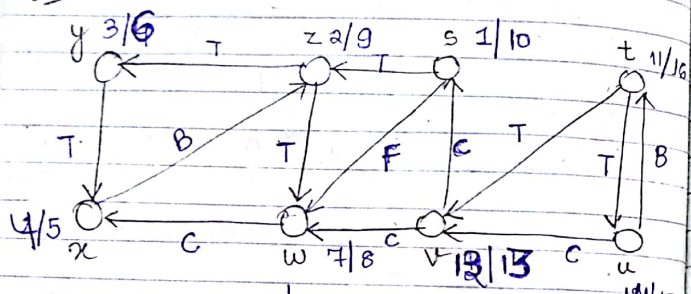
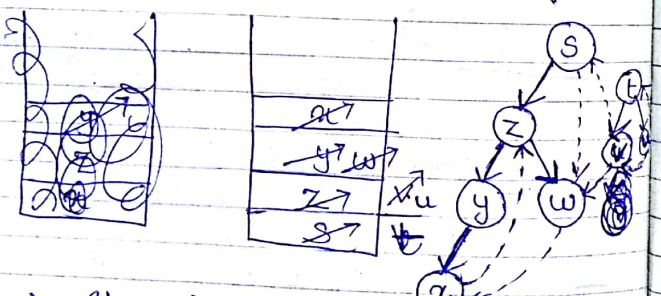
→ The graph having back edge that is a cycle. (after applying DFS).

18/9/18



x → z	s → z, w
y → x	v → w, s
z → y, w	u → v, t
w → x	t → v, u



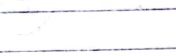
Note :- Starting from root, if different paths then cross edge.

19/9/18

Topological sorting

Applicable for Directed acyclic graph (DAG).
(all edges have direction and it is acyclic)

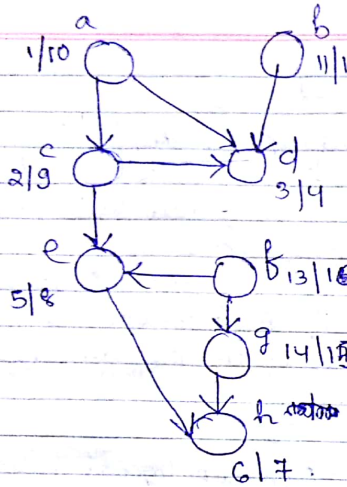
If a graph G is DAG, then topological sorting of the vertices V is a linear ordering of V such that each edge (u, v) in a graph u appears before v in the linear ordering.

(u, v)  (in topological ordering u should come before v).

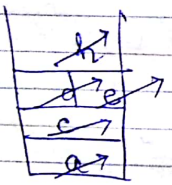
u will have finishing time greater than v when we apply DFS.

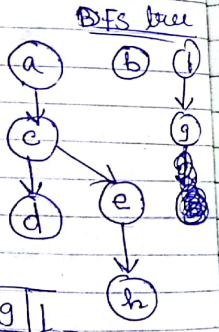
Topological Sort (G).

1. DFS(G), compute f[v] // where f[v] = finishing time of v.
2. As each vertex finished, put it in the front of the list.
3. Output



a = c d
b = d
c = d e
d = –
e = h
f – e g
g – h.
h – –





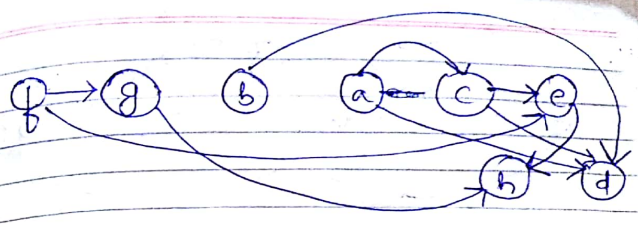
Linked list | d | h | e | c | a | b | g | f |

(see in reverse manner)

i.e.,

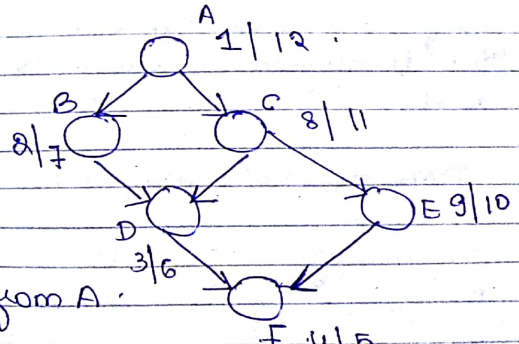
f g b a c e h d

(As soon as the node is getting finished add it to the front of the linked list.

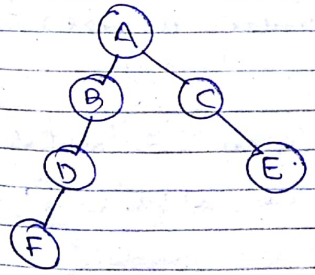


All the edges should go from left to right. If it moves from right to left then our topological sorting is wrong.

