CS402 ADA – Unit 3

Dynamic Programming Concept

(RGPV Exam-Oriented Notes | Easy Hinglish)

A. Introduction

Dynamic Programming, short form **DP**, ek algorithm design technique hai jo complex problems ko small subproblems mein divide karke solve karti hai.

DP ka main idea:

"Same calculation baar-baar mat karo, ek baar solve karo aur answer store kar lo."

Real-Life Analogy

Maan lo tum Mathematics ka ek question solve kar rahe ho jisme same formula baar-baar use ho raha hai.

Agar tum har baar formula dobara solve karoge, time waste hoga.

Better way:

Pehli baar answer nikalo, notebook mein likh lo, next time directly use kar lo.

Yahi Dynamic Programming hai.

B. Definition

Exam Definition

Dynamic Programming is an algorithm design technique used to solve optimization problems by dividing them into overlapping subproblems, solving each subproblem only once, and storing their results for future use.

Easy Explanation

DP ka simple meaning:

Bade problem ko chhote problems mein todna, chhote answers save karna, aur unse final answer banana.

C. Core Concept

Dynamic Programming mostly optimization problems mein use hoti hai.

Optimization means:

- Maximum profit
- Minimum cost
- Shortest path
- Best decision
- Minimum number of steps

DP tab use hoti hai jab problem mein ye 2 properties ho:

1. **Optimal Substructure**
2. **Overlapping Subproblems**

D. Dynamic Programming Fundamentals

1. Optimal Substructure

Definition

A problem has optimal substructure if its optimal solution can be constructed from optimal solutions of its subproblems.

Easy Explanation

Final best answer chhote best answers se banta hai.

Example

Shortest path A to C:

A ---- B ---- C

A to C ka shortest path agar A → B → C hai, to A to B aur B to C bhi shortest hone chahiye.

---

2. Overlapping Subproblems

Definition

A problem has overlapping subproblems if the same subproblems are solved again and again.

Easy Explanation

Same calculation baar-baar repeat ho rahi ho.

Example: Fibonacci

F(5)
= F(4) + F(3)

F(4)
= F(3) + F(2)

Yahan **F(3)** baar-baar calculate ho raha hai.

DP isko store kar leta hai.

---

3. Memoization

Definition

Memoization is a top-down DP technique in which results of subproblems are stored and reused later.

Easy Explanation

Recursive solution + storage table = Memoization.

Direction

Top to Bottom

---

4. Tabulation

Definition

Tabulation is a bottom-up DP technique in which solutions of smaller subproblems are calculated first and stored in a table.

Easy Explanation

Sabse chhote problem se start karo, table fill karo, final answer tak jao.

Direction

Bottom to Top

---

5. State Definition

Definition

State represents a subproblem in Dynamic Programming.

Easy Explanation

State batata hai ki current problem kis condition mein solve ho raha hai.

Example:

0/1 Knapsack mein state:
DP[i][w]
Meaning:

First i items use karke capacity w mein maximum profit.

---

6. Recurrence Relation

Definition

Recurrence relation is a formula that expresses a problem in terms of smaller subproblems.

Easy Explanation

DP ka formula jisse table fill hoti hai.

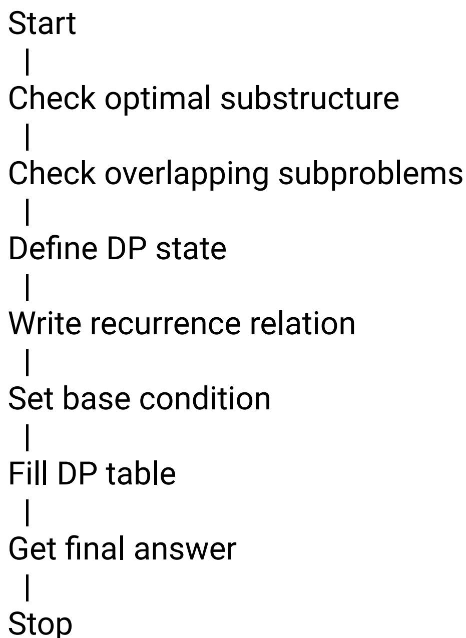
Example Fibonacci:
F(n) = F(n-1) + F(n-2)

E. Working of Dynamic Programming

General Steps

1. Identify problem type.
2. Check optimal substructure.
3. Check overlapping subproblems.
4. Define state.
5. Write recurrence relation.
6. Decide base condition.
7. Fill DP table.
8. Extract final answer.

Flowchart



General Pseudocode

```
DynamicProgramming(problem)

Define DP table

Initialize base cases

for each state:
   compute answer using recurrence relation
   store answer in DP table

return final answer
```

F. Complete Dry Run: Fibonacci Using DP

Problem

Find Fibonacci number:

F(5)

Formula

F(n) = F(n-1) + F(n-2)
Base cases:
F(0) = 0
F(1) = 1

Step-by-Step Table

n	F(n)
0	0
1	1

Now calculate:

For n = 2

F(2) = F(1) + F(0)
= 1 + 0
= 1

For n = 3

F(3) = F(2) + F(1)
= 1 + 1
= 2

For n = 4

F(4) = F(3) + F(2)
= 2 + 1
= 3

For n = 5

F(5) = F(4) + F(3)
= 3 + 2
= 5

Final table:

n	F(n)
0	0
1	1
2	1
3	2
4	3
5	5

Final Answer:
F(5) = 5

G. Complexity Analysis

Without DP Fibonacci

Recursive solution repeats same calls again and again.

Time Complexity:
$O(2^n)$

With DP Fibonacci

Each value is calculated once.

Time Complexity:
$O(n)$
Space Complexity:
$O(n)$

Best Case

$O(n)$

Average Case

$O(n)$

Worst Case

$O(n)$
Because every subproblem is solved exactly once.

