

Branch - AND - Bound

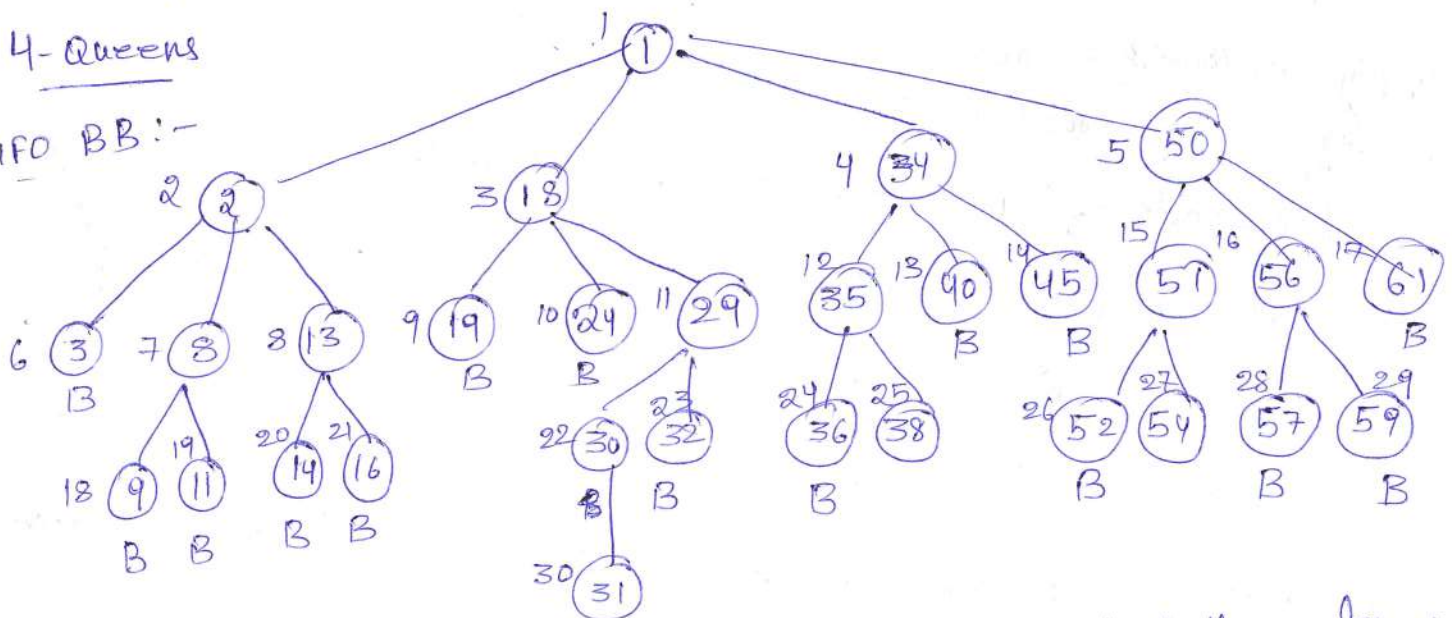
Branch-and-bound refers to all state space search method in which all children of the E-node are generated before any other live node can become the E-node.

2 types

1. a BFS-like state space search, called FIFO search, such as list of live nodes in a first-in-first-out list (or queue)
2. D search-like state space search, called LIFO search, such as list of live nodes in a last-in-first-out list (or stack)

4-Queens

1. FIFO BB:-



Numbers outside the nodes give the order in which the nodes are generated in FIFO BB.

At the time answer node, node 31, is reached live nodes left are 38 and 54.

2. Least cost (LC) search:-

E-node gives any preference to a node that has a very good chance of getting the search to an answer node quickly.

At node 30, one move to 31 would lead to answer node, but FIFO requires the expansion of all live nodes generated, before node 30 was expanded.

To speed up the search for an answer node by using an "intelligent" ranking func $\hat{c}(\cdot)$ for live nodes, to select next E-node.

Assign ranks on the basis of cost needed to reach an answer node from the live node.

Possible cost measures

1. The no. of nodes in the subtree 'x' that need to be generated before an answer node is generated.

2. The no. of levels the nearest answer node (in the subtree x) is from x.

Apply method 2:- measure 2:-

From fig Cost of root is 4

↑
bcoz node 31 is four levels from node 1.

Cost " (18) & (34) - is 3

" " (29) & (35) - is 2

" " (30) & (38) - is 1

The costs of all remaining nodes on levels 2, 3 & 4 are respectively greater than 3, 2, & 1. & etc - -

Let $\hat{g}^1(x)$ be an estimate of an additional effort needed to reach an answer node from x. Node x is assigned a rank using a func $\hat{c}(\cdot)$ such that $\hat{c}(x) = f(h(x)) + \hat{g}^1(x)$, where $h(x)$ is the cost of reaching 'x' from the root and $f(\cdot)$ is any nondecreasing func.

A search strategy that uses a cost func $\hat{c}(n) = f(h(n)) + \hat{g}(n)$ (2) to select the next E-node, would always choose for its next E-node a live node with least $\hat{c}(\cdot)$. Hence, this is called LC-search (least cost).

If $\hat{g}(n) \equiv 0$ and $f(h(n)) = \text{level of node } n$, then a LC-search generates nodes by levels. This is same as BFS.

If $f(h(n)) \equiv 0$ and $\hat{g}(n) \geq \hat{g}(y)$, whenever y is a child of n , then the search is essentially a D-search.

An LC-search coupled with bounding functions is called as LC branch-and-bound search.

In LC-searches, cost function $c(\cdot)$ is defined as: if x is an answer node, then $c(x)$ is the cost (level, computational difficulty) of reaching x from the root of the state space tree. If x is not an answer node, then $c(x) = \infty$ providing the subtree x contains no answer node; otherwise $c(x) =$ the cost of a min cost answer node in the subtree x .

Ex 1 The 15-puzzle : An Example:-

Consists of 15 numbered tiles on a square frame with capacity of 16 tiles. For the given initial arrangement, we need to generate Goal arrangement, through a series of legal moves.

Fig(a)

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

Initial

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Goal.

