Unit-V

Branch and Bound

1.General Method

Branch and Bound refers to all state space search methods in which all children of the E-Node are generated before any other live node becomes the E-Node.

Branch and Bound is the generalization of both graph search strategies, BFS and D-search.

- A BFS like state space search is called as FIFO (First in first out) search as the list of live nodes in a first in first out.
- A D-search like state space search is called as LIFO (last in first out) search as the list of live nodes in a last in first out list.

✓ 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. In other words, an E-node is a node currently being expanded.

✓ Dead node is a generated anode that is not be expanded or explored any further. All children of a dead node have already been expanded.

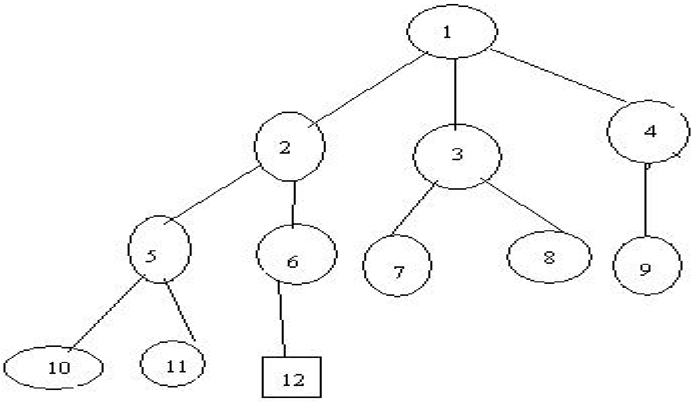
3 types of search strategies:

1. FIFO (First In First Out)
2. LIFO (Last In First Out)
3. LC (Least Cost) Search

FIFO Branch and Bound Search:

For this we will use a data structure called Queue. Initially Queue is empty.

Example:



Assume the node 12 is an answer node (solution)

In FIFO search, first we will take E-node as a node 1.

Next we generate the children of node 1. We will place all these live nodes in a queue.

2	3	4			

Now we will delete an element from queue, i.e. node 2, next generate children of node 2 and place in this queue.

	3	4	5	6	

Next, delete an element from queue and take it as E-node, generate the children of node 3, 7, 8 are children of 3 and these live nodes are killed by bounding functions. So we will not include in the queue.

		4	5	6	

Again delete an element an from queue. Take it as E-node, generate the children of 4. Node 9 is generated and killed by boundary function.

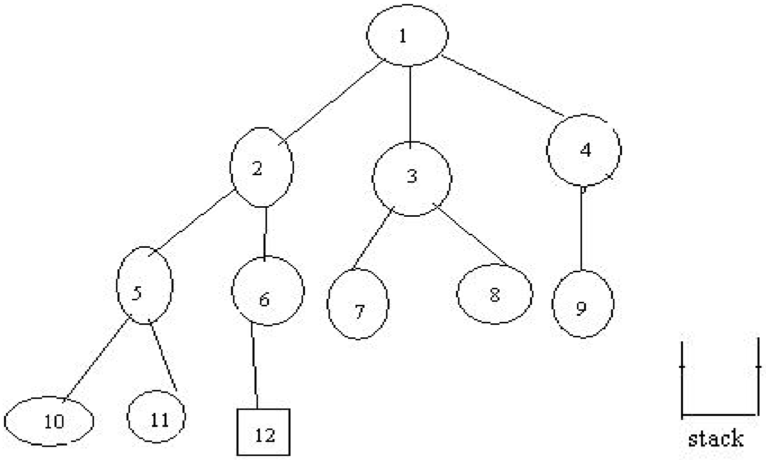
			5	6	

Next, delete an element from queue. Generate children of nodes 5, i.e., nodes 10 and 11 are generated and by boundary function, last node in queue is 6. The child of node 6 is 12 and it satisfies the conditions of the problem, which is the answer node, so search terminates.

LIFO Branch and Bound Search:

For this we will use a data structure called stack. Initially stack is empty.

Example:



Generate children of node 1 and place these live nodes into stack.

2
3
4

stack

Remove element from stack and generate the children of it, place those nodes into stack. 2 is removed from stack. The children of 2 are 5, 6. The content of stack is,

5
6
3
4

stack

Again remove an element from stack, i.,e node 5 is removed and nodes generated by 5 are 10, 11 which are killed by bounded function, so we will not place 10, 11 into stack.

6
3
4

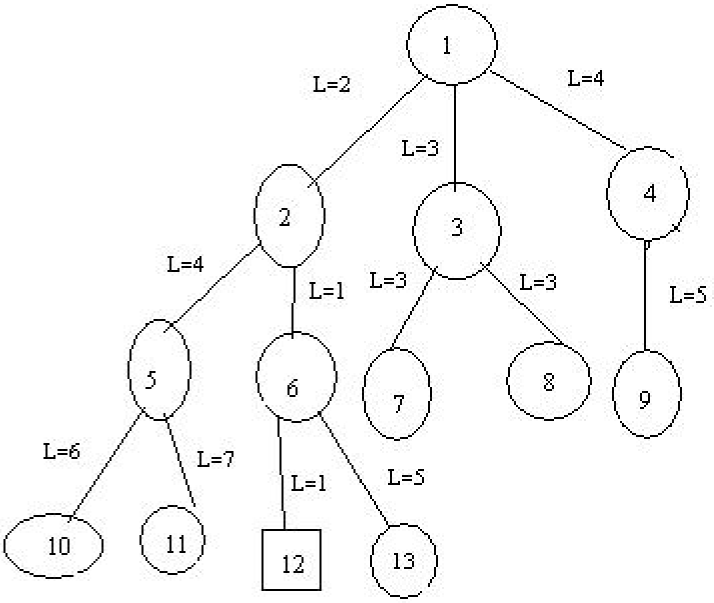
stack

Delete an element from stack, i.,e node 6. Generate child of node 6, i.,e 12, which is the answer node, so search process terminates.

