Matrix chain Multiplication

$C[i,j]= \min \{ C[i,k]+ C[k+1,j] + d_{i-1} d_k d_j\}$

$I \le k < j$

A1 A2 A3 A4

3 2 2 4 4 2 2 5

$d_0\ d_1\ \ d_1\ d_2\ \ d_2\ d_3\ \ d_3\ d_4$

	1	2	3	4
1	0	24	28	**58**
2		0	16	36
3			0	40
4				0

j-i=1 combinations

$c[1,2] = \min\{ c[1,1] + c[2,2] +d_0d_1d_2\} = 0+0+3 \times 2 \times 4= 24$

I, j , k=1

$c[2,3]=\{ c[2,2] +c[3,3] + d_1d_2d_3\} = 0 + 0+ = 16$

$c[3,4]= \{c[3,3]+ c[4,4]+ d_2d_3d_4= 40$

j-i=2 combinations

$c[1,3]= \min\{c[1,1] +c[2,3] + d_0d_1d_3 ,\ c[1,2] +c[3,3]+d_0d_2d_3\}$

$\min\{ 0+16+ 12, 24 +0+24\}$

28

I j **k=1**,2

$c[2,4]= \min\{c[2,2]+ c[3,4]+d_1d_2d_4,\ c[2,3]+c[4,4]+d_1d_3d_4\}$

$=\min\{0+40+40, 16+0+20\}$

36

k=2,**3**

j-i=3 combinations

c[1,4]= { c[1,1]+c[2,4]+d0d1d4,   c[1,2]+c[3,4]+d0d2d4, c[1,3]+c[4,4]+ d0d3d4}

0+36+24, 24+40+60, 28 + 0+30

k=1,2,**3**

((A1)(A2 A3))A4)

Longest Common subsequence

S1= abcd n

S2= bd string of length m, subsequences possible= 2^m

bruteforce= O(n 2^m)

LCS= bd

O(m n)

If (A[i]=B[j])

LCS[I,j]= 1+ LCS[i-1,j-1]

Else

LCS[I,j]=max (LCS[i-1,j], LCS[i,j-1])

S1= STONE (5)

S2= LONGEST (7)

			S	E	Q	U	E	N	C	E
		0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
S	1	0	1	1	1	1	1	1	1	1
E	2	0	1	2	2	2	2	2	2	2
Q	3	0	1	2	3	3	3	3	3	3
U	4	0	1	2	3	4	4	4	4	4
E	5	0	1	2	3	4	5	5	5	5
L	6	0	1	2	3	4	5	5	5	5

S E Q U E

S1= sequence (8)

S2- sequel (6)

0/1 Knapsack Problem

The problem can be stated as

maximize $\sum_{n}^{1} p_i x_i$ ———— (1)

subject to $\sum_{n}^{1} w_i x_i \leq m$ ———— (2)

and $0 \leq x_i \leq 1$, $1 \leq i \leq n$. ———— (3)

Xi= 0,1

o1,o2,o3…on

Sack capacity=m

X1 X2 X3….XN-l,X_N

m-W_N, XN=1, PROFIT= PN

f(n-1)=

XN=0

m, profit=0

Initially,

$S^0 = \{(0, 0)\}$

$S^i_1 = \{(P, W) \mid (P-p_i, W-w_i) \in S^i\}$

S^{i+1}_1 can be computed by merging S^i and S^i_1

Purging rule

(p1,w1) (p2,w2)

P1<p2 and w1>w2

(p1,w1) discard

Consider the knapsack instance M = 8, and n = 4

(w1,w2,w3,w4) = (2, 3, 4, 5) (p1,p2,p3,p4) = (1, 2, 5, 6)

Initially, S^0 = {(0, 0)}

S^0_1= {(1,2)}

S^1= {(1,2), **(0, 0)**}

S^1_2= {(3,5),(2,3)}

S^2= {(0, 0), (1,2), (**2,3**), (3,5)}

S^2_3= {(5,4)(6,6)(7,7)}

S^3= {(0,0)(1,2**)(2,3)(**5,4)(3,5) (6,6)(7,7)}

S^3_4= {(6,5)(7,7)(8,8)}

S^4= {(0,0)(1,2)(2,3)(5,4)(3,5) (6,6)(7,7), (6,5**)(8,8)}**

(8,8)€S^4

(8,8) does not belong to S^3

..X4=1

m=8, w4=5 remaining capacity= 8-5=3

(2,3) €S^3

(2,3) €S^2

X3=0

(2,3) €S^2

(2,3) not belong to €S^1

X2=1

REMAINING CAPACITY=3, W2=3

REMANING CAPACITY=3-3=0

(0,0) BELONGS TO S_1

(0,0) BELONGS TO S_0

X1=0

{X1,X2,X3,X4}= (0,1,0,1}

PROFIT= 8

Table method for 0/1 knapsack problem

P	W	O/K	0	1	2	3	4	5	6	7	8
0	0	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**

x4=1

w4=5, p4=6

remaining profit= 8-6=2

x3=0

x2=1

w2=3, p2=2

remaining profit = 2-2 =0

x1=0

(x1,x2,x3,x4)= (0,1,0,1)

V[i,w]= max { V[i-1,w], V[i-1,w-w[i]] +p[i] }

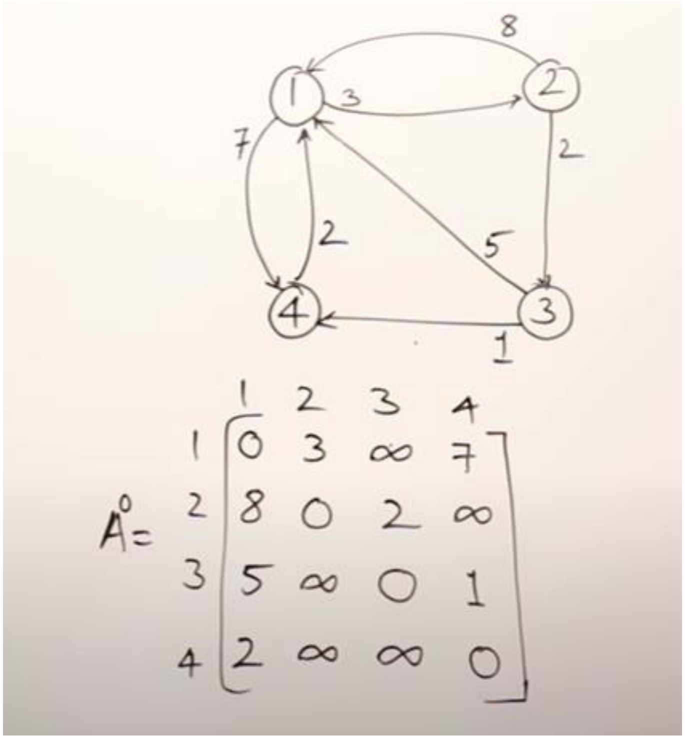
V[4,7]= max{ V[3,7], V[3, 7-5] + 6}

Max{ 7, 1+6}

V[4,8] = max {v[3,8], V[3,3]+ 6}

Max{ 7, 2+6}= 8

All Pairs Shortest path problem



A^1 (indicates the path by considering vertex 1 as intermediate vertex) – First row and first column of A^0 remains same

