Program : **B.Tech**

Subject Name: **Analysis and Design of Algorithm**

Subject Code: **IT-403**

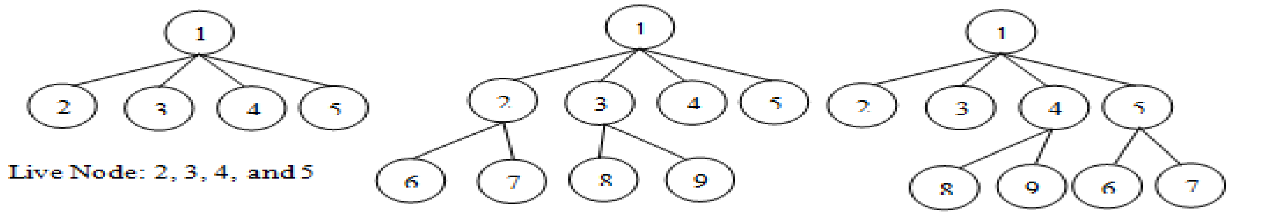
Semester: **4th**

Unit-5 Introduction to branch & bound method

Introduction:

Branch & Bound (B & B) is general algorithm (or Systematic method) for finding optimal solution of various optimization problems, especially in discrete and combinatorial optimization.

- ➢ The B&B strategy is very similar to backtracking in that a state space tree is used to solve a problem.
- ➢ The differences are that the B&B method
- ✓ Does not limit us to any particular way of traversing the tree.
- ✓ It is used only for optimization problem
- ✓ It is applicable to a wide variety of discrete combinatorial problem.
- ➢ B&B is rather general optimization technique that applies where the greedy method & dynamic programming fail.
- ➢ It is much slower, indeed (truly), it often (rapidly) leads to exponential time complexities in the worst case.
- ➢ The term B&B refers to all state space search methods in which all children of the "E-node" are generated before any other "live node" can become the "E-node"
- ✓ **Live node** is a node that has been generated but whose children have not yet been generated.
- ✓ **E-node** is a live node whose children are currently being explored.
- ✓ **Dead node** is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.



FIFO Branch & Bound (BFS) Children of E-node are inserted in a queue.

LIFO Branch & Bound (D-Search) Children of E-node are inserted in a stack.

- ➢ Two graph search strategies, BFS & D-search (DFS) in which the exploration of a new node cannot begin until the node currently being explored is fully explored.
- ➢ Both BFS & D-search (DFS) generalized to B&B strategies.
- ✓ **BFS** like state space search will be called FIFO (First In First Out) search as the list of live nodes is "First-in-first-out" list (or queue).
- ✓ **D-search (DFS)** Like state space search will be called LIFO (Last In First Out) search as the list of live nodes is a "last-in-first-out" list (or stack).
- ➢ In backtracking, bounding function are used to help avoid the generation of sub-trees that do not contain an answer node.
- ➢ We will use 3-types of search strategies in branch and bound

1) FIFO (First In First Out) search

2) LIFO (Last In First Out) search

3) LC (Least Count) search

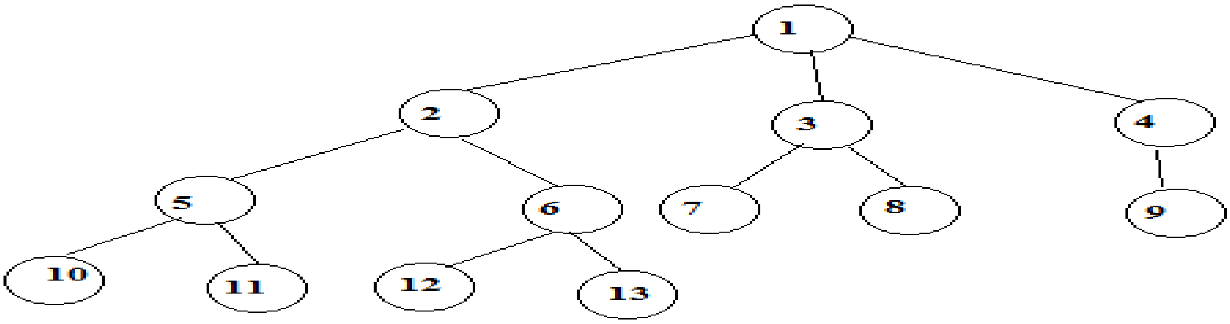
FIFO B&B:

FIFO Branch & Bound is a BFS.

