UNIT-III

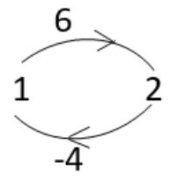
All-Pairs Shortest Paths problem: -

Given a graph *G*, we have to find a shortest path between every pair of vertices. That is, for every pair of vertices *(i, j)*, we have to find a shortest from *i* to *j*.

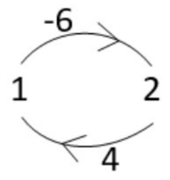
Let $G = (V, E)$ be a weighted graph with *n* vertices. Let *c* be the cost adjacency matrix for G such that $c[i, i]=0, 1\leq i\leq n$ and $c[i, j]=\infty$ if $i \neq j$ and $<i, j>$ $E(G)$.

When no edge has a negative length, the All-Pairs Shortest Paths problem may be solved by applying Dijkstra's greedy Single-Source Shortest Paths algorithm *n* times, once with each of the *n* vertices as the source vertex. This process results in $O(n^3)$ solution to the All-Pairs problem.

We will develop a dynamic programming solution called **Floyd's algorithm** which also runs in $O(n^3)$ time but works even when the graph has negative length edges (provided there are no negative length cycles).



Graph with positive cycle
Cycle length = 2
Floyd's algorithm works



Graph with negative cycle
Cycle length = -2
Floyd's algorithm doesn't work

We need to determine a matrix $A[1:n,1:n]$ such that $A[i, j]$ is the length of the shortest path from vertex *i* to vertex *j*.

The matrix *A* is initialized as $A^0[i, j] = c[i, j], 1 \leq i \leq n, 1 \leq j \leq n$. The algorithm makes *n* passes over *A*. In each pass, *A* is transformed. Let $A^1, A^2, A^3, \ldots, A^n$ be the transformations of *A* on the *n* passes.

Let $A^k[i, j]$ denote the length of the shortest path from *i* to *j* that has no intermediate vertex larger than *k*. That means, the path possibly passes through the vertices $\{1, 2, 3, \ldots, k\}$, but not through the vertices $\{k+1, k+2, \ldots, n\}$.

So, $A^n[i, j]$ gives the length of the shortest path from *i* to *j* because all intermediate vertices $\{1, 2, 3, \ldots, n\}$ are considered.

How to determine $A^k[i, j]$ for any $k\geq 1$?

A shortest path from *i* to *j* going through no vertex higher than *k* may or mayn't go through *k*.

If the path goes through *k*, then $A^k[i, j]=A^{k-1}[i, k]+A^{k-1}[k, j]$, following the principle of optimality.

If the path doesn't go through *k*, then no intermediate vertex has index greater than *(k-1)*.

Hence $A^k[i, j]=A^{k-1}[i, j]$.

By combining both cases, we get

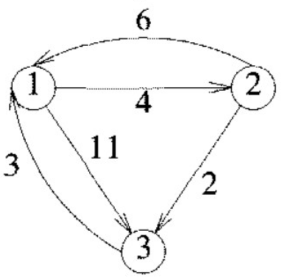
$A^k[i, j]=min\{ A^{k-1}[i, j], A^{k-1}[i, k]+A^{k-1}[k, j]\}, k\geq 1.$

This recurrence can be solved for A^n by first computing A^1, then A^2, then A^3, and so on. In the algorithm, the same matrix *A* is transformed over *n* passes and so the superscript on *A* is not needed. The k^{th} pass explores whether the vertex *k* lies on an optimal path from *i* to *j*, for all *i* and *j*. We assume that the shortest path *(i, j)* contains no cycles.

```
Algorithm Allpaths(c, A, n)
{
    //c[1:n,1:n] is the cost adjacency matrix of a graph with n vertices.
    //A[i, j] is length of a shortest path from vertex i to vertex j.
    //c[i, i]=0, for 1≤i≤n.
    for i:=1 to n do
        for j:=1 to n do
            A[i, j]=c[i, j]; //copy c into A
    for k:=1 to n do
        for i=1 to n do
            for j=1 to n do
                A[i, j]:=min(A[i, j], A[i, k]+A[k, j]);
}
```

Time complexity, $T(n)=O(n^3)$.

<u>EXAMPLE</u>: - Find out shortest paths between all pairs of vertices in the following digraph.