$A^1[2,3]$= min { $A^0[2,1]$ + $A^0[1,3]$, $A^0[2,3]$}

= min{ 8+ ∞, 2}= 2

$A^1[2,4]$= min { $A^0[2,1]$ + $A^0[1,4]$, $A^0[2,4]$}

= min{ 8+ 7, ∞}= 15

$A^1[4,2]$= min { $A^0[4,1]$ + $A^0[1,2]$, $A^0[4,2]$}

= min{ 2+ 3, ∞}= 5

$A^1[4,3]$= min { $A^0[4,1]$ + $A^0[1,3]$, $A^0[4,3]$}

Min{ 2+∞, ∞}= ∞

A^1 =

	1	**2**	**3**	**4**
1	0	3	∞	7
2	8	0	2	**15**
3	5	8	0	1
4	2	**5**	∞	0

General formula for k^{th} vertex as intermediate vertex

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

A^2

	1	**2**	**3**	**4**
1	0	3	5	7
2	8	0	2	**15**
3	5	8	0	1
4	2	**5**	7	0

$A^2[1,3]= \min\{ A^1 [1,2] + A^1[2,3], A^1[1,3]\}$

Min{ 3+2, ∞}= 5

$A^2[1,4]= \min\{ A^1 [1,2] + A^1[2,4], A^1[1,4]\}$

Min{3+15, 7}=7

$A^2[3,1]= \min\{ A^1 [3,2] + A^1[2,1], A^1[3,1]\}$

Min{8+8, 5}= 5

$A^2[3,4]= \min\{ A^1 [3,2] + A^1[2,4], A^1[3,4]\}$

Min{8+15, 1}= 1

$A^2[4,3]= \min\{ A^1 [4,2] + A^1[2,3], A^1[4,3]\}$

Min{5+2, ∞} = 7

Algorithm= $O(n^3)$

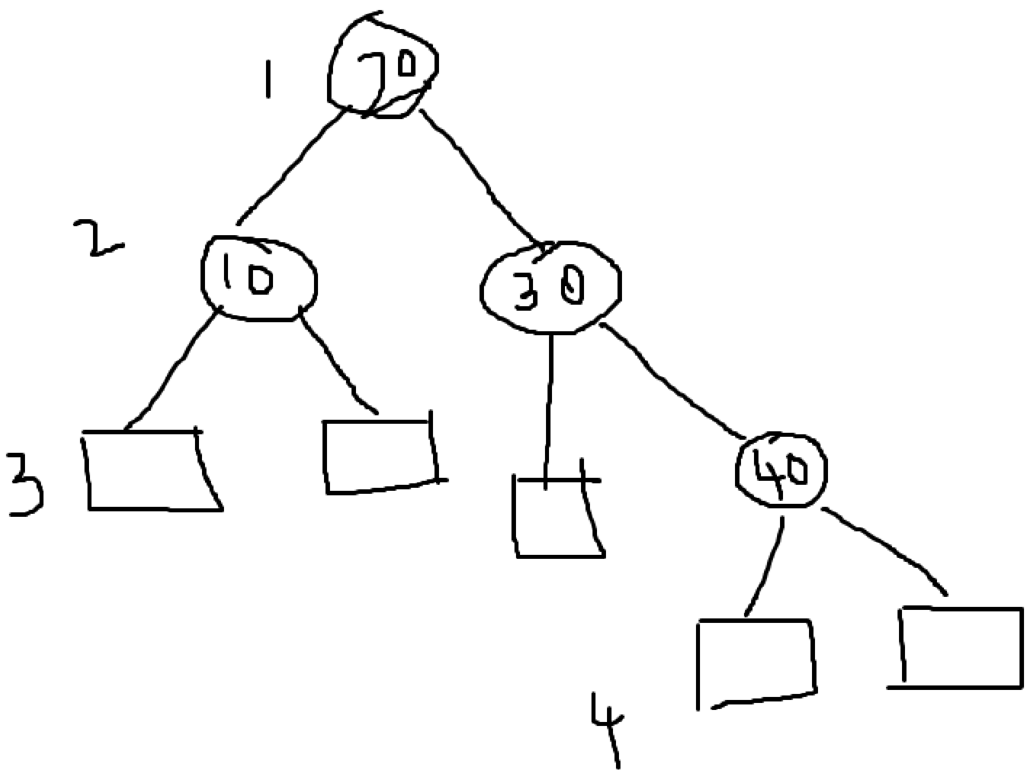
```
For(k=1,k<=n;k++)
{
For (i=1,i<=n,i++)
For(j=1; j<=n; j++)
A[i,j]= min{ A[i,k]+A[k,j], A[i,j]}
}
```

Optimal Binary Search Trees

Number of keys-n

Total BST possible- (2n C n) / (n+1)

Keys= 10,20,30,40



Cost of BST (C)= Cost of Successful searches + Cost of unsuccessful searches

C= p_i X level(Internal node) + q_i X level(External node-1)

Let (p1, p2, p3, p4) = (0.1, 0.2, 0.1, 0.2),

(q0, q1, q2, q3, q4) = (0.1,0.5,0.15,0.05,0.05)

SC= 0.1 X 2 + 0.2 X 1 + 0.1 X 2 + 0.2 X3 = 1.2

USC= 0.1 X 2 + 0.5 X 2 + 0.15 X 2 + 0.05 X 3 + 0.05 X 3 =

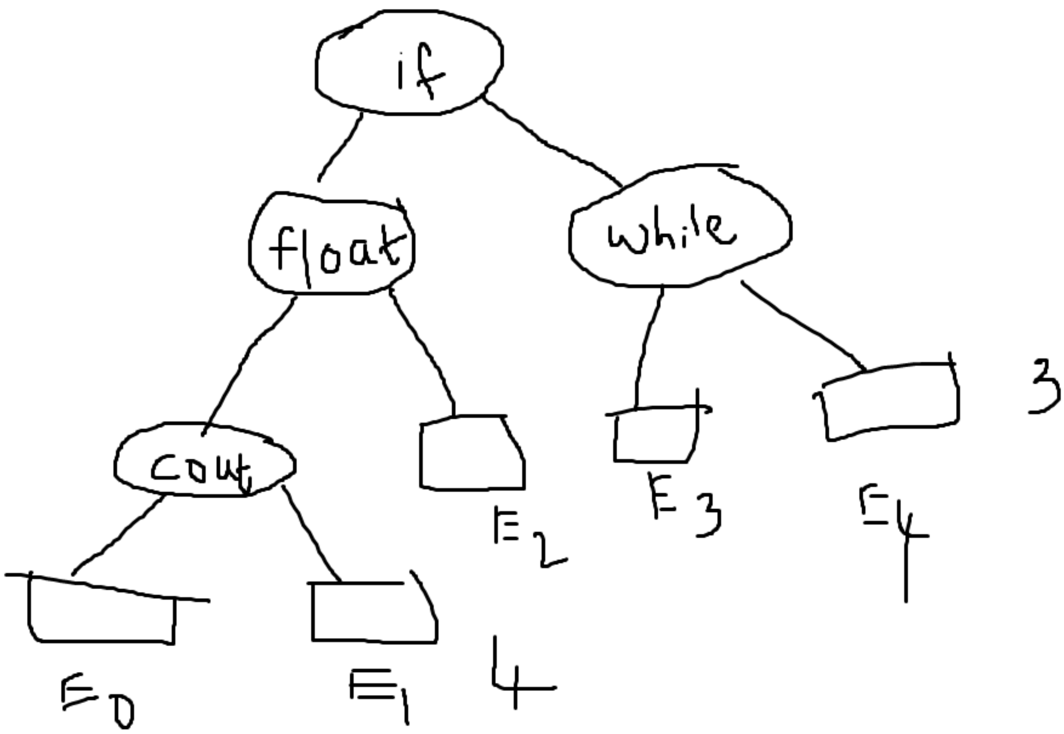
C=SC + USC=

(a1,a2,a3,a4)= (cout, float, if, while)

P(1:4)= (1/20, 1/5, 1/10, 1/20)

Q(0:4)= (1/5,1/10,1/5,1/20,1/20)

Find the cost of BST given as



Cost= SC+ USC

$$C[i,j] = min_{i<k\leq j}\{C[i,k-1] + C[k,j] + w[i,j]\}$$

w[i,j]= w[i,j-1] + p(j) + q(j)

C(i,i)=0, r(i,i)=0, w(i,i)= q(i)