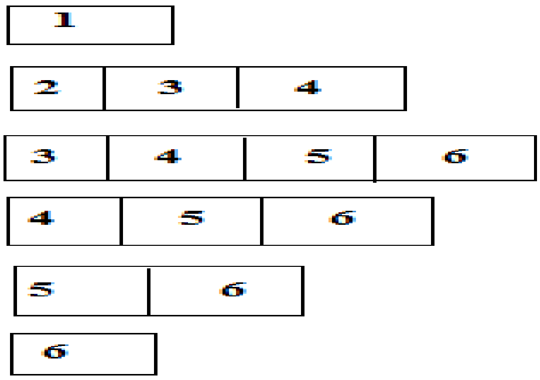
In this, children of E-Node (or Live nodes) are inserted in a queue.

Implementation of list of live nodes as a queue

- ✓ Least()▯ Removes the head of the Queue
- ✓ Add()▯ Adds the node to the end of the Queue



Assume that node ‘12’ is an answer node in FIFO search, 1st we take E-node has ‘1’



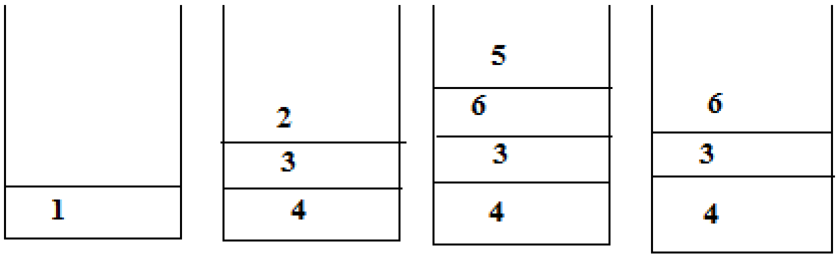
LIFO B&B:

LIFO Brach & Bound is a D-search (or DFS).

In this children of E-node (live nodes) are inserted in a stack

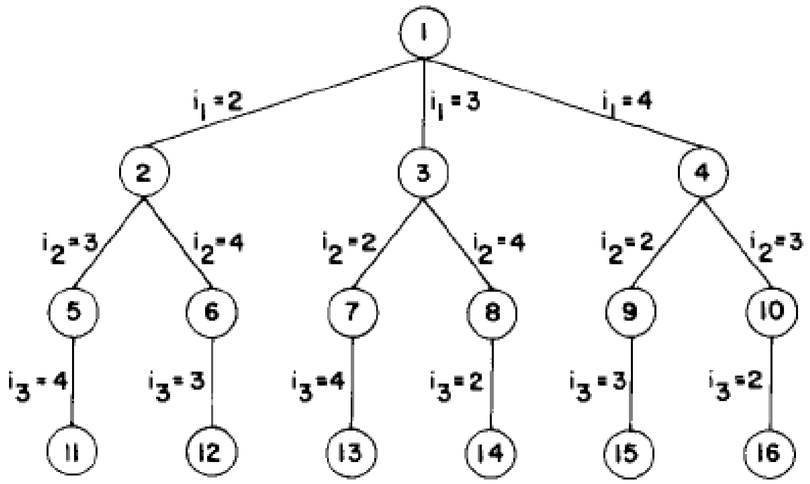
Implementation of List of live nodes as a stack

- ✓ Least()▯ Removes the top of the stack
- ✓ ADD()▯Adds the node to the top of the stack.



Travelling Salesman Problem:

Def:- Find a tour of minimum cost starting from a node S going through other nodes only once and returning to the starting point S.
Time Conmlexity of TSP for Dynamic Programming algorithm is $O(n^2 2^n)$
B&B algorithms for this problem, the worest case complexity will not be any better than $O(n^2 2^n)$ but good bunding functions will enables these B&B algorithms to solve some problem instances in much less time than required by the dynamic programming alogrithm.
Let G=(V,E) be a directed graph defining an instances of TSP.
Let C_{ij} cost of edge <i, j>
$C_{ij} = \infty$ if $<i, j> \notin E$
|V|=n total number of vertices.
Assume that every tour starts & ends at vertex 1.
Solution Space S= {1, Π , 1 / Π is a permutation of (2, 3. 4. ----n) } then |S|=(n-1)!
The size of S reduced by restricting S
Sothat $(1, i_1, i2, \text{-----} i_{n-1}, 1\} \in S$ iff $<i_j, i_{j+1}> \in E$. $O \le j \le n-1$, $i_0\text{-}i_n=1$
S can be organized into "State space tree".
Consider the following Example



State space tree for the travelling salesperson problem with n=4 and $i_0=i_4=1$

The above diagram shows tree organization of a complete graph with |V|=4.
Each leaf node 'L' is a solution node and represents the tour defined by the path from the root to L.

Node 12 represents the tour.
$i_0=1, i_1=2, i_2=4, i_3=3, i_4=1$

Node 14 represents the tour.
$i_0=1, i_1=3, i_2=4, i_3=2, i_4=1$.