LC (Least Cost) Branch and Bound Search:

In both FIFO and LIFO Branch and Bound the selection rules for the next E-node in rigid and blind. The selection rule for the next E-node does not give any preferences to a node that has a very good chance of getting the search to an answer node quickly.

In this we will use ranking function or cost function. We generate the children of E-node, among these live nodes; we select a node which has minimum cost. By using ranking function we will calculate the cost of each node.



Initially we will take node 1 as E-node. Generate children of node 1, the children are 2, 3, 4. By using ranking function we will calculate the cost of 2, 3, 4 nodes is ĉ =2, ĉ =3, ĉ =4 respectively. Now we will select a node which has minimum cost i.,e node 2. For node 2, the children are 5, 6. Between 5 and 6 we will select the node 6 since its cost minimum. Generate children of node 6 i.,e 12 and 13. We will select node 12 since its cost (ĉ =1) is minimum. More over 12 is the answer node. So, we terminate search process.

Control Abstraction for LC-search:

Let **t** be a state space tree and c() a cost function for the nodes in t. If x is a node in t, then c(x) is the minimum cost of any answer node in the sub tree with root x. Thus, c(t) is the cost of a minimum-cost answer node in t.

LC search uses ĉ to find an answer node. The algorithm uses two functions

Least-cost()
Add_node()

Least-cost() finds a live node with least c(). This node is deleted from the list of live nodes and returned.

Add_node() to delete and add a live node from or to the list of live nodes.

Add_node(x)adds the new live node x to the list of live nodes.The list of live nodes be implemented as a min-heap.

BOUNDING

- A branch and bound method searches a state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E-node.
- A good bounding helps to prune (reduce) efficiently the tree, leading to a faster exploration of the solution space. Each time a new answer node is found, the value of upper can be updated.

- Branch and bound algorithms are used for optimization problem where we deal directly only with minimization problems. A maximization problem is easily converted to a minimization problem by changing the sign of the objective function.

2. 0/1 KNAPSACK PROBLEM (LCBB)

There are n objects given and capacity of knapsack is M. Select some objects to fill the knapsack in such a way that it should not exceed the capacity of Knapsack and maximum profit can be earned. The Knapsack problem is maximization problem. It means we will always seek for maximum p_1x_1 (where p_1 represents profit of object x_1).

A branch bound technique is used to find solution to the knapsack problem. But we cannot directly apply the branch and bound technique to the knapsack problem. Because the branch bound deals only the minimization problems. We modify the knapsack problem to the minimization problem. The modifies problem is,

$$\text{minimize } -\sum_{i=1}^{n} p_i x_i$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i \leq m$$

$$x_i = 0 \text{ or } 1, \quad 1 \leq i \leq n$$

Example: Consider the instance M=15, n=4, (p_1, p_2, p_3, p_4) = 10, 10, 12, 18 and (w_1, w_2, w_3, w_4)=(2, 4, 6, 9).

Solution: knapsack problem can be solved by using branch and bound technique. In this problem we will calculate lower bound and upper bound for each node.

Arrange the item profits and weights with respect of profit by weight ratio. After that, place the first item in the knapsack. Remaining weight of knapsack is 15-2=13. Place next item w_2 in knapsack and the remaining weight of knapsack is 13-4=9. Place next item w_3, in knapsack then the remaining weight of knapsack is 9-6=3. No fraction are allowed in calculation of upper bound so w_4, cannot be placed in knapsack.

Profit= p_1+p_2+ p_3,=10+10+12

So, Upper bound=32

To calculate Lower bound we can place w_4 in knapsack since fractions are allowed in calculation of lower bound.

Lower bound=10+10+12+ (3/9*18)=32+6=38

Knapsack is maximization problem but branch bound technique is applicable for only minimization problems. In order to convert maximization problem into minimization problem we have to take negative sign for upper bound and lower bound.

Therefore, upper bound (U) =-32

Lower bound (L)=-38

We choose the path, which has minimized difference of upper bound and lower bound. If the difference is equal then we choose the path by comparing upper bounds and we discard node with maximum upper bound.

Now we will calculate upper bound and lower bound for nodes 2, 3

For node 2, x_1=1, means we should place first item in the knapsack.