Fig(b)

The only legal moves are ones in which a tile adjacent to empty spot (E.s) is moved to E.s.

Each move creates a new arrangement of the tiles.

These arrangements are called the states of the puzzle.

The initial and goal arrangements are called the initial & goal states.

A state is reachable from initial state iff there is a sequence of legal moves from initial state to this state.

- 16! different arrangements of tiles on the frame exists.

of these only one-half are reachable from any given initial state.
So, determine whether the goal state is reachable from the initial state.

Let position (i) be the pos num in the initial state of the tile numbered i .

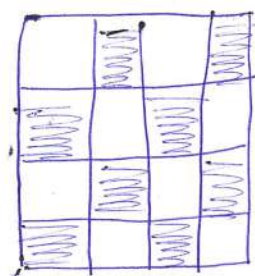
Ex:- position (16) denote pos of empty spot.

For any state, let $lev(i)$ be the no. of tiles j such that $j < i$ and position(j) > position(i).

Ex For any state, of Fig (a) $lev(1) = 0$, $lev(4) = 1$ and $lev(12) = 6$.

Let $x = 1$ if in the initial state the empty spot is at one of the shaded positions of Fig(c)

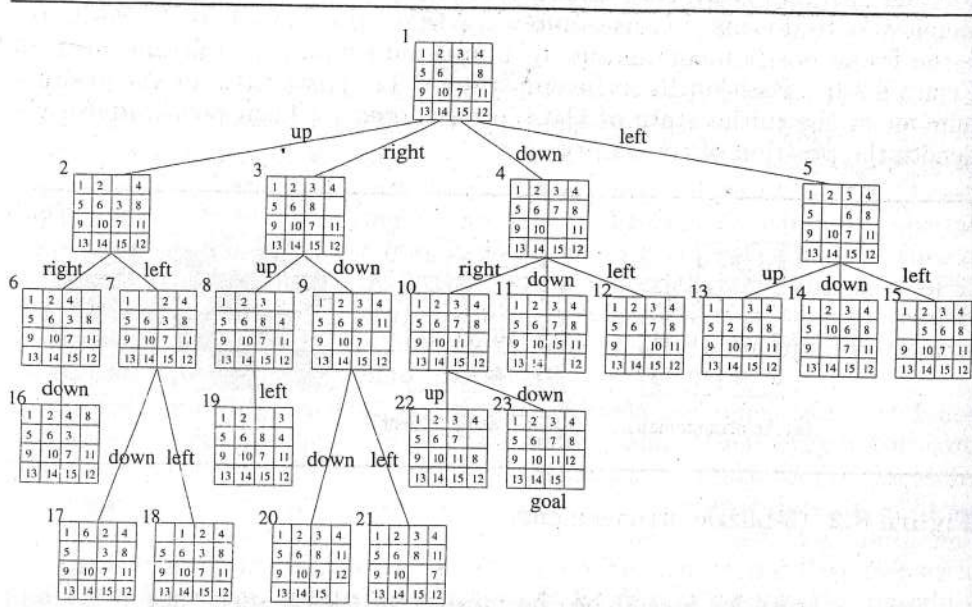
and $x = 0$ if it is at one of the remaining positions.



Fig(c)

Theorem:-

The goal state of Fig (b) is reachable from the initial state iff $\sum_{i=1}^{16} lev(i) + x$ is even.



Edges are labeled according to the direction in which the empty space moves

Figure 8.3 Part of the state space tree for the 15-puzzle

No node p has a child state that is the same as p 's parent.
The subtree eliminated in this way is already present in the tree and may root parent (p)

A depth-first state space tree generation will result in the subtree in the following figure

385

Fig 4

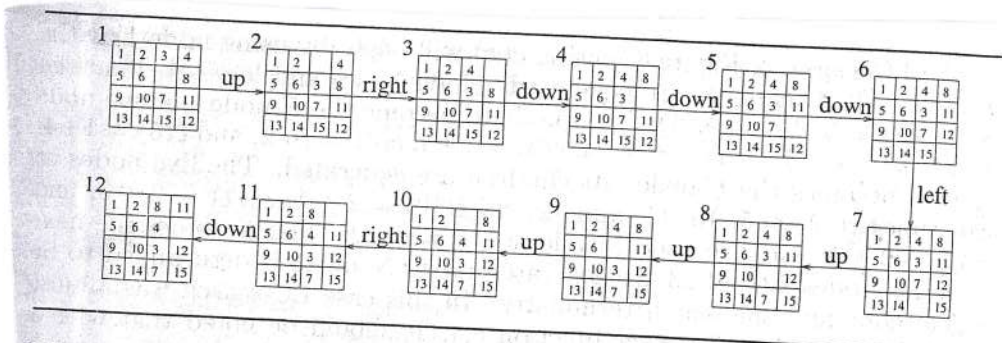


Figure 8.4 First ten steps in a depth first search

Associate a cost $c(n)$ with each node 'n' in the state space tree.
cost $c(n)$ is the length of a path from the root to a nearest goal node if any in a subtree with root n.

In Fig 3, $c(1) = c(4) = c(10) = c(23) = 3$.

Begin with root as the E-node and generate a child node with $C()$ - value the same as the root.

So, children nodes 2, 3, & 5 are eliminated and only node 4 becomes a live node. This ~~me~~ becomes the next E-node.

Its first child, node 10, has $c(10) = c(4) = 3$.

The remaining children are not generated.

Node 4 dies and node 10 becomes the E-node.

In generating node 10's children, node 22 is killed immediately as $c(22) > 3$.

Node 23 is generated next.

It is a goal node and the search terminates.

In this search strategy, the only nodes to become E-nodes are nodes on the path from the root to a nearest goal node.

But it is impractical to compute $c(n)$.

$\Rightarrow \hat{c}(n) = f(n) + g(n) - f(n)$ - length of path from the root to node n.
 $g(n)$ - is an estimate of the length of a path from node n to the goal.

$\hat{g}(u)$ = number of nonblank tiles not in their goal position. (4)

Ex.