H. Advantages

1. Avoids repeated calculation.
2. Reduces time complexity.
3. Gives optimal solution.
4. Useful for optimization problems.
5. Converts exponential solutions into polynomial solutions.

I. Disadvantages

1. Requires extra memory.
2. DP table can become large.
3. State and recurrence relation can be difficult to form.
4. Not suitable for every problem.

J. Applications

Dynamic Programming is used in:

1. 0/1 Knapsack
2. Floyd-Warshall Algorithm
3. Multistage Graph

4. Reliability Design
5. Matrix Chain Multiplication
6. Longest Common Subsequence
7. Shortest Path Problems

K. Common Mistakes

▯ DP aur Greedy ko same samajhna.
▯ DP stores previous results, Greedy immediate best choice leta hai.

▯ Recurrence relation skip karna.
▯ DP answer mein recurrence relation zaroor likho.

▯ Base case nahi likhna.
▯ Base condition DP table ka starting point hota hai.

▯ Table ka final answer galat cell se lena.
▯ Final answer usually last cell / required state se milta hai.

L. Viva Questions

Q1. What is Dynamic Programming?

Dynamic Programming is a technique that solves overlapping subproblems once and stores their results.

Q2. What are the two main properties of DP?

1. Optimal Substructure
2. Overlapping Subproblems

Q3. What is Memoization?

Top-down DP technique using recursion and storage.

Q4. What is Tabulation?

Bottom-up DP technique using table filling.

Q5. Give examples of DP algorithms.

0/1 Knapsack, Floyd-Warshall, Multistage Graph.

M. Exam Keywords

Write these keywords in exam:

- Optimal Substructure
- Overlapping Subproblems
- Memoization
- Tabulation
- State
- Recurrence Relation
- DP Table

- Optimization Problem
- Store and Reuse
- Polynomial Time

N. Memory Tricks

DP Meaning

DP = Don’t Repeat

Same calculation repeat mat karo.

DP Properties

OO

Optimal Substructure
Overlapping Subproblems

DP Steps

S-R-B-T

State
Recurrence
Base Case
Table

O. Comparison Tables

DP vs Greedy

Basis	Dynamic Programming	Greedy
Approach	Solves all subproblems	Chooses local best
Storage	Stores results	Usually no storage
Decision	Considers multiple possibilities	One immediate choice
Optimality	More reliable	Not always optimal
Example	0/1 Knapsack	Fractional Knapsack

DP vs Divide and Conquer

Basis	Dynamic Programming	Divide and Conquer
Subproblems	Overlapping	Independent
Storage	Stores results	Usually no storage
Repetition	Avoids repeated work	May repeat work
Example	Fibonacci DP	Merge Sort
Use	Optimization	Sorting/ searching

P. RGPV Exam Answers

2 Mark Answer

Dynamic Programming is an algorithm design technique that solves overlapping subproblems only once and stores their results for future use.

---

5 Mark Answer

Dynamic Programming is used to solve optimization problems by dividing them into smaller overlapping subproblems. It stores the result of each subproblem to avoid repeated calculation.

The two main properties are:

1. Optimal Substructure
2. Overlapping Subproblems

DP can be implemented using:

- Memoization
- Tabulation

Examples:

- 0/1 Knapsack
- Floyd-Warshall Algorithm
- Multistage Graph

---

7 Mark Answer

Dynamic Programming is an algorithmic technique used for solving optimization problems. It divides the problem into smaller subproblems, solves each subproblem once, and stores the answer in a table.

It works when the problem has:

1. Optimal Substructure
2. Overlapping Subproblems

Important terms:

- Memoization: Top-down approach
- Tabulation: Bottom-up approach
- Recurrence Relation: Formula for solving states
- State: Representation of subproblem

Example:

Fibonacci:

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

Using DP, repeated calculations are avoided.

Applications:

- 0/1 Knapsack
- Floyd-Warshall
- Reliability Design
- Multistage Graph

---

10 Mark Topper Answer

Dynamic Programming is an important algorithm design technique used for optimization problems. It is based on solving smaller subproblems and storing their results so that repeated computation can be avoided.

A problem can be solved using DP if it satisfies two properties:

1. Optimal Substructure

The optimal solution of the original problem can be formed using optimal solutions of subproblems.

2. Overlapping Subproblems

The same subproblems occur repeatedly during computation.

DP has two approaches:

Memoization

It is a top-down approach using recursion and storage.

Tabulation

It is a bottom-up approach using table filling.

General steps:

1. Define state.
2. Write recurrence relation.
3. Initialize base cases.
4. Fill DP table.

5. Obtain final answer.

Example:

Fibonacci series:
F(n) = F(n-1) + F(n-2)
Without DP, repeated calculations occur. With DP, each value is calculated once and stored.

Applications:

- 0/1 Knapsack
- Multistage Graph
- Floyd-Warshall Algorithm
- Reliability Design

Conclusion:

Dynamic Programming is useful for reducing time complexity and solving optimization problems efficiently.

Q. One Page Revision Sheet

Dynamic Programming

Definition:
Solve overlapping subproblems once and store their answers.

Main Properties

1. Optimal Substructure
2. Overlapping Subproblems

Approaches

Memoization

Top-down + Recursion

Tabulation

Bottom-up + Table

Important Terms

State = Subproblem
Recurrence = Formula
Base Case = Starting value
DP Table = Stored answers

General Steps

State
□
Recurrence
□
Base Case
□
Table

Final Answer

Complexity Idea

Without DP:
O(2^n)
With DP:
O(n)

Examples

- 0/1 Knapsack
- Floyd-Warshall
- Multistage Graph
- Reliability Design

Memory Trick

DP = Don't Repeat

0/1 Knapsack Problem

(RGPV Exam-Oriented Complete Notes | Easy Hinglish)

A. Introduction

Suppose tumhare paas ek bag (Knapsack) hai.