Find C, w, r values for all possible (j-i) values i.e. 0,1,2,3,4

N=4, (a1,a2,a3,a4)= (do,if,int,while), p(1:4)= (3,3,1,1),

q(0:4)= (2,3,1,1,1)

j-i=0

C(0,0)=0, r(0,0)=0, w(0,0)= q(0)= 2

C(1,1)=0, r(1,1)=0, w(1,1)= q(1)= 3

C(2,2)=0, r(2,2)=0, w(2,2)= q(2)= 1

C(3,3)=0, r(3,3)=0, w(3,3)= q(3)= 1

C(4,4)=0, r(4,4)=0, w(4,4)= q(4)= 1

j-i=1

w(0,1)= w(0,0)+p(1)+q(1)= 2+3+3=8

C(0,1)= k=1 { C(0,0) +C(1,1)} + w(0,1)= 8

r(0,1)= 1

w(1,2)= w(1,1)+p(2)+q(2)= 3+3+1=7

C(1,2)= k=2 { C(1,1) +C(2,2)} + w(1,2)= 7

r(1,2)= 2

w(2,3)= w(2,2) + p(3) + q(3)= 1+1+1=3

C(2,3)= k=3 { C(2,2) +C(3,3)} + w(2,3)= 3

r(2,3)= 3

w(3,4)= w(3,3) + p(4) + q(4)= 1+1+1=3

$C(3,4)= k=4 \{ C(3,3) +C(4,4)\} + w(3,4)= 3$

r(3,4)= 4

j-i=2

$w(0,2)= w(0,1) + p(2) + q(2) = 8+3+1=12$

$C(0,2)= k=\mathbf{1},2 \min \{ C(0,0) +C(1,2) , C(0,1) + C(2,2)\} + w(0,2)$

$\mathrm{Min}\{\mathbf{0+7}, 8+0\} + 12 = 19$

$r(0,2)= 1$

$w(1,3)= w(1,2) + p(3) + q(3)= 7+1+1=9$

$C(1,3) = k=\mathbf{2},3 \min \{ C(1,1) +C(1,2) , C(1,2) + C(2,3)\} + w(1,3)$

$\mathrm{Min}\{\mathbf{0+3}, 7+0\} + 9 = 12$

$r(1,3)= 2$

$w(2,4)= w(2,3) + p(4) + q(4)= 3+1+1=5$

$C(2,4) = k=3,4 \min \{ C(2,2) +C(3,4) , C(2,3) + C(4,4)\} + w(2,4)$

$\mathrm{Min}\{0+3, 3+0\} + 5 = 8$

r(2,4)= **3 or 4**

j-i=3

$w(0,3)= w(0,2) + p(3) + q(3)= 12+1+1=14$

$C(0,3) = k=1,\mathbf{2},3 \min \{ C(0,0) +C(1,3) , C(0,1) + C(2,3), C(0,2) + C(3,3)\} + w(0,3)$

$\mathrm{Min}\{0+12, \mathbf{8+3}, 19+0\} + 14 = 25$

r(0,3)= **2**

$w(1,4)= w(1,3) + p(4) + q(4)= 9+1+1=11$

$C(1,4) = k=\mathbf{2},3,4 \min \{ C(1,1) +C(2,4) , C(1,2) + C(3,4), C(1,3) + C(4,4)\} + w(1,4)$

Min{0**+8**, 7+3, 12+0} + 11 = 19

r(0,3)= **2**

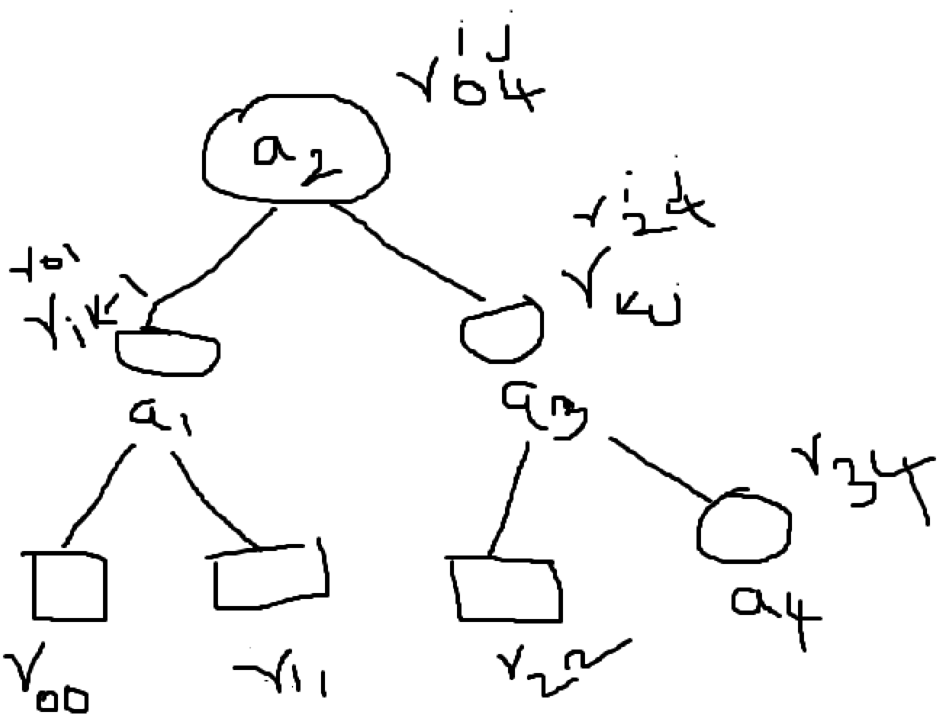
j-i=4

w(0,4)= w(0,3) + p(4) + q(4)= 14+1+1=16

C(0,4) = k=1,**2**,3,4 min { C(1,0) +C(1,4) , C(0,1) + C(2,4), C(0,2) + C(3,4), C(0,3)+ C(4,4)} + w(1,4)