1	2	3	4
5	6		8
9	10	11	12
13	14	15	7

→ here $\hat{g}(u) = 1$

b'coz only tile 7 is not in its final spot
but it needs many ^{moves} more than $\hat{g}(u)$

So, $\hat{c}(u)$ is a lower bound on $c(u)$

Fig 3 LC-search

Using $\hat{c}(u)$ begin using node 1 as E-node. All its children are generated. Node 1 dies & leaves behind live nodes 2, 3, 4 & 5.

Next node to become the E-node is a live node with least $\hat{c}(u)$

Then $\hat{c}(2) = 1+4$

$\hat{c}(3) = 1+4$

$\hat{c}(4) = 1+2$

$\hat{c}(5) = 1+4$

Node 4 becomes E-node. Its children are generated.

The live nodes at this time are 2, 3, 5, 10, 11, 12

$\hat{c}(10) = 2+1$

→ live node = 10, This becomes next E-node.

$\hat{c}(11) = 2+3$

$\hat{c}(12) = 2+3$

Next nodes 22 & 23 are generated

23 is goal & search terminates.

Control Abstractions for LC-Search

Let t be a state space tree

$cc()$ - a cost func for the nodes in t .

If x is a node in t , then $c(x)$ is the min cost of any answer node in the subtree with root x .

$c(t)$ - is the cost of a min-cost answer node in t .

But, as $cc()$ is impractical to compute, we use a heuristic $\hat{c}$ to estimate $cc()$

listnode = record {

listnode *next, *parent; float cost;

}

Algorithm LCSearch (t)

// Search t for an answer node

{ if t is an answer node then output t and return;

$E := t$ // E-node

Initialize the list of live nodes to be empty;

repeat

{ for each child x of E do

{ if x is an answer node then o/p the path from x to t and return;

Add(x); // x is a new live node.

$(x \rightarrow \text{parent}) := E$ // Pointer for path to root.

}

if there are no more live nodes then

{ write ("No answer node"); return;

}

$E := \text{Least}();$

$\}$ until (false);

$\}$

$\text{Add}(x)$ - ~~to delete and~~ ^{to} ~~add~~ ^{new} a live node with least $\hat{c}(x)$ from ~~list of live nodes~~. to add a new live node x to list of live nodes.

$\text{Least}()$ - finds a live node with least $\hat{c}(x)$

- used to delete a live node with least cost, from list of live nodes

$\text{LCSearch}()$

- ops the path from the answer node it finds to the root node t .

Associate a parent field which gives the parent of node x .

When an answer node g is found, path from g to t can be determined

by following a seq of parent values starting from the current

E-node (which is the parent of g) and ending at node t .

LC search terminates only when either an answer node is found or the entire state space tree has been generated and searched.

Similarity b/w LC search & BFS & D-search of a state space tree.

- If the list of live nodes is implemented as a queue with $\text{Least}()$ and $\text{Add}(x)$ being algos to delete an elt from and add an elt to the queue, then LC search will be transformed to a FIFO search schema.

- If the list of live nodes is implemented as a stack with $\text{Least}()$ and $\text{Add}(x)$ being algos to delete and add elts to the stack with $\text{Least}()$ and $\text{Add}(x)$, then $\text{LCSearch}()$ will carry out a LIFO search of the state space tree.

So, LC, FIFO & LIFO are same, only difference is implementation of the list of live nodes i.e. in selection rule to obtain next E-node.

Bounding:

A branch-and-bound method searches a state space using any search mechanism in which all the children of the E-node are generated before another node becomes E-node.

Assume each answer node x has a cost $c(x)$ associated with it and that a min-cost answer node is to be found.

3 common search strategies - FIFO
- LIFO
- LC.

Cost fun $\hat{c}(\cdot)$ such that $\hat{c}(x) \leq c(x)$ is used to provide lower bounds on solutions obtainable from any node x .

If 'upper' is an upper bound on the cost of a min-cost soln, then all live nodes x with $\hat{c}(x) > \text{upper}$ may be killed as all answer nodes reachable from x to have cost $c(x) \geq \hat{c}(x) > \text{upper}$.

The starting value for 'upper' can be obtained by some heuristic or can be set to ∞ .

Each time a new answer node is found, the value of 'upper' can be updated.

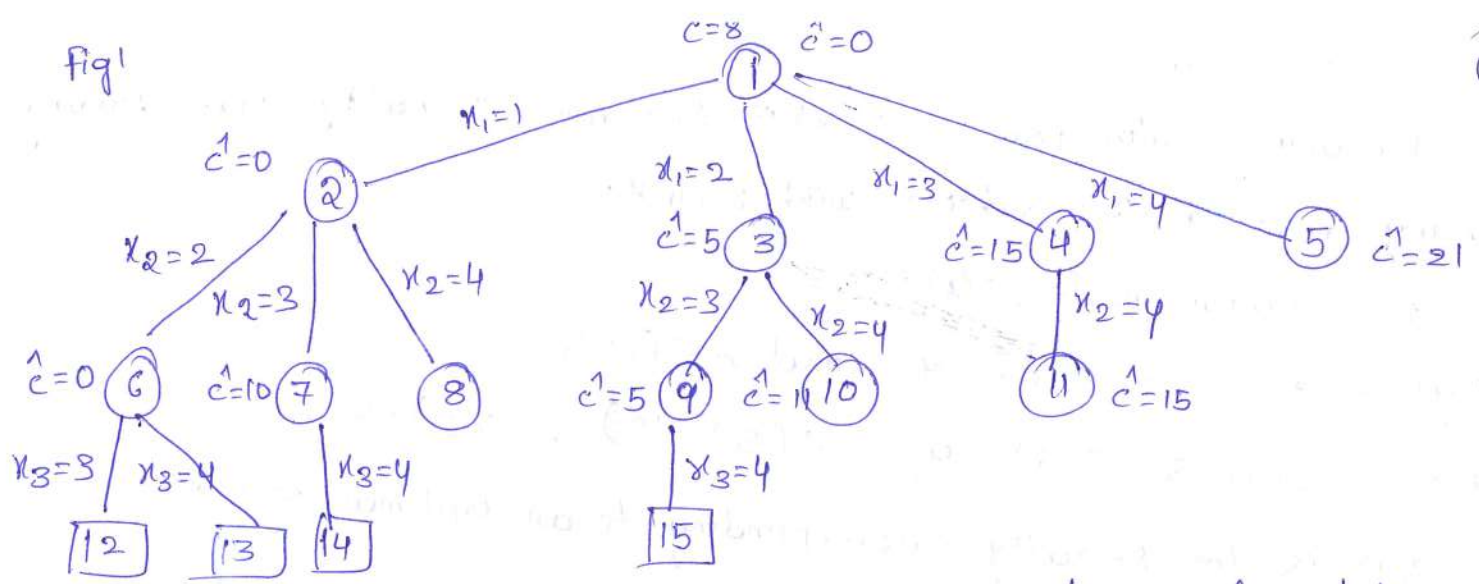
Ex Job sequencing with deadlines
 $n=4$, $(p_1, d_1, t_1) = (5, 1, 1)$, $(p_2, d_2, t_2) = (10, 3, 2)$, $(p_3, d_3, t_3) = (6, 2, 1)$ and $(p_4, d_4, t_4) = (3, 1, 1)$.