Capacity:
W = 50 kg
Aur kuch items hain:

Item	Weight	Profit
A	10	60
B	20	100
C	30	120

Question:

Kaunse items choose karein taki profit maximum ho aur bag ki capacity exceed na ho?

Yahi **0/1 Knapsack Problem** hai.

Why "0/1" ?

Har item ke liye sirf 2 choices hain:
0 = Item mat lo

1 = Item lo
Fraction allowed nahi hai.

Isi liye naam:

0/1 Knapsack

Real-Life Analogy

Exam mein books carry karni hain.

Bag mein limited space hai.

Har book ka:

- Weight alag
- Importance alag

Goal:

Maximum useful books carry karo.

B. Definition

Exam Definition

0/1 Knapsack is a Dynamic Programming optimization problem in which each item can either be selected completely or not selected at all, with the objective of maximizing profit under a given weight capacity.

Easy Explanation

Rule:

▯ Item pura lo

OR

▯ Item bilkul mat lo

▯ Half item allowed nahi.

Difference from Fractional Knapsack

Feature	Fractional	0/1
Division Allowed	Yes	No
Technique	Greedy	Dynamic Programming

Item Selection	Fraction	Full Item Only

C. Core Concept

Keyword 1: Weight

Item ka weight.

Keyword 2: Profit

Item ka value/benefit.

Keyword 3: Capacity

Bag kitna weight carry kar sakta hai.

Keyword 4: State

Most Important

State:
DP[i][w]
Meaning:

First i items use karke capacity w mein maximum profit.

Optimal Substructure

Final optimal solution small optimal solutions se banta hai.

Overlapping Subproblems

Same states baar-baar calculate hote hain.

DP unhe store kar leta hai.

Why Greedy Fails?

Example:

Item	Weight	Profit

A	10	60
B	20	100
C	30	120

Capacity:
50
Greedy hamesha optimal answer nahi deta.

Isliye DP use karte hain.

D. Recurrence Relation

Most Important Formula

If item not selected:
DP[i][w]

= DP[i-1][w]

If item selected:
Profit[i]

+ DP[i-1][w-weight[i]]

Final Formula:
DP[i][w]

=

max(

DP[i-1][w],

Profit[i]

+

DP[i-1][w-weight[i]]

)
□ RGPV mein bahut important.

E. Working

Steps

Step 1

Create DP table.

Step 2

Initialize first row and first column as 0.

Step 3

For every item:

Check:

Take item

OR

Leave item

Step 4

Store maximum value.

Step 5

Last cell gives answer.

Flowchart

```
Start
  |
Create DP Table
  |
For Each Item
  |
Can Item Fit?
 / \
No Yes
 |   |
Copy Max(Take,Not Take)
 |
Fill Table
 |
Last Cell = Answer
 |
Stop
```

F. Complete Dry Run

Example

Capacity:
W = 5
Items:

Item	Weight	Profit
A	1	6
B	2	10
C	3	12

DP Table

Rows = Items

Columns = Capacity

Initial Table

Item\W	0	1	2	3	4	5
0	0	0	0	0	0	0
A						
B						
C						

Fill Row A

Weight = 1

Profit = 6

Item\W	0	1	2	3	4	5
0	0	0	0	0	0	0
A	0	6	6	6	6	6

Fill Row B

Weight = 2

Profit = 10

For Capacity 3:

Take:
10 + 6 = 16
Not Take:
6
Choose:
16
Table:

Item\W	0	1	2	3	4	5
0	0	0	0	0	0	0
A	0	6	6	6	6	6
B	0	6	10	16	16	16

Fill Row C

Weight = 3

Profit = 12

For Capacity 5:

Take:
12 + 10

= 22
Not Take:
16
Choose:
22
Final Table

Item\W	0	1	2	3	4	5
0	0	0	0	0	0	0
A	0	6	6	6	6	6
B	0	6	10	16	16	16
C	0	6	10	16	18	22

Final Answer

Last Cell:
DP[3][5]

= 22
Maximum Profit:
22

G. Algorithm

Pseudocode

```
Knapsack(W,wt,profit,n)

Create DP table

for i=0 to n

  for w=0 to W

    if i==0 or w==0

      DP[i][w]=0

    else if wt[i] <= w

      DP[i][w]

      = max(

        profit[i]

        + DP[i-1][w-wt[i]],

        DP[i-1][w]

      )

    else

      DP[i][w]

      = DP[i-1][w]

return DP[n][W]
```

H. Complexity Analysis

Table Size:
n × W

Time Complexity

O(nW)

Space Complexity

O(nW)

Best Case

O(nW)

Average Case

O(nW)

Worst Case

O(nW)

Why O(nW)?

Because every cell of DP table is filled exactly once.

I. Advantages

1

Always gives optimal solution.

2

Avoids repeated calculations.

3

Efficient compared to brute force.

4

Dynamic Programming based.

J. Disadvantages

1

Requires extra memory.

2

Large DP table for large capacity.

3

Implementation slightly complex.

K. Applications

Resource Allocation

Budget planning.

Cargo Loading

Container optimization.

Investment Planning

Maximum returns.

Project Selection

Maximum profit projects.

L. Common Mistakes

▯ Confusing with Fractional Knapsack.

▯ No fractions allowed.

▯ Forgetting recurrence relation.

▯ Must write formula.

▯ Taking greedy approach.

▯ Use DP table.

M. Viva Questions

What is 0/1 Knapsack?

DP optimization problem.

Why called 0/1?

Item either selected or rejected.

Technique used?

Dynamic Programming.

Complexity?

O(nW)

Is item division allowed?

No.

