Module - 4

Dynamic programming:

- It was designed for optimizing multistage decision p

Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions.

- Dynamic programming is mainly an optimization over plain recursion.
- Dynamic programming relies on a principle of optimality. This principle states that in an optimal sequence of decisions or choice, each subsequence must also be optimal.
- Dynamic programming is used where we have problems, which can be divided into similar subproblems, so that their results can be re-used.

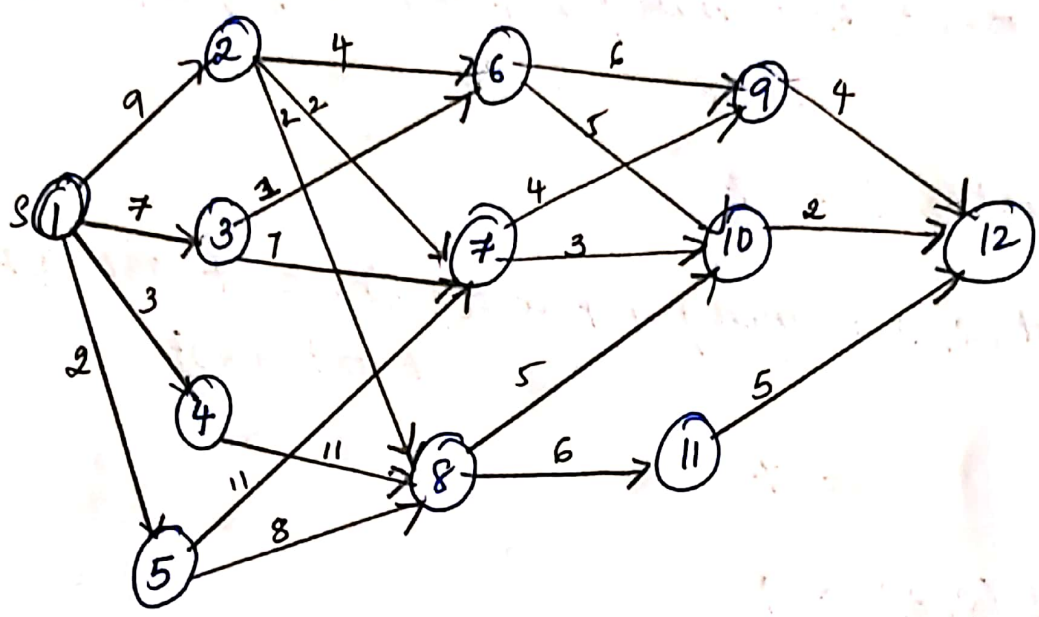
Ex: - optimal merge pattern

An optimal merge pattern tells which pair of files should be merged at each step. As a decision sequence, problem is to decide which pair of files should be merged first, which pair second and so on.

An optimal sequence of decisions is a least - cost sequence.

Multistage Graph

- It is a directed weighted graph, $G = (V, E)$, vertices are divided into stages such that, the edges connects vertex from one stage to another. First stage and last stage has single vertex only.
- It is mainly applied in resource allocation.
- The vertex s is the source, and t is the sink.
- $c(i, j)$ – cost of edge (i, j).
- The cost of the path from s to t is the sum of the costs of the edges on the path.
- Multistage graph problem is to find a minimum-cost path from s to t.
- Dynamic programming solution can be applied to find a optimal solution i,e 'principle of optimality'



V	1	2	3	4	5	6	7	8	9	10	11	12
cost	16	7	9	18	15	7	5	7	4	2	5	0
d.	2/3	7	6	8	8	10	10	10	12	12	12	12

Consider last - 1 stage – IV stage

$$\text{cost}(\overset{\text{stage}}{4}, \overset{\text{vertex}}{9}) = 4$$

$$\text{cost}(4, 10) = 2$$

$$\text{cost}(4, 11) = 5$$

III stage :

$$\text{cost}(3, 6) = \min\{\text{cost}(6, 9) + \text{cost}(4, 9), \text{cost}(6, 10) + \text{cost}(4, 10)\}$$

$$= \min\{6 + 4, 5 + 2\}$$

$$= 7$$

$$cost(3, 7) = min\{4 + cost(4, 9), 3 + cost(4, 10)\}$$

$$= 5$$

$$cost(3, 8) = 7$$

$$cost(2, 2) = min\{4 + cost(3, 6), 2 + cost(3, 7), 1 + cost(3, 8)\}$$

$$= 7$$

$$cost(2, 3) = 9$$

$$cost(2, 4) = 18$$

$$cost(2, 5) = 15$$

$$cost(1, 1) = min\{9 + cost(2, 2), 7 + cost(2, 3), 3 + cost(2, 4), 2 + cost(2, 5)\}$$

$$= 16.$$

stage vertex

$$cost(i, j) = min\{c(j, l) + cost(i+1, l)\}$$

$$l \in V_{i+1}$$

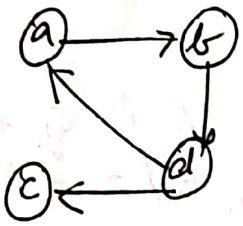
$$\langle j, l \rangle \in E$$

```
Algorithm FGraph (G, k, n, P)
// The input is a k-stage graph G=(V,E) with n vertices.
// indexed in order of stages. E is the set of edges
   and c[i,j] is the cost. P[1:k] is the
   minimum-cost path.
{
    cost[n] := 0;
    for j := n-1 to 1 step -1 do
    {
        Let r be a vertex such that (j,r) is
        an edge of G and c[j,r] + cost[r]
            is minimum;
        cost[j] = cost[j,r] + cost[r];
        d[j] = r;
    }
    P[1] = 1;
    P[k] = n;
    for j := 2 to k-1 do
        P[j] = d[P[j-1]]
}
```

Transitive Closure:

- The transitive closure of a directed graph with 'n' vertices can be defined as n by n boolean matrix $T = \{ t_{ij} \}$, in which the element in the i^{th} row $(1 \le i \le n)$ & j^{th} column $(1 \le j \le n)$ is 1 if there exists a non-trivial directed path

Example:



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \qquad T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Adjacency matrix — Transitive closure.