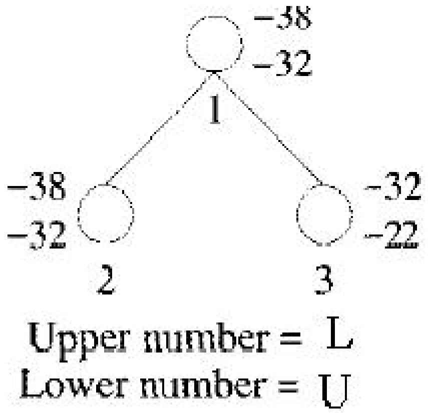
U=10+10+12=32, make it as -32

L=10+10+12+ (3/9*18) = 32+6=38, we make it as -38

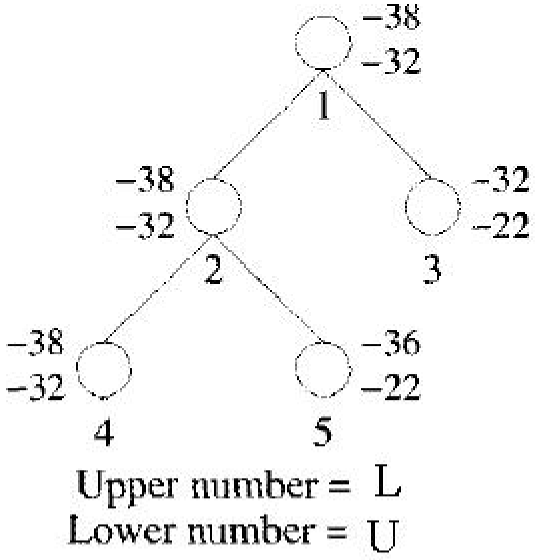
For node 3, x_1=0, means we should not place first item in the knapsack.

U=10+12=22, make it as -22

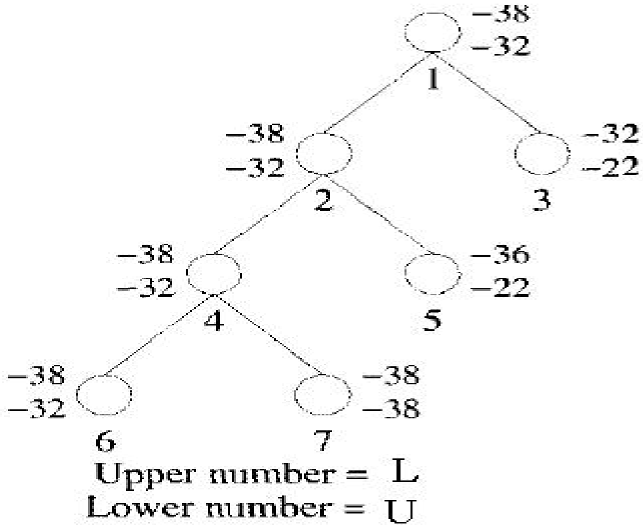
L=10+12+ (5/9*18) = 10+12+10=32, we make it as -32



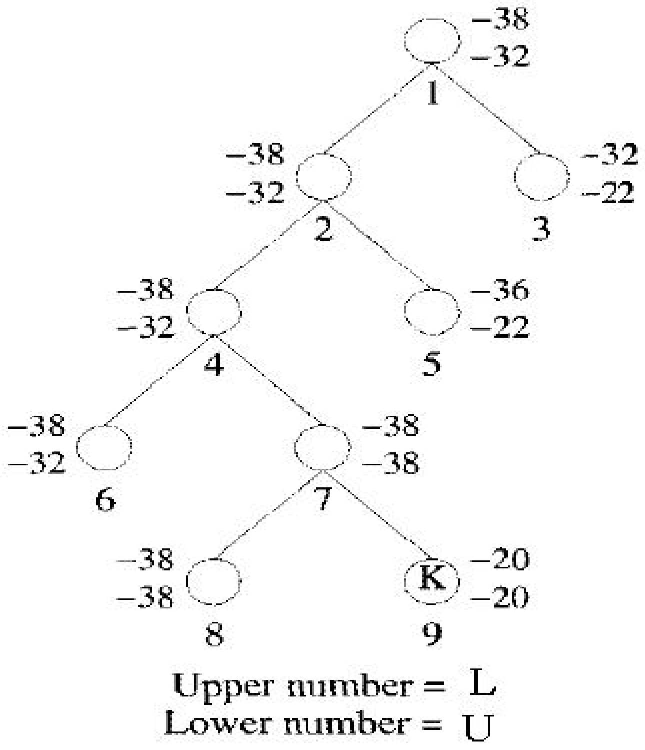
Choose node 2, since it has minimum cost.



Now we will calculate lower bound and upper bound of node 4 and 5.
Choose node 4, since it has minimum cost.



Now we will calculate lower bound and upper bound of node 6 and 7.
Choose node 7, since it has minimum cost and choosing node 6 is infeasible**.**



Now we will calculate lower bound and upper bound of node 8 and 9.

Choose node 8 as it has minimum cost. Discard node 9.

1→2→4→7→8

$X_1=1$

$X_2=1$

$X_3=0$

$X_4=1$

The solution for 0/1 knapsack problem is (x_1, x_2, x_3, x_4)=(1, 1, 0, 1)

Maximum profit is:

∑pixi=10*1+10*1+12*0+18*1

10+10+18=38.

FIFO Branch-and-Bound Solution

Now, let us trace through the FIFOBB algorithm using the same knapsack instance as in above Example.

Initially the root node, node 1 of following Figure, is the E-node and the queue of live nodes is empty. Since this is not a solution node, upper is initialized to u(l) = -32.

We assume the children of a node are generated left to right. Nodes 2 and 3 are generated and added to the queue (in that order).

The value of upper remains unchanged. Node 2 becomes the next E- node. Its children, nodes 4 and 5, are generated and added to the queue.

Node 3, the next-node, is expanded. Its children nodes are generated; Node 6 gets added to the queue. Node 7 is immediately killed as L (7) > upper.