N. Exam Keywords

Write these keywords:

- Dynamic Programming
- Optimization Problem
- Profit Maximization
- Weight Constraint
- DP Table
- State
- Recurrence Relation
- Optimal Substructure
- Overlapping Subproblems

O. Memory Tricks

Rule

Take

OR

Not Take

Mnemonic:
TNT
Take

Not Take

Choose Maximum

P. Comparison Table

0/1 Knapsack vs Fractional Knapsack

Feature	0/1	Fractional
Technique	DP	Greedy
Fraction Allowed	No	Yes
Optimal by Greedy	No	Yes
Complexity	O(nW)	O(n log n)
Item Selection	Full/ Reject	Fraction Possible

Q. RGPV Exam Answers

2 Mark Answer

0/1 Knapsack is a Dynamic Programming problem in which each item is either selected completely or rejected completely to maximize profit under a weight constraint.

---

5 Mark Answer

0/1 Knapsack is an optimization problem solved using Dynamic Programming.

Features:

- No item division
- Profit maximization
- Weight constraint

Recurrence:
DP[i][w]

=

max(

DP[i-1][w],

profit[i]

+

DP[i-1][w-wt[i]]

)
Complexity:

O(nW)

---

7 Mark Answer

Explain 0/1 Knapsack Problem

0/1 Knapsack is a Dynamic Programming optimization problem.

Each item:

- Selected completely
  OR
- Rejected completely

State:
DP[i][w]
Recurrence:
DP[i][w]

=

max(

DP[i-1][w],

profit[i]

+

DP[i-1][w-wt[i]]

)
Complexity:

O(nW)

Applications:

- Resource Allocation
- Budget Planning

---

10 Mark Topper Answer

0/1 Knapsack Problem

Definition

0/1 Knapsack is a Dynamic Programming optimization problem used to maximize profit within a given capacity.

Properties

1. Optimal Substructure
2. Overlapping Subproblems

State

DP[i][w]

Recurrence Relation

DP[i][w]

=

max(

DP[i-1][w],

profit[i]

+

DP[i-1][w-wt[i]]

)

Algorithm

1. Create DP table.
2. Initialize base cases.
3. Fill table.
4. Extract answer from last cell.

Complexity

O(nW)

Applications

- Resource Allocation
- Investment Planning
- Cargo Loading

Conclusion

0/1 Knapsack is one of the most important Dynamic Programming problems and guarantees an optimal solution.

One Page Revision Sheet

0/1 Knapsack

Goal:

Maximum Profit

Technique

Dynamic Programming

State

DP[i][w]

Recurrence

DP[i][w]

=

max(

DP[i-1][w],

profit[i]

+
DP[i-1][w-wt[i]]

)

Rule

Take

OR

Not Take

Choose Maximum

Complexity

O(nW)

Properties

1. Optimal Substructure
2. Overlapping Subproblems

Memory Trick

TNT

Take

Not Take

Choose Maximum

Most Important Question

▯ **Explain 0/1 Knapsack Problem using Dynamic Programming with suitable example and complexity analysis.**

Multistage Graph

(RGPV Exam-Oriented Complete Notes | Easy Hinglish)

A. Introduction

Multistage Graph Dynamic Programming ka ek important application hai.

Iska objective:

▯ Source vertex se Destination vertex tak **minimum cost path** find karna.

Real-Life Analogy

Suppose tum Bhopal se Delhi jana chahte ho.

Direct route nahi hai.

Tumhe different stages se jana padega:
Bhopal

▯

Gwalior

▯

Agra

▯

Delhi
Har route ki cost alag hai.

Question:

Kaunsa path choose kare taki total cost minimum ho?

Yehi Multistage Graph Problem hai.

B. Definition

Exam Definition

A Multistage Graph is a directed weighted graph whose vertices are divided into stages and edges are allowed only from one stage to the next stage.

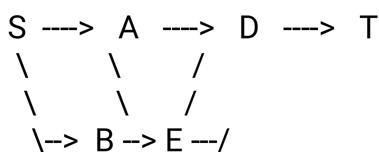
Easy Explanation

Multistage Graph mein:

- Graph stages mein divide hota hai.
- Edge sirf next stage mein jati hai.
- Goal = Minimum Cost Path.

Structure of Multistage Graph

Stage 1 Stage 2 Stage 3 Stage 4



Where:

S = Source

T = Destination

C. Core Concept

Keyword 1: Stage

Graph ke vertices ka group.

Example:

Stage-1 = Source

Stage-2 = Intermediate Nodes

Stage-3 = Intermediate Nodes

Stage-4 = Destination

Keyword 2: Source

Starting node.

Keyword 3: Destination

Final node.

Keyword 4: Cost

Edge weight.

Optimal Substructure

Agar destination tak minimum path chahiye,

to har intermediate node se destination tak bhi minimum path hona chahiye.

Overlapping Subproblems

Same node se destination tak ka minimum cost repeatedly calculate ho sakta hai.

DP us cost ko store kar leta hai.

Why Dynamic Programming Works?

Instead of:
Checking every possible path
DP:
Stores minimum cost of each node
and reuses it.

D. Working

DP Idea

Start from destination.

Move backward.

Calculate minimum cost for every node.

Formula

Most Important Formula

For any node i:
Cost(i)

=

min

[

Cost(i,j)

+

Cost(j)

]
Where:
Cost(i,j)
=
edge cost

Cost(j)
=
minimum cost from j to destination

Step-by-Step Procedure

Step 1

Assign destination cost = 0

Step 2

Move backward stage-wise.

Step 3

For every node:

Calculate all possible path costs.

Step 4

Choose minimum.

Step 5

Store result.

Step 6

Continue until source node.