TSP is solved by using LC Branch & Bound:

To use LCBB to search the travelling salesperson "State space tree" first define a cost function C(.) and other 2 functions Ĉ(.) & u(.)
Such that $\hat{C}(r) \le C(r) \le u(r)$ for all nodes r.
Cost C(.) is the solution node with least C(.) corresponds to a shortest tour in G.

C(A)={Length of tour defined by the path from root to A if A is leaf

Cost of a minimum-cost leaf in the sub-tree A, if A is not leaf }

From 1 Ĉ(r) ≤ C(r) then Ĉ(r) □ is the length of the path defined at node A.

From previous example the path defined at node 6 is i_0, i_1, i_2=1, 2, 4 & it consists edge of <1,2> & <2,4>

Abetter Ĉ(r) can be obtained by using the reduced cost matrix corresponding to G.

- A row (column) is said to be reduced iff it contains at least one zero & remaining entries are non negative.
- A matrix is reduced iff every row & column is reduced.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

(a) Cost Matrix

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Reduced Cost Matrix

L = 25

Given the following cost matrix:

$$\begin{bmatrix} inf & 20 & 30 & 10 & 11 \\ 15 & inf & 16 & 4 & 2 \\ 3 & 5 & inf & 2 & 4 \\ 19 & 6 & 18 & inf & 3 \\ 16 & 4 & 7 & 16 & inf \end{bmatrix}$$

- ➢ The TSP starts from node 1: **Node 1**
- ➢ Reduced Matrix: To get the lower bound of the path starting at node 1

Row # 1: reduce by 10 $\begin{bmatrix} inf & 10 & 20 & 0 & 1 \\ 15 & inf & 16 & 4 & 2 \\ 3 & 5 & inf & 2 & 4 \\ 19 & 6 & 18 & inf & 3 \\ 16 & 4 & 7 & 16 & inf \end{bmatrix}$	Row #2: reduce 2 $\begin{bmatrix} inf & 10 & 20 & 0 & 1 \\ 13 & inf & 14 & 2 & 0 \\ 3 & 5 & inf & 2 & 4 \\ 19 & 6 & 18 & inf & 3 \\ 16 & 4 & 7 & 16 & inf \end{bmatrix}$	Row #3: reduce by 2 $\begin{bmatrix} inf & 10 & 20 & 0 & 1 \\ 13 & inf & 14 & 2 & 0 \\ 1 & 3 & inf & 0 & 2 \\ 19 & 6 & 18 & inf & 3 \\ 16 & 4 & 7 & 16 & inf \end{bmatrix}$
Row # 4: Reduce by 3: $\begin{bmatrix} inf & 10 & 20 & 0 & 1 \\ 13 & inf & 14 & 2 & 0 \\ 1 & 3 & inf & 0 & 2 \\ 16 & 3 & 15 & inf & 0 \\ 16 & 4 & 7 & 16 & inf \end{bmatrix}$	Row # 5: Reduce by 4 $\begin{bmatrix} inf & 10 & 20 & 0 & 1 \\ 13 & inf & 14 & 2 & 0 \\ 1 & 3 & inf & 0 & 2 \\ 16 & 3 & 15 & inf & 0 \\ 12 & 0 & 3 & 12 & inf \end{bmatrix}$	Column 1: Reduce by 1 $\begin{bmatrix} inf & 10 & 20 & 0 & 1 \\ 12 & inf & 14 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 15 & inf & 0 \\ 11 & 0 & 3 & 12 & inf \end{bmatrix}$
Column 2: It is reduced.	Column 3: Reduce by 3 $\begin{bmatrix} inf & 10 & 17 & 0 & 1 \\ 12 & inf & 11 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & 12 & inf \end{bmatrix}$	Column 4: It is reduced. Column 5: It is reduced.

The reduced cost is: RCL = 25

So the cost of node 1 is: Cost (1) = 25

The reduced matrix is:

Cost (1) = 25

$$\begin{bmatrix} inf & 10 & 17 & 0 & 1 \\ 12 & inf & 11 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & 12 & inf \end{bmatrix}$$

- **Choose to go to vertex 2: Node 2**

  - Cost of edge <1,2> is: A(1,2) = 10
  - Set row #1 = inf since we are choosing edge <1,2>
  - Set column # 2 = inf since we are choosing edge <1,2>
  - Set A(2,1) = inf
  - The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & 11 & 2 & 0 \\ 0 & inf & inf & 0 & 2 \\ 15 & inf & 12 & inf & 0 \\ 11 & inf & 0 & 12 & inf \end{bmatrix}$$