Min{0+19, **8+8**, 19+3, 25+0} + 16 = 32

r(0,4)= **2**



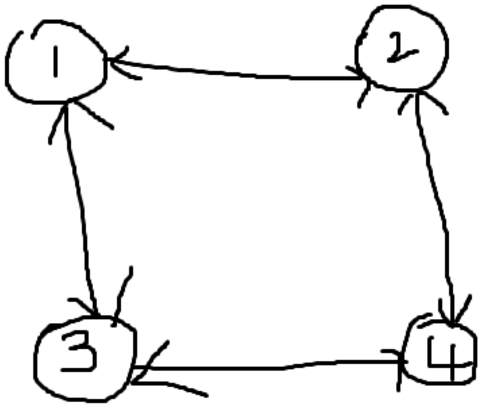
i=0, j=4, k=2

I=0, j=1, k=1

Rik-1= r00

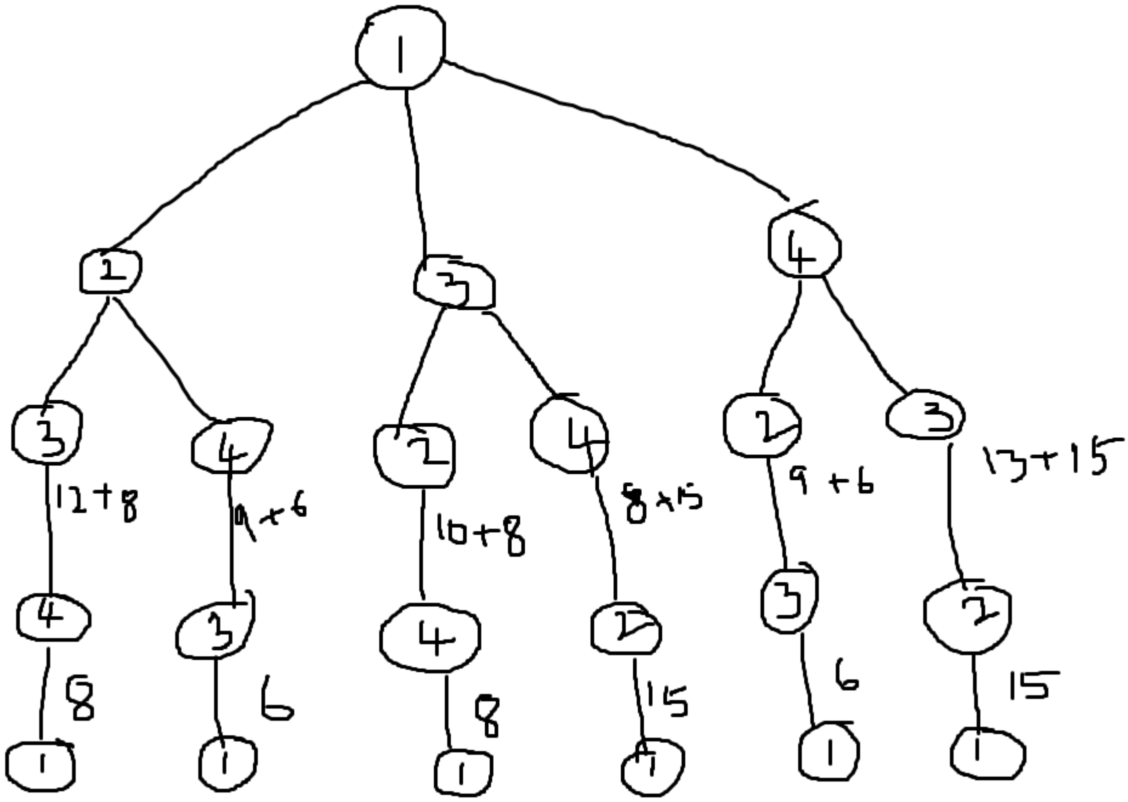
Travelling Sales Person problem

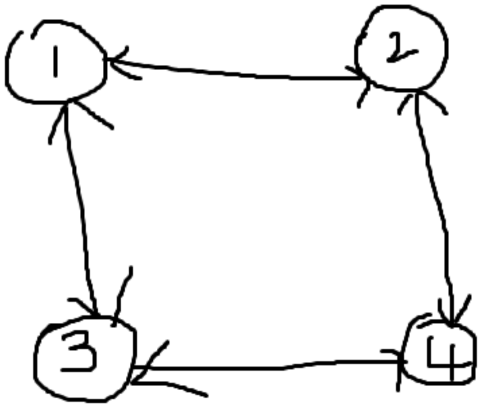
G=(V,E), source=1, V-{1}



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

Brute-force Method





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

g(1, {2,3,4}) = min { C1k + g{2,3,4}-k}

g(i,S)= min k € S { Cik + g (k, S-{k})}

g(2, ø)= C21= 5

g(3, ø)= C31=6

g(4, ø)= C41=8

g(2,{3})= k=3 {C23 + g(3, ø)}= 9+6=15

i=2, S= {3}-{3}

g(2,{4})= {C24 + g(4, ø)}= 10+8=18

g(3,{2})= {C32 + g(2, ø)}= 13+5=18

g(3,{4})= {C34 + g(4, ø)}= 12+8=20

g(4,{2})= {C42 + g(2, ø)}= 8+5=13

g(4,{3})= {C43 + g(3, ø)}= 9+6=15

g(2,{3,4})= min k=3 or **4** {C23 + g(3, {4}), C24 + g(4,{3})}

min {9+20, **10+15**}= 25

g(3,{2,4})= min k=2 or **4** {C32 + g(2, {4}), C34 + g(4,{2})}

min {13+18, **12+13**}= 25

g(4,{2,3})= min k=**2** or 3 {C42 + g(2, {3}), C43 + g(3,{2})}

min {8**+15**, 9+17}= 23

g(1,{2,3,4})= min k=**2**,3,4 { C12 + g(2,{3,4}), C13+ g(3,{2,4}), C14 + g(4,{2,3})}

=min{ **10+15**, 15+25, 20+23} = **35**

Path= 1 -> 2 -> 4 -> 3 -> 1

$$
\begin{array}{c c}
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 16 & 11 & 6 \\ 8 & 0 & 13 & 16 \\ 4 & 7 & 0 & 9 \\ 5 & 12 & 2 & 0 \end{bmatrix}
\end{array}
$$

g(1,{2,3,**4**})= min {C12 +g(2, {3,4}), C13+g(3,{2,4}), C14+ g(4,{2,3})}

=min{ 16+22, 11+28, **6+17**} **= 23**

Path- 1->4->3->2->1

g(2,ø)= C21= 8

g(3, ø)= C31= 4

g(4, ø)= C41=5

g(2, {3})= {C23 + g(3, ø)}= 13+ 4 =17

g(2,{4})= {C24 + g(4, ø)}= 16+5= 21

g(3,{2})= {C32 + g(2, ø)}= 7+8= 15

g(3, {4})= {C34 + g(4, ø)}= 9+5= 14

g(4,{2})= {C42 +g(2, ø)}= 12 +8=20

g(4, {3})= {C43 + g(3, ø)}= 2+4=6

g(2, {3,**4**})= min{C23 +g(3, {4}), C24+g(4,{3})}

=min{ 13+14, **16+6**}= 22

g(3, {**2**,4})= min(C32+g(2,{4}), C34+g(4,{2})}

=min{**7+21**, 9+20}= 28

g(4,{2,**3**})= min(C42+ g(2,{3}), C43+g(3,{2})}

=min{ 12+17**, 2+15**} = 17

Reliability Design Problem

In this section we will discuss the problem based on multiplicative optimization function. We will consider such a system in which many devices are connected in series, as

Devices Connected in series

When devices are connected together then it is a necessity that each device should work properly. The probability that device „i" will work properly is called reliability of that device.

$r_1=r_2=....=r_n= 0.99$

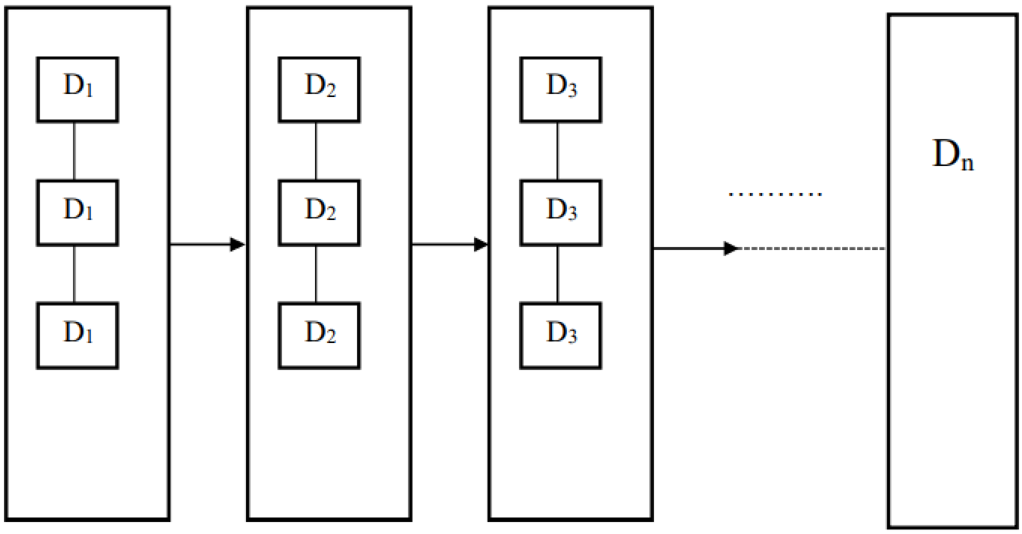
Reliability of the system where devices are connected in series=$\prod ri$

Suppose consider a system with three devices D1,D2,D3

Reliability = 0.99 * 0.99 * 0.99= 0.97

To make system more reliable, multiple copies of the same device is used at each stage. The system with multiple copies looks as given below

Multiple devices connected together



Probability of malfunction of a device = 1-relability

Let 3 copies of D1 are available with r1=0.99, all copies not working properly at stage 1= $(1-r_1)^3$ = $(1-0.99)^3$= 0.00001

All 3 copies working proper (reliability of stage 1)= 1- $(1-r_1)^3$ = 1- 0.00001 = 0.9999

Reliability design problem is to find minimal number of duplicate copies at each stage to maximize the reliability with in given cost constraints. Formally defined as

D1=C1,m1

D2=C2,m2

D3=C3,m3, C

C1 *m1 +C2*m2 + C3*m3 ≤ C

maximize $\prod_{1 \leq i \leq n} \Phi_i(m_i)$

subject to $\sum_{1 \leq i \leq n} C_i m_i \leq C$ where C_i is the cost of each device and C is the maximum allowable cost of the system.