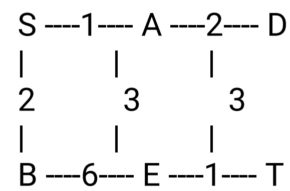
Flowchart

```
Start
  |
Destination Cost = 0
  |
Move Backward
  |
Calculate Cost
  |
Choose Minimum
  |
Store Result
  |
Source Reached?
 / \
No Yes
|   |
Repeat Stop
```

E. Complete Dry Run

Example Graph

Stage1 Stage2 Stage3 Stage4



Edges

Edge	Cost
S-A	1
S-B	2
A-D	2
A-E	3
B-E	6
D-T	3
E-T	1

Step 1

Destination
Cost(T)=0

Step 2

Node D
Cost(D)

=

3 + 0

= 3

Node E
Cost(E)

=

1 + 0

= 1

Step 3

Node A

Path 1:
A → D → T

2 + 3

= 5
Path 2:
A → E → T

3 + 1

= 4
Choose minimum:
Cost(A)=4

Step 4

Node B
B → E → T

6 + 1

= 7
Therefore:
Cost(B)=7

Step 5

Source S

Path 1:
S → A → E → T

1 + 4

= 5
Path 2:
S → B → E → T

2 + 7

= 9
Choose minimum:
Cost(S)=5

Final Answer

Minimum Cost:
5
Optimal Path:
S → A → E → T

DP Cost Table

Node	Cost
T	0
D	3
E	1
A	4
B	7
S	5

Path Reconstruction

Stored decision:
S → A

A → E

E → T
Hence:
S → A → E → T

F. Algorithm

Pseudocode

```
MultistageGraph()

Cost[n] = 0

for i=n-1 to 1

  Cost[i] = infinity

  for each edge(i,j)

     Cost[i]

     = min(

       Cost[i],

       EdgeCost(i,j)

       + Cost[j]

     )

return Cost[1]
```

G. Complexity Analysis

Suppose:
V = vertices

E = edges

Time Complexity

Each edge processed once.

O(E)

Space Complexity

Cost array:

O(V)

Best Case

O(E)

Average Case

O(E)

Worst Case

O(E)

H. Advantages

1

Efficient shortest path computation.

2

Uses Dynamic Programming.

3

Avoids repeated calculations.

4

Optimal solution guaranteed.

I. Disadvantages

1

Works only for multistage graphs.

2

Requires stage structure.

3

Not suitable for arbitrary graphs.

J. Applications

Network Routing

Project Planning

Resource Allocation

Transportation Systems

Decision Making Problems

K. Common Mistakes

□ Starting from source.

□ Start from destination.

□ Forgetting backward calculation.

□ DP works backward.

□ Not storing costs.

□ Store intermediate results.

L. Viva Questions

What is Multistage Graph?

A graph divided into stages.

Which technique is used?

Dynamic Programming.

Objective?

Minimum Cost Path.

From where do we start calculations?

Destination Node.

Complexity?

O(E)

M. Exam Keywords

Write these keywords:

- Dynamic Programming
- Multistage Graph
- Source Vertex
- Destination Vertex
- Minimum Cost Path
- Stage-wise Computation
- Backward Calculation
- Cost Table
- Optimal Substructure
- Path Reconstruction

N. Memory Trick

Multistage Graph Rule

Destination

▯

Backward

▯

Minimum Cost

▯

Source

Mnemonic:

DBMS

Destination

Backward

Minimum Cost

Source

O. Comparison Table

Multistage Graph vs Dijkstra

Feature	Multistage Graph	Dijkstra
Technique	DP	Greedy
Graph Type	Multistage	General Weighted Graph
Direction	Backward	Forward
Goal	Minimum Cost Path	Shortest Path
Complexity	O(E)	O(E log V)

P. RGPV Exam Answers

2 Mark Answer

A Multistage Graph is a directed weighted graph divided into stages where edges exist only between consecutive stages. It is solved using Dynamic Programming to find the minimum cost path.

---

5 Mark Answer

Multistage Graph is a Dynamic Programming problem used to find the minimum cost path from source to destination.

Steps:

1. Set destination cost = 0.
2. Move backward.
3. Compute minimum cost.
4. Store results.
5. Find source cost.

Complexity:

O(E)

---

7 Mark Answer

Explain Multistage Graph

A Multistage Graph is a graph whose vertices are divided into stages.

Dynamic Programming computes minimum cost from destination to source.

Formula:

Cost(i)

=

min

[

Cost(i,j)

+

Cost(j)

]

Applications:

- Routing
- Planning
- Optimization

Complexity:

O(E)

---

10 Mark Topper Answer

Multistage Graph

Definition

A Multistage Graph is a directed weighted graph divided into stages and solved using Dynamic Programming.

Properties

- Source Node
- Destination Node
- Stage-wise Vertices
- Directed Edges

Algorithm

1. Destination cost = 0.
2. Move backward.

3. Compute minimum cost.
4. Store costs.
5. Reconstruct path.

Formula

Cost(i)

=

min

[

Cost(i,j)

+

Cost(j)

]

Complexity

Time:

O(E)

Space:

O(V)

Applications

- Routing
- Project Scheduling
- Optimization Problems

Conclusion

Multistage Graph efficiently finds minimum cost paths using Dynamic Programming.

One Page Revision Sheet

Multistage Graph

Goal:

Minimum Cost Path

Technique

Dynamic Programming

Rule

Start From Destination

Move Backward

Formula

Cost(i)

=

min

[

Cost(i,j)

+

Cost(j)

]

Steps

Destination Cost = 0

□

Backward Calculation

□

Store Costs

□

Source Cost

Complexity

Time:

O(E)

Space:

O(V)

Memory Trick

DBMS

Destination

Backward

Minimum Cost

Source

Most Important Question

� **Explain Multistage Graph using Dynamic Programming with suitable example, cost table, path reconstruction, and complexity analysis.**

Floyd-Warshall Algorithm

(RGPV Exam-Oriented Notes | Easy Hinglish)

A. Introduction

Floyd-Warshall Algorithm ek Dynamic Programming algorithm hai jo graph ke **har vertex se har vertex tak shortest path** find karta hai.

Simple words:

Dijkstra ek source se sab tak shortest path nikalta hai.
Floyd-Warshall sabhi vertices ke beech shortest path nikalta hai.