- Transitive closure can be generated by DFS or BFS.

Warshall's Algorithm:

Warshall's algorithm constructs transitive closure of a given digraph with n vertices through a series of n by n boolean matrices.

$$R^{(0)} \ldots R^{(k-1)}, R^{k} \ldots R^{n}$$

- where the element $r_{ij}^{(k)}$ in the i^{th} row and j^{th} column of matrix $R^{(k)}$ is equal to 1, iff there exists a directed path from i^{th} vertex to the j^{th} vertex with each intermediate vertex if any numbered not higher than k.

$R^{(0)} \rightarrow$ Adjacency matrix itself

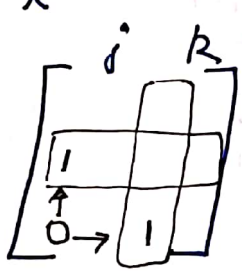
Recurrence Relation for Warshall's Algorithm:

$$r_{ij}^{(k)} = r_{ij}^{k-1} \quad \text{or} \quad \left(r_{ik}^{k-1} \text{ and } r_{kj}^{k-1}\right)$$

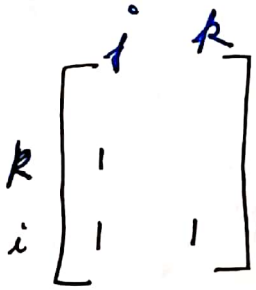
i,e (1) If an element r_{ij} is 1 in R^{k-1}, it remains 1 in R^k

(2) If an element r_{ij} is 0 in R^{k-1}, it has to be changed to 1 in R^k iff the element in its row 'i' and column 'k' & the element in its column j & row k are both 1's in R^{k-1}

$R^{k-1} =$



$\Rightarrow R^k =$



Algorithm:

Algorithm Warshall $(A[1 \cdots n][1 \cdots n])$

// Computes transitive closure

// Input - Adjacency matrix A of a digraph

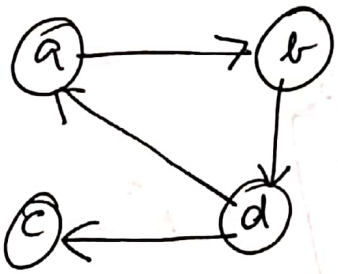
// with n vertices

// Output: Transitive closure.

{ $R^{(0)} = A$

```
for k=1 to n do
    for i=1 to n do
        for j=1 to n do
            R^k[i,j] = R^(k-1)[i,j] or (R^(k-1)[i,k] and
                                        R^(k-1)[k,j])
    return R^k
}
```

problem : obtain the transitive closure for the following digraph using warshall's Algorithm:



Solution:

Adjacency matrix

$$A = R^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$R^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

highlight 1st row & 1st column

- observe 1st in column & row

(d, a) = 1 & (a, b) = 1, then (d, b) = 1

$R^{(1)}$ =

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

Highlight 2nd row & 2nd column :

$(a, b) = 1$, $(b, d) = 1$

$\therefore (a, d) = 1$

$(d, b) = 1$, $(b, d) = 1$

$\therefore (d, d) = 1$

$R^{(2)}$ =

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

d, c = 1, so no new paths.

$R^{(3)}$ =

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

$(a, d) = 1$ { $(d, a) = 1$ $\therefore (a, a) = 1$; $(d, b) = 1$ $(a, b) = 1$; $(d, c) = 1$ $(a, c) = 1$; $(d, d) = 1$ $(a, d) = 1$

$(b, d) = 1$ { $(d, a) = 1$ $\therefore (b, a) = 1$; $(d, b) = 1$ $(b, b) = 1$; $(d, c) = 1$ $(b, c) = 1$; $(d, d) = 1$ $(b, d) = 1$

$R^{(4)}$ =

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

Transitive closure.

Floyd's Algorithm:

It computes the distance matrix D of a weighted graph with 'n' vertices through a series of n by n matrices.

$$D^{(0)} \ldots D^{(k-1)}, D^{(k)} \ldots D^{(n)}$$

where the element $d_{ij}^{(k)}$ in the i^{th} row and j^{th} column of matrix $D^{(k)}$ $(k = 0, 1, 2 \ldots n)$ is equal to the length of the shortest path among all paths from the i^{th} vertex to j^{th} vertex, with each intermediate vertex, if any, numbered not higher than k.

→ $D^{(0)}$ is the cost adjacency matrix

Recurrence Relation for Floyd's Algorithm:

$$d_{ij}^{(k)} = \min \left\{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\} \quad \forall \; k \geq 1$$

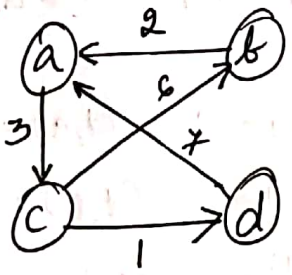
$$\boxed{d_{ij}^{(0)} = a_{ij}}$$

- the element in the i^{th} row & j^{th} column of the current distance matrix $D^{(k-1)}$ is replaced by the sum of the elements in the same row i & the k^{th} column & in the same column j & the k^{th} column if the latter sum is smaller than its current value.

Algorithm

```
ALGORITHM Floyd (A[1 ... n][A ... n])
//Finds the shortest path b/w all pair of vertices.
// Input - the cost adjacency matrix A
// output - the distance matrix D
{
    D = W
    for k=1 to n do
        for i=1 to n do
            for j=1 to n do
                D[i,j] = min [D[i,j], D[i,k] + D[k,j]]
    return D
}
```

Problem: Apply Floyd's Algorithm to the following graph & find the shortest paths b/w all pairs of vertices.



eg!