2 solns - var size
- fixed size.

Job i requires t_i units of processing time. If the processing is not completed by deadline d_i , then a penalty p_i is incurred. Objective is to select a subset J of the ' n ' jobs such that all jobs in J can be completed by their deadline, with minimum penalty.

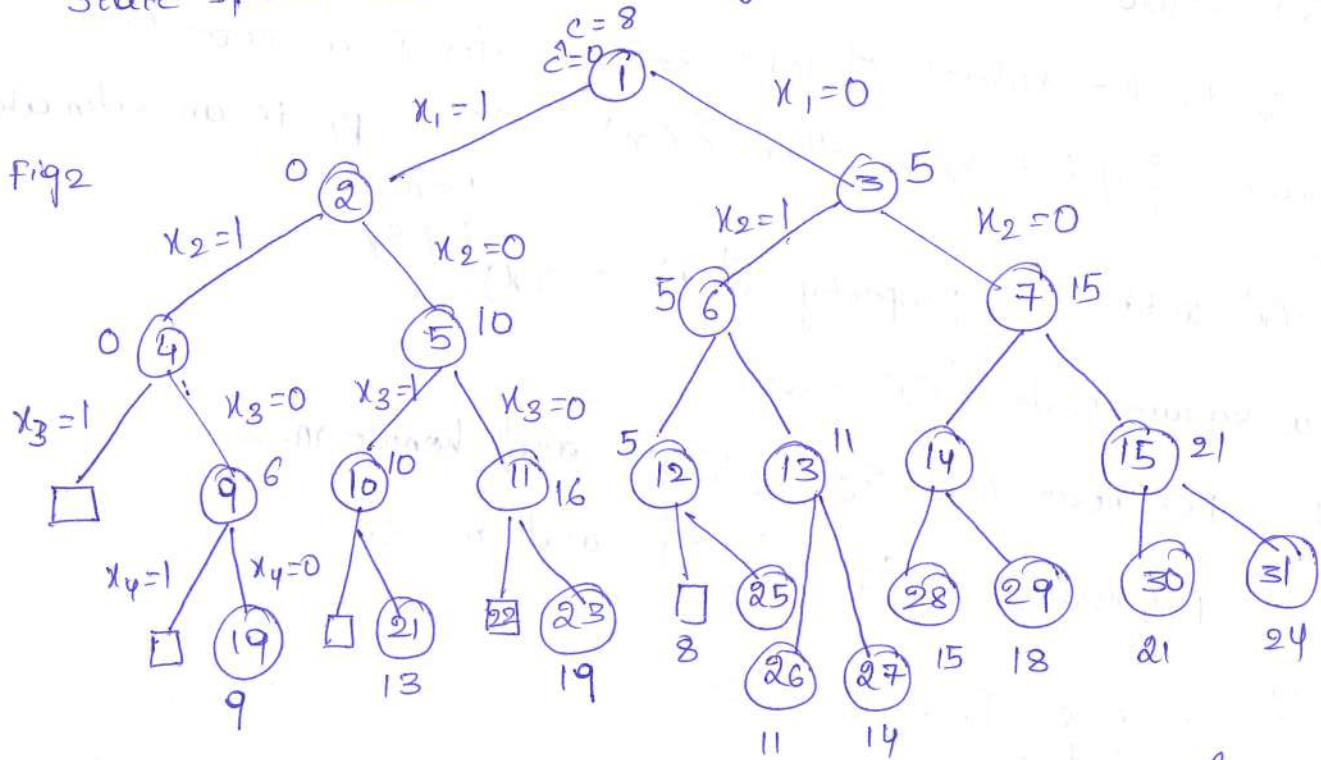
(6)

Fig 1



state space tree corresponding to var tuple size formulation.

Fig 2



State space tree corresponding to fixed tuple size formulation.

Square nodes represent infeasible subsets.

In Fig 1 all non square nodes are answer nodes.

Node 9 represents an optimal soln and is the only min-cost answer node. For this node $J = \{2, 3\}$ and the penalty cost = 8.

In Fig 2 only non square leaf nodes are answer nodes.

Node 25 represents the optimal soln and is also a min-cost answer node. This node $J = \{2, 3\}$ & penalty cost = 8.

→ For both figures.

For any circular node x , $c(x)$ is the min penalty corresponding to any node in the subtree with root x .

for square node, $c(x) = \infty$

Fig 1 :- $c(3) = 8$, $c(2) = 9$ and $c(1) = 8$

Fig 2 $c(1) = 8$, $c(2) = 9$, $c(5) = 13$, $c(6) = 8$.

$c(i)$ is the penalty corresponding to an optimal selection J .

⇒ $\hat{c}(x)$ bound :- $\hat{c}(x) \leq c(x) \forall x$

Let S_x be the subset of jobs selected for J at node x .

If $m = \max \{i / i \in S_x\}$, then $\hat{c}(x) = \sum_{\substack{i < m \\ i \notin S_x}} p_i$ is an estimate

for $c(x)$ with the property $\hat{c}(x) \leq c(x)$.