B. Definition

Exam Definition

Floyd-Warshall Algorithm is a Dynamic Programming algorithm used to find shortest paths between all pairs of vertices in a weighted graph.

Easy Explanation

Agar graph mein A, B, C, D vertices hain, to Floyd-Warshall find karega:

- A se B
- A se C
- B se D
- C se A
  sabhi shortest paths.

C. Core Concept

Problem Type

All Pair Shortest Path Problem

Matlab:
Every vertex � Every other vertex

Important Condition

� Positive weights allowed
� Negative weights allowed
� Negative cycle allowed nahi hota

D. Dynamic Programming Idea

Floyd-Warshall ka main idea:

Agar i se j tak directly jane se zyada cost aa rahi hai, aur i → k → j se kam cost aa rahi hai, to path update kar do.

Most Important Formula

D[i][j] = min(D[i][j], D[i][k] + D[k][j])

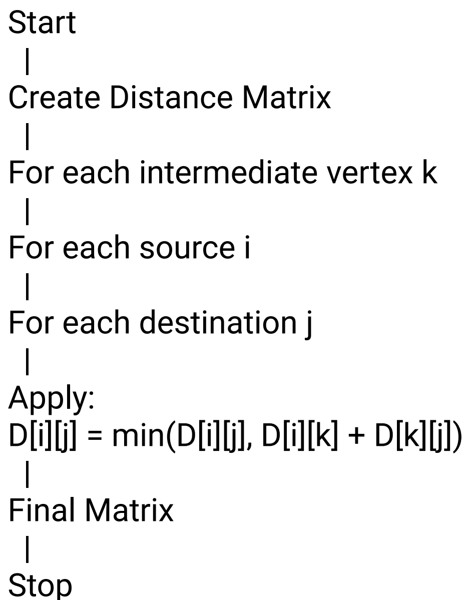
Meaning:

- `D[i][j]` = current shortest distance from i to j
- `D[i][k]` = distance from i to intermediate vertex k
- `D[k][j]` = distance from k to j

E. Working Procedure

1. Create distance matrix.
2. Put `0` on diagonal.
3. Put edge weights where direct edge exists.
4. Put `∞` where no direct edge exists.
5. Pick each vertex as intermediate vertex.
6. Update matrix using formula.
7. Final matrix gives all-pair shortest paths.

Flowchart



F. Pseudocode

FloydWarshall(D, n)

for k = 1 to n:

```
  for i = 1 to n:
     for j = 1 to n:

        if D[i][j] > D[i][k] + D[k][j]:

           D[i][j] = D[i][k] + D[k][j]

return D
```

G. Complete Dry Run

Example Graph

```
A ----3---- B
|           |
7           2
|           |
C ----1---- D
```

Extra edge:

A ----8---- D

Edges:

Edge	Weight
A B	3
A C	7
A D	8
B D	2
C D	1

Initial Matrix D

From/To	A	B	C	D
A	0	3	7	8
B	∞	0	∞	2
C	∞	∞	0	1
D	∞	∞	∞	0

Formula

D[i][j] = min(D[i][j], D[i][k] + D[k][j])

---

k = A as intermediate

A se kisi aur vertex ka shortcut useful nahi hai because A tak baaki vertices se path ∞ hai.

Matrix same rahegi.

---

k = B as intermediate

Check A → D:

Current:
A → D = 8
Via B:
A → B + B → D = 3 + 2 = 5
Update:
A → D = 5
Matrix:

From/To	A	B	C	D
A	0	3	7	5
B	∞	0	∞	2
C	∞	∞	0	1
D	∞	∞	∞	0

---

k = C as intermediate

Check A → D:

Current:
A → D = 5
Via C:
A → C + C → D = 7 + 1 = 8
5 is smaller, so no update.

Matrix same.

---

k = D as intermediate

D se aage koi outgoing useful edge nahi hai.

Final Matrix same.

Final Shortest Path Matrix

From/To	A	B	C	D
A	0	3	7	5
B	∞	0	∞	2
C	∞	∞	0	1
D	∞	∞	∞	0

Final Result

Shortest path:

A D = 5

Path:

A B D

H. Complexity Analysis

Floyd-Warshall mein 3 nested loops hote hain:

```
for k
  for i
    for j
```

Time Complexity

$O(n^3)$

Space Complexity

$O(n^2)$

Because distance matrix store karni padti hai.

Best / Average / Worst Case

All cases:

$O(n^3)$

Reason:

Loops hamesha complete run karte hain.

I. Advantages

1. Simple algorithm.
2. All-pair shortest path find karta hai.
3. Negative edge weights handle kar sakta hai.
4. Easy matrix-based implementation.
5. Transitive closure problems mein useful.

J. Disadvantages

1. Time complexity $O(n^3)$, large graphs ke liye slow.
2. Space complexity $O(n^2)$.
3. Negative cycle handle nahi karta.
4. Sparse graph ke liye inefficient ho sakta hai.

K. Applications

- Routing algorithms
- Network optimization
- Road network shortest path
- Airline route planning
- Graph reachability
- City-to-city shortest distance calculation

L. Common Mistakes

▯ Dijkstra aur Floyd-Warshall same likhna.
▯ Dijkstra = single source, Floyd = all pairs.

▯ Negative edge allowed nahi likhna.
▯ Negative edge allowed hai, but negative cycle nahi.

▯ Formula bhool jana.
▯ Formula sabse important hai:

D[i][j] = min(D[i][j], D[i][k] + D[k][j])

M. Viva Questions

Q1. What is Floyd-Warshall Algorithm?

It is a DP algorithm for all-pair shortest path.

Q2. Time complexity?

$O(n^3)$

Q3. Space complexity?

$O(n^2)$

Q4. Can it handle negative edges?

Yes, but not negative cycles.

Q5. Which technique is used?

Dynamic Programming.