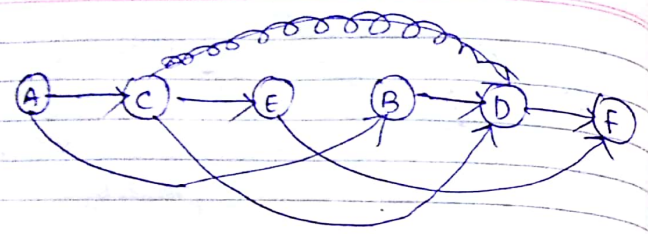
Q.



start from A.



| A | C | E | B | D | F |

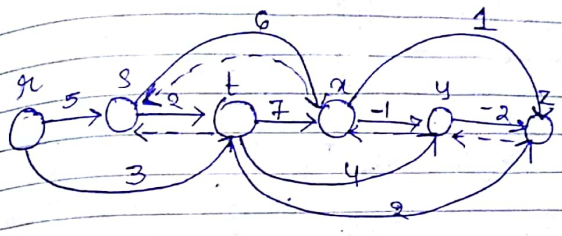


This is the topological sorting.

One application of topological sorting is single source shortest path of DAG (we can use dijkstra), But topological sorting is the better method.

DAG - Shortest Path (G, w, s).

① Topological - Sort (G)
② Initialize - Single source (s).
③ for each vertex u, taken in topological ordering
{
for each v ∈ Adj[u]
do Relax (u, v, w)
}



let us say s is the source.

Initialize - single source (s)

	s	r	t	x	y	z
d[v]	0	∞	∞	∞	∞	∞	∞
π[v]	nil	nil	nil	nil	nil	nil	nil

① s
d[t] = 2.
π[t] = s.
d[x] = 6
π[x] = t.

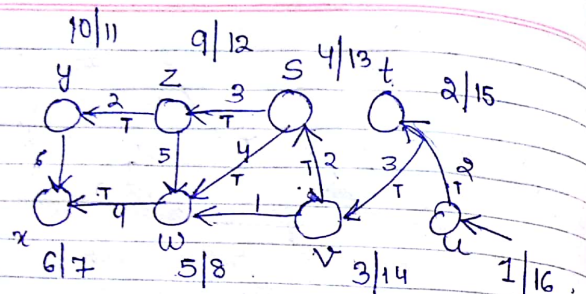
② t
d[y] = 6.
π[y] = t.
d[z] = 4.
π[z] = t.

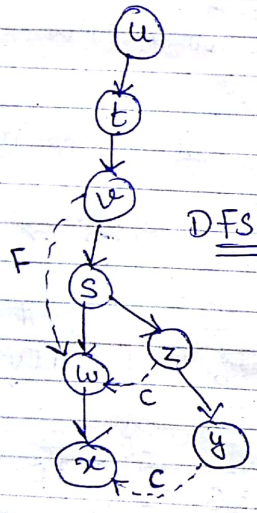
Shortest path b/w s to z is s → x → y → z

③ x
d[y] = 6 - 1 = 5.
π[y] = x.
z
d[z] = 6 - 2 = 4 (✗)

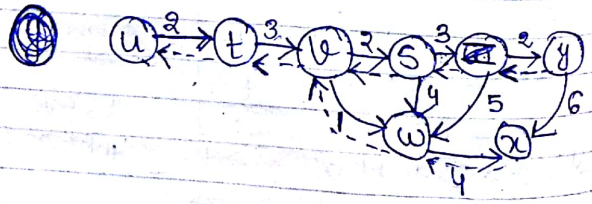
④ y
d[z] = 5 - 2 = 3
π[z] = y

Q





DFS tree.



① u	② t	③ v.
d[t] = 2	d[v] = 5.	d[s] = 7.
π[t] = u	π[v] = t.	π[s] = v
		d[w] = 6.
		π[w] = v
④ s	⑤ z	⑥ w
d[z] = 10	d[y] = 12	d[x] = 10
π[z] = s.	π[y] = z	π[x] = w.

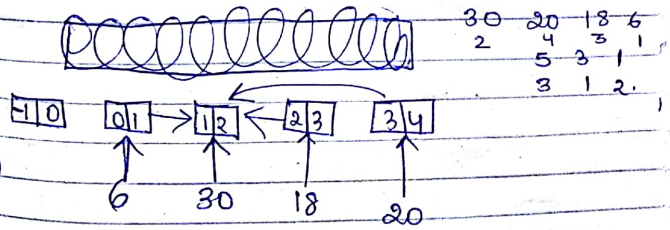
20/9/18

Tutorial - 4.

Q n = 7.

$(p_1, p_2, \cdots p_7) = (3, 5, 20, 18, 1, 6, 30)$.

$(d_1, d_2, \cdots d_7) = (1, 3, 4, 3, 2, 1, 2)$.