$1 <= mi <= u_i$ and integer $1 \leq i \leq n$

First step: find the number of copies of each device at every stage by considering cost constraint. Let u_i be the number of copies of ith device and computed by

Ctotal= C1 + C2 + …Cn

C is the total cost given

Each device we have to consider atleast one copy so

Remaining cost= C- Ctotal

u_i=(C- Ctotal) /Ci

Second step: compute the reliability of stage 1 with different number of copies of D1

Third Step: when considering second stage, compute reliability by combining with first stage values. Repeat this until last stage.

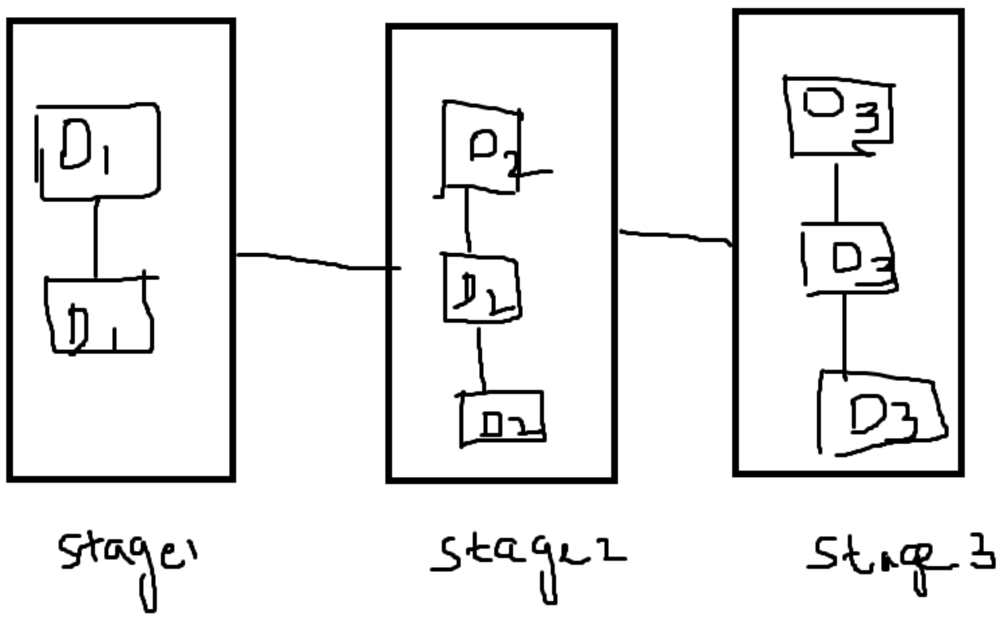
Ex: D1, D2, D3 (r1, r2,r3)= (0.9, 0.8, 0.5), (c1,c2,c3)= (30,15,20), **C=105**

u1= floor((105- (30+15+20))/30) =1.3=2 copies of D1

u2=floor(40/15)=2.6=3

u3=40/20=3

system will look like



$\mathbf{S^1_1}$= {(0.9, 30)}

S^1_2= {(0.99, 60)}

Reliability with two copies= $1-(1-0.9)^2$ = 1- 0.01= 0.99

$\mathbf{S^1}$**= {(0.9, 30), {(0.99, 60)}**

S^2_1= {(0.8, 15)}

$\mathbf{S^2_2}$= {(0.96, 30)} $(1-(1-0.8)^2)$= 0.96)

S^2_3= {(0.992, 45)} $(1-(1-0.8)^3)$= 0.992)

Only stage 2= **{(0.8, 15), (0.96, 30), (0.992, 45)}**

Combine stage 2 with stage 1 to get reliability after stage 2 i.e. S^2

S^2 = { (0.9 * 0.8, 30+15), (0.9 * 0.96, 30+30), (0.9* 0.992, 30+45),

(0.99 * 0.8, 60+15), **(0.99 * 0.96, 60+30)**, **(0.99* 0.992, 60+45)**}

S^2 = {(0.72, 45), (0.864,60), **(0.792,75),** (0.892,75)} (purging rule)

S^2 = {(0.72, 45), (0.864,60), (0.892,75)}

S^3_1= {(0.5, 20)}

S^3_2= {(0.75, 40)} (1-(1-0.5)^2)= 0.75)

S^3_3= {(0.875, 60)} (1-(1-0.5)^3)= 0.875)

Only stage 3 = { (0.5, 20), (0.75,40) (0.875,60)}

Combining stage 3 with S^2 gives reliability at stage 3

S^3= { (0.36, 65), (0.432,80), (0.44, 95), (0.63,85), (0.648,100), **(,115), (,110), (,120), (,135)}**

S^3= { (0.36, 65), (0.432,80), (0.44, 95), (0.63,85), **(0.648,100)**}

Maximum reliability= 0.648

(0.648,100) is obtained by multiplying (0.864,60) and {(0.75, 40)}

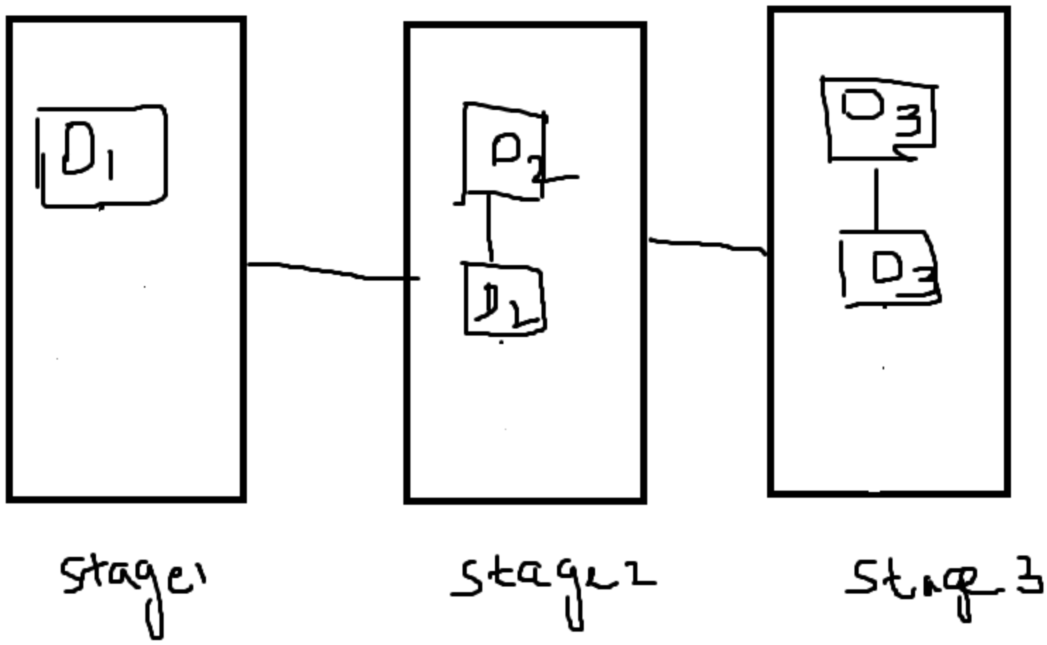
So at stage 3 two copies of D3 will be used

(0.864,60) is obtained by multiplying (0.96,30) and (0.9, 30)