N. Exam Keywords

Write these in exam:

- Dynamic Programming
- All Pair Shortest Path
- Distance Matrix
- Intermediate Vertex
- Weighted Graph
- Negative Edge
- No Negative Cycle
- Matrix Updating
- $O(n^3)$

O. Memory Trick

Formula Trick

Direct vs Via

Direct path:

D[i][j]

Via path:

D[i][k] + D[k][j]

Choose minimum.

Mnemonic:

DVM

Direct
Via
Minimum

P. Comparison Table

Floyd-Warshall vs Dijkstra

Feature	Floyd-Warshall	Dijkstra
Problem	All Pair Shortest Path	Single Source Shortest Path
Technique	Dynamic Programming	Greedy
Negative Edge	Allowed	Not allowed

Negative Cycle	Not allowed	Not allowed
Complexity	$O(n^3)$	$O(E \log V)$
Representation	Matrix	Graph/ List

Q. RGPV Exam Answers

2 Mark Answer

Floyd-Warshall Algorithm is a Dynamic Programming algorithm used to find shortest paths between all pairs of vertices in a weighted graph.

---

5 Mark Answer

Floyd-Warshall Algorithm solves the all-pair shortest path problem using Dynamic Programming. It updates the distance matrix by checking whether a path through an intermediate vertex gives a shorter distance.

Formula:

$D[i][j] = \min(D[i][j], D[i][k] + D[k][j])$

Time complexity is $O(n^3)$.

---

7 Mark Answer

Floyd-Warshall Algorithm is a Dynamic Programming algorithm used for finding shortest paths between every pair of vertices.

Steps:

1. Create initial distance matrix.
2. Put 0 on diagonal.
3. Put edge weights for direct edges.
4. Put ∞ where no edge exists.
5. Use each vertex as intermediate vertex.
6. Update matrix using:

$D[i][j] = \min(D[i][j], D[i][k] + D[k][j])$

Complexity:

$O(n^3)$

Conclusion:

It is useful for all-pair shortest path problems.

---

10 Mark Topper Answer

Floyd-Warshall Algorithm

Definition:
Floyd-Warshall Algorithm is a Dynamic Programming based algorithm used to find shortest paths between all pairs of vertices in a weighted graph.

Concept:
It checks whether a path from vertex i to vertex j through an intermediate vertex k is shorter than the direct path.

Formula:

D[i][j] = min(D[i][j], D[i][k] + D[k][j])

Algorithm:

```
for k = 1 to n:
    for i = 1 to n:
        for j = 1 to n:
            D[i][j] = min(D[i][j], D[i][k] + D[k][j])
```

Complexity:

- Time: $O(n^3)$
- Space: $O(n^2)$

Applications:

- Routing
- Network optimization
- Road distance calculation
- Graph reachability

Conclusion:
Floyd-Warshall is a simple and powerful DP algorithm for all-pair shortest path problems.

R. One Page Revision Sheet

Floyd-Warshall

Goal: All Pair Shortest Path

Technique: Dynamic Programming

Formula:

D[i][j] = min(D[i][j], D[i][k] + D[k][j])

Meaning:

Direct path vs path via k

Initialization:

- Diagonal = 0
- Edge exists = weight
- No edge = ∞

Complexity:

Time = $O(n^3)$
Space = $O(n^2)$

Allowed:

� Negative edges

Not Allowed:

 Negative cycles

Memory Trick:

DVM = Direct, Via, Minimum

Most Important Question:

 Explain Floyd-Warshall Algorithm with example and complexity analysis.

Reliability Design

(RGPV Exam-Oriented Notes | Easy Hinglish)

A. Introduction

Reliability Design Dynamic Programming ka ek important application hai.

Iska objective:

 System ki reliability (trustworthiness) ko maximize karna.

 Cost budget ke andar rehna.

Real-Life Analogy

Suppose ek company server system bana rahi hai.

Har component ki reliability alag hai.

Component	Reliability
Component A	0.90
Component B	0.95

Agar system fail ho gaya to company ko loss hoga.

Question:

Limited budget mein maximum reliable system kaise design karein?

Yahi Reliability Design Problem hai.

B. Definition

Exam Definition

Reliability Design is a Dynamic Programming optimization problem in which system reliability is maximized under a given cost constraint.

Easy Explanation

Goal:
Maximum Reliability

+

Minimum Cost
System ko itna reliable banao ki failure ki probability minimum ho.

Why Needed?

Real-world systems:

- Computer Networks
- Power Systems
- Aircraft Systems
- Communication Systems

mein failure bahut costly hota hai.

C. Core Concept

Keyword 1: Reliability

Probability that a component works correctly.

Example:
Reliability = 0.95
Meaning:

95% chance component sahi kaam karega.

Keyword 2: Failure Probability

Failure

=

1 - Reliability
Example:

$1-0.95=0.05$

Keyword 3: Redundancy

Extra backup component add karna.

Example:
Main Server

+

Backup Server

Keyword 4: Cost

Extra reliability increase karne ke liye additional cost lagti hai.

Dynamic Programming Idea

DP use hoti hai:

- Best reliability store karne ke liye
- Repeated calculations avoid karne ke liye
- Budget constraint satisfy karne ke liye

Optimal Substructure

Best overall design small optimal component designs se banta hai.

Overlapping Subproblems

Same cost combinations repeatedly calculate hote hain.

DP unhe store kar leta hai.

D. Reliability Formula

Single Component

Reliability:
R

Failure Probability

1-R

Parallel Redundant Components

Most Important Formula

If two identical components:

Reliability:
$R = 1-(1-R_1)(1-R_2)$
Example:
If:
$R_1 = 0.9$

$R_2 = 0.9$
Then:
R

=

1-(0.1×0.1)

=

0.99
Reliability increased.

Why Redundancy Helps?

If one component fails,

backup component may still work.

Hence reliability increases.

E. Working

General Procedure

Step 1

Identify components.

Step 2

Find reliability and cost of each component.

Step 3

Consider redundancy options.