Adjacency matrix $A = D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Highlight 1st row & 1st Column & observe non zero, non infinity values.

$D^{(0)}$ =

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

(b,a)=2 (a,c)=3

∴ (b,c)= min (b, 3+2) = 5

(d,a) = 6 (a,c)=3

∴(d,c) = min(b, 6+3) = 9

$D^{(1)}$ =

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	7	0	1
d	6	∞	9	0

(c,b) = 7 (b,a) = 2

∴(c, a)= min (b, 7+2) = 9

(b,c)=5 (c,b)= 7 ∴ (b,b) → wont change

$D^{(2)}$ =

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

(a,c)=3 (c,a)=9 ∴ (a,a)- wont change

(a,c) =3 (c,b)=7 ∴ (a,b)= min(b, 7+3)= 10

(a,c)= 3 (c,d)=1 ∴ (a,d)= Min(b,3+1)=4

(b,c)=5 { (c,a)=9 ∴ (b,a) - wont change

(c,b)=7 (b,b) - wont change

(c,d)=1 (b,d) = Min(b, 5+1)=6

(d,c)=9 { (c,a)= 9 (d,a) = wont change

(c,b)=7 (d,b) = Min(b, 9+7)=16

(c,d)=1 (d,d)= No change.

$D^{(3)}$ =

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	9	7	0	1
d	6	16	9	0

$(a,d) = 4$ $\begin{cases} (d,a) = 6 \\ (d,b) = 16 \\ (d,c) = 9 \\ (d,d) = 0 \end{cases}$

$\therefore (a,a)$ – No change

$\therefore (a,b)$ – No change

$\therefore (a,c)$ – No change

$\therefore (a,d)$ – No change

$(b,d) = 6$ $\begin{cases} (d,a) = 6 \\ (d,b) = 16 \\ (d,c) = 9 \\ (d,d) = 0 \end{cases}$

$\therefore (b,a)$ = No change

$\therefore (b,b)$ = No change

$\therefore (b,c)$ = No change

$\therefore (b,d)$ = No change

$(c,d) = 1$ $\begin{cases} (d,a) = 6 \\ (d,b) = 16 \\ (d,c) = 9 \\ (d,d) = 0 \end{cases}$

$\therefore (c,a) = \min(9, 1+6) = 7$

$\therefore (c,b)$ = No change

$\therefore (c,c)$ = No change

$\therefore (c,d)$ = No change

$D^4 =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

Distance Matrix.

Problem

① Solve the following to find transitive closure:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

② Solve the all pairs shortest paths problem for the digraph given a weight matrix

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Optimal Binary Search Tree:

- It is a binary search tree which provides the smallest possible search time for a given sequence of access probabilities.

Example:

Let there be four keys A, B, C, D with probabilities 0.1, 0.2, 0.4 and 0.3 respectively. The number of binary search tree that can be formed

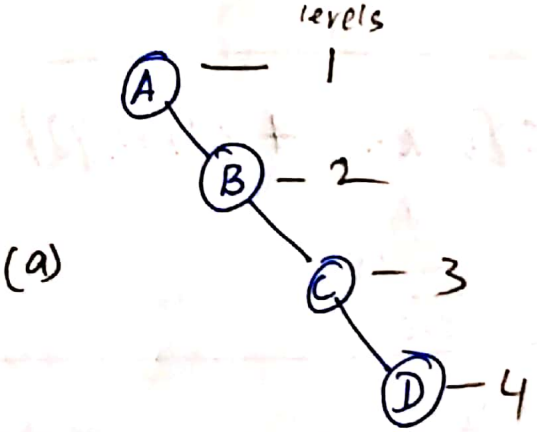
$$= \frac{^{2n}C_n}{n+1}$$

so 4 keys

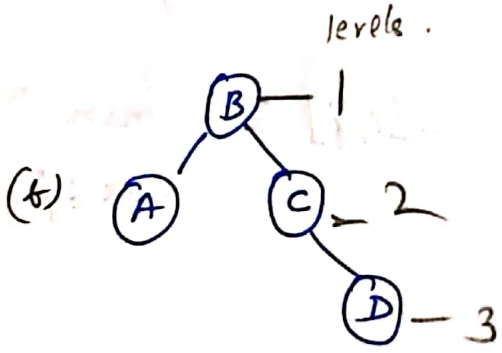
$$= \frac{^{2*4}C_4}{5} \qquad \frac{^{8}C_4}{5} = \frac{8!/4!(8-4)!}{5}$$

$$= 14$$

Consider 2 possible BST with A, B, C, D



(a)



(b)

- Average number of comparisions in a successful search in tree (a) is

$$0.1*1 + 0.2*2 + 0.4*3 + 0.3*4$$
$$= 2.9$$

tree (b) →

$$0.1*2 + 0.2*1 + 0.4*2 + 0.3*3 = 2.1$$

using exhaustive search approach is unrealistic:
As finding solutions for all different binary tree will grow to $\frac{4^n}{n^{1.5}} \to \infty$

Dynamic programming Approach:

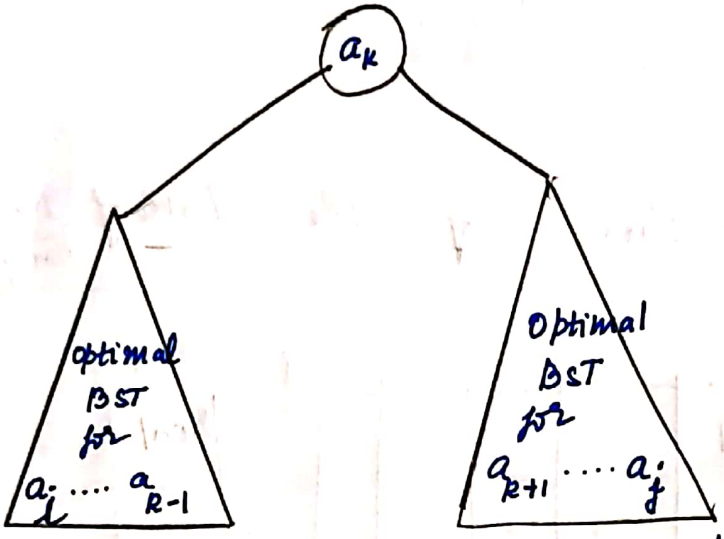
- Let $a_1, \ldots a_n$ → distinct keys ordered from the smallest to largest.
- $P_1, \ldots \ldots P_n$ → probability of searching for them.
- Let $C[i,j]$ – smallest average number of comparisions made in successful search in a binary search tree T_i^j made up of keys $a_i \ldots a_j$, where i, j are some integer indices, $1 \le i \le j \le n$.