  - The matrix is reduced:
  - RCL = 0
  - The cost of node 2 (Considering vertex 2 from vertex 1) is:

  Cost(2) = cost(1) + A(1,2) = 25 + 10 = 35

- **Choose to go to vertex 3: Node 3**

  - Cost of edge <1,3> is: A(1,3) = 17 (In the reduced matrix
  - Set row #1 = inf since we are starting from node 1
  - Set column # 3 = inf since we are choosing edge <1,3>
  - Set A(3,1) = inf
  - The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & inf & 2 & 0 \\ inf & 3 & inf & 0 & 2 \\ 15 & 3 & inf & inf & 0 \\ 11 & 0 & inf & 12 & inf \end{bmatrix}$$

  Reduce the matrix: Rows are reduced

  The columns are reduced except for column # 1:

  Reduce column 1 by 11:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 1 & inf & inf & 2 & 0 \\ inf & 3 & inf & 0 & 2 \\ 4 & 3 & inf & inf & 0 \\ 0 & 0 & inf & 12 & inf \end{bmatrix}$$

  The lower bound is: RCL = 11

  The cost of going through node 3 is:

  cost(3) = cost(1) + RCL + A(1,3) = 25 + 11 + 17 = 53

- Choose to go to vertex 4: **Node 4**

  Remember that the cost matrix is the one that was reduced at the starting vertex 1

  Cost of edge <1,4> is: A(1,4) = 0

Set row #1 = inf since we are starting from node 1
Set column # 4 = inf since we are choosing edge <1,4>
Set A(4,1) = inf
The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & inf & 0 \\ 0 & 3 & inf & inf & 2 \\ inf & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & inf & inf \end{bmatrix}$$

Reduce the matrix: Rows are reduced
Columns are reduced
The lower bound is: RCL = 0

The cost of going through node 4 is:

cost(4) = cost(1) + RCL + A(1,4) = 25 + 0 + 0 = 25

- **Choose to go to vertex 5: Node 5**

- Remember that the cost matrix is the one that was reduced at starting vertex 1
- Cost of edge <1,5> is: A(1,5) = 1
- Set row #1 = inf since we are starting from node 1
- Set column # 5 = inf since we are choosing edge <1,5>
- Set A(5,1) = inf
- The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & 2 & inf \\ 0 & 3 & inf & 0 & inf \\ 15 & 3 & 12 & inf & inf \\ inf & 0 & 0 & 12 & inf \end{bmatrix}$$

Reduce the matrix:

Reduce rows:

Reduce row #2: Reduce by 2

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 10 & inf & 9 & 0 & inf \\ 0 & 3 & inf & 0 & inf \\ 15 & 3 & 12 & inf & inf \\ inf & 0 & 0 & 12 & inf \end{bmatrix}$$

Reduce row #4: Reduce by 3

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 10 & inf & 9 & 0 & inf \\ 0 & 3 & inf & 0 & inf \\ 12 & 0 & 9 & inf & inf \\ inf & 0 & 0 & 12 & inf \end{bmatrix}$$

Columns are reduced

The lower bound is: RCL = 2 + 3 = 5

The cost of going through node 5 is:

cost(5) = cost(1) + RCL + A(1,5) = 25 + 5 + 1 = 31

In summary:

So the live nodes we have so far are:

- ✓ 2: cost(2) = 35, path: 1->2
- ✓ 3: cost(3) = 53, path: 1->3
- ✓ 4: cost(4) = 25, path: 1->4
- ✓ 5: cost(5) = 31, path: 1->5

Explore the node with the lowest cost: Node 4 has a cost of 25

Vertices to be explored from node 4: 2, 3, and 5

Now we are starting from the cost matrix at node 4 is:

Cost (4) = 25

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & inf & 0 \\ 0 & 3 & inf & inf & 2 \\ inf & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & inf & inf \end{bmatrix}$$

- ➢ **<u>Choose to go to vertex 2: Node 6 (path is 1->4->2)</u>**

Cost of edge <4,2> is: A(4,2) = 3

Set row #4 = inf since we are considering edge <4,2>

Set column # 2 = inf since we are considering edge <4,2>

Set A(2,1) = inf

The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & 11 & inf & 0 \\ 0 & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & 0 & inf & inf \end{bmatrix}$$

Reduce the matrix: Rows are reduced

Columns are reduced

The lower bound is: RCL = 0

The cost of going through node 2 is:

cost(6) = cost(4) + RCL + A(4,2) = 25 + 0 + 3 = 28

- **Choose to go to vertex 3: Node 7 (path is 1->4->3)**

Cost of edge <4,3> is: A(4,3) = 12
Set row #4 = inf since we are considering edge <4,3>
Set column # 3 = inf since we are considering edge <4,3>
Set A(3,1) = inf
The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & inf & inf & 0 \\ inf & 3 & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & 0 & inf & inf & inf \end{bmatrix}$$

