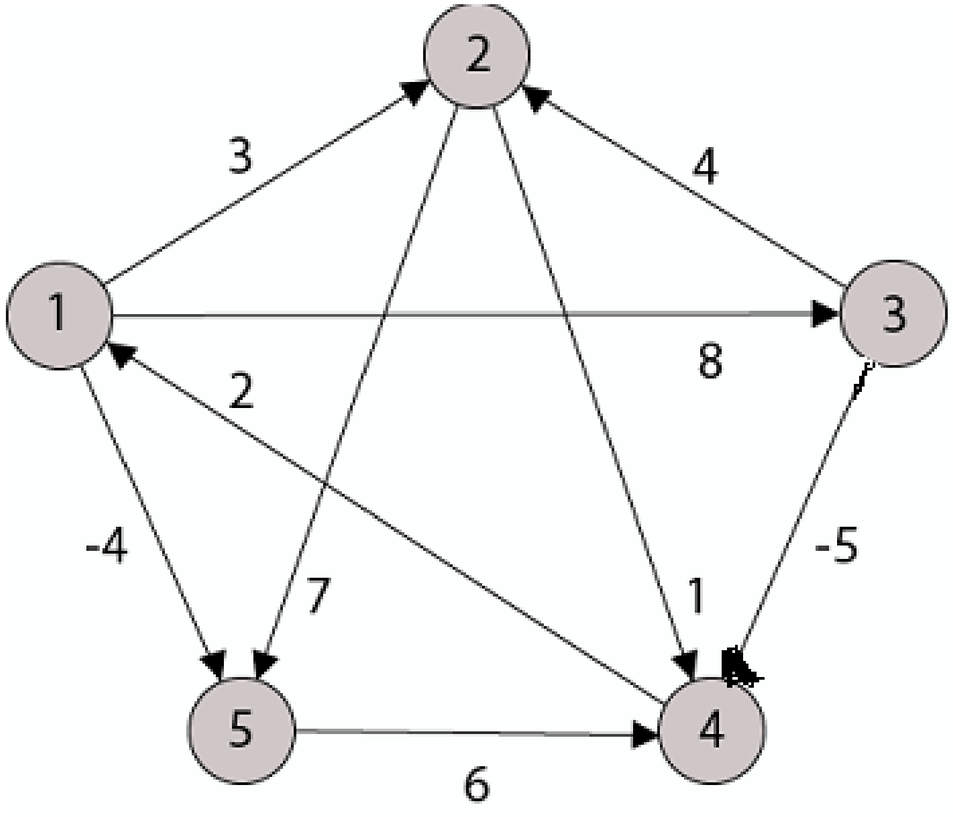



$$D_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \end{array}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Practice Example Using Floyd Warshall Algorithm

Problem: Find All Pairs Shortest Path.

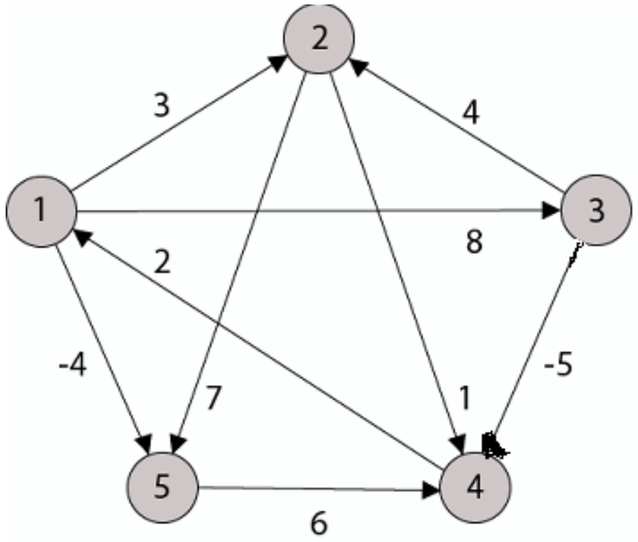



~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Practice Example Using Floyd Warshall Algorithm

Problem: Find All Pairs Shortest Path.