For a square node $\hat{c}(x) = \infty$

Fig 1, For node 6, $S_6 = \{1, 2\}$ and hence $m = 2$.

for node 7, $S_7 = \{1, 3\}$ and $m = 3$.

$$\therefore \sum_{\substack{i < 2 \\ i \notin S_2}} p_i = p_2 = 10.$$

Fig 2, node 12 corresponds to the omission of job 1 and hence penalty of 5.

node 13 corresponds to omission of jobs 1 and 3 and penalty is 11.

→ upper bound $u(x)$ on the cost of a min-cost answer node in the subtree x is $u(x) = \sum_{i \notin S_x} p_i$.

$u(x)$ is the cost of the soln S_x corresponding to node x .

FIFO Branch-and-bound :- for sum-of-subsets problem

(7)

A FIFO branch-and-bound algo for the job sequencing problem begin with upper = ∞ . (or upper = $\sum_{1 \leq i \leq n} P_i$) as an upper bound on the cost of the min-cost answer node.

Fig 1, start with node 1. Nodes with 2, 3, 4 and 5 are generated.

$u(2) = 19$ $\rightarrow$ correspond to inclusion job 1; $u(2)$ is obtained by summing the penalties of all the other jobs.

$u(3) = 14$ $\rightarrow$ upper is updated, when node 3 is generated.

$u(4) = 18$

$u(5) = 21$.

Since $\hat{c}(4)$ and $\hat{c}(5)$ are greater than upper, nodes 4 and 5 gets killed or bounded, only nodes 2 and 3 remain alive.

Node 2 becomes the next E-node.

Its children, nodes 6, 7 & 8 are generated.

$u(6) = 9$, so upper is updated to 9.

$\hat{c}(7) = 10 > \text{upper}$, node 7 gets killed.

Node 8 is infeasible and so it is killed.

Next, node 3 becomes the E-node.

Nodes 9 and 10 are now generated.

$u(9) = 8 \rightarrow$ so upper becomes 8.

Cost $\hat{c}(10) = 11 > \text{upper}$, this node is killed.

Next E-node is node 6.

Both its children are infeasible.

Node 9's only child is also infeasible.

Final min cost $z = 8$, The min cost answer node is ____.

391

26/11/20

FIFO-based branch & bound with appropriate $\hat{c}(\cdot) \leq u(\cdot)$ is called FIFOBB.

LC Branch-and-bound - for sum-of-subsets problem.

Refer

Fig 1: - Search begin with upper = ∞
node 1 is first E-node.

when node 1 is expanded nodes 2, 3, 4 & 5 are generated in that order.

As in FIFO BB, upper is updated 14 when node 3 is generated.
nodes 4 and 5 are killed, as $\hat{c}(4) > \text{upper}$, & $\hat{c}(5) > \text{upper}$.
Node 2 is next E-node as $\hat{c}(2) = 0$, $\hat{c}(3) = 5$.

Nodes 6, 7 & 8 are generated.

Upper is updated to 9, when node 6 is generated
so, node 7 is killed as $\hat{c}(7) = 10 > \text{upper}$.

Node 8 is infeasible and so killed.

The only live nodes now are nodes 3 and 6.

Node 6 is the next E-node as $\hat{c}(6) = 0 < \hat{c}(3)$.

Both its children are infeasible.

Node 3 becomes the next E-node.

when node 9 is generated, upper is updated to 8 as

$$u(9) = 8$$

So, node 10 with $\hat{c}(10) = 11$ is killed on generation.

Node 9 becomes the next E-node.

Its only child is infeasible. No live nodes remain.

This search terminates with node 9 representing the min-cost answer node.

LC(FIFO)-based branch-and-bound algo with an appropriate

$\hat{c}(\cdot)$ and $u(\cdot)$ is called LCB (LIFO BB)

Traveling Salesperson:-

(8)

Let $G = (V, E)$ be a directed graph defining an instance of the TSP.

Cost $c_{ij} = \infty$, if $(i, j) \notin E$

Let $|V| = n$

without loss of generality - tour starts from vertex 1 and ends at vertex 1.

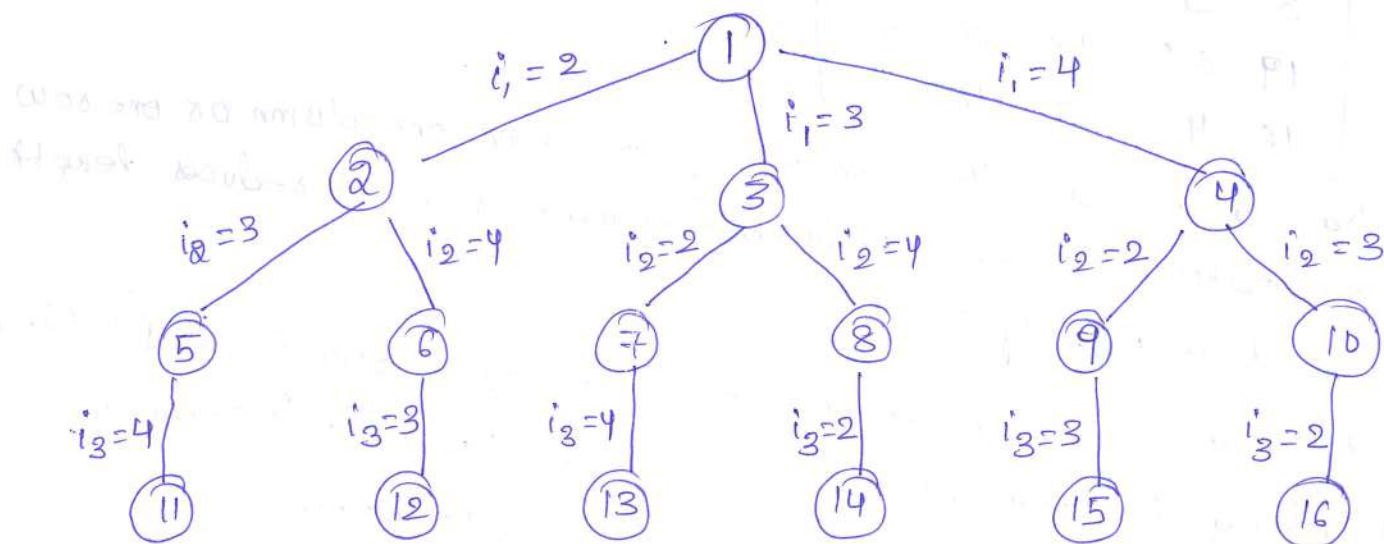
$|S| = (n-1)!$; - S-solution space.

where $S = \{1, \pi, 1/\pi\}$ is a permutation of $(2, 3, \dots, n)$

The size of S can be reduced by restricting S so that

$(1, i_1, i_2, \dots, i_n, 1) \in S$ if $(i_j, i_{j+1}) \in E$, $0 \leq j \leq n-1$, and $i_0 = i_n = 1$.