6	30	18	20

(1)

1	2	3	4
6	30	18	20

1	2	3	4
6	30	18	20

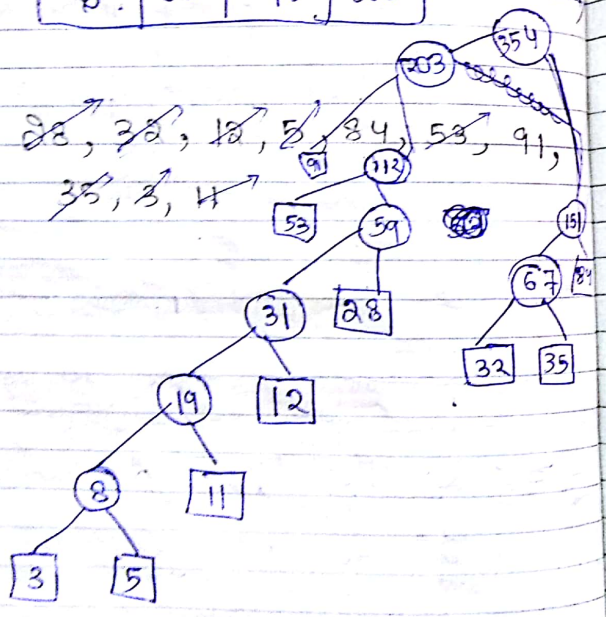
(Forward)

(2)

1	2	3	4
6	30	18	20

(Backward)

Q. 22, 30, 12, 5, 84, 53, 91, 35, 3, 4



1083
1183

= 182 + 159 + 112 + 60 + 66 + 35 + 21 + 252 + 128 + 126

= 1182

= 3×32 + 3×35 + 2×84 + 2×91 + 3×53 + 28×4 + 5×12 + 6×11 + 7×5 + 7×3

= 96 + 105 + 168 + 182 + 159 + 60 + 66 + 112 + 35 + 21

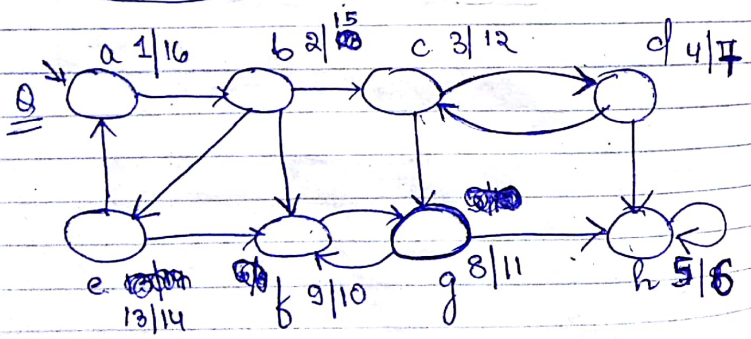
= 1004 → Ans.

25/9/18

Problems to find out the strongly connected components of a graph.

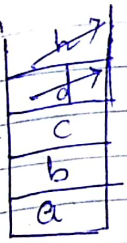
Strongly connected component of a directed graph G which consist of set of vertices and set of edges is a maximal set of vertices C which is a subset of V such that for every pair of vertices u & v in C we have both path from u to v and v to u. such that u and v are reachable from each other.

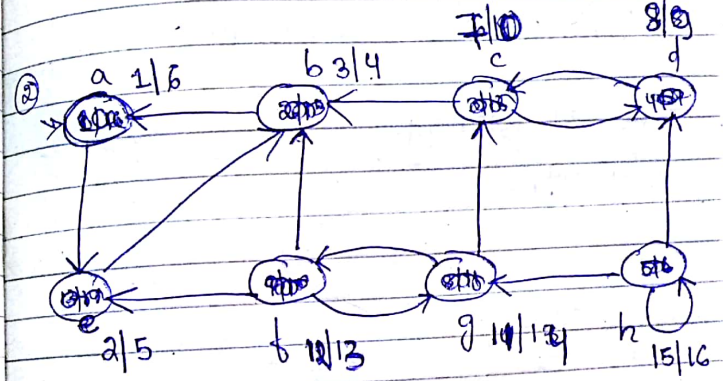
$G(V, E)$
$C \subseteq V$



① Call DFS and compute the finishing time.
② Compute transpose of graph G^T.

Source a.

① 

② 

③ To call DFS for transpose of a graph Call DFS (G^T) but consider vertices in order of decreasing finishing time.

④ The strongly connected components:- take the start time of components till there is a break in the start time.

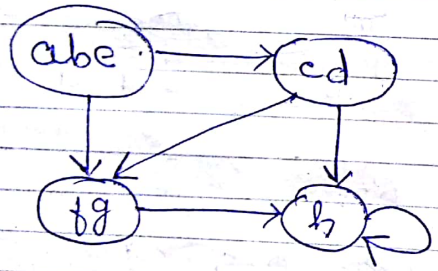
abe ① component.

cd ② component.

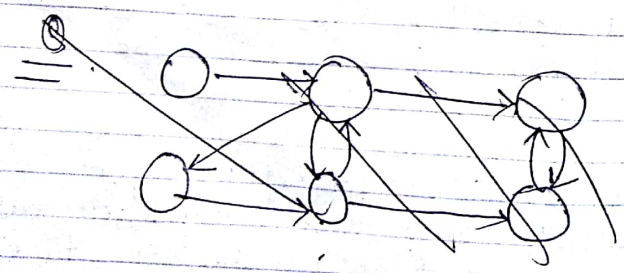
fg ③ component.

h ④ component.