**Step (i)** When k = 0




$$D^{(0)} = \begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & \infty \\ 2 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Practice Example Using Floyd Warshall Algorithm

**Step (ii)** When k =1

$d_{ij}^{(k)} = \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$

$d_{14}^{(1)} = \min (d_{14}^{(0)}, d_{11}^{(0)} + d_{14}^{(0)})$

$d_{14}^{(1)} = \min ( \infty , 0 + \infty ) = \infty$

$d_{15}^{(1)} = \min (d_{15}^{(0)}, d_{11}^{(0)} + d_{15}^{(0)})$

$d_{15}^{(1)} = \min (-4, 0 + -4) = -4$

$d_{21}^{(1)} = \min (d_{21}^{(0)}, d_{21}^{(0)} + d_{11}^{(0)})$

$d_{21}^{(1)} = \min (\infty,\infty + 0) = \infty$

$d_{23}^{(1)} = \min (d_{23}^{(0)}, d_{21}^{(0)} + d_{13}^{(0)})$

$d_{23}^{(1)} = \min ((\infty,\infty + 8) = \infty$

$d_{31}^{(1)} = \min (d_{31}^{(0)}, d_{31}^{(0)} + d_{11}^{(0)})$

$d_{31}^{(1)} = \min (\infty,\infty + 0) = \infty$

$d_{35}^{(1)} = \min (d_{35}^{(0)}, d_{31}^{(0)} + d_{15}^{(0)})$

$d_{35}^{(1)} = \min (\infty,\infty + (-4)) = \infty$

$d_{42}^{(1)} = \min (d_{42}^{(0)}, d_{41}^{(0)} + d_{12}^{(0)})$

$d_{42}^{(1)} = \min (\infty, 2 + 3) = 5$

$d_{43}^{(1)} = \min (d_{43}^{(0)}, d_{41}^{(0)} + d_{13}^{(0)})$

$d_{43}^{(1)} = \min (\infty, 2 + 8) = 10$

$d_{45}^{(1)} = \min (d_{45}^{(0)}, d_{41}^{(0)} + d_{15}^{(0)})$

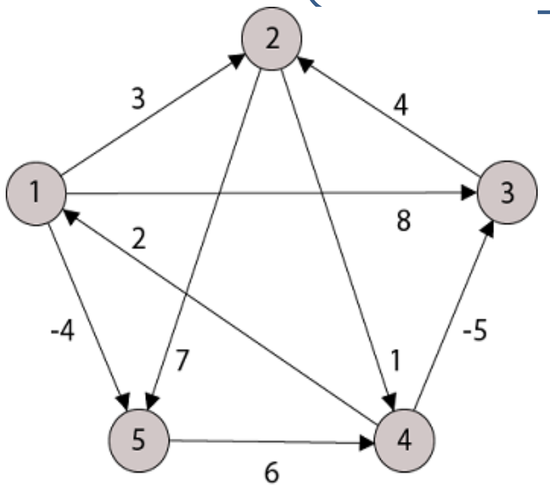
$d_{45}^{(1)} = \min (\infty, 2 + (-4)) = -2$

$d_{51}^{(1)} = \min (d_{51}^{(0)}, d_{51}^{(0)} + d_{11}^{(0)})$

$d_{51}^{(1)} = \min (\infty,\infty + 0) = \infty$

$$D^{(0)} = \begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & \infty \\ 2 & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

$$D_{ij}^{(1)} = \begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & \infty \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$




~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Practice Example Using Floyd Warshall Algorithm

Problem: Find All Pairs Shortest Path.

**Step (iii)** When k = 2

$$D_{ij}^{(2)} = \begin{matrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & 11 \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

**Step (iv)** When k = 3

$$D_{ij}^{(3)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & -5 & 11 \\ 2 & 5 & 10 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Practice Example Using Floyd Warshall Algorithm

Problem: Find All Pairs Shortest Path.

**Step (v)** When k = 4

$$D_{ij}^{(4)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ 3 & 0 & 11 & 1 & -1 \\ -3 & 0 & 0 & -5 & -7 \\ 2 & 5 & 10 & 0 & -2 \\ 8 & 11 & 16 & 6 & 0 \end{matrix}$$

**Step (vi)** When k = 5

$$D_{ij}^{(5)} = \begin{matrix} 0 & 3 & 8 & 3 & -4 \\ 3 & 0 & 11 & 1 & -1 \\ -3 & 0 & 0 & -5 & -7 \\ 2 & 5 & 10 & 0 & -2 \\ 8 & 11 & 16 & 6 & 0 \end{matrix}$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU

# Time Complexity of Floyd Warshall Algorithm

**FLOYD - WARSHALL (W)**

1. n ← rows [W].
2. $D^0 \leftarrow W$
3. for k ← 1 to n
4. do for i ← 1 to n
5. do for j ← 1 to n
6. do $d_{ij}^{(k)} \leftarrow \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. return $D^{(n)}$

**n*n*n**

**Total Time= n*n*n = $O(n^3)$**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU


# Happy Learning!

If you have any doubts, or queries , can be discussed in the C-11, Room 310, AB-1.

or share it on WhatsApp 8586968801

if there is any suggestion or feedback about slides, please write it on gaurav.kumar@gla.ac.in

~Dr Gaurav Kumar, Asst. Prof, CEA, GLAU