Node 4 is expanded next. Nodes 8 and 9 are generated and added to the queue.

Then Upper is updated to u(9) = -38, Nodes 5 and 6 are the next two nodes to become B-nodes. Neither is expanded as for each, L > upper.

Node 8 is the next E-node. Nodes 10 and 11 are generated; Node 10 is infeasible and so killed. Node 11 has L (11) > upper and so is also killed. Node 9 is expanded next.

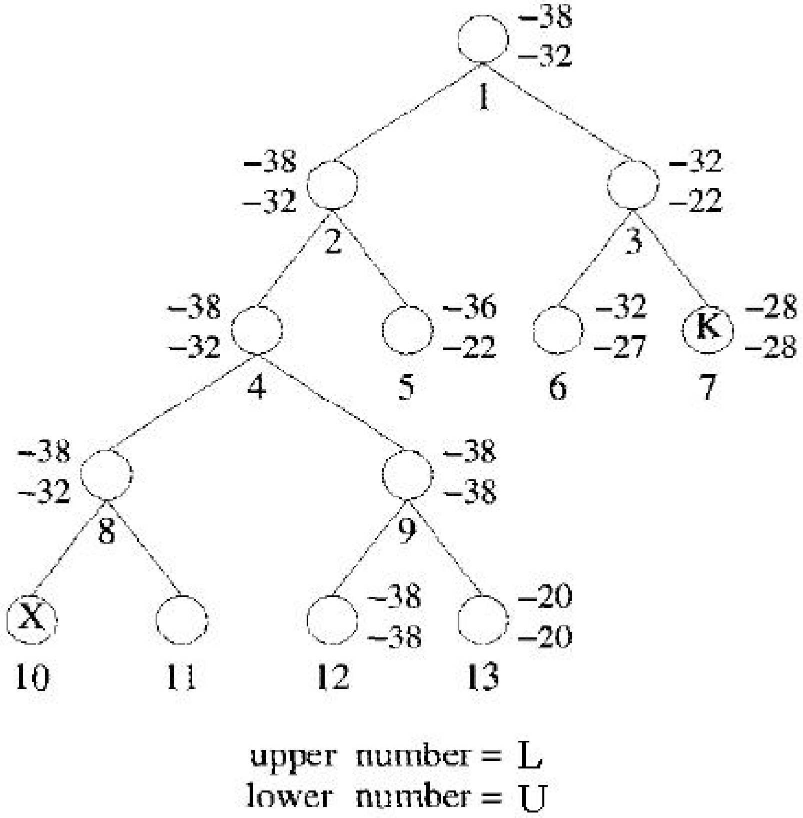
When node 12 is generated, 'Upper and ans are updated to -38 and 12 respectively.

Node 12 joins the queue of live nodes. Node 13 is killed before it can get onto the queue of live nodes as L (13) > upper.

The only remaining live node is node 12.

It has no children and the search terminates.

The value of upper and the path from node 12 to the root is output. So solution is X_1=1, X2=1, X3=0, X4=1.



3.TRAVELLING SALES PERSON PROBLEM

Let G = (V', E) be a directed graph defining an instance of the traveling salesperson problem. Let Cij equal the cost of edge (i, j), Cij = ∞ if (i, j) != E, and let IVI = n, without loss of generality, we can assume that every tour starts and ends at vertex 1.

Procedure for solving travelling sales person problem

1. Reduce the given cost matrix. A matrix is reduced if every row and column is reduced. This can be done as follows:

   Row Reduction:

   a) take the minimum element from first row, subtract it from all elements of first row, next take minimum element from the second row and subtract it from second row. Similarly apply the same procedure for all rows.

   b) Find the sum of elements, which were subtracted from rows.

   c) Apply column reductions for the matrix obtained after row reduction.

   Column Reduction:

   d) Take the minimum element from first column, subtract it from all elements of first column, next take minimum element from the second column and subtract it from second column. Similarly apply the same procedure for all columns.

   e) Find the sum of elements, which were subtracted from columns.

   f) Obtain the cumulative sum of row wise reduction and column wise reduction.

   Cumulative reduced sum=Row wise reduction sum + Column wise reduction sum.

   Associate the cumulative reduced sum to the starting state as lower bound and α as upper bound.

Calculate the reduced cost matrix for every node.

a) If path (i,j) is considered then change all entries in row **i** and column **j** of A to α.

b) Set A(j,1) to α.

c) Apply row reduction and column reduction except for rows and columns containing only α. Let **r** is the total amount subtracted to reduce the matrix.

d) Find ĉ(S)= ĉ(R)+A(i,j)+r.

Repeat step 2 until all nodes are visited.

Example: Find the LC branch and bound solution for the travelling sales person problem whose cost matrix is as follows.

$$\text{The cost matrix is } \begin{pmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

Step 1: Find the reduced cost matrix

Apply now reduction method:

Deduct 10 (which is the minimum) from all values in the 1st row.

Deduct 2 (which is the minimum) from all values in the 2nd row.

Deduct 2 (which is the minimum) from all values in the 3rd t row.

Deduct 3 (which is the minimum) from all values in the 4th row.

Deduct 4 (which is the minimum) from all values in the 5th row.

$$\text{The resulting row wise reduced cost matrix} = \begin{matrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{matrix}$$

Row wise reduction sum = 10+2+2+3+4=21.

Now apply column reduction for the above matrix:

Deduct 1 (which is the minimum) from all values in the 1st column.