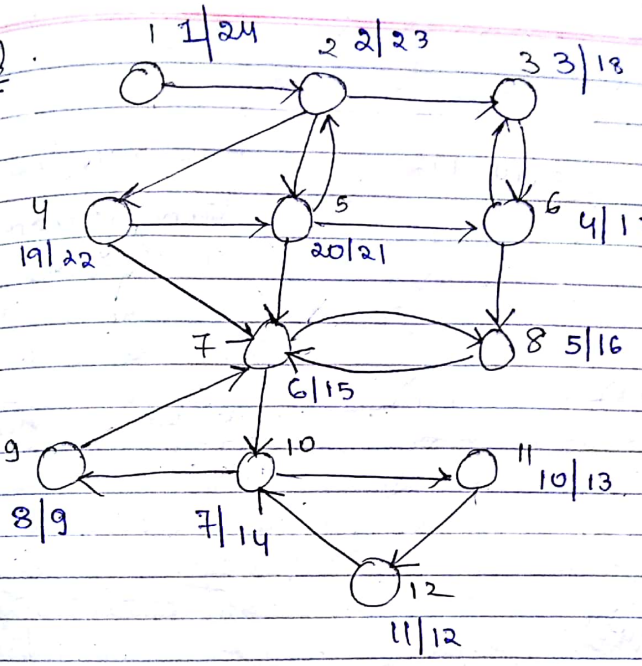
I got 4 strongly connected components of graph.



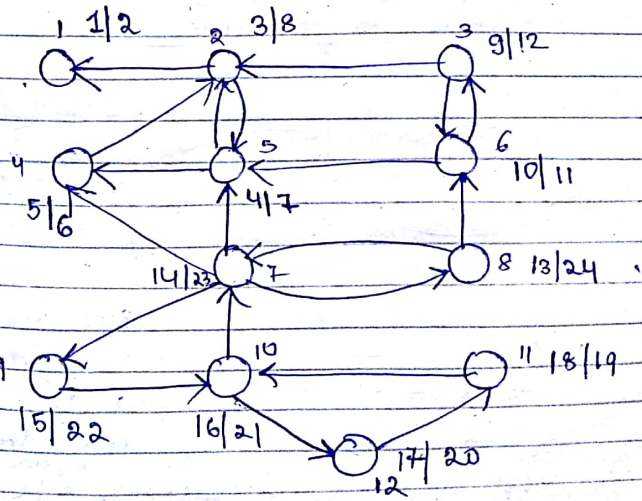
Q.

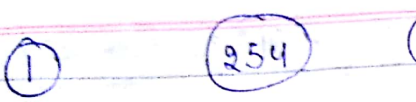


Q.

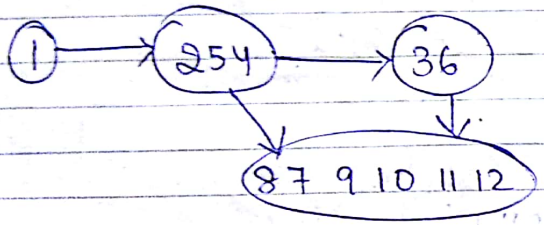


$(G)^T =$



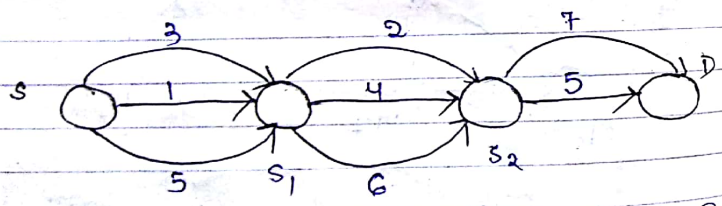


These are the strongly connected components.



26/9/18.

Dynamic Programming



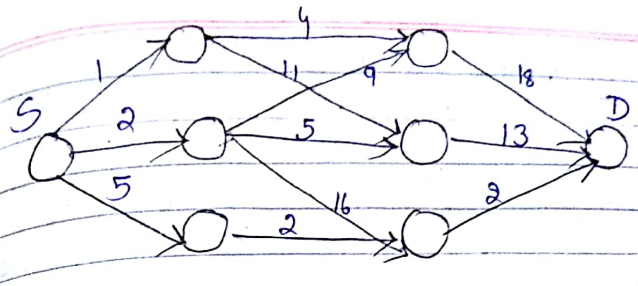
$s \to s_1 = 1$

$s_1 \to s_2 = 4$

$s_2 \to D = 5$

Shortest path = 1+4+5 = 10

{ greedy approach as problem are independent of one another.



According to greedy approach shortest path = 1+4+18 = 23

But actually shortest path is 5+2+2 = 9.

So for this type of prob we use dynamic programming.

Two parts of greedy programming.

① Optimal Substructure.

② Overlapping Subproblems.

A problem exhibit optimal substructure if an optimal solution to the problem contains within it optimal solution to the subproblem.