Step-1: **<u>$A^0 \leftarrow c$</u>**

0	4	11
6	0	2
3	∞	0

Step-2: Using, $A^1[i, j] = \min\{A^0[i, j], A^0[i, 1]+A^0[1, j]\}$

<u>A^1</u>

0	4	11
6	0	2
3	7	0

Step-3: Using, $A^2[i, j] = \min\{A^1[i, j], A^1[i, 2]+A^1[2, j]\}$

<u>A^2</u>

0	4	6
6	0	2
3	7	0

Step-4: Using, $A^3[i, j] = \min\{A^2[i, j], A^2[i, 3]+A^2[3, j]\}$

<u>A^3</u>

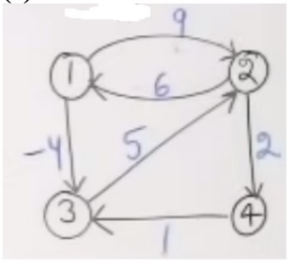
0	4	6
5	0	2
3	7	0

<u>All-Pairs Shortest Paths:</u>

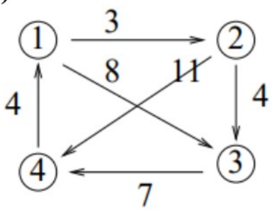
$1 \rightarrow 2$	$2 \rightarrow 3$	$3 \rightarrow 1$
$1 \rightarrow 2 \rightarrow 3$	$2 \rightarrow 3 \rightarrow 1$	$3 \rightarrow 1 \rightarrow 2$

<u>Exercises:</u>

(1)



(2)



__The Traveling Salesperson problem__**:** -

→Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

→Let the cities be represented by vertices in a graph and the distances between them be represented by weights on edges in the graph.

→Let *G(V, E)* be the directed graph with *n* vertices.
→Let graph *G* be represented by cost adjacency matrix *C[1:n, 1:n]*.
→For simplicity, *C[i, j]* is denoted by C_{ij}. If $<i, j> \notin E(G)$, $C_{ij}=\infty$; otherwise $C_{ij}\geq 0$.

→A **tour** of *G* is a directed cycle that includes every vertex in *V*. The cost of a tour is the sum of the costs of the edges on the tour.
→The travelling salesperson problem aims at finding an optimal tour (i.e., a tour of minimum cost).

→Let the given set of vertices be *{1, 2, …., n}*. In our discussion, we shall consider a tour be a simple path that starts at vertex *1* and terminates at *1*.

→Every possible tour can be viewed as consisting of an edge *<1, k>* for some $k\in V$-{1} and a path from vertex *k* to vertex *1*.
The path from vertex *k* to vertex *1* goes through each vertex in the set *V-{1, k}* exactly once.

→Suppose the tour consisting of the edge *<1, k>* (for some $k\in V$-{1}) followed by a path from *k* to *1* is an optimal tour. Then the path from *k* to *1* should also be optimal. Thus, the principle of optimality holds.

→Let *g(i, S)* denote the length of the shortest path starting at vertex *i*, going through all vertices in set *S*, and terminating at vertex *1*. Thus *g(1, V-{1})* gives the length of an optimal tour.
→If the optimal tour consists of the edge *<1, k>*, then from the principle of optimality, it follows that $g(1, V-\{1\})= C_{1k}+g(k, V-\{1,k\})$.
→But we don't know for which value of *k* the tour will be optimal. We know that *k* take can any value from the set *{2, 3, …, n}*.
→The RHS of the above recurrence can be evaluated for each value of *k* and the minimum of those results should be assigned to *g(1, V-{1})* resulting in the following recurrence equation:

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

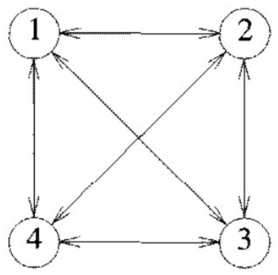
→Generalizing equation *(1)* for *i* not in *S*, we obtain

$$g(i, S)=\min_{\forall j\in S}\{C_{ij}+g(j, S-\{j\})\} \text{ ------ } (2)$$

→Equation *(1)* can be solved for *g(1, V-{1})* if we know *g(k, V-{1,k})* for all choices of *k* which can be obtained by using equation *(2)*.
→Clearly, $g(i, \emptyset)=C_{i1},\ 1\leq i\leq n$.

→The optimal tour can be found by noting the vertex which resulted in minimum cost at each stage. An algorithm that proceeds to find an optimal tour by making use of *(1)* and *(2)* will require $O(n^2 2^n)$ time.