Deduct 3 (which is the minimum) from all values in the 2nd column.

$$\text{The resulting column wise reduced cost matrix (A)} = \begin{matrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 3 & 12 & \infty \end{matrix}$$

Column wise reduction sum = 1+0+3+0+0=4.

Cumulative reduced sum = row wise reduction + column wise reduction sum.
=21+ 4 =25.

This is the cost of a root i.e. node 1, because this is the initially reduced cost matrix.

The lower bound for node is 25 and upper bound is ∞.

Starting from node 1, we can next visit 2, 3, 4 and 5 vertices. So, consider to explore the paths (1, 2), (1,3), (1, 4), (1,5).

The tree organization up to this as follows;

Variable **i** indicate the next node to visit.

Now consider the path (1, 2)

Change all entries of row 1 and column 2 of A to ∞ and also set A (2, 1) to ∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{matrix}$$

Apply row and column reduction for the rows and columns whose rows and column are not completely ∞. Then the resultant matrix is

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

Row reduction sum = 0 + 0 + 0 + 0 = 0
Column reduction sum = 0 + 0 + 0 + 0= 0
Cumulative reduction(r) = 0 + 0=0
Therefore, as ĉ(S)= ĉ(R)+A(1,2)+r ➔ ĉ(S)= 25 + 10 + 0 = 35.

Now consider the path (1, 3)

Change all entries of row 1 and column 3 of A to ∞ and also set A (3, 1) to ∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{matrix}$$

Apply row and column reduction for the rows and columns whose rows and column are not completely ∞

Then the resultant matrix is = $\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{matrix}$

Row reduction sum = 0
Column reduction sum = 11
Cumulative reduction(r) = 0 +11=11
Therefore, as ĉ(S)= ĉ(R)+A(1,3)+r
ĉ(S)= 25 + 17 +11 = 53.

Now consider the path (1, 4)

Change all entries of row 1 and column 4 of A to ∞ and also set A(4,1) to ∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{matrix}$$

Apply row and column reduction for the rows and columns whose rows and column are not completely ∞

$$\text{Then the resultant matrix is } = \begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{matrix}$$

Row reduction sum = 0
Column reduction sum = 0

Cumulative reduction(r) = 0 +0=0

Therefore, as ĉ(S)= ĉ(R)+A(1,4)+r

ĉ(S)= 25 + 0 +0 = 25.

Now Consider the path (1, 5)

Change all entries of row 1 and column 5 of A to ∞ and also set A(5,1) to ∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{matrix}$$

Apply row and column reduction for the rows and columns whose rows and column are not completely ∞

$$\text{Then the resultant matrix is =} \begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{matrix}$$

Row reduction sum = 5

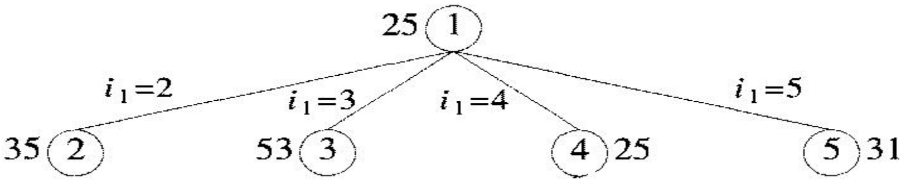
Column reduction sum = 0

Cumulative reduction(r) = 5 +0=0

Therefore, as ĉ(S)= ĉ(R)+A(1,5)+r

ĉ(S)= 25 + 1 +5 = 31.

The tree organization up to this as follows:



Numbers outside the node are $\hat{c}$ values

The cost of the between (1, 2) = 35, (1, 3) = 53, (1, 4) = 25, (1, 5) = 31. The cost of the path between (1, 4) is minimum. Hence the matrix obtained for path (1, 4) is considered as reduced cost matrix.

∞ ∞ ∞ ∞
∞
12 ∞ 11 ∞
0
A = 0 3 ∞ ∞
2
∞ 3 12 ∞
0

$$\begin{matrix} 11 & 0 & 0 & \infty & \infty \end{matrix}$$

The new possible paths are (4, 2), (4, 3) and (4, 5).

Now consider the path (4, 2)

Change all entries of row 4 and column 2 of A to ∞ and also set A(2,1) to ∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{matrix}$$

Apply row and column reduction for the rows and columns whose rows and column are not completely ∞

Then the resultant matrix is =

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{matrix}$$

Row reduction sum = 0

Column reduction sum = 0

Cumulative reduction(r) = 0 +0=0

Therefore, as ĉ(S)= ĉ(R)+A(4,2)+r

ĉ(S)= 25 + 3 +0 = 28.

Now consider the path (4, 3)

Change all entries of row 4 and column 3 of A to ∞ and also set A(3,1) to ∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{matrix}$$

Apply row and column reduction for the rows and columns whose rows and column are not completely ∞

Then the resultant matrix is =

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{matrix}$$

Row reduction sum = 2

Column reduction sum = 11

Cumulative reduction(r) = 2 +11=13

Therefore, as ĉ(S)= ĉ(R)+A(4,3)+r

ĉ(S)= 25 + 12 +13 = 50.