Overlapping subproblems :- All the subproblems are dependent on one another.

0/1 Knapsack problems

Item	Value	Weight	
1	1	1	1
2	6	5	1.2
3	18	5	3.6
4	22	6	3.5
5	28	7.	4

Capacity = 11.

According to greedy approach

According to P/W in decreasing order.

28	18	22	6	1
7	5	6	5	1

According to greedy approach with 0/1 Knapsack problem.

5 & 6 will be the ideal case (As it will give max. profit).

With 7 there will be no combination in the problem to make Capacity = 11

OPT(i)
= Max profit of subset of item 1 .... i.

eg:- OPT(5) = 1, 2, 3, 4, 5 items's maximum profit.

OPT(2) = 1, 2 (we are comparing 1, 2 item to find maximum profit)

Opt(i, w)
= Max Profit of subset of item 1 ... i with weight limit w.

Case 1 → We don't choose i
we can choose item (1...i-1) with weight limit w.

Case 2 → We choose i item.
We can choose item from (1...i-1) with weight limit (w - w_i).

$$opt(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ Opt(i-1, w) & \text{if } w_i > w. \\ \max\{opt(i-1, w), & \text{if } w_i \le w \\ v_i + opt(i-1, w-w_i)\} & \text{(otherwise)} \end{cases}$$

Knapsack 0/1

Input → n, $W_1 \cdots W_n$, $V_1 \cdots V_N$

```
for w = 0 to W
    M[0, w] = 0
for i = 0 to n   M[i, 0] = 0;
for i = 1 to n
    for w = 1 to W
        if (wi > w)
            M[i, w] = M[i-1, w]
        else
            M[i, w] = max{M[i-1, w],
                          vi + M[i-1, w-wi]}
```

Return M[n, W]

	0	1	2	3	4	5	6	7	8	9	10	11
{1}	0	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	1	6	7	7	7	7	7	7	7	7	7
{1,2,3}	0	1	6	7	7	18	19	24	25	25	25	25
{1,2,3,4}	0	1	6	7	7	18	22	24	28	29	29	40
{1,2,3,4,5}	0	1	6	7	7	18	22	28	29	34	35	40

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

The maximum profit is 40.

Pseudo code for seeing which item is in knapsack

```
let i = n    k = W
if M[i, k] ≠ M[i-1, k]
    mark ith item in knapsack.
    i = i-1,  k = k - wi.
else
    i = i-1.
```

4th, 3rd item is in knapsack according to the above code.

① $i = 5$ $k = 11$.
$M[5,11] \neq M[4,11]$ (false)
$i = 4$.

② $M[4,11] \neq M[3,11]$ (T)
4th in knapsack
$i = 3$; $k = 11-6 = 5$.

③ $M[3,5] \neq M[2,5]$ (T)
3rd in knapsack.
$i = 2$; $k = 5-5 = 0$

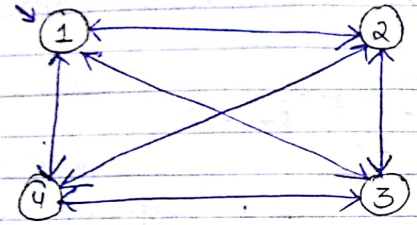
④ $M[2,0] \neq M[1,0]$ (X)
$i = 1$.

⑤ $M[1,0] \neq M[0,0]$
$i = 0$ (F)

So finally we have 4th & 3rd in knapsack

28/9/18

Travelling Salesman Problem



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

Let G is a directed graph with set of vertices V and set of edges E and edge cost
c_{ij} → cost of edge b/w i and j (starting from i)
$c_{ij} > 0$.
else $c_{ij} \to \infty$ $(i,j) \notin E$ (no edge b/w i and j)

We will define a tour,
A tour of a G is a directed simple cycle that includes every vertex in V.

simple cycle :- If starting from SN then coming back to SN

$g(i, S)$ → we will start from i going through all the vertices in subset of S and coming back to source node.

$g(2, \{3,4\})$ → Starts from 2 travelling 3 and 4 terminating at 1.

We are finally looking for :-

$g(1, V-\{1\})$.

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

$$g(i,S) = \min_{j \in S} (c_{ij} + g(j, S-\{j\}))$$ → eq ①

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

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

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

$g(2, \{3\}) = C_{23} + g(3, \phi) = 15$

$g(2, \{4\}) = 18$

$g(3, \{2\}) = 18$

$g(3, \{4\}) = 20$

$g(4, \{2\}) = 13$

$g(4, \{3\}) = 15$

$g(2,\{3,4\}) = \min(c_{23} + g(3,\{4\}), c_{24} + g(4,\{3\}))$
$= \min(9+20, 10+15) = 25$

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

$g(4,\{2,3\}) \quad \min(8+15, 9+18)$
$= 23$

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

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

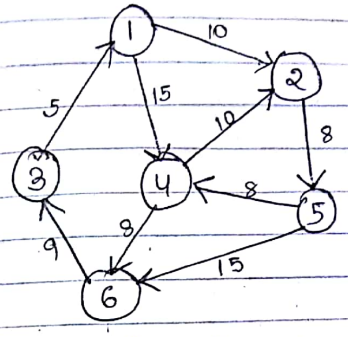
$= \min(35, 40, 43)$

$= 35.$

A tour of this length can be constructed if we retain with each $g(i,S)$ the value of j that minimizes the RHS of equation (1).