$$C[i,j] = \min_{i \le k \le j} \left\{ C[i, k-1] + C[k+1, j] \right\} + \sum_{s=i}^{j} P_s$$

$\forall \; 1 \le i \le j \le n$

Note:

- $c[i, i-1] = 0 \quad \forall\ 1 \le i \le n+1$
  - number of comparisions in empty tree
- $c[i, i] = p_i \quad \forall\ 1 \le i \le n$
  - one node binary search tree with a_i



Binary Search Tree (BST) with root a_k and two optimal binary search subtrees T_i^{k-1} and T_{k+1}^{j}

$$c[i,j] = \min_{i \le k \le j} \left\{ p_k \cdot 1 + \sum_{s=i}^{k-1} p_s \cdot (\text{level of } a_s \text{ in } T_i^{k-1} + 1) \right.$$

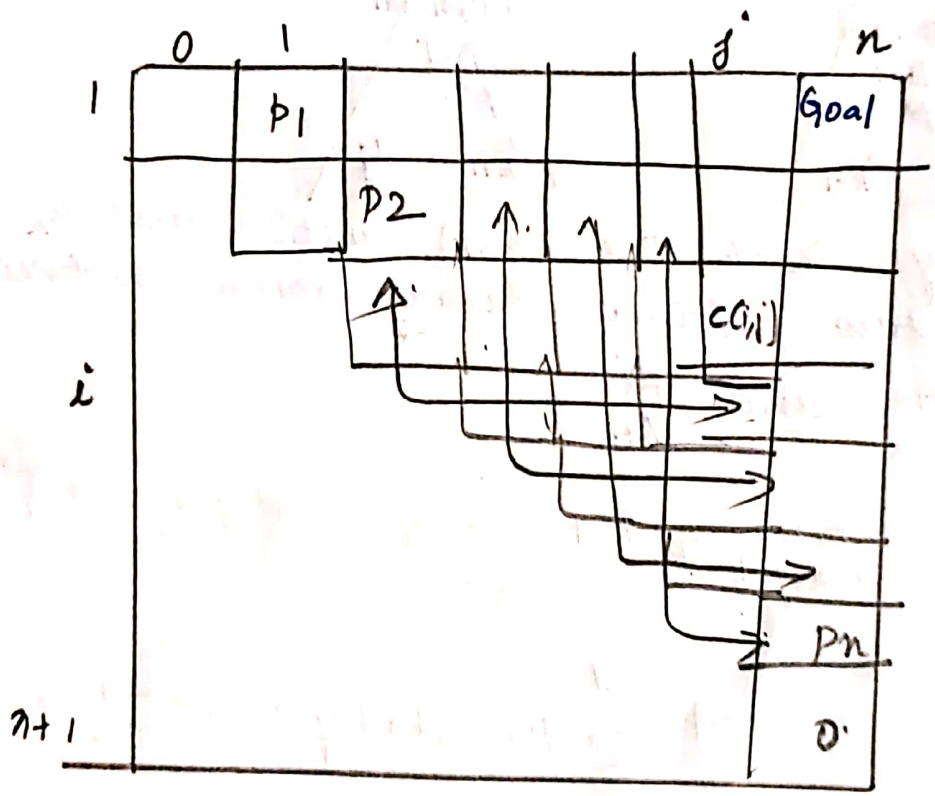
$$\left. + \sum_{s=i}^{j} p_s \cdot (\text{level of } a_s \text{ in } T_{k+1}^{j} + 1) \right\}$$

$$= \quad \vdots$$

$$= \boxed{\min_{i \le k \le j} \left\{ c[i, k-1] + c[k+1, j] \right\} + \sum_{s=i}^{j} p_s}$$

	a	b	c	d
a	a	ab	abc	abcd
b	x	b	bc	bcd
c	x	x	c	cd
d	x	x	x	d

General structure of the table of Dynamic programming



Ex: Let us illustrate the algorithm by applying it to the four-key set we used at the beginning of this section:

key	A	B	C	D
probability	0.1	0.2	0.3	0.3

Soln.:

Initial table ~~look~~ !

Main Table

	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

Root Table.

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

Compute $c[1,2]$ for $k = 1$ & 2

$$c[i,j] = \min_{i \le k \le j} \{ c[i,k-1] + c[k+1,j] \} + \sum_{s=i}^{j} P_s$$

$$c[1,2] = \min \begin{cases} k=1 : c[1,0] + c[2,2] + 0.3 \\ k=2 : c[1,1] + c[3,2] + 0.3 \end{cases}$$

$k=1 : 0 + 0.2 + 0.3 = 0.5$

$= \min$

$k=2 : 0.1 + 0 + 0.3 = 0.3$

$= 0.4$

$C[1,2] = 0.4$ with $k = 2$

- update the table with $C[1,2]$ with 0.4 & $R[1,2] = 2$

Now compute $C[2,3]$ for $k = 2, 3$

$$C[2,3] = \min \begin{cases} k=2 \rightarrow C[2,1] + C[3,3] + 0.6 \\ k=3 \rightarrow C[2,2] + C[4,3] + 0.6 \end{cases}$$

$$= \min \begin{cases} k=2 \rightarrow 0 + 0.4 + 0.6 = 1.0 \\ k=3 \rightarrow 0.2 + 0 + 0.6 = 0.8 \end{cases}$$