.Now consider the path (4, 5)

Change all entries of row 4 and column 5 of A to ∞ and also set A(5,1) to ∞.

$$\begin{matrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \end{matrix}$$

0 3 ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞
∞ 0 0 ∞ ∞

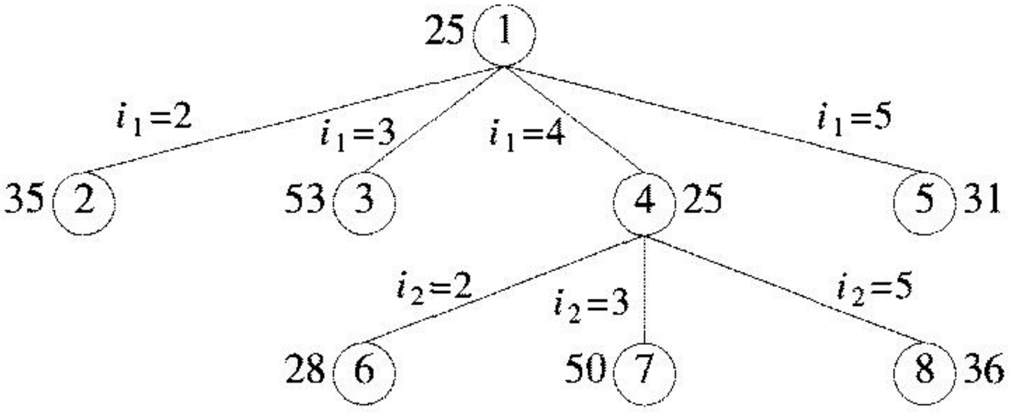
Apply row and column reduction for the rows and columns whose rows and column are not completely ∞

Then the resultant matrix is =

∞ ∞ ∞ ∞ ∞
1 ∞ 0 ∞ ∞
0 3 ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞
∞ 0 0 ∞ ∞

Row reduction sum =11
Column reduction sum = 0
Cumulative reduction(r) = 11 +0=11
Therefore, as ĉ(S)= ĉ(R)+A(4,5)+r
ĉ(S)= 25 + 0 +11 = 36.

The tree organization up to this as follows:



Numbers outside the node are ĉ values

The cost of the between (4, 2) = 28, (4, 3) = 50, (4, 5) = 36. The cost of the path between (4, 2) is minimum. Hence the matrix obtained for path (4, 2) is considered as reduced cost matrix. A=

∞ ∞ ∞ ∞ ∞
∞ ∞ 11 ∞ 0
0 ∞ ∞ ∞ 2
∞ ∞ ∞ ∞ ∞
11 ∞ 0 ∞ ∞

The new possible paths are (2, 3) and (2, 5).

Now Consider the path (2, 3):

Change all entries of row 2 and column 3 of A to ∞ and also set A(3,1) to ∞.

∞ ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ 2
∞ ∞ ∞ ∞ ∞
11 ∞ ∞ ∞ ∞

Apply row and column reduction for the rows and columns whose rows and

column are not
completely ∞

∞ ∞ ∞ ∞
∞
∞ ∞ ∞ ∞
∞
Then the resultant matrix is = ∞ ∞ ∞ ∞
0
∞ ∞ ∞ ∞
∞
0 ∞ ∞ ∞
∞

Row reduction sum =13
Column reduction sum = 0
Cumulative reduction(r) = 13 +0=13
Therefore, as ĉ(S)= ĉ(R)+A(2,3)+r
ĉ(S)= 28 + 11 +13 = 52.

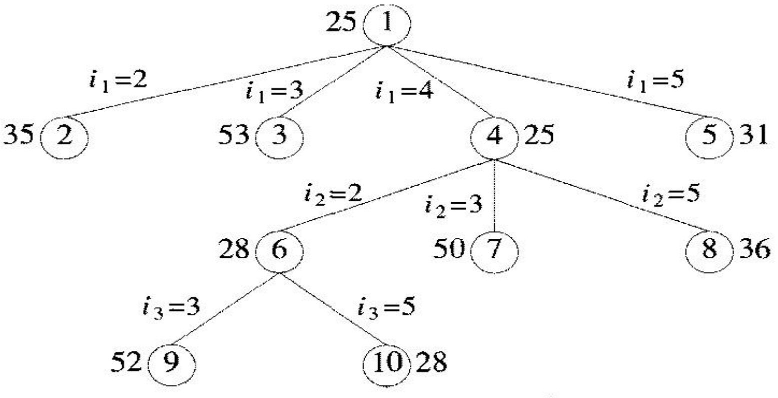
Now Consider the path (2, 5):
Change all entries of row 2 and column 5 of A to ∞ and also set A(5,1) to ∞.

∞ ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞
0 ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞
∞ ∞ 0 ∞ ∞

Apply row and column reduction for the rows and columns whose rows and column are not
completely ∞

∞ ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞
Then the resultant matrix is =0 ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞
∞ ∞ 0 ∞ ∞

Row reduction sum =0
Column reduction sum = 0
Cumulative reduction(r) = 0 +0=0
Therefore, as ĉ(S)= ĉ(R)+A(2,5)+r ➔ ĉ(S)= 28 + 0 +0 = 28.
The tree organization up to this as follows:



Numbers outside the node are $\hat{c}$ values

The cost of the between (2, 3) = 52 and (2, 5) = 28. The cost of the path between (2, 5) is minimum. Hence the matrix obtained for path (2, 5) is considered as reduced cost matrix.

$$A = \begin{matrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{matrix}$$

The new possible path is (5, 3).