State space tree for TSP with $n=4$ and $i_0 = i_4 = 1$



Each leaf node L is a soln node and represents the tour defined by the path from the root to L .

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

LCBB for TSP:-

Define cost fun $c(\cdot)$ and $c^1(u)$, $u(\cdot)$, as $c^1(x) \leq c(x) \leq u(x)$

The cost $c(.)$ is such that the soln made with least $c(.)$ corresponds to a shortest tour in G .

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \\ \text{if } A \text{ is a leaf.} \\ \text{Cost of a min-cost-leaf in the subtree } A, \\ \text{if } A \text{ is not a leaf.} \end{cases}$$

$\# c^1(.)$, such that $c^1(A) \leq c(A)$ for all A ; is the length of the path defined at node A .

Path defined at nodes, is $i_0, i_1, i_2 = 1, 2, 4$, with edges $(1,2)$ & $(2,4)$.

A better $c^1(.)$ can be obtained by using reduced cost matrix

corresponding to G .

→ Solve the following instance of TSP using LCBB.
Let the cost matrix is

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

Subtract a constant 't' from every entry in one column or one row of cost matrix, to get reduced cost matrices, this reduces length of every tour exactly t-

$\#$ If 't' is chosen to be the min entry in row i (col j), then subtracting it from all entries in row i (col j), introduces a zero into row i (col j). Repeat this if needed.

Step 1

Row reduction a) Subtract 10, 2, 2, 3, 4 from ~~all~~ rows 1 to 5 i.e. 1, 2, 3, 4, 5
Col reduction b) ^{next} " 1 and 3 from cols 1 & 3

Row reduction

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 & 10 \\ 15 & \infty & 16 & 4 & 2 & 2 \\ 3 & 5 & \infty & 2 & 4 & 2 \\ 19 & 16 & 18 & \infty & 3 & 3 \\ 16 & 4 & 7 & 16 & \infty & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

$= 21$

9

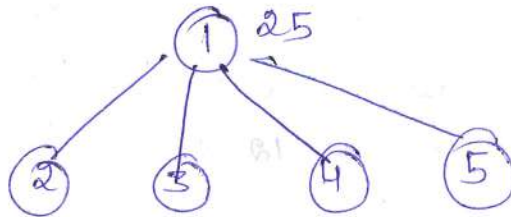
Column Reduction

$$\begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

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

$$\begin{bmatrix} 1 & - & 3 & - & - \end{bmatrix} = 4$$

Total amount subtracted, $t = 21 + 4 = 25$



With every node in state space tree, associate a reduced cost matrix.

Let A be the reduced cost matrix for node R . Let S be the child of R such that the edge (R, S) corresponds to including edge (i, j) in the tour. If S is not a leaf node then the reduced cost matrix for node S can be obtained as follows.

1. Change all entries in row i col j of A to ∞
2. Set (j, i) to ∞ .

3. Apply row reduction & col reduction except for rows i columns containing only ∞ .

4. The total cost for node '3' can be calculated as,

$$\hat{C}(s) = \hat{C}(R) + A(i, j) + \gamma$$

where γ is the total amount subtracted in step 3.

For leaf nodes $\hat{C}(i) = C(i)$

Step 2:-

→ Consider path (1, 2) : node 2

Row 1 = ∞
Col 2 = ∞
(2, 1) = ∞

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

Row reduction = col reduction = 0

$$\Rightarrow \hat{C}(2) = \hat{C}(1) + A(1, 2) + \gamma = 25 + 10 + 0 = 35$$

→ Consider path (1, 3) : node 3

Row 1 = ∞
Col 2 = ∞
(3, 1) = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & \infty & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 12 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

11

→ Apply row reduction & column reduction, $\gamma = 11$

$$\hat{C}(3) = \hat{C}(1) + A(1, 3) + \gamma = 25 + 17 + 11 = 53$$

→ Consider path (1, 4) : node 4

$\gamma = 0$

$$\begin{aligned} \hat{C}(4) &= \hat{C}(1) + A(1, 4) + \gamma \\ &= 25 + 0 + 0 \\ &= 25 \end{aligned}$$

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

→ consider path (1,5) : node 5

(10)

Row reduction Row 2 - 2

Row 4 - 3

Result is

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

$$\begin{aligned} C^1(5) &= C^1(1) + A(1,5) + \delta \\ &= 25 + 1 + 5 = 31 \end{aligned}$$

Step 3

$$\begin{aligned} \text{Since min cost} &= \min \{ 35, 53, 25, 31 \} \\ &= 25 \end{aligned}$$

Select node 4

⇒ Start with matrix

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

⇒ Path (1-4-2) : node 6

change all entries of fourth row & second column to ∞ .

$$\text{Set } \boxed{A(2,1) = \infty}$$

Apply row reduction & column reduction, $\delta = 0$

$$\begin{aligned} C^1(2) &= C^1(4) + A(4,2) + \delta \\ &= 25 + 3 + 0 \end{aligned}$$

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

→ Path (1, 4, 3) ; node 7.

$$4^{\text{th}} \text{ row} = \infty$$

$$3^{\text{rd}} \text{ col} = \infty$$

$$\text{Set}(3, 1) = \infty$$

$$\text{Row reduction : row 3} = 2$$

$$\text{Col " : col 1} = 11$$

$$\Rightarrow r = 2 + 11 = 13$$

$$\begin{aligned} C^1(3) &= C^1(4) + A(4, 3) + r \\ &= 25 + 12 + 13 = 50 \end{aligned}$$