$C[2,3] = 0.8$ with $k = 3$

- update the table with $C[2,3]$ with 0.8 & $R[2,3] = 3$

Now compute $3, 4$ for $k = 3$ & 4

$$C[3,4] = \min \begin{cases} k=3 \rightarrow C[3,2] + C[4,4] + 0.7 \\ k=4 \rightarrow C[3,3] + C[5,4] + 0.7 \end{cases}$$

$$= \min \begin{cases} k=3 \rightarrow 0 + 0.3 + 0.7 = 1.0 \\ k=4 \rightarrow 0.4 + 0 + 0.7 = 1.1 \end{cases}$$

- Now compute $C[2,4]$ for $k = 2,3,4$

$$C[2,4] = \min \begin{cases} k=2 \rightarrow C[2,1] + C[3,4] + 0.9 \\ k=3 \rightarrow C[2,2] + C[4,4] + 0.9 \\ k=4 \rightarrow C[2,3] + C[5,4] + 0.9 \end{cases}$$

$$= \min \begin{cases} k=2 \rightarrow 0 + 1.0 + 0.9 = 1.9 \\ k=3 \rightarrow 0.2 + 0.3 + 0.9 = 1.4 \\ k=4 \rightarrow 0.8 + 0 + 0.9 = 1.7 \end{cases}$$

$$\boxed{C[2,4] = 1.4 \quad \text{with} \quad k=3}$$

Update the table with $C[2,4] = 1.4$ & $R[2,4] = 3$

- Now compute $C[1,4]$ for $k = 1,2,3,4$

$$C[1,4] = \min \begin{cases} k=1 \Rightarrow C[1,0] + C[2,4] + 1.0 \\ k=2 \Rightarrow C[1,1] + C[3,4] + 1.0 \\ k=3 \Rightarrow C[1,2] + C[4,4] + 1.0 \\ k=4 \Rightarrow C[1,3] + C[5,4] + 1.0 \end{cases}$$

$$C[1,4] = \min \begin{cases} k=1 \rightarrow 0 + 1.4 + 1 = 2.4 \\ k=2 \rightarrow 0.1 + 1.0 + 1.0 = 2.1 \\ k=3 \Rightarrow 0.4 + 0.3 + 1.0 = 1.7 \\ k=4 \Rightarrow 1.1 + 0 + 1.0 = 2.1 \end{cases}$$

$\therefore\ C[3,4] = 1.0$ with $k=3$

update the table with $C[3,4] = 1.0$ & $R[3,4] = 3$

Now compute $C[1,3]$ for $k=1, 2$ and 3.

$$C[1,3] = \min \begin{cases} k=1 \rightarrow C[1,0] + C[2,3] + 0.7 \\ k=2 \rightarrow C[1,1] + C[3,3] + 0.7 \\ k=3 \rightarrow C[1,2] + C[4,3] + 0.7 \end{cases}$$

$$= \min \begin{cases} k=1 \rightarrow 0 + 0.8 + 0.7 = 1.5 \\ k=2 \rightarrow 0.1 + 0.4 + 0.7 = 1.2 \\ k=3 \rightarrow 0.4 + 0 + 0.7 = 1.1 \end{cases}$$

$\therefore\ C[1,3] = 1.1$ with $k=3$

update the table with $C[1,3] = 1.1$ and $R[1,3] = 3$.

C[1,4] = 1.7 with k=3

-update the table with C[1,4] = 1.7 and R[1,4] = 3

Final Table

Main Table

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

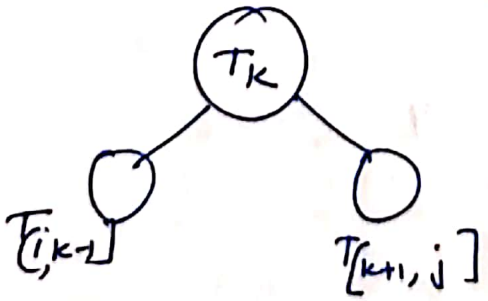
Root Table

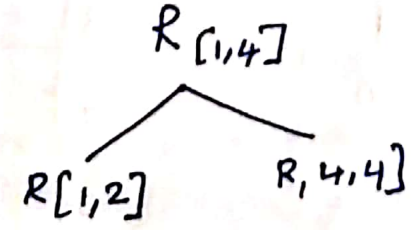
optimal

	1	2	3	4
1	1	2	3	3
2		2	3	3
3			3	3
4				4

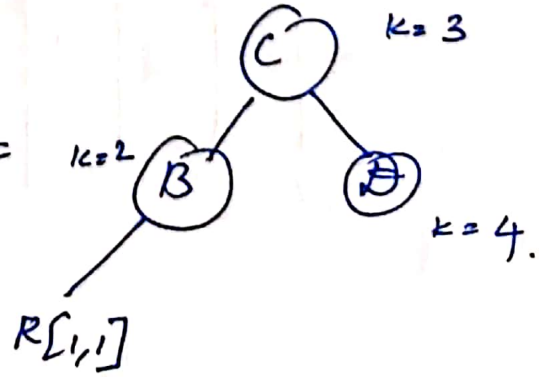
- To build a tree R[1][n] ;i.e R[1][4] is

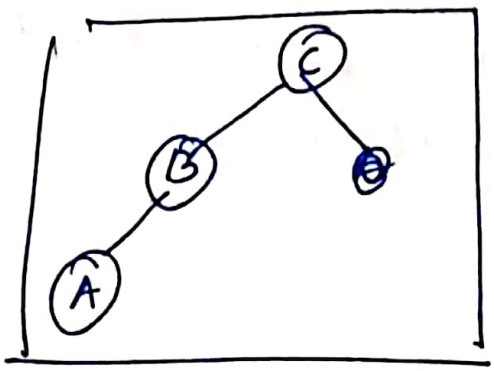
Root → 3 ;i.e C





=





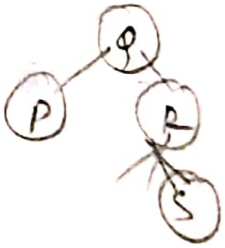
The optimal Binary search Tree

Ex: Construct optimal Binary Search Tree for the following data.