Now c***onsider the path (5, 3):***

Change all entries of row 5 and column 3 of A to ∞ and also set A(3,1) to ∞. Apply row and column reduction for the rows and columns whose rows and column are not completely ∞

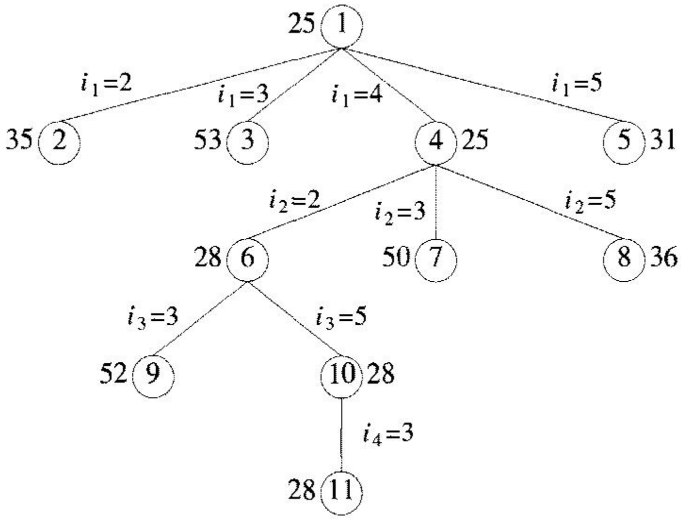
Then the resultant matrix is $= \begin{matrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{matrix}$

Row reduction sum =0
Column reduction sum = 0
Cumulative reduction(r) = 0 +0=0
Therefore, as ĉ(S)= ĉ(R)+A(5,3)+r
ĉ(S)= 28 + 0 +0 = 28.

The path travelling sales person problem is:

1→4→2→5→3→1**:**

The minimum cost of the path is: 10+2+6+7+3=28.



Numbers outside the node are $\hat{c}$ values

4.NP-Complete and NP-Hard problems

Basic concepts

For many of the problems we know and study, the best algorithms for their solution have computing times can be clustered into two groups;

1. Solutions are bounded by the **polynomial**- Examples include Binary search O(log n), Linear search O(n), sorting algorithms like merge sort O(n log n), Bubble sort O(n^2) &matrix multiplication O(n^3) or in general O(n^k) where k is a constant.
2. Solutions are bounded by a non-polynomial-Examples include travelling salesman problem O($n^2 2^n$) & knapsack problem O($2^{n/2}$). As the time increases exponentially, even moderate size problems cannot be solved.

So far, no one has been able to device an algorithm which is bounded by the polynomial for the problems belonging to the non-polynomial. However impossibility of such an algorithm is not proved.

Non deterministic algorithms:

We also need the idea of two models of computer (Turing machine): deterministic and non- deterministic. A deterministic computer is the regular computer we always thinking of; a non- deterministic computer is one that is just like we're used to except that is has unlimited parallelism, so that any time you come to a branch, you spawn a new "process" and examine both sides.

When the result of every operation is uniquely defined then it is called **deterministic algorithm**.

When the outcome is not uniquely defined but is limited to a specific set of possibilities, we call it **non deterministic** algorithm.

We use new statements to specify such **non deterministic** algorithms.

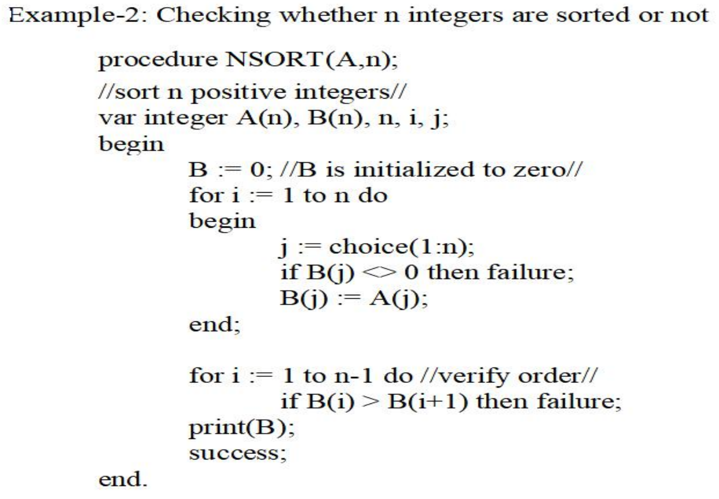
- **choice(S) -** arbitrarily choose one of the elements of set S
- **failure -** signals an unsuccessful completion
- **success -** signals a successful completion

The assignment ***X* = *choice(1:n)*** could result in X being assigned any value from the integer ***range[1..n]***. There is no rule specifying how this value is chosen.

"The nondeterministic algorithms terminates unsuccessfully if there is no set of choices which leads to the successful signal".

Example-1: Searching an element **x** in a given set of elements A(1:n). We are required to determine an index j such that A(j) = x or j = 0 if x is not present.

```
j := choice(1:n);
if A(j) = x then write(j); success(); endif
write('0'); failure();
```



Example-2: Checking whether n integers are sorted or not

```
procedure NSORT(A,n);
//sort n positive integers//
var integer A(n), B(n), n, i, j;
begin
        B := 0; //B is initialized to zero//
        for i := 1 to n do
        begin
                j := choice(1:n);
                if B(j) <> 0 then failure;
                B(j) := A(j);
        end;

        for i := 1 to n-1 do //verify order//
                if B(i) > B(i+1) then failure;
        print(B);
        success;
end.
```

"A nondeterministic machine does not make any copies of an algorithm every time a choice is to be made. Instead it has the ability to correctly choose an element from the given set".