EXAMPLE: - Consider the directed graph below and its edge lengths given by cost adjacency matrix.



$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

Find optimal sales person tour from vertex1.

Answer:-

$g(1,\{2,3,4\})=\min\{C_{12}+g(2,\{3,4\}),C_{13}+g(3,\{2,4\}),C_{14}+g(4,\{2,3\})\}$

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

$g(3,\{4\})=C_{34}+g(4,Ø)=C_{34}+C_{41}$
$=12+8=20.$

$g(4,\{3\})=C_{43}+g(3,Ø)=C_{43}+C_{31}$
$=9+6=15.$

$g(2,\{3,4\})=\min\{9+20,10+15\}$
$=25.$

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

$g(2,\{4\})=C_{24}+g(4,Ø)=C_{24}+C_{41}$
$=10+8=18.$

$g(4,\{2\})=C_{42}+g(2,Ø)=C_{42}+C_{21}$
$=8+5=13.$

$g(3,\{2,4\})=\min\{18+13,12+13\}$
$=\min\{31, 25\}$
$=25.$

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

$g(2,\{3\})=C_{23}+g(3,Ø)=C_{23}+C_{31}$
$=9+6=15.$

$g(3,\{2\})=C_{32}+g(2,Ø)=C_{32}+C_{21}$
$=13+5=28.$

$g(4,\{2,3\})=\min\{8+15,9+28\}$
$=\min\{23,37\}$
$=23.$

$g(1,\{2,3,4\})=\min\{10+25,15+25,20+23\}$
$=\min\{35,40,43\}$
$=35.$

$g(1,\{2,3,4\})=35.$

Construction of the optimal tour step by step:-

$g(1,\{2,3,4\})=C_{12}+g(2,\{3,4\}) \rightarrow 1\rightarrow2$

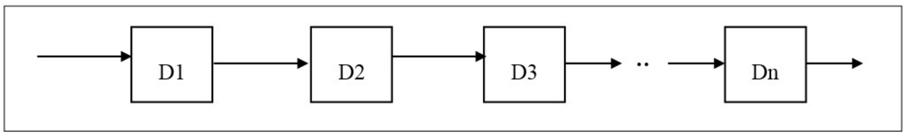
$g(2,\{3,4\})=C_{24}+g(4,\{3\}) \rightarrow 1\rightarrow2\rightarrow4$

$g(4,\{3\})=C_{43}+g(\{3,Ø\}) \rightarrow 1\rightarrow2\rightarrow4\rightarrow3\rightarrow1$

So, the optimal tour is:- $1\rightarrow2\rightarrow4\rightarrow3\rightarrow1$.

Reliability Design: -

We need to design a system that functions with maximum reliability. The system consists of n devices connected in series as shown below: -

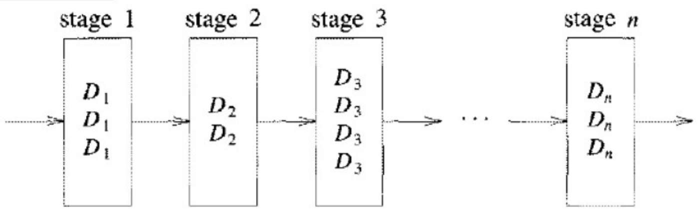


Let r_i be the reliability of device D_i (that is, r_i is the probability that device i will function properly). Then the reliability of the entire system is .

Even if the individual devices are very reliable, the reliability of the entire system may not be very good.
For example, if $n=10$ and $r_i=0.99$, $i \leq 1 \leq 10$, then = $(0.99)^{10} = 0.904$.

Hence it is desirable to have multiple copies of the same device connected in parallel so as to improve the reliability of the system.

EXAMPLE: -



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

Let us assume that the reliability of stage i with m_i copies is denoted by a function $\Phi_i(m_i)$.
So, $\Phi_i(m_i)=1-(1-r_i)^{mi}$.
Then, the reliability of the entire system is, .

The reliability design problem is to use multiple copies (as many copies as possible) of the devices at each stage so as to maximize the reliability of the entire system. However, this is to be done under a cost constraint.

Let C_i be the cost of each unit of D_i and C be the maximum allowed cost (budget) of the system being designed. Then the reliability problem is mathematically formulated as: -

maximize
subject to
$\leq C$
where, m_i is an integer and $m_i \geq 1$, for all i.