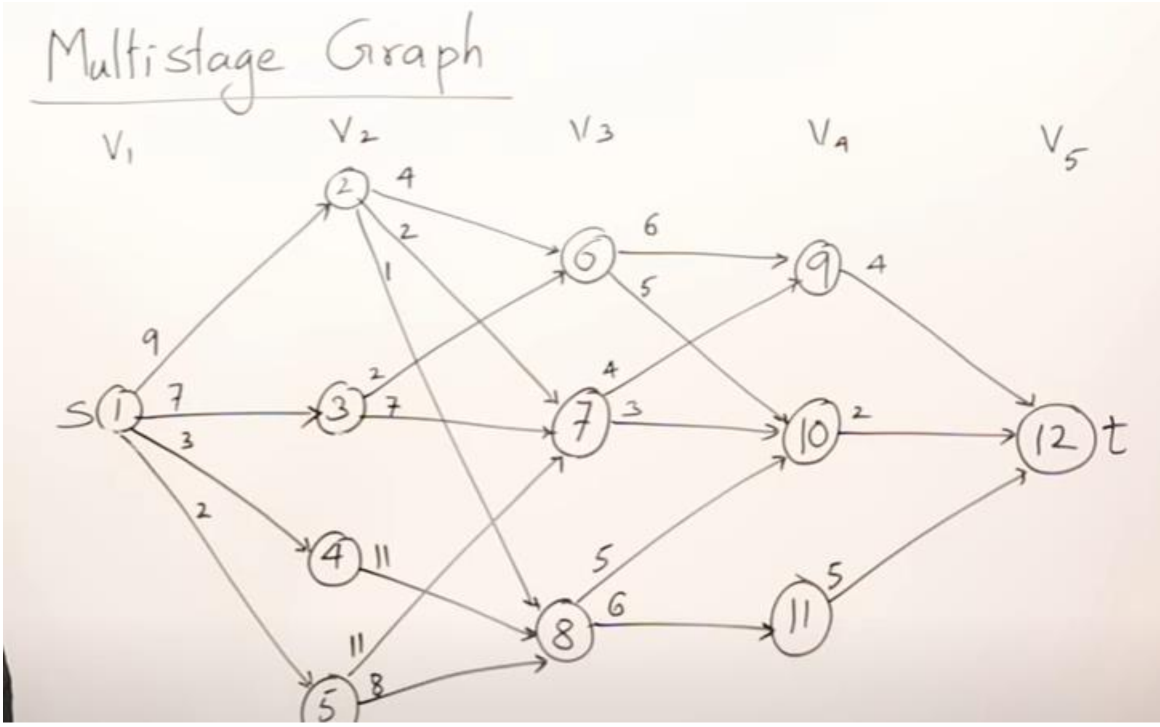
So at stage 2 two copies of D2 will be used

Remaining cost= 100- 40-30= 30

So stage 1 one copy of D1 will be used. So the final system is



D1
D2
D1
D3
D3
stage1
stage2
stage 3



Multi-stage graph is a directed weighted graph G= (V,E)

Vertices are divided into disjoint sets V1, V2, …..Vk

Edges E € (Vi, Vi+1)

Find the minimum cost path from source to sink

Two approaches are there

1. Forward method
2. Backward method

Forward Method:

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

Cost(stage, vertex) need to be calculated from 5th stage

Cost(5, 12)=0

Cost(4, 9) = 4

Cost(4,10)= 2

Cost(4,11)= 5

Cost(3, 6) = min l= 9,**10** { C(6,9) + cost (4,9), C(6,10) + cost(4,10)}

= min {6+4, **5+2**} = 7

Cost(3, 7) = min { C(7,9) + cost (4,9), C(7,10) + cost(4,10)}

= min {4+4, **3+2**} = 5

Cost(3, 8) = min { C(8,10) + cost (4,10), C(8,11) + cost(4,11)}

= min {**5+2**, 6+5} = 7

Cost(2,2)= min l € stage 3 = 6,**7**,8 { C(2,6) + cost (3,6), C(2,7) +cost(3,7), C(2,8) +cost(3,8) }

=min { 4+7, **2+5**, 1+7} = 7

Cost(2,3)= min l € stage 3 = **6**,7 { C(3,6) + cost (3,6), C(3,7) +cost(3,7}

=min { **2+7**, 7+5} = 9

Cost(2,4)= min { C(4,8) +cost(3,8) }

=min { 11+7} = 18

Cost(2,5)= min { C(5,7) +cost(3,7), C(5,8) +cost(3,8) }

=min {11+5, **8+7**} = 15

Cost(1,1)= min l € stage 2 = **2,3**,4,5 { C(1,2) + cost(2,2), C(1,3) + cost(2,3), C(1,4) + cost(2,4), C(1,5) + cost(2,5)}

= min {**9+7, 7+9**, 3+18, 2+15} = **16**

Cost(i,j) = min l € i+1 { C(j,l) + cost(i+1, l) }

i-   Stage, j- vertex , l- vertex at next stage

d(1,1)= 2

d(2,2)= 7

d(3,7) = 10

d(4,10)=12

so shortest path from source to sink = 1->2 ->7->10->12

Backward method:

Cost(i,j) = min l € i-1 { C(l,j) + cost(i-1, l) }

i- Stage, j- vertex , l- vertex at previous stage

Cost(1,1)=0

Cost(2,2)= 9

Cost(2,3)= 7

Cost(2,4)=3

Cost(2,5)=2

Cost(3,6)= min { C(2,6)+cost(2,2), C(3,6)+cost(2,3)}

=min {4+9, **2+7**}= 9

Cost(3,7)= min { C(2,7)+cost(2,2), C(3,7)+cost(2,3), C(5, 7) + cost(2,5) }

=min {**2+9**, 7+7, 11+2}= 11

Cost(3,8)= min { C(2,8)+cost(2,2), C(4,8)+cost(2,4), C(5, 8) + cost(2,5) }

= min { **1+9**, 11+3, **8+2**}= 10

Cost (4, 9) = min{C(6,9) +cost(3,6), C(7,9) + cost(3,7)}

= min {**6+9, 4+11**} = 15

Cost (4, 10) = min{C(6,10) +cost(3,6), C(7,10) + cost(3,7)}

= min {**5+9, 3+11**} = 14

Cost(4,11)= c(8,11) + cost(3,8) = 6+ 10= 16

Cost (5,12) = min { C(9,12) + cost (4,9), C(10,12) + cost (4,10), C(11,12) +cost(4,11) }

= min { 4+15**, 2+14**, 5+16} = 16

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	0	9	7	3	2	9	11	10	15	14	16	16
d	1	1	1	1	1	3	2	2/5	6/7	6/7	8	10

d(5,12)= 10

d(4,10)= 6

d(3,6) =3

d(2,3)= 1

so shortest path with accost of 16 = 12<-10<-6<-3<-1

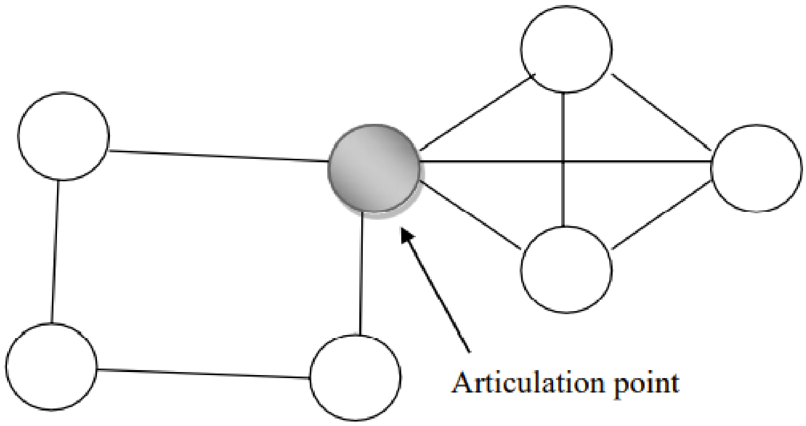
Bi-connected Components

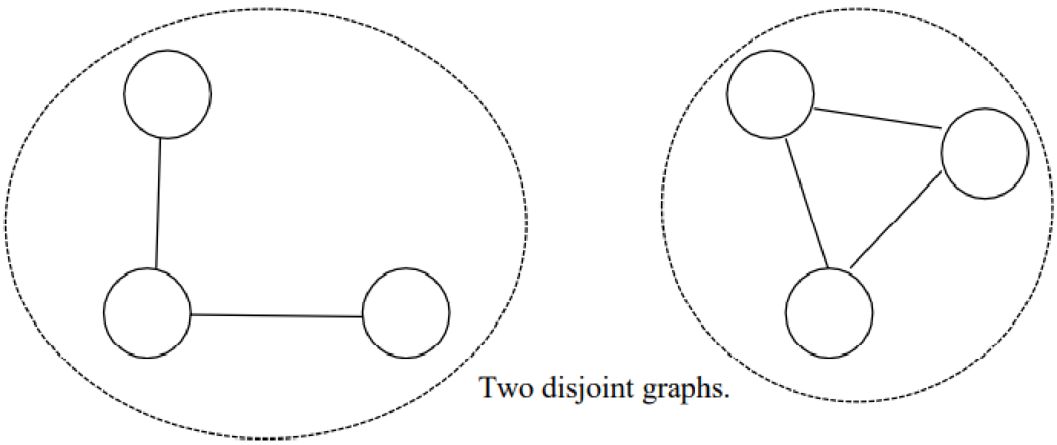
A "graph" means always an undirected graph. A vertex v in a connected graph G is an articulation point if and only if the deletion of vertex v together with all edges incident to v disconnects the graph into two or more nonempty components.