Reduce the matrix:

Reduce row #3: by 2:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & inf & inf & 0 \\ inf & 1 & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 11 & 0 & inf & inf & inf \end{bmatrix}$$

Reduce column # 1: by 11

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 1 & inf & inf & inf & 0 \\ inf & 1 & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 0 & 0 & inf & inf & inf \end{bmatrix}$$

The lower bound is: RCL = 13

So the RCL of node 7 (Considering vertex 3 from vertex 4) is:

Cost(7) = cost(4) + RCL + A(4,3) = 25 + 13 + 12 = 50

- Choose to go to vertex 5: **Node 8** (path is 1->4->5)

Cost of edge <4,5> is: A(4,5) = 0
Set row #4 = inf since we are considering edge <4,5>
Set column # 5 = inf since we are considering edge <4,5>
Set A(5,1) = inf
The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & inf & inf \\ 0 & 3 & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & 0 & 0 & inf & inf \end{bmatrix}$$

Reduce the matrix:
Reduced row 2: by 11

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 1 & inf & 0 & inf & inf \\ 0 & 3 & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & 0 & 0 & inf & inf \end{bmatrix}$$

Columns are reduced
The lower bound is: RCL = 11
So the cost of node 8 (Considering vertex 5 from vertex 4) is:
Cost(8) = cost(4) + RCL + A(4,5) = 25 + 11 + 0 = 36

In summary: So the live nodes we have so far are:

- ✓ 2: cost(2) = 35, path: 1->2
- ✓ 3: cost(3) = 53, path: 1->3
- ✓ 5: cost(5) = 31, path: 1->5
- ✓ 6: cost(6) = 28, path: 1->4->2
- ✓ 7: cost(7) = 50, path: 1->4->3
- ✓ 8: cost(8) = 36, path: 1->4->5
- Explore the node with the lowest cost: Node 6 has a cost of 28
- Vertices to be explored from node 6: 3 and 5
- Now we are starting from the cost matrix at node 6 is:

Cost (6) = 28

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & 11 & inf & 0 \\ 0 & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & 0 & inf & inf \end{bmatrix}$$

- **Choose to go to vertex 3: Node 9 (path is 1->4->2->3)**

Cost of edge <2,3> is: A(2,3) = 11
Set row #2 = inf since we are considering edge <2,3>
Set column # 3 = inf since we are considering edge <2,3>
Set A(3,1) = inf
The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & inf & inf & inf \end{bmatrix}$$

Reduce the matrix: Reduce row #3: by 2

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 11 & inf & inf & inf & inf \end{bmatrix}$$

Reduce column # 1: by 11

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \end{bmatrix}$$

The lower bound is: RCL = 2 +11 = 13

So the cost of node 9 (Considering vertex 3 from vertex 2) is:

Cost(9) = cost(6) + RCL + A(2,3) = 28 + 13 + 11 = 52

- **<u>Choose to go to vertex 5: Node 10 (path is 1->4->2->5)</u>**

Cost of edge <2,5> is: A(2,5) = 0
Set row #2 = inf since we are considering edge <2,3>
Set column # 3 = inf since we are considering edge <2,3>
Set A(5,1) = inf
The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & 0 & inf & inf \end{bmatrix}$$

Reduce the matrix: Rows reduced
Columns reduced
The lower bound is: RCL = 0
So the cost of node 10 (Considering vertex 5 from vertex 2) is:
Cost(10) = cost(6) + RCL + A(2,3) = 28 + 0 + 0 = 28

In summary: **So the live nodes we have so far are:**

- ✓ 2: cost(2) = 35, path: 1->2
- ✓ 3: cost(3) = 53, path: 1->3
- ✓ 5: cost(5) = 31, path: 1->5
- ✓ 7: cost(7) = 50, path: 1->4->3
- ✓ 8: cost(8) = 36, path: 1->4->5
- ✓ 9: cost(9) = 52, path: 1->4->2->3
- ✓ 10: cost(2) = 28, path: 1->4->2->5
- Explore the node with the lowest cost: Node 10 has a cost of 28
- Vertices to be explored from node 10: 3
- Now we are starting from the cost matrix at node 10 is:

Cost (10)=28

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & 0 & inf & inf \end{bmatrix}$$

➢ **Choose to go to vertex 3: Node 11 (path is 1->4->2->5->3)**

Cost of edge <5,3> is: A(5,3) = 0

Set row #5 = inf since we are considering edge <5,3>

Set column # 3 = inf since we are considering edge <5,3>

Set A(3,1) = inf

The resulting cost matrix is:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \end{bmatrix}$$