A deterministic interpretation of the nondeterministic algorithm can be done by making unbounded parallelism in the computation. Each time a choice is to be made, the algorithm makes several copies of itself, one copy is made for each of the possible choices.

Decision vs Optimization algorithms

An optimization problem tries to find an optimal solution.

A decision problem tries to answer a yes/no question. Most of the problems can be specified in decision and optimization versions.

For example, Traveling salesman problem can be stated as two ways

- Optimization - find hamiltonian cycle of minimum weight,
- Decision - is there a hamiltonian

cycle of weight ≤ k? For graph coloring problem,

- Optimization – find the minimum number of colors needed to color the vertices of a graph so that no two adjacent vertices are colored the same color
- Decision - whether there exists such a coloring of the graph's vertices with no more than m colors?

Many optimization problems can be recast in to decision problems with the property that the decision algorithm can be solved in polynomial time if and only if the corresponding optimization problem.

P, NP, NP-Complete and NP-Hard classes

NP stands for Non-deterministic Polynomial time.

Definition: P is a set of all decision problems solvable by a deterministic algorithm in polynomial time.

Definition: NP is the set of all decision problems solvable by a nondeterministic algorithm in polynomial time. This also

implies P ⊆ NP

Problems known to be in P are trivially in NP — the nondeterministic machine just never troubles itself to fork another process, and acts just like a deterministic one. One example of a problem not in P but in NP is ***Integer Factorization***.

But there are some problems which are known to be in NP but don't know if they're in P. The traditional example is the decision-problem version of the Travelling Salesman Problem (***decision-TSP***). It's not known whether decision-TSP is in P: there's no known poly-time solution, but there's no proof such a solution doesn't exist.

There are problems that are known to be neither in P nor NP; a simple example is to enumerate all the bit vectors of length n. No matter what, that takes 2^n steps.

Now, one more concept: given decision problems P and Q, if an algorithm can transform a solution for P into a solution for Q in polynomial time, it's said that Q is poly-time reducible (or just reducible) to P.

The most famous unsolved problem in computer science is "whether P=NP or P≠NP? "

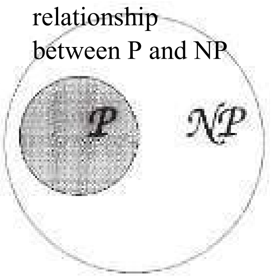


Figure: Commonly believed relationship between P and NP

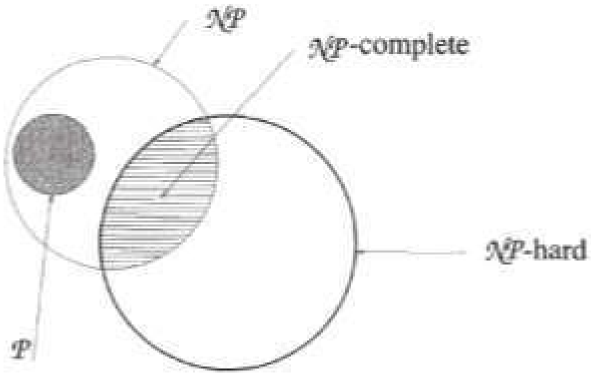


Figure: Commonly believed relationship between P, NP, NP- Complete and NP-hard problems

Definition: A decision problem D is said to be **NP-complete** if:

1. it belongs to class NP
2. every problem in NP is polynomially reducible to D

The fact that closely related decision problems are polynomially reducible to each other is not very surprising. For example, Hamiltonian circuit problem is polynomially reducible to the decision version of the traveling salesman problem.

NP-Complete problems have the property that it can be solved in polynomial time if all other NP-Complete problems can be solved in polynomial time. i.e if anyone ever finds a poly-time solution to one NP-complete problem, they've automatically got one for ***all*** the NP-complete problems; that will also mean that P=NP.

Example for NP-complete is CNF-satisfiability problem. The CNF-satisfiability problem deals with boolean expressions. This is given by Cook in 1971. The CNF-satisfiability problem asks whether or not one can assign values true and false to variables of a given boolean expression in its CNF form to make the entire expression true.

Over the years many problems in NP have been proved to be in P (like Primality Testing). Still, there are many problems in NP not proved to be in P. i.e. the question still remains whether P=NP? NP Complete Problems helps in solving this question. They are a subset of NP problems with the property that all other NP problems can be reduced to any of them in polynomial time. So, they are the hardest problems in NP, in terms of running time. If it can be showed that any NP-Complete problem is in P, then all problems in NP will be in P (because of NP-Complete definition), and hence P=NP=NPC.

NP Hard Problems - These problems need not have any bound on their running time. If any NP-Complete Problem is polynomial time reducible to a problem X, that problem X belongs to NP-Hard class. Hence, all NP-Complete problems are also NP-Hard. In other words if a NP-Hard problem is non-deterministic polynomial time solvable, it is a NP- Complete problem. Example of a NP problem that is not NPC is Halting Problem.

If a NP-Hard problem can be solved in polynomial time then all NP-Complete can be solved in polynomial time.

“All NP-Complete problems are NP-Hard but not all NP-Hard problems are not NP-Complete.”NP-Complete problems are subclass of NP-Hard

The more conventional optimization version of Traveling Salesman Problem for finding the shortest route is NP-hard, not strictly NP-complete.