→ Path (1, 4, 5) ; node 8

$$4^{\text{th}} \text{ row} = \infty$$

$$5^{\text{th}} \text{ col} = \infty$$

$$A(5, 1) = \infty$$

$$\text{Row reduction : row 2} = 11$$

$$r = 11 + 0 = 11$$

$$\begin{aligned} C^1(5) &= C^1(4) + A(4, 5) + r \\ &= 25 + 0 + 11 \\ &= 36 \end{aligned}$$

$$\Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

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

Step 4

$$\text{Min cost} = \min \{ 28, 50, 36 \} = 28, \text{ So select node 2.}$$

$$\text{So, start step 4 with } A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

→ consider (1-4-2-3)

$$2^{\text{nd}} \text{ row} = \infty$$

$$3^{\text{rd}} \text{ col} = \infty$$

$$\text{Set}(3, 1) = \infty$$

$$\text{Row reduction, row 2} = 2$$

$$\Rightarrow r = 2 + 11 = 13$$

$$\begin{aligned} C^1(3) &= C^1(2) + A(2, 3) + r \\ &= 28 + 11 + 13 \\ &= 52 \end{aligned}$$

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

→ Consider (1-4-2-5)

$$c^1(5) = c^1(2) + A(5,3) + \delta$$

$$= 28 + 0 + 0$$

$$= 28$$

⇒

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

2nd row = ∞

5th col = ∞

A(5,1) = ∞

row red & col red ⇒ δ = 0

Step 5

$$\text{Min cost} = \min \{ 52, 28 \} = 28$$

Select node 5.

Consider (1-4-2-5-3) ⇒

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞

5th row = ∞

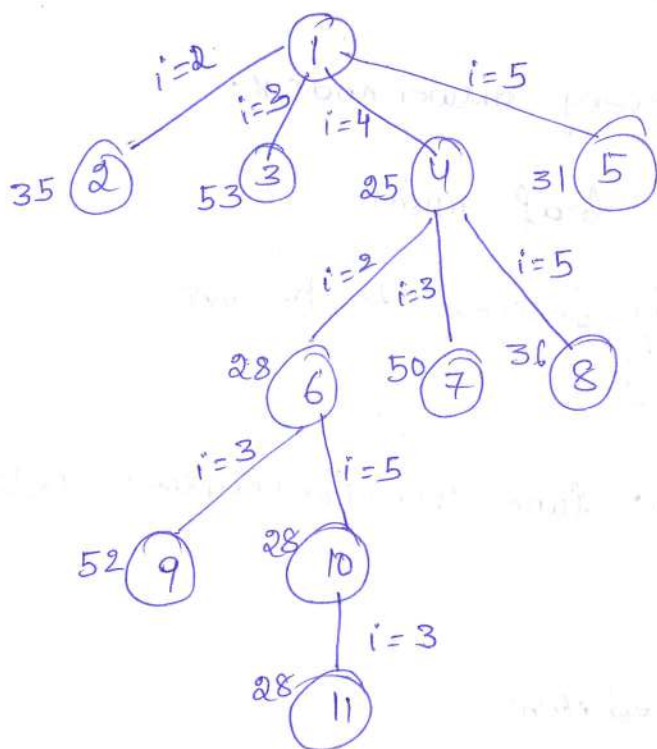
3rd col = ∞

A[3,1] = ∞

δ = 0

$$c^1(3) = c^1(5) + A(5,3) + \delta$$

$$= 28 + 0 + 0 = 28$$



The path is 1-4-2-5-3-1

$$\text{Min cost} = 10 + 6 + 2 + 7 + 3 = 28$$

0/1 Knapsack Problem:-

To solve 0/1 knapsack problem, through branch-and-bound technique, modify the statement

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

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

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

Assume fixed size tuple formulation for the soln space.

Every leaf node in the state space tree representing an assignment for which $\sum_{1 \leq i \leq n} w_i x_i \leq m$ is an answer or soln node.

All other nodes are infeasible.

Define $c(x) = - \sum_{1 \leq i \leq n} P_i x_i$ for every answer node x .

$c(x) = \infty$ for infeasible leaf nodes.

For nonleaf nodes, $c(x)$ is recursively defined to be min

$$\{c(\text{child}_1(x)), c(\text{child}_2(x))\}$$

Algorithm UBound $(C_p, cw, k, m) \rightarrow \text{func } u(\cdot)$ for knapsack problem
/* C_p is current profit total

cw is " weight "

k is index of the last removed item

m is knapsack size

$w[i] \leq P[i]$ are respectively the weight $\leq$ profit of the i th object.

$\{$ $b := C_p ; c := cw$

for $i := k+1$ to n do

$\{$ if $(c + w[i] \leq m)$ then

$\{$ $c := c + w[i] ; b = b - P[i] ;$

return $b ;$

$\}$

Let $\hat{c}(x)$ and $\hat{u}(x)$ are the two cost fncs such that

$$\hat{c}(x) \leq c(x) \leq \hat{u}(x) \text{ for every node } x.$$

Search begins at root node. Compute lower & upper bounds.

Considers first var x_1 . x_1 takes 0 or 1. compute lower & upper bounds for all vars of the same level.

Select the node whose cost is min

$$c(x) = \min \{ c(\text{left child}(x)), c(\text{right child}(x)) \}$$

The problem is solved in $x_1, x_2, \dots, x_n$ level wise.

→ Draw a portion of state space tree generated by LCBP by following knapsack problem $n=4$, $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$,
 $(W_1, W_2, W_3, W_4) = (2, 4, 6, 9)$, $m=15$.

knapsack is a maximization problem, but branch and bound technique is applied for minimization problem only.

$$\text{So, } (P_1, P_2, P_3, P_4) = (-10, -10, -12, -18)$$

place first item in bag $\Rightarrow$ remaining wt = $15 - 2 = 13$.