Reduce the matrix: Rows reduced

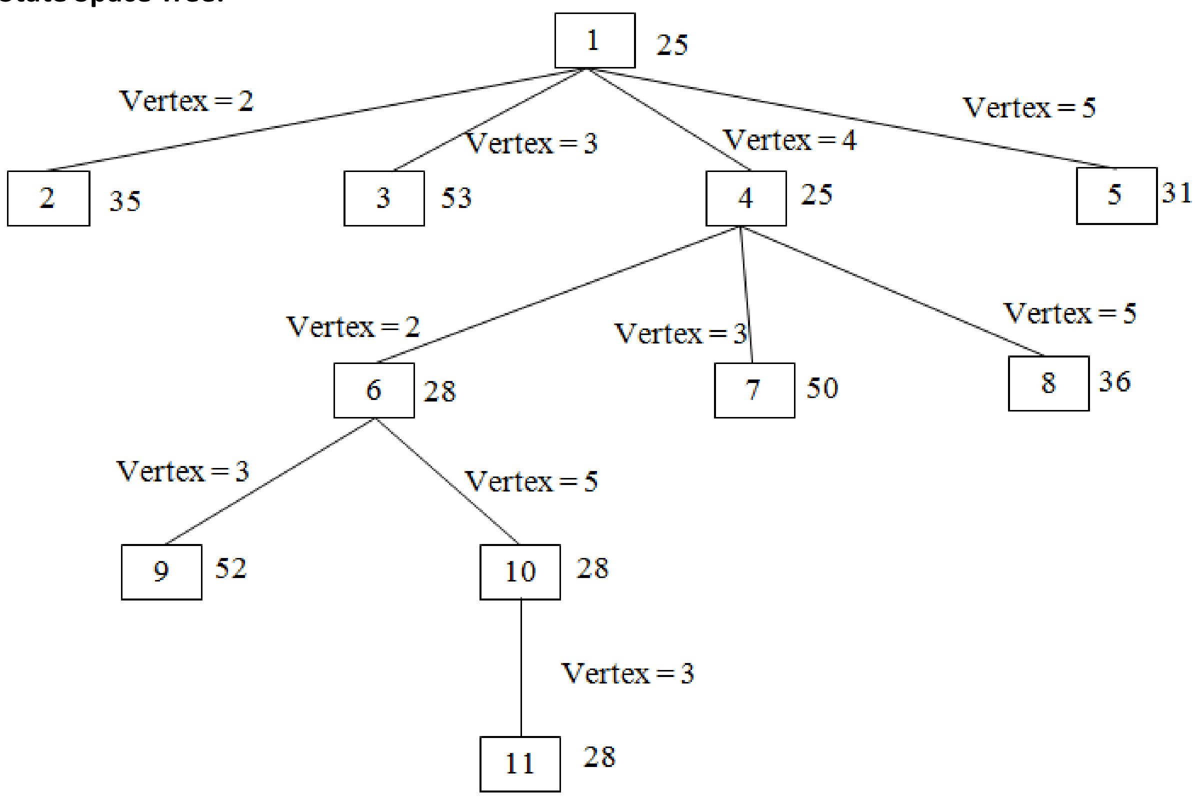
Columns reduced

The lower bound is: RCL = 0

So the cost of node 11 (Considering vertex 5 from vertex 3) is:

Cost(11) = cost(10) + RCL + A(5,3) = 28 + 0 + 0 = 28

State Space Tree:



Lower Bound Theory

To find a function g(n) which is a lower bound on the time that algorithm must take, So if f(n) is the time for some algorithm then we can write f(n)=Ω(g(n) where g(n) is lower bound for f(n)Or we can say f(n) >= c * g(n) for all n>n 0 Where c & n 0 are constants For many problems it is easy to calculate the lower bound on n inputs e.g. consider the set of problems to find the maximum of an ordered set of n integers. Clearly every integer must be examined at least once. So Ω(n) is a lower bound for that. For matrix multiplication we have 2n 2 inputs so the lower bound can be Ω(n 2) For an unordered set the searching algorithm will take Ω(n) as the lower bound. For an ordered set it will take Ω(logn) as the lower bound. Similarly all the sorting algorithms can not sort in less then Ω(nlogn) time so Ω(nlogn) is the lower bound for sorting algorithms.

For all sorting & searching we use decision trees/comparison trees for finding the lower bound.

-Internal nodes represent comparisons

-Leaves represent outcomes

-The running time of the algorithm = the length of the path taken.

-Worst-case running time = height of tree.