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

Q



	1	2	3	4	5	6
1	0	10	∞	15	∞	∞
2	∞	0	∞	∞	8	∞
3	5	∞	0	∞	∞	∞
4	∞	10	∞	0	∞	8
5	∞	∞	∞	8	0	15
6	∞	∞	9	∞	∞	0

$g(2,\phi) = \infty$
$g(3,\phi) = 5$
$g(4,\phi) = \infty$
$g(5,\phi) = \infty$
$g(6,\phi) = \infty$

$g(2,\{3\}) = \infty$
$g(2,\{4\}) = \infty$
$g(2,\{5\}) = \infty$
$g(2,\{6\}) = \infty$

$g(3,\{2\}) = \infty$
$g(3,\{4\}) = \infty$
$g(3,\{5\}) = \infty$
$g(3,\{6\}) = \infty$

$g(4,\{2\}) = \infty$
$g(4,\{3\}) = \infty$
$g(4,\{5\}) = \infty$
$g(4,\{6\}) = \infty$

$g(5,\{2\}) = \infty$
$g(5,\{3\}) = \infty$
$g(5,\{4\}) = \infty$
$g(5,\{6\}) = \infty$

$g(6,\{2\}) = \infty$ $g(6,\{3\}) = 14$ $g(6,\{4\}) = \infty$ $g(6,\{5\}) = \infty$

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

$g(2,\{3,5\}) =$

$g(2,\{3,6\}) =$

$g(2,\{4,5\}) =$

$g(2,\{4,6\}) =$

$g(2,\{5,6\}) =$

$g(3,\{2,4\}) =$
$g(3,\{2,5\}) =$
$g(3,\{2,6\}) =$
$g(3,\{4,5\}) =$
$g(3,\{4,6\}) =$
$g(3,\{5,6\}) =$

$g(4,\{2,3\}) =$
$g(4,\{2,5\}) =$
$g(4,\{2,6\}) =$
$g(4,\{3,5\}) =$
$g(4,\{3,6\}) =$
$g(4,\{5,6\}) =$

$g(5,\{2,3\}) =$
$g(5,\{2,4\}) =$
$g(5,\{2,6\}) =$
$g(5,\{3,4\}) =$
$g(5,\{3,6\}) =$
$g(5,\{4,6\}) =$

$g(6,\{2,3\}) =$
$g(6,\{2,4\}) =$
$g(6,\{2,5\}) =$
$g(6,\{3,4\}) =$
$g(6,\{3,5\}) =$
$g(6,\{4,5\}) =$

$g(2,\{3,4,5\}) =$

$g(2,\{3,4,6\}) =$

$g(2,\{4,5,6\}) =$

$g(2,\{3,5,6\}) =$

03/10/18.

Longest common subsequence.

		yj	B	D	C	A	B	A
	xi	0	0	0	0	0	0	0
A		0	0↑	0↑	0↑	1↖	1←	1↖
B		0	1↖	1←	1←	1↑	2↖	2←
C		0	1↑	1↑	2↖	2←	2↑	2↑
B		0	1↖	1↑	2↑	2↑	3↖	3←
D		0	1↑	2↖	2↑	2↑	3↑	3↑
A		0	1↑	2↑	2↑	3↖	3↑	4↖
B		0	1↖	2↑	2↑	3↑	4↖	4↑

longest common subsequence = 4.

A B C B D A B
B D C A B A

→ BCBA is the BDAB longest Common subsequence

It is of length 4.

We are comparing any x[i] with y[j] ⇒ (x[i] = y[j]) = $x_{i-1} + y_{j-1} + 1$

⇒ x[i] ≠ y[j] = max(x_i, y_{i-1}; x_{i-1}, y_i).

Algorithm:

```
LCS(X, Y)
m ← length(X)
n ← length(Y)
for i ← 0 to m.
    do c[i,0] ← 0.
```

```
for j ← 0 to n
    do c[0,j] ← 0.

for i ← 1 to m.
    for j ← 1 to n.
        if xi = yj then
            c[i,j] ← c[i-1,j-1]+1
                b[i,j] ← "↖"
    else if c[i-1,j] ≥ c[i,j-1]
    then c[i,j] ← c[i-1,j];
        b[i,j] ← "↑"

    else c[i,j] ← c[i,j-1]
        b[i,j] ← "←".

    return c & b.
```

```
Print LCS(b, X, i, j)
    if i=0 or j=0
        then return.

    if b[i,j] ← "↑"
    then Print LCS[b, X, i-1, j-1]
        Print Xi

    else if b[i,j] = "↑"
        then Print LCS[b, X, i-1, j]
    else
        Print LCS[b, X, i, j-1]
```

O/P = BCBA.