Step 4

Calculate reliability.

Step 5

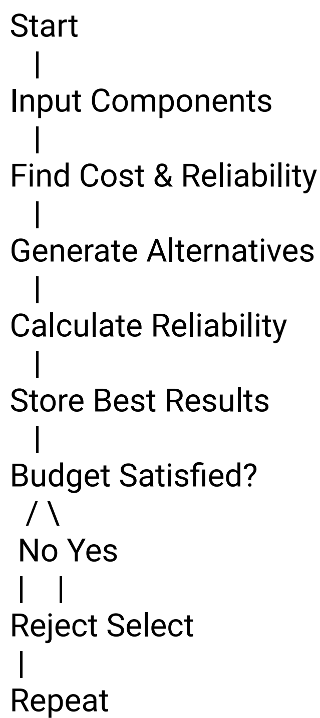
Use DP table.

Step 6

Choose design with:
Maximum Reliability

within Budget

Flowchart



F. Complete Dry Run

Example

Budget:
10 Units

Component Options

Copies	Cost	Reliability
1	2	0.80
2	4	0.96
3	6	0.992

Suppose:

Total Budget:
10

DP Table Concept

Cost	Best Reliability
2	0.80
4	0.96
6	0.992
8	Better Combination
10	Maximum Reliability

Choose combination giving highest reliability under budget.

Reliability Calculation Example

Single component:
R=0.8

Two parallel components:
R

=

1-(1-0.8)^2
=1-(0.2)^2
=1-0.04
=0.96

Three parallel components:
R

=

1-(0.2)^3
=0.992

Reliability increased significantly.

G. Dynamic Programming State

State:
DP[i][b]
Meaning:

Best reliability using first i components within budget b.

Recurrence Idea

Either:
Take component
OR
Do not take component
Store best reliability.

H. Algorithm (Conceptual)

Pseudocode

```
ReliabilityDesign()

Initialize DP table

for each component

  for each budget

    compute reliability

    update DP table

return maximum reliability
```

I. Complexity Analysis

Suppose:
n = components

B = budget

Time Complexity

O(nB)

Space Complexity

O(nB)

Best Case

O(nB)

Average Case

O(nB)

Worst Case

O(nB)O(nB)O(nB)

J. Advantages

1

Produces optimal design.

2

Maximizes reliability.

3

Handles budget constraints.

4

Avoids repeated calculations.

K. Disadvantages

1

Requires extra memory.

2

Complex calculations.

3

Large DP table for large budget.

L. Applications

Computer Networks

Reliable communication.

Data Centers

Backup systems.

Aircraft Systems

Safety systems.

Power Systems

Reliable power supply.

Space Missions

Critical hardware design.

M. Common Mistakes

□ Confusing reliability with probability of failure.

□ Reliability = Working probability.

□ Forgetting redundancy formula.

□ Must write:
R = 1-(1-R□)(1-R□)

□ Ignoring budget constraint.

□ Reliability must be optimized within budget.

N. Viva Questions

What is Reliability Design?

A DP optimization problem.

Objective?

Maximum reliability.

Why redundancy is used?

To increase reliability.

Technique used?

Dynamic Programming.

State representation?

DP[i][b]

O. Exam Keywords

Write these keywords:

- Reliability
- Failure Probability
- Redundancy
- Dynamic Programming
- Cost Constraint
- Optimization
- Backup Components
- Reliability Maximization
- DP Table
- Budget Allocation

P. Memory Trick

Reliability Rule

More Backup

□

More Reliability

Mnemonic:

RBC

Reliability

Backup

Cost

Q. Comparison Table

Reliability Design vs 0/1 Knapsack

Feature	Reliability Design	0/1 Knapsack
Goal	Max Reliability	Max Profit
Constraint	Budget	Capacity
Technique	DP	DP
State	DP[i][b]	DP[i][w]
Optimization	Reliability	Profit

R. RGPV Exam Answers

2 Mark Answer

Reliability Design is a Dynamic Programming optimization problem used to maximize system reliability under a given budget constraint.

---

5 Mark Answer

Reliability Design improves system reliability by selecting optimal components and redundancy under a limited budget.

Features:

- Dynamic Programming
- Reliability Maximization
- Cost Constraint

Reliability Formula:

$R = 1-(1-R_1)(1-R_2)$

---

7 Mark Answer

Explain Reliability Design

Reliability Design is a Dynamic Programming problem used to maximize reliability while satisfying budget constraints.

Concepts:

- Reliability
- Failure Probability
- Redundancy

Formula:

R = 1-(1-R[illegible])(1-R[illegible])

State:

DP[i][b]

Applications:

- Networks
- Data Centers
- Aircraft Systems

---

10 Mark Topper Answer

Reliability Design

Definition

Reliability Design is a Dynamic Programming optimization problem used to maximize system reliability under a specified cost constraint.

Concept

Reliability can be improved using redundant components.

Formula

For parallel components:

R = 1-(1-R[illegible])(1-R[illegible])

DP State

DP[i][b]

Algorithm

1. Define components.
2. Calculate reliability.
3. Create DP table.
4. Store best reliability.
5. Select optimal design.

Complexity

Time:

O(nB)

Space:

O(nB)

Applications

- Computer Networks

- Aircraft Systems
- Power Systems

Conclusion

Reliability Design helps create highly reliable systems using Dynamic Programming while keeping costs under control.

One Page Revision Sheet

Reliability Design

Goal:

Maximum Reliability

Technique

Dynamic Programming

Formula

R = 1-(1-R□)(1-R□)

State

DP[i][b]

Concepts

Reliability

□

Redundancy

□

Budget

□

Optimization

Complexity

Time:

O(nB)

Space:

O(nB)

Memory Trick

RBC

Reliability

Backup

Cost

Most Important Question

� Explain Reliability Design using Dynamic Programming with reliability formula, DP state, and complexity analysis.