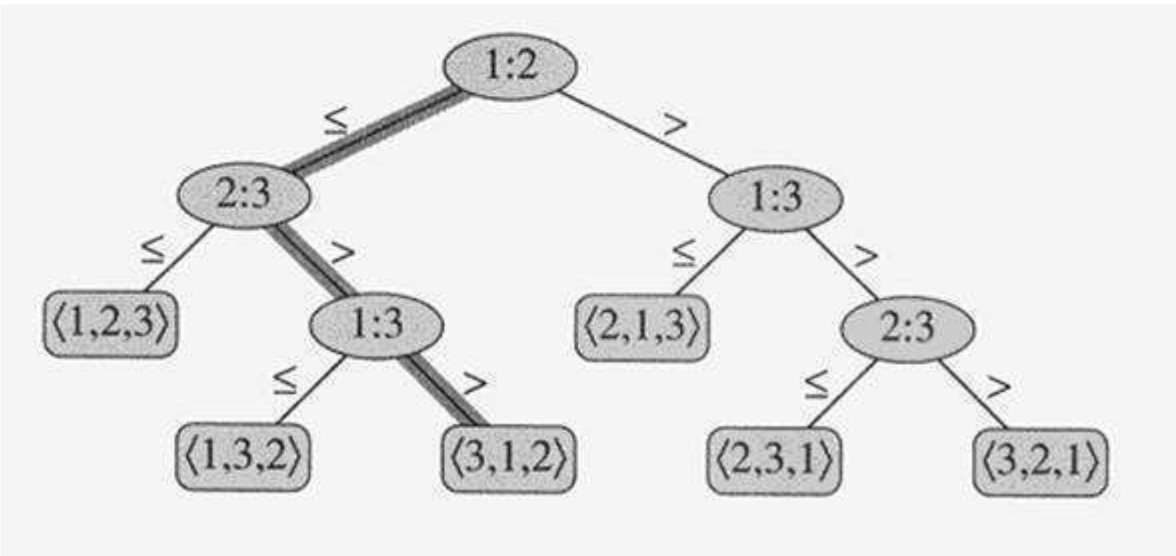
ITS USE::: EXAMPLE: Sorting

o Decision tree for sorting 3 elements Sort *a1, a2, …, an*

Each internal node is labeled i:j for i, j {1, 2,…, n}.

The left subtree shows subsequent comparisons if *ai aj*.

The right subtree shows subsequent comparisons if *ai aj*.



NP Completeness & NP Hard Problems

Basic concepts:

NP: Nondeterministic Polynomial time

The problems has best algorithms for their solutions have "Computing times", that cluster into two groups

Group 1	Group 2
➢ Problems with solution time bound by a polynomial of a small degree. ➢ It also called "Tractable Algorithms" ➢ Most Searching & Sorting algorithms are polynomial time algorithms ➢ **Ex:** Ordered Search **(O (log n)),** Polynomial evaluation **O(n)** Sorting **O(n.log n)**	➢ Problems with solution times not bound by polynomial (simply non polynomial) ➢ These are hard or intractable problems ➢ None of the problems in this group has been solved by any polynomial time algorithm ➢ **Ex:** Traveling Sales Person **$O(n^2 2^n)$** Knapsack **$O(2^{n/2})$**

No one has been able to develop a polynomial time algorithm for any problem in the 2nd group (i.e., group 2) So, it is compulsory and finding algorithms whose computing times are greater than polynomial very quickly because such vast amounts of time to execute that even moderate size problems cannot be solved.

Theory of NP-Completeness:

Show that may of the problems with no polynomial time algorithms are computational time algorithms are computationally related.

There are two classes of non-polynomial time problems

- NP-Hard
- NP-Complete

NP Complete Problem: A problem that is NP-Complete can solved in polynomial time if and only if (iff) all other NP-Complete problems can also be solved in polynomial time.

NP-Hard: Problem can be solved in polynomial time then all NP-Complete problems can be solved in polynomial time.

All NP-Complete problems are NP-Hard but some NP-Hard problems are not know to be NP-Complete.

Concept of Nondeterministic Algorithms:

Algorithms with the property that the result of every operation is uniquely defined are termed as deterministic algorithms. Such algorithms agree with the way programs are executed on a computer. Algorithms which contain operations whose outcomes are not uniquely defined but are limited to specified set of possibilities. Such algorithms are called nondeterministic algorithms. The machine executing such operations is allowed to choose any one of these outcomes subject to a termination condition to be defined later.

To specify nondeterministic algorithms, there are 3 new functions.
Choice(S)⍰ arbitrarily chooses one of the elements of sets S
Failure ()⍰ Signals an Unsuccessful completion
Success ()⍰ Signals a successful completion.