Here, $m_i \geq 1$ means that at each stage i, at least one unit of device D_i should be taken.

Assuming that each $C_i>0$, each m_i must be in the range $1 \leq m_i \leq u_i$, where

EXAMPLE: - C=100, n=3, $C_1=15$, $C_2=20$, $C_3=25$.

= 3
= 3
= 2

The dynamic programming approach finds the optimal solution for $m_1, m_2, m_3, \ldots, m_n$.
An optimal sequence of decisions, (i.e., a decision for each m_i), can result in an optimal solution.

Solution to the Reliability Design problem using backward recurrence equation:
Let $f(i, x)$ denote the reliability of an optimal solution by considering the stages $i, i\text{-}1, \ldots, 1$ and with remaining funds (or, budget) x.

Then the reliability of an optimal solution to the original problem is given by $f(n, C)$.

The first decision is made on m_n whose value can be chosen from the set $\{1, 2, 3, \ldots, u_n\}$.

Once a value for m_n has been chosen, the remaining decisions must be made so as to use the remaining funds $(C\text{-}C_n m_n)$ in an optimal way. Using the principle of optimality, we can write:

$f(n, C)=$

For any, $f(i, x), i\geq 1$, this equation generalizes to

$f(i, x)=$

Clearly, $f(0, x) =1$ for all x, $0\leq x\leq C$.

Solution using Sets with Ordered pairs: -
We solve the above recurrence equations using the method similar to that used for 0/1 knapsack problem.
Let S^i consists of tuples (f, x) that may result from the various decision sequences for $m_1, m_2, m_3, \ldots, m_i$, where f indicates the reliability of the system by considering stages 1 to i and x indicates the corresponding cost incurred (or, funds spent so far).
Obtaining S^i from S^{i-1}: -
Assume that there is u_i total no. of possible values for m_i. Then we will obtain u_i number of subsets: $S_1^i, S_2^i, S_3^i, \ldots,$ from set S^{i-1}. Then S^i can be obtaining by merging all those subsets.
So,
$S^i = S_1^i \cup S_2^i \cup S_3^i \cup \ldots \cup .$

$S_j^i, 1\leq j\leq u_i$ can be obtained from all tuples (f, x) of S^{i-1} as follows:

$S_j^i= \{(f * (j), x+j*\}, 1\leq j \leq u_i.$

If there is a tuple (f, x) in S_j^i such that $(x+)>C$, that tuple can be removed from S_j^i because such a tuple will not leave adequate funds to complete the system.

While merging the sets $S_1^i, S_2^i, S_3^i, \ldots,$ to obtain S^i, the tuples have to be arranged in the increasing order of reliabilities and then dominance rule (or, purging rule) has to be applied.

Dominance rule:-
The tuple (f_2, x_2) dominates (f_1, x_1) iff $f_2\geq f_1$ and $x_2\leq x_1$. The dominated tuple (f_1, x_1) is discarded from the set S^i.

Example:-
Design a 3-stage system in an optimal way with the following data. The costs are \$30, \$15, \$20. The cost of the system can't be more than \$105. The reliability of each device type is 0.9, 0.8 and 0.5 respectively.
Ans:-
Given $C_1=30, C_2=15, C_3=20, C=105$
$r_1=0.9, r_2=0.8, r_3=0.5$.

$= = = 2$

$= = = 3$

$= = = 3$

$S^0 = \{(1,0)\}$.
Obtaining S^1 from S^0: -
Since $u_1=2$, the possible choices for m_1 are 1 and 2.

For $m_1=1$: -
$\Phi_1(m_1)=\Phi_1(1)=1-(1-r_1)^{m1}= 1-(1-0.9)^1 = 1- 0.1 = 0.9$
The cost incurred is $m_1c_1=1*30=30$
$S_1^1= \{(1*0.9,0+30)\} = \{(0.9,30)\}$

For $m_1=2$: -
$\Phi_1(m_1) =\Phi_1(2) =1-(1-r_1)^{m1}= 1-(1-0.9)^2 = 1- 0.01 = 0.99$
The cost incurred is $m_1c_1=2*30=60$
$S_2^1= \{(1*0.99,0+60)\} = \{(0.99,60)\}$
Now,
$S^1 = S^1_1 \, U \, S_2^1$
$S^1= \{(0.9,30),(0.99,60)\}$.