Q. Y = B D C A B.
X = A B C B.

	yi	B	D	C	A	B
xi	0	0	0	0	0	0
A	0	0↑	0↑	0↑	1↖	1←
B	0	(1)↖	1←	1←	1↑	2↖
C	0	1↑	1↑	(2)↖	2←	2↑
B	0	1↑	1↑	2↑	2↑	(3)↖

BCB. (Longest chain = 3).

7/10/18

Matrix Chain Multiplication

$A_1 A_2 A_3 \ldots\ldots A_n$ we have n matrices, we want to multiply these matrices.

If we want to multiply $A_1 A_2 A_3 A_4$ then we can do it many ways like $(A_1(A_2(A_3 A_4)))$

$(A_1 A_2)(A_3 A_4)$ ..... etc

$$\begin{bmatrix} - & - & - & - \\ - & - & - & - \\ 1 & 2 & 3 & 4 \end{bmatrix}_{3\times4} \quad \begin{bmatrix} a & - \\ b & - \\ c & - \\ d & - \end{bmatrix}_{4\times2}$$

$$C = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}_{3\times2}$$

$\Rightarrow a_{31} = a\cdot1 + b\cdot2 + c\cdot3 + d\cdot4$

```
MatMul (A, B)
  for i ← to row[A]
    for j ← 1 to col[B]
      C[i,j] ← 0
      for K ← 1 to col[A]
        do C[i,j] = C[i,j] + A[i,K] * B[K,j]
```

$A = p \times q \quad B = q \times r$

So $A \times B = C$

$C = p \times r$.

To obtain C we have to do $p \times q \times r$ multiplication in total.

Eg:- $A_1 A_2 A_3$.

$A_1 = 10 \times 100$

$A_2 = 100 \times 5$

$A_3 = 5 \times 50$

$(A_1 A_2) A_3$.

Case 1 $A_1 \quad A_2 \quad A_3\ (5\times50)$

(10×5)

$(10 \times 100 \times 5)$

$= 5000$

$10 \times 5 \times 50$

$= 2500$

7500 = Total multiplication.

Case 2 $A_1 (A_2 A_3)$

$(10\times100) \quad (100\times50)$

$100 \times 5 \times 50$

$= 25000$

$10 \times 100 \times 10$

$= 50000$

75000 = Total multiplication

By just changing the order of the matrix multiplication, the inner scalar multiplication is been increased 10 times.

Problem:- You are given chain of matrices $A_1, A_2 \cdots A_n$ a chain of n matrices where any matrix A_i has dimension

$A_i \longrightarrow P_{i-1} \times P_i$

So you have to fully parenthesize the product in such a way that reduces the scalar multiplication.

$m[i,j]$ → from i to j the no. of scalar multiplication needed.

If $i = j$ → only one matrix so no. of scalar multiplication = 0

If $i < j$ → $i \le k < j$.

Suppose we have split it in k matrix

Q:- $m[1,3]$

k = can be 1 or 2.

$A_1(A_2A_3)$ $(A_1A_2)A_3$.

Eg: $m[1,5]$ $k = 3$.

$= m(1,3) + m(4,5)$ = total scalar multiplication

$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k < j} \left(m[i,k] + m[k+1,j] + P_{i-1} P_k P_j \right) \end{cases}$$

Q $A_1\ A_2\ A_3\ A_4$

$A_1 = 5 \times 4$

$A_2 = 4 \times 6$

$A_3 = 6 \times 2$

$A_4 = 2 \times 7$.

$A_1 = 5 \times 4$
$A_2 = 4 \times 6$
$A_3 = 6 \times 2$
$A_4 = 2 \times 7$

$P_0 = 5$ $(A_1 = P_0 \times P_1)$
$P_1 = 4$ $(A_2 = P_1 \times P_2)$
$P_2 = 6$ $(A_3 = P_2 \times P_3)$
$P_3 = 2$
$P_4 = 7$. $(A_4 = P_3 \times P_4)$

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

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

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

$m[1-2] = p_0 \cdot p_1 \cdot p_2$

$= 5 \times 4 \times 6 = 120$

$m[2-3] = p_1 \times p_2 \times p_3$

$= 4 \times 6 \times 2 = 48$

$m[3-4] = p_2 \times p_3 \times p_4 = 6 \times 2 \times 7 = 84$

$m[1-3]$

When $k = 1$ and $k = 2$

$\min((m[1,1] + m[2,3] + p_0 p_1 p_3),$
$m[1,2] + m[3,3] + p_0 p_2 p_3)$

$= \min(0 + 48 + 40, 120 + 0 + 60)$

$= 88.$

$m[2,4] =$

$\min(m[2,2] + m[3,4] + p_1 p_2 p_4,$
$m[2,3] + m[4,4] + p_1 p_3 p_4)$

$= \min(0 + 3 + 168, 48 + 0 + 56)$

$= 104.$

$m(1,4)$

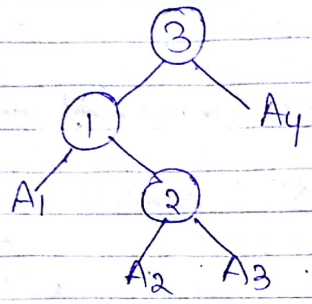
$= \min(m[1,1] + m(2,4) + p_0 p_1 p_4,$
$m[1,2] + m(3,4) + p_0 p_2 p_4,$
$m[1,3] + m[4,4] + p_0 p_3 p_4)$

$= \min(0 + 104 + 140, 120 + 84 + 210,$
$88 + 0 + 70)$

$= 158$

So our final answer is 158.