Example for Non Deterministic algorithms:

<table>
<tr><td>Algorithm Search(x){
//Problem is to search an element x
//output J, such that A[J]=x; or J=0 if x is not in A
J:=Choice(1,n);
if(A[J]:=x) then {
Write(J);
Success();
}
else{
write(0);
failure();
}</td><td>Whenever there is a set of choices that leads to a successful completion then one such set of choices is always made and the algorithm terminates.

A Nondeterministic algorithm terminates unsuccessfully if and only if (iff) there exists no set of choices leading to a successful signal.</td></tr>
<tr><td colspan="2">Nondeterministic Knapsack algorithm</td></tr>
<tr><td>Algorithm DKP(p, w, n, m, r, x){
W:=0;
P:=0;
for i:=1 to n do{
x[i]:=choice(0, 1);
W:=W+x[i]*w[i];
P:=P+x[i]*p[i];
}
if((W>m) or (P<r)) then Failure();
else Success();
}</td><td>p given Profits
wgiven Weights
n Number of elements (number of p or w)
mWeight of bag limit
P[]Final Profit
W[]Final weight</td></tr>
</table>

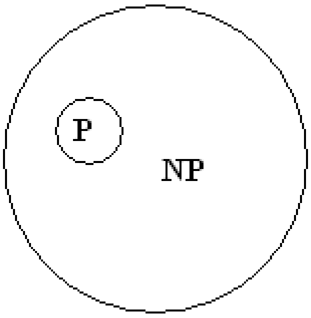
The Classes NP-Hard & NP-Complete:

For measuring the complexity of an algorithm, we use the input length as the parameter. For example, An algorithm A is of polynomial complexity p() such that the computing time of A is

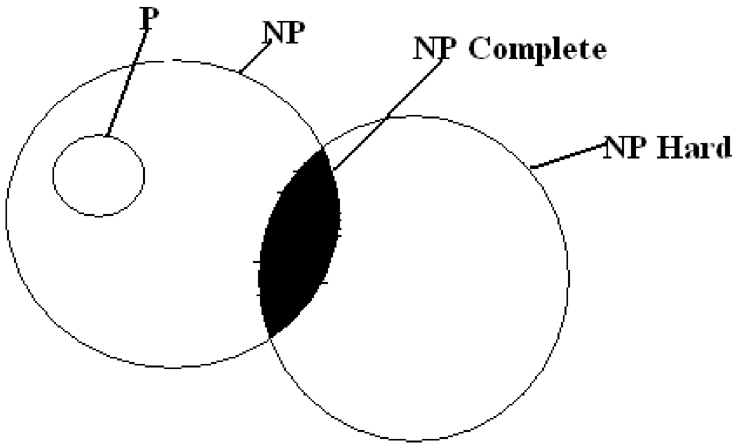
O(p(n)) for every input of size n.
Decision problem/ Decision algorithm: Any problem for which the answer is either zero or one is decision problem. Any algorithm for a decision problem is termed a decision algorithm.
Optimization problem/ Optimization algorithm: Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an optimization problem. An optimization algorithm is used to solve an optimization problem.

P is the set of all decision problems solvable by deterministic algorithms in polynomial time.
NP is the set of all decision problems solvable by nondeterministic algorithms in polynomial time.

Since deterministic algorithms are just a special case of nondeterministic, by this we concluded that **P ⊆NP**



Commonly believed relationship between P & NP
The most famous unsolvable problems in Computer Science is Whether P=NP or P≠NP
In considering this problem, s.cook formulated the following question.
If there any single problem in NP, such that if we showed it to be in 'P' then that would imply that P=NP.
Cook answered this question with
Theorem: Satisfiability is in P if and only if (iff) P=NP
Notation of Reducibility
Let L_1 and L_2 be problems, Problem L_1 reduces to L_2 (written $\mathbf{L_1 \alpha L_2}$) iff there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time
This implies that, if we have a polynomial time algorithm for L_2, Then we can solve L_1 in polynomial time.
Here α is a transitive relation i.e., $\mathbf{L_1 \alpha L_2}$ and $\mathbf{L_2 \alpha L_3}$ then $\mathbf{L_1 \alpha L_3}$
A problem L is NP-Hard if and only if (iff) satisfiability reduces to L ie., **Statisfiability α L**
A problem L is NP-Complete if and only if (iff) L is NP-Hard and **L Є NP**



Commonly believed relationship among P, NP, NP-Complete and NP-Hard

Most natural problems in NP are either in P or NP-complete.

Examples of NP-complete problems:

- Packing problems: SET-PACKING, INDEPENDENT-SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3-COLOR, CLIQUE.
- Constraint satisfaction problems: SAT, 3-SAT.