Obtaining S^2 from S^1: -
Since $u_2=3$ the possible choices for m_2 are 1, 2 and 3.

For $m_2=1$:-
$\Phi_2(m_2)=\Phi_2(1)=1-(1-r_2)^{m2}= 1-(1-0.8)^1 = 1-0.2 =0.8$
The cost incurred is $m_2c_2=1*15=15$
$S_1^2=\{(0.9*0.8,30+15),(0.99*0.8,60+15)\}=\{(0.72,45),(0.792,75)\}$

For $m_2=2$:-
$\Phi_2(m_2)=\Phi_2(2)=1-(1-r_2)^{m2}= 1-(1-0.8)^2 =0.96$
The cost incurred is $m_2c_2=2*15=30$
$S_2^2=\{(0.9*0.96,30+30),(0.99*0.96,60+30)\}=\{(0.864,60),(0.9504,90)\}$
$= \{(0.864,60)\}$
Note: The tuple $(0.9504, 90)$ was removed from S_2^2 because there are no adequate funds to include even one copy of D_3 device in the 3rd stage.

For $m_2=3$:-
$\Phi_2(m_2)=\Phi_2(3)=1-(1-r_2)^{m2}= 1-(1-0.8)^3 =0.992$
The cost incurred is $m_2c_2=3*15=45$
$S_3^2=\{(0.9*0.992,30+45),(0.99*0.992,60+45)\}=\{(0.8928,75),(0.98208,105)\}$
$=\{(0.8928,75)\}$
Now,
$S^2 = S_1^2 \, U \, S_2^2 \, U \, S_3^2 =\{(0.72,45),(0.792,75),(0.864,60), (0.8928,75)\}$
By purging the tuple $(0.792,75)$ from S^2,
$S^2=\{(0.72,45),(0.864,60),(0.8928,75)\}$.

Obtaining S^3 from S^2:-
Since $u_3=3$ the possible choices for m_3 are 1, 2 and 3.

For $m_3=1$:-
$\Phi_3(m_3)=\Phi_3(1)=1-(1-r_3)^{m3}= 1-(1-0.5)^1 =0.5$
The cost incurred is $m_3c_3=1*20=20$
$S_1^3=\{(0.72*0.5,45+20),(0.864*0.5,60+20),(0.8928*0.5,75+20)\}$
$=\{(0.36,65),(0.432,80),(0.4464,95)\}$

For $m_3=2$:-
$\Phi_3(m_3)=\Phi_3(2)=1-(1-r_3)^{m3}= 1-(1-0.5)^2=0.75$
The cost incurred is $m_3c_3=2*20=40$
$S_2^3 =\{(0.72*0.75,45+40),(0.864*0.75,60+40),(0.8928*0.75,75+40)\}$
$=\{(0.54,85),(0.648,100),(0.6696,115)\}$
$=\{(0.54,85),(0.648,100)\}$

For $m_3=3$:-
$\Phi_3(m_3)=\Phi_3(3)=1-(1-r_3)^{m3}= 1-(1-0.5)^3 =0.875$
The cost incurred is $m_2c_2=3*20=60$
$S_3^3=\{(0.72*0.875,45+60),(0.864*0.875,60+60),(0.8928*0.875,75+60)\}$
$=\{(0.63,105),(0.756,120),(0.7812,135)\}$
$=\{(0.63,105)\}$

Now,
$S^3=S_1^3 \, U \, S_2^3 \, U \, S_3^3$
$=\{(0.36,65),(0.432,80),(0.4464,95),(0.54,85),(0.63,105),(0.648,100)\}$
$=\{(0.36,65),(0.432,80),(0.54,85),(0.648,100)\}$

The best optimal design has a reliability of *0.648* and cost of *$100.*

Tracing back the values of m_i :

→ The last tuple of S^3 is *(0.648,100)* and it is present in S_2^3 subset. So, $m_3=2$.

→ The tuple *(0.648,100)* of S_2^3 came from the tuple *(0.864,60)* of S^2.

→ The tuple *(0.864,60)* of S^2 is present in S_2^2 subset. So, $m_2=2$.

→ The tuple *(0.864,60)* of S_2^2 came from the tuple *(0.9,30)* of S^1.

→ The tuple *(0.9,30)* of S^1 is present in S_1^1 subset. So, $m_1=1$.

So, the optimal solution is: $(m_1, m_2, m_3) = (1, 2, 2)$.