A graph G is biconnected if and only if it contains no articulation points.

Definition of Articulation point: Let G = (V, E) be a connected undirected graph, then an articulation point of graph G is a vertex whose removal disconnects graph G. This articulation point is a kind of cut-vertex.

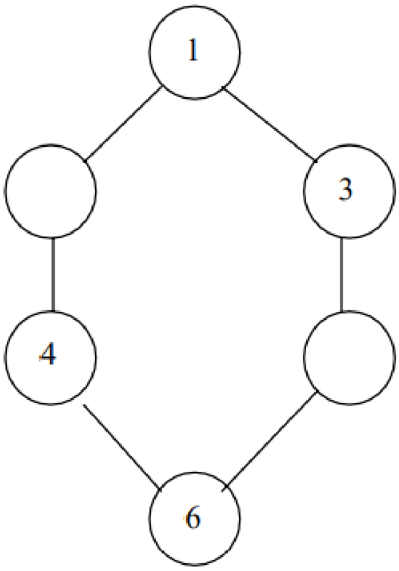
Eg:





A graph G is said to be bi-connected if it contains no articulation points.

Eg:



Identification of Articulation Point

1. Build DFS tree of the given graph G
   While building the DFS tree we can classify the tree edges into four categories:
   i) Tree edge: It is an edge in depth first search tree.
   ii) Back edge: It is an edge (u, v) which is not in DFS tree and v is an ancestor of u. It basically indicates a loop.
   iii) Forward edge: An edge (u, v) which is not in search tree and u is an ancestor of v.
   iv) Cross edge: An edge (u, v) not in search tree and v is neither an ancestor nor a descendant of u.
2. Find the lowest dfn (L) reachable from each vertex by taking one back edge
3. Find Articulations point by
   i) The root of the DFS tree is an articulation if it has two or more children.
   ii) A leaf node of DFS tree is not an articulation point.

iii) If u is any internal node then it is not an articulation point if and only if from every child w of u it is possible to reach an ancestor of u using only a path made up of descendants of w and back edge.

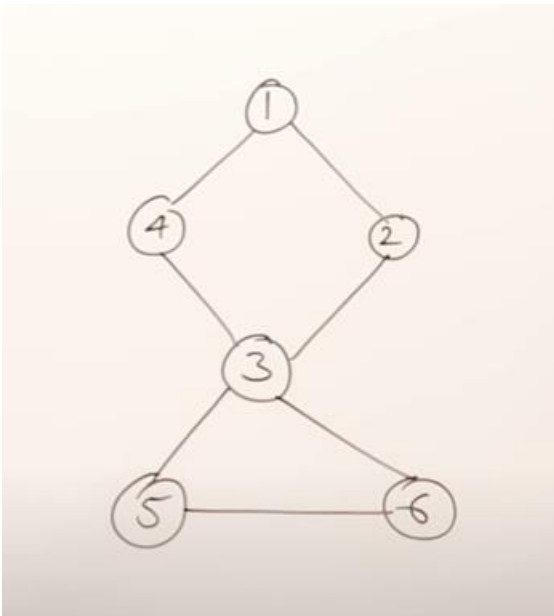
This observation leads to a simple rule as,

$$Low[u] = min \ \{ \ dfn[u], \ min\{Low[w]/w \text{ is a child of } u\}, \ min\{dfn[w]/(u, w) \text{ is a back edge}\}\}$$

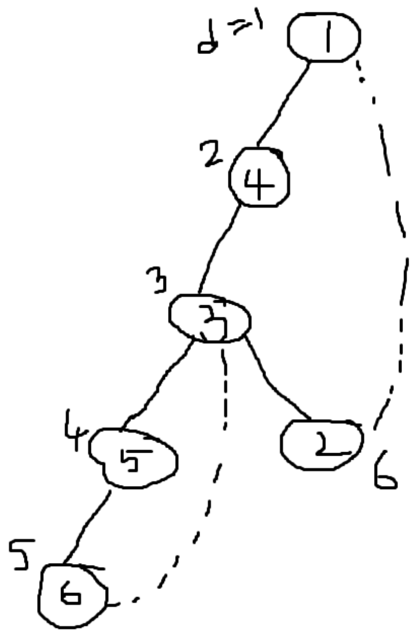
Where Low[u] is the lowest depth first number that can be reached from u using a path of descendants followed by at most one back edge. The vertex u is an articulation point if u is child of w such that

$$L[w] \geq dfn[u]$$

Eg:



1. DFS tree



V	1	2	3	4	5	6
Dfn	1	6	3	2	4	5
L	1	1	1	1	3	3

2. Find L for each vertex

Low[u] = min { dfn[u], min{Low[w]/w is a child of u}, min{dfn[w]/(u, w) is a back edge}}

L[6]= min {5, dfn(3)}= {5,3} = 3

L[2]= min {dfn(2), dfn(1)} = {6,1} = 1

L[5]= min { dfn(5), L(6) } = (4,3} = 3

L[3]= min { dfn(3), min (L(5), L(2))} = {3,1} = 1

L[4]= min { dfn(4), L(3) }= {2,1} = 1

L[1]= min {dfn(1), L(4)} = {1,1} = 1

3. Identify Articulation point

The vertex u is an articulation point if u is child of w such that

$L[w] \geq dfn[u]$

u=4, w=3

$L[w] \geq dfn[u]$ is true

$L[3] \geq dfn(4) = 1 \geq 2$ is false , 4 is not an articulation point

u=3, w=5

$L[5] \geq dfn[3]$

$3 \geq 3$ is true so 3 is an articulation point

u=5, w=6

$L[6] \geq dfn(5)$

$3 \geq 4$ is false , 5 is not an articulation point

Algorithm for articulation points:

```
Algorithm Art(u, v)
//The vertex u is a starting vertex for depth first search. V is its parent if any in the depth first
//spanning tree. It is assumed that the global array dfn is initialized to 0 and that the global
//variable num is initialized to 1. N is the number of vertices in G.

   {
    dfn[u] := num; L[u] := num; num := num+1;

    for each vertex w adjacent from u do
    {

     if(dfn[w] = 0) then
     {

      Art(w, u); // w is unvisited
      L[u] := min(L[u], L[w]);
     }
     else if(w≠v) then L[u] := min(L[u], dfn[w]);
   }
 }
```

Analysis: The Art has a complexity O(n+e) where e is the number of edges in G, the articulation points of G can be determined in O(n+e) time.

Identification of Bi-connected components

• A bi-connected graph G = (V, E) is a connected graph which has no articulation point.

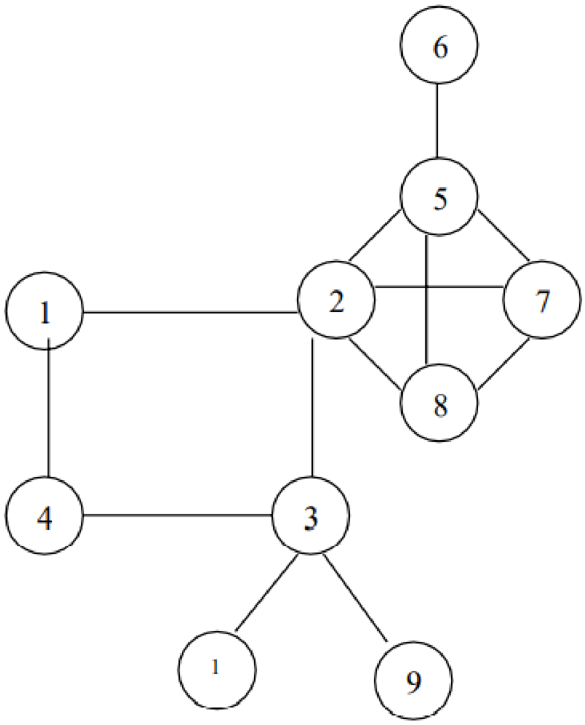
• A bi-connected component of a graph G is maximal biconnected sub graph. That means it is not contained in any larger bi-connected sub graph of G.