Key	P	Q	R	S
probability	1/20	1/5	1/10	1/20.

} Take LCM

P	Q	R	S
1	4	2	1



Sol

	0	1	2	3	4
1	0	1	6	10	13
2		0	4	8	12
3			0	2	4
4				0	1
5					0

	1	2	3	4
1	1	2	2	2
2		2	2	2
3			3	3
4				4.

- If the keys are in nondecreasing order, then running time of the algorithm to $\Theta(n^2)$.

Algorithm : optimalBST (P[1 ... n])

// Finds an optimal binary tree by dynamic programming

// Input : An array P[1 ... n] of search probabilities for a sorted list of n keys.

// output: Average number of comparisions in successful searches in the optimal BST and table R of subtrees root in the optimal BST.

```
for i ← 1 to n do
        c[i, i-1] ← 0;
        c[i, i] ← p[i];
        R[i, i] ← i;
c[n+1, n] ← 0;
for d ← 1 to n-1 do
        for i ← 1 to n-d do
                j = i + d;
                minval ← ∞;
                for k ← i to j do
                        if c[i, k-1] + c[k+1, j] < minval
                                minval ← c[i, k-1] + c[k+1, j];
                                kmin ← k;
                R[i, j] = kmin;
                sum ← p[i];
                for s ← i+1 to j do
                        sum ← sum + p[s];
                c[i, j] ← minval + sum
return c[1, n], R
```

The knapsack problem & memory functions

Problem statement:

Given 'n' objects of known weights $(w_1, w_2 \cdots w_n)$ and profits $(v_1, v_2 \cdots v_n)$ & the knapsack capacity $-W$,

the task is to find most valuable subset of items that fit into the knapsack.

- All the weights and knapsack capacity are +ve integers, the item values do not have to integers.

Dynamic program Approach:

- Lets consider an instance of the problem with first 'i' items, $1 \le i \le n$ with the weights $w_1 \cdots w_i$, values (profits) $v_1 \cdots v_i$ & knapsack capacity j, $1 \le j \le w$.

- Let $v[i, j]$ be the value of the optimal solution to the instance i.e value of the most valuable subset of the first i items that fit onto the knapsack of capacity j.

- This subset can be divided into 2 categories:

(i) Those that do not include i^{th} item &

(ii) Those that include i^{th} item.

Recurrance Relation :

(i) Among the subsets that do not include i^{th} item, the value of an optimal subset is $V[i-1, j]$.

(ii) Among the subsets that include i^{th} item hence $(j - w_i \geq 0)$, an optimal subset is made up of this item & an optimal subset of the first $(i-1)$ items that fit into the knapsack of capacity $j - w_i$. The value of such an optimal subset is $V_i + V[i-1, j-w_i]$

∴ The recurrence relation is given by :

$$V[i,j] = \begin{cases} V[i-1, j] \\ \max \{ V[i-1, j],\ V_i + V[i-1, j-w_i] \} \end{cases}$$

if $j - w_i \geq 0$

with initial conditions:

$V[0, j] = 0$ for $j \geq 0$ &

$V[i, 0] = 0$ for $i \geq 0$

problem: Find an optimal solution for the following instance of knapsack problem using Dynamic programming:

item	weight	value
1	2	12
2	1	10
3	3	20
4	2	15

capacity, W = 5

0/1

since n=4 & W=5, the solution matrix will have n+1 rows & W+1 columns as shown below:

	Capacity j					
i	0	1	2	3	4	5
0						
1						
2						
3						
4						

- we know from the recurrence relation that $V[i,j] = 0$ for $i = j = 0$.

[i=0 indicates no objects & j=0 indicates knapsack ~~capacity~~ is not filled].

- From the recurrence relation

we know $V[i,j] = 0$

So first row and first column = 0

- Consider $i=1$, $w_i = 2$, $v_i = 12$ & compute the values for $j = 1, 2, 3, 4, 5$.

i.e $V[i,j] = V[i-1, j]$ if $(j - w_i < 0 \ \& \ j < w_i)$

$V[i,j] = \max [V[i-1, j], \ v_i + V[i-1, \ j - w_i]]$ if $(j - w_i \geq 0 \ \& \ j \geq w_i)$

$V[1,1] = V[0,1] = 0$

$V[1,2] = \text{Max}[V[0,2], \ 12 + V[0,0]] = \max(0, 12) = 12$

$V[1,3] = \text{Max}[V[0,3], \ 12 + V[0,1]] = \max(0, 12) = 12$

$V[1,4] = \max[V[0,4], \ 12 + V[0,2]] = \max(0, 12) = 12$

$V[1,5] = \max[V[0,5], \ 12 + V[0,3]] = \max(0, 12) = 12$

→ Consider $i = 2$, $w_i = 1$, $v_i = 10$ & compute the values for j:

$V[2,1] = \max[V[1,1], \ 10 + V[1,0]] = \max(0, 10) = 10$

$V[2,2] = \max[V[1,2], \ 10 + V[1,1]] = \max(12, 10) = 12$

$V[2,3] = \max[V[1,3], \ 10 + V[1,2]] = \max[12, 22] = 22$

$V[2,4] = \max[V[1,4], \ 10 + V[1,3]] = \max(12, 22) = 22$

$V[2,5] = \max[V[1,5], \ 10 + V[1,4]] = \max[12, 22) = 22$

- consider $i=3$, $w_i=3$, $v_i=20$ & compute the values:

$V[3,1] = V[2,1] = 10$

$V[3,2] = V[2,2] = 12$

$V[3,3] = \max[V[2,3], 20+V[2,0]] = \max(22, 20) = 22$

$V[3,4] = \max[V[2,4], 20+V[2,1]] = \max(22, 30] = 30$

$V[3,5] = \max[V[2,5], 20+V[2,2]] = \max[22, 32] = 32$

- consider $i=4$, $w_i=2$, $v_i=15$

$V[4,1] = V[3,1] = 10$

$V[4,2] = \max[V[3,2], 15+V[3,0]] = \max[12, 15] = 15$

$V[4,3] = \max[V[3,3], 15+V[3,1]] = \max(22, 15) = 22$

$V[4,4] = \max[V[3,4], 15+V[3,2]] = \max(30, 27) = 30$

$V[4,5] = \max[V[3,5], 15+V[3,3]] = \max(32, 37] = 37$

	Capacity j					
i	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Solve the following instance of knapsack problem using dynamic programming:

item	weight	value
1	3	25
2	2	20
3	1	15
4	4	40
5	5	50

$W = 6$

Travelling Salesman problem (TSP)

Problem statement:

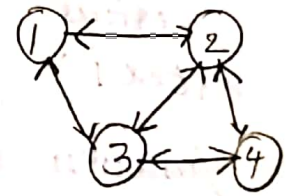
Given → • 'n' cities & distances between them

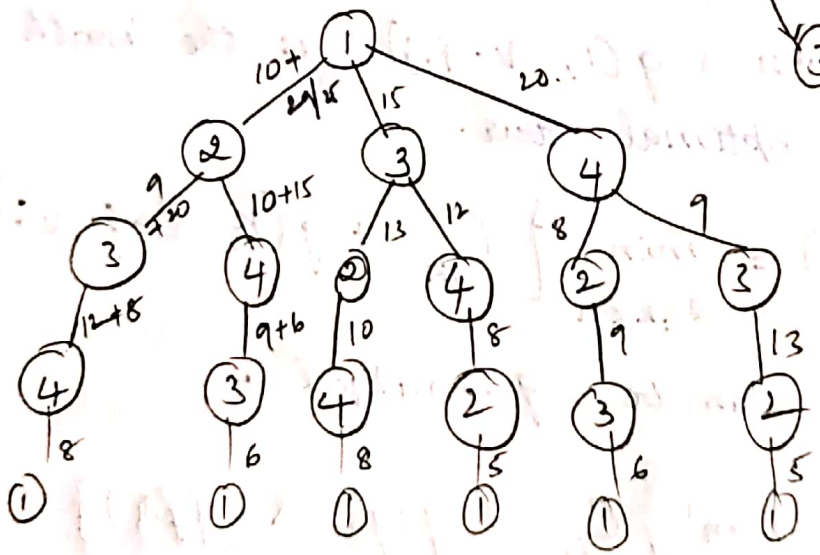
• source (starting point).

The task is: the salesperson starts his tour from source and visits all (n-1) cities exactly one and returns back to the source. Objective is - to finish this tour at minimum cost.

Dynamic programming Approach:

Ex:





	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

cost

$$g(1, \{2,3,4\}) = \min \left(C_{1k} + g(k, \underbrace{\{2,3,4\} - \{k\}}_{\text{Remaining}}) \right)$$

from 1 to 2, 3, 4.

subtract which is considered

- TSP can be modelled as a graph $G = (V, E)$, where vertices represent cities, weights linked to edges represent distance [cost involved between adjacent vertices]

- Consider source vertex as 1
- Tour is a simple path that starts and ends at vertex 1
- Every tour consists of an edge $(1, k)$ for some $k \in V - \{1\}$ & a path from vertex k to vertex 1
- Let $g(i, s)$ be length of shortest path starting as vertex i, going through all vertices in S & terminating at vertex 1
- The function $g(1, V - \{1\})$ is the length of an optimal tour.

$$g(1, V - \{1\}) = \min_{2 \le k \le n} \{ C_{1k} + g(k, V - \{1, k\}) \}$$

- The equation can be generalized to:

$$g(i, s) = \min_{j \in s} \{ C_{ij} + g(j, S - \{j\}) \} \quad \text{for } i \notin S$$

The base case is given by:

$$g(i, \phi) = C_{i1}, \quad 1 \le i \le n$$

In summary: equations used to solve TSP are:

(i) $g(i, \phi) = C_{i1}$ $\quad 1 \le i \le n, \quad i \neq 1$

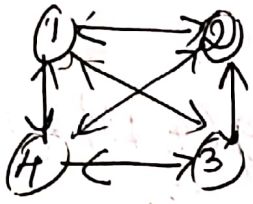
(ii) $g(i, s) = \min_{j \in s} \{ C_{ij} + g(j, s - \{j\}) \}$

for $i \neq 1, i \notin s, 1 \notin s$

& compute this for $|s| < n-1$

(iii) $g(1, V - \{1\}) = \min_{2 \le k \le n} \{ C_{1k} + g(k, V - \{1, k\}) \}$

Problem → solve the following graph for TSP



$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 5 & 9 & 0 \end{bmatrix} \end{array}$$

Soln use base case:

$g(i, \phi) = C_{i1}$ for $1 \le i \le n, i \neq 1$ as $s = \{\phi\}$, $|s| = 0$

$g(2, \phi) = C_{21} = 5$

$g(3, \phi) = C_{31} = 6$

$g(4, \phi) = C_{41} = 8$

use

$$g(i,s) = \min_{j \in s} \{ C_{ij} + g(j, s - \{j\}) \}$$

for $i \neq 1$, $i \notin s$, $1 \notin s$

$|s| = 1$

for $i = 2$

$$g(2,\{3\}) = \min_{j \in \{3\}} \{ C_{23} + g(3,\phi) \} = 9+6 = 15$$

$$g(2,\{4\}) = \min_{j \in \{4\}} \{ C_{24} + g(4,\phi) \} = 10+8 = 18$$

for $i = 3$

$$g(3,\{2\}) = \min_{j \in \{2\}} \{ C_{32} + g(2,\phi) \} = 13+5 = 18$$

$$g(3,\{4\}) = \min_{j \in \{4\}} \{ C_{34} + g(4,\phi) \} = 12+8 = 20$$

for $i = 4$

$$g(4,\{2\}) = \min_{j \in \{2\}} \{ C_{42} + g(2,\phi) \} = 8+5 = 13$$

$$g(4,\{3\}) = \min_{j \in \{3\}} \{ C_{43} + g(3,\phi) \} = 9+6 = 15$$

$|s| = 2$

for $i = 2$

$$g(2,\{3,4\}) = \min_{j \in (3,4)} \{ C_{23} + g(3,4),\ C_{24} + g(4,3) \}$$

$$= \min(9+20,\ 10+15) = \min(29, 25)$$

$$= 25$$

for $i=3$

$$g(3,\{2,4\}) = \min_{j \in 2,4} \{C_{32} + g(2,4),\ C_{34} + g(4,2)\}$$

$$= \min(13+18,\ 12+13) = \min(31,25) = \underline{25}$$

for $i=4$

$$g(4,\{2,3\}) = \min_{j \in 2,3} \{C_{42} + g(2,3),\ C_{43} + g(3,2)\}$$

$$= \min(8+15,\ 9+18)$$

$$= \min(23,\ 27) = 23$$

since $|s| < n+$ i.e $|s| < 3$ is met, the iteration stops here:

$$g(1, v-\{1\}) = \min_{2 \le k \le n} \{C_{1k} + g(k,\ V - \{1,k\})\}$$

$$g(1,\{1,2,3,4\}-\{1\}) = \min_{k=2,3,4} \{C_{1k} + g(k,\{1,2,3,4\} - \{1,k\})\}$$

$$g(1,\{2,3,4\}) = \min\{C_{12} + g(2,\{3,4\}),$$

$$C_{13} + g(3,\{2,4\}),$$

$$C_{14} + g(4,\{2,3\})\}$$

$$= \min(10+25,\ 15+25,\ 20+23)$$

$$= \min(35,\ 40,\ 43)$$

$$= \underline{35}.$$

$\therefore g(1, \{2,3,4\}) = 35$ - optimal solution.

$g(1, \{2,3,4\}) = c_{12} + g(2, \{3,4\})$

$c_{24} + g(4,3)$

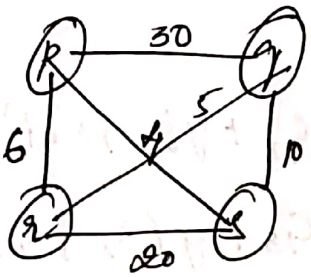
$c_{43} + g(3, \phi)$

c_{31}

$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

$\therefore$ optimal tour is $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ with optimal cost of 35.

Ex: using dynamic programming solve the following instance of TSP

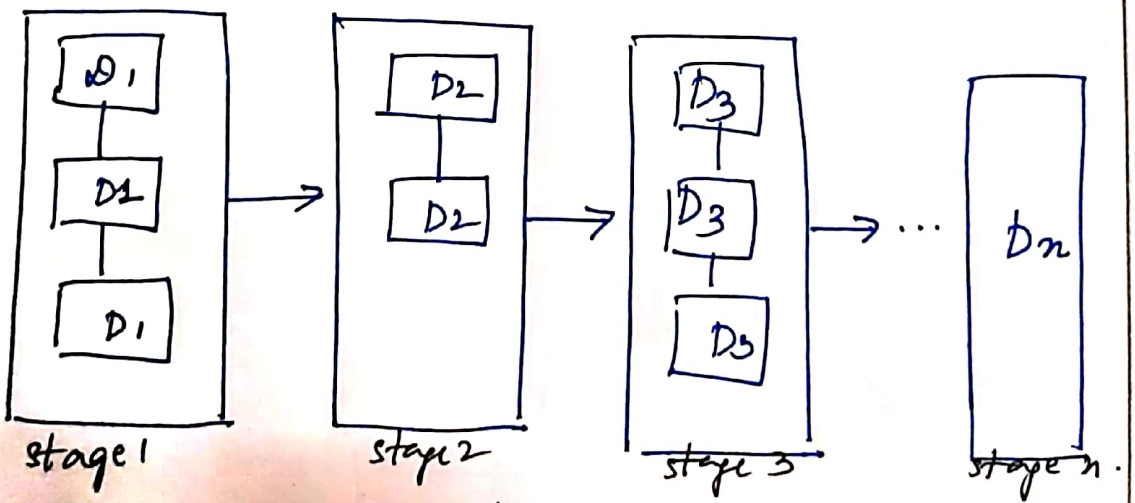


Bellman-Ford Algorithm:

- It is single source shortest path Algo.
- Dijkstra's algorithm works correct on graphs, which have edges weight as positive numbers. If the edge weight are negative, Dijkstras Algo may not give the correct shortest path.
- Belman-Ford Algo is used to obtain the shortest paths from a given vertex (source) to all other vertices in a graph with general (+ve or -ve) weights.

Reliability Design:

- set up a system which consists of set of devices. Each device is associated with cost and reliability.
- the probability that device 'i' will work properly is called Reliability of that device.
- Let r_i be the reliability of device D_i, then reliability of entire s/m is πr_i.
- Even if reliability of individual device is very good but reliability of entire system may not be good.
- To obtain good performance from entire system, individual devices can be duplicated.
- different devices are connected in series & duplicated devices are connected in parallel



- Let m_i be the copies of device D_i, then $(1-r_i)^{m_i}$ be the probability that all m_i have a malfunction. Hence stage reliability is $1-(1-r_i)^{m_i}$