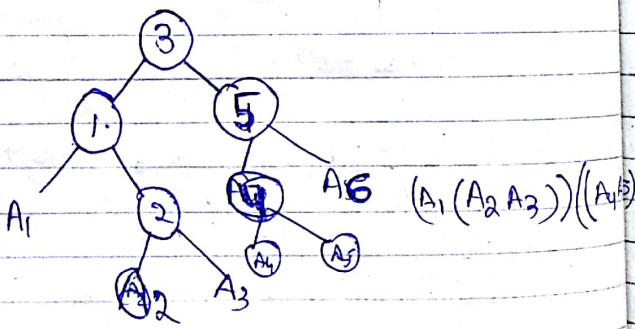
and our pattern is -



$(A_1(A_2A_3))A_4$ → Ans.

Eg:-

2	3	4	5	6	
1	1	3	3	3	1
	2	3	3	3	2
		3	3	3	3
			4	5	4
				5	5



$(A_1(A_2A_3))((A_4A_5)A_6)$

~~$((A_1(A_2A_3))(A_4A_5))A_6$~~

10/10/18

Backtracking:-

Generally you propose solution in the form of tupple.

n tuple $(x_1 \dots x_n)$

$(\overset{0}{4}, \overset{1}{8}, \overset{2}{1}, \overset{3}{2})$ → sort this is ascending order.

So our solution is a 4-tuple solution
= (2, 3, 1, 0)
→ indexes.

8-queen problem. (Imp).

8×8 chess board, we place to place 8 queen in such a manner that no queen attack each other.

The solution will be a 8 tuple solution
ith queen will go to the ith row.
So 1st queen will go on 1st row.
So we will show columns in tupples.

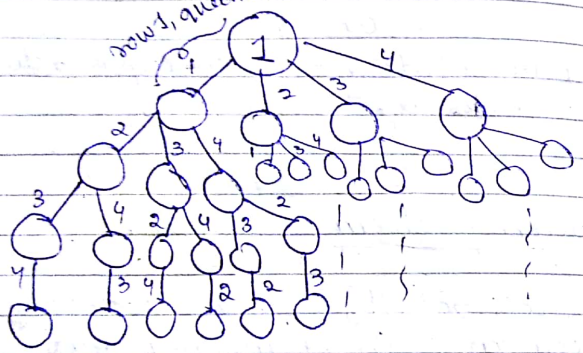
8-tupple (2, 5, 7, 4, 1, 8, 6, 3)
Sol:

↓ 1st queen in 2nd row.

↑ 8th queen in 8th row & 3rd column

Permutation tree is used for back-tracking problems.
Here tree edges will represent column and levels will represent row.

We are taking 4 queen problem.



Problem state :- Every node in this tree is known as problem state.

State space
Every path from a root to other node is a state space.

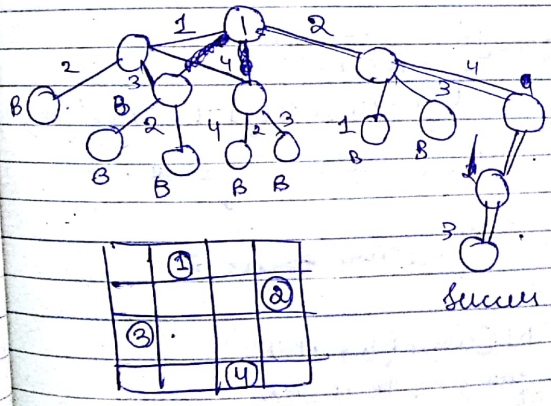
Only that path where we will get sucess is known as solution state.

Live node.
which has been generated, and all those whose children are not generated.

Dead node.
which cannot be expanded further or all the children are already generated.

E-node.
Node being expanded. The live node whose children are currently being expanded.

Bounding function :- is the function which is use to kill a live node without generating all their children.



Upper right to lower left diagonal, row + column value is same.

Upper left to lower right, diagonal = row - column value is same.

Ques
(i,j) & (k,l) (two co-ordinates)
$i-j = k-l$ or $i+j = k+l$.

or $j-l = i-k$ or $j-l = k-i$

$$|j-l| = |i-k|$$

Place (k,i)
$x[1 \cdots (k-1)]$

```
{
  for j = 1 to k-1 do
      if (x[j] = i)
        or (Abs(x[j]-i) = Abs(j-k))
      then return false.
  else
      return true.
}
```

Algo. NQueen(k, n)

```
for i = 1 to n do
{
  if place(k,i) then
  {
      x[k] = i;
      if (k=n) then
          write(x[1 ... n]), else
          NQueen(k+1, n)
  }
}
```