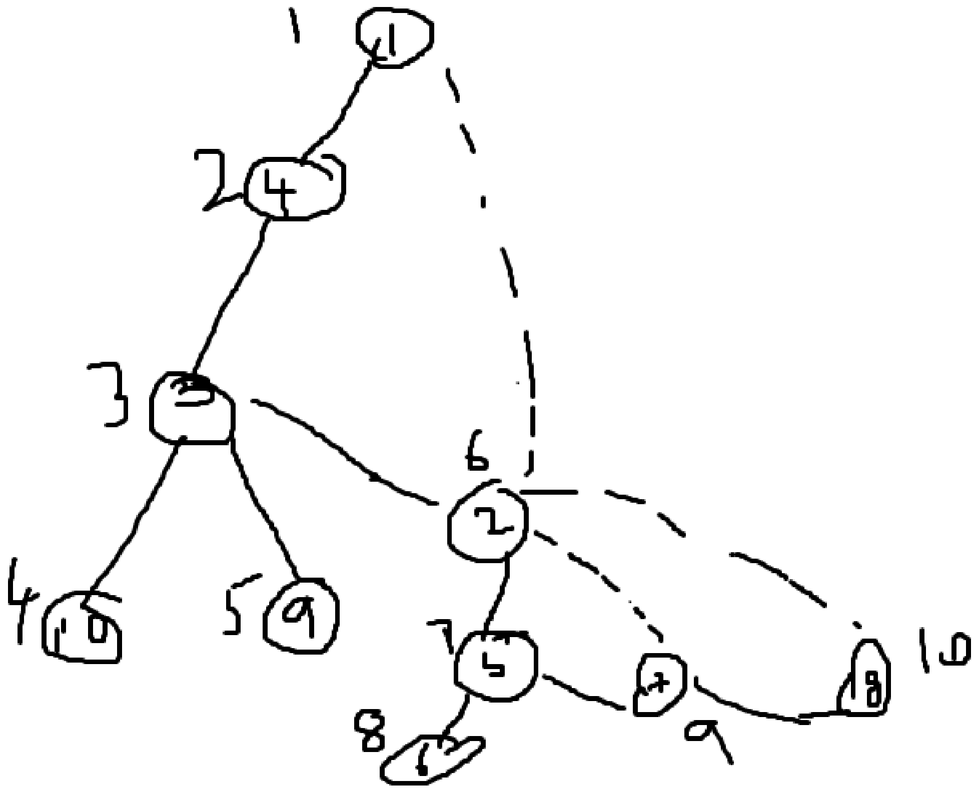
• Some key observations can be made in regard to bi-connected components of graph.

i) Two different bi-connected components should not have any common edges.

ii) Two different bi-connected components can have common vertex.

iii) The common vertex which is attaching two (or more) bi-connected components must be an articulation point.





V	1	2	3	4	5	6	7	8	9	10
Dfn	1	6	3	2	7	8	9	10	5	4
L										

L[10]=dfn[10]= 4

L[9] = dfn[9]= 5

L[6]= 8

L[8]= min {10, dfn(2)} = min {10,6}= 6

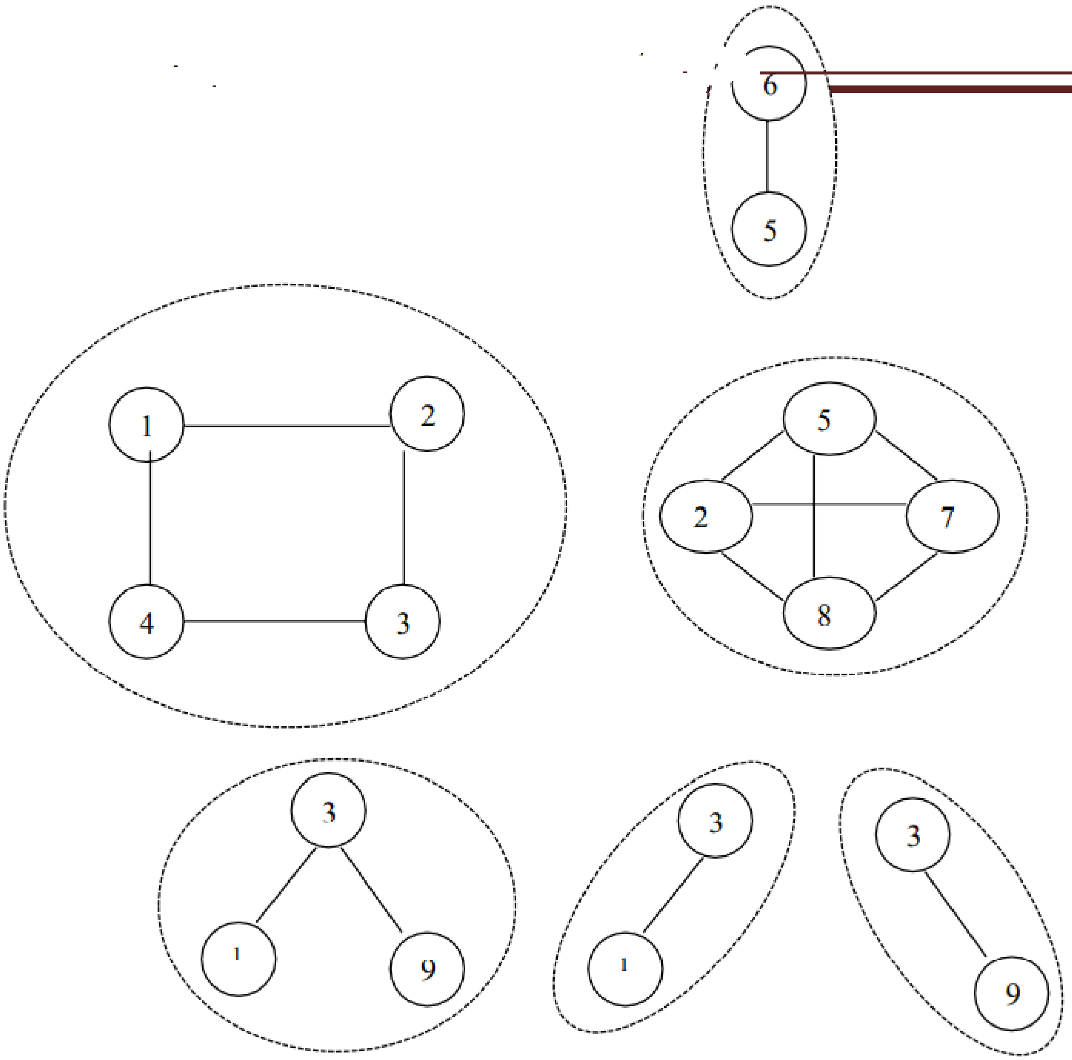
L[3]= min {dfn(3), min (4,5,1)}= 1

L(7)= min {dfn(7), L(8), dfn(2)}= {9,6,6}= 6

u=3, w=10

L[10] ≥ dfn(3)

$4 \geq 3$ = true



Algorithm for Bi-connected components Algorithm BiComp(u, v)

// u is a start vertex for depth first search. v is its parent if any in the depth first spanning tree. It //is assumed that the global array dfn is initialized to zero and that the global variable num is //initialized to 1. n is the number of vertices in G.

{

dfn[u] := num;

L[u] := num;

num := num+1;

for each vertex w adjacent from u do

{

if((v≠w) and (dfn[w] < dfn[u])) then

add (u, w) to the top of a stack s;

if (dfn[w] = 0) then

```
{
        if (L[w] ≥ dfn[u]) then
        {
                write ("New bicomponent");
                repeat
                {
                        Delete an edge from the top of stack s;
                        Let this edge be (x, y);
                        write (x, y);
                } until (((x, y) = (u, w)) or ((x, y) = (w, u)));
        }
        BiComp(w, u); // w is unvisited
        L[u] := min(L[u], L[w]);
}
else if (w≠v) then L[u] := min(L[u], dfn[w]);
   }
}
```