" second " " $\Rightarrow$ " " = $13 - 4 = 9$

third " " $\Rightarrow$ " " = $9 - 6 = 3$

4th item doesn't fit in bag. Fractions are not allowed in calculation of upper bound.

$$\therefore \text{Profit earned} = -10 - 10 - 12 = -32 = \text{Upper bound.}$$

→ To calculate lower bound, place 4th item in bag, since fractions

$$\text{are allowed} = -10 - 10 - 12 - \left(\frac{3}{9} \times 18\right) = -38.$$

$$\hat{u}(1) = -32, \hat{c}(1) = -38.$$

→ for node 2, $x_1 = 1$, place item in bag

$$\hat{c}(2) = -10 - 10 - 12 - \frac{3}{9} \times 18 = -38$$

$$\hat{u}(2) = -10 - 10 - 12 = -32.$$

For node 3, $x_1 = 0$, Do not place first item in bag

$$\hat{c}(3) = -10 - 12 - \frac{5}{9} \times 18 = -32$$

$$\hat{u}(3) = -10 - 12 = -22$$

Select the min of lower bounds i.e

$$\min \{ \hat{c}(2), \hat{c}(3) \} = \min \{ -38, -32 \} = -38 = \hat{c}(2)$$

choose node 2. $\Rightarrow \boxed{x_1 = 1}$

For node 4 ($x_2 = 1$)

$$\hat{c}(4) = -10 - 10 - 12 - \left(\frac{3}{9} \times 18 \right) = -38$$

$$\hat{u}(4) = -10 - 10 - 12 = -32$$

For node 5 ($x_2 = 0$)

$$\hat{c}(5) = -10 - 12 - \left(\frac{7}{9} \times 18 \right) = -36$$

$$\hat{u}(5) = -10 - 12 = -22$$

$$\therefore \min \{ \hat{c}(4), \hat{c}(5) \} = \min \{ -38, -36 \} = -38$$

$\therefore$ Node 4 is selected. $\Rightarrow \boxed{x_2 = 1}$

For Node 6 ($x_3 = 1$)

$$\hat{u}(6) = -10 - 10 - 12 = -32$$

$$\hat{c}(6) = -10 - 10 - 12 - \frac{3}{9} \times 18 = -38$$

For Node 7 ($x_3 = 0$)

$$\hat{u}(7) = -10 - 10 - 18 = -38$$

$$\hat{c}(7) = -10 - 10 - 18 = -38$$

Since lower bounds are same, select min of upper bound

$$\min \{ \hat{u}(6), \hat{u}(7) \} = \min \{ -32, -38 \} = -38$$

$\therefore$ node 7 is selected i.e $\boxed{x_3 = 0}$

$\rightarrow$ For Node 8 ($x_4 = 1$)

$$\hat{u}(8) = -10 - 10 - 18 = -38$$

$$\hat{c}(8) = -10 - 10 - 18 = -38$$

For node 9 ($x_4=0$)

$$\hat{u}(9) = -10 - 10 = -20$$

$$\hat{c}(9) = -10 - 10 = -20$$

$$\min \{ \hat{c}(8), \hat{c}(9) \} = \min \{ -38, -20 \} = -38$$

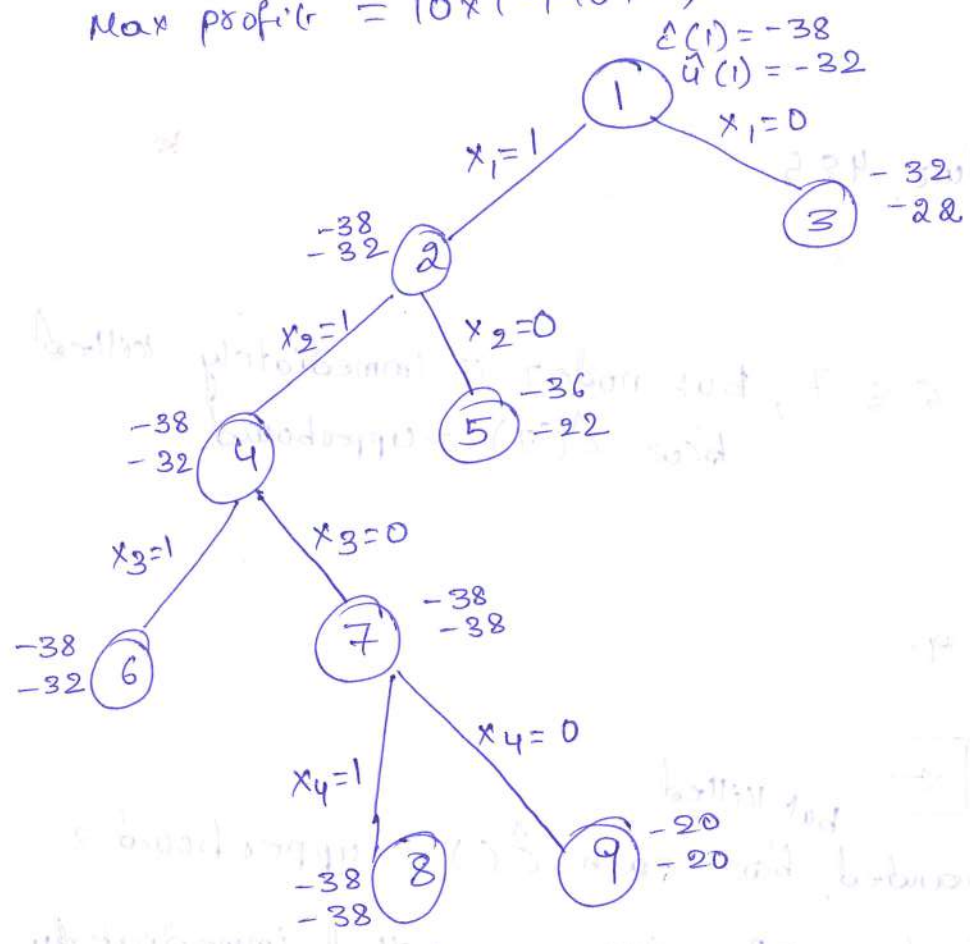
$\therefore$ Node 8 is selected.

$$\therefore \boxed{x_4 = 1}$$

The path is 1-2-4-7-8

$$\text{Soln is } (x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$$

$$\text{Max profit} = 10 \times 1 + 10 \times 1 + 12 \times 0 + 18 \times 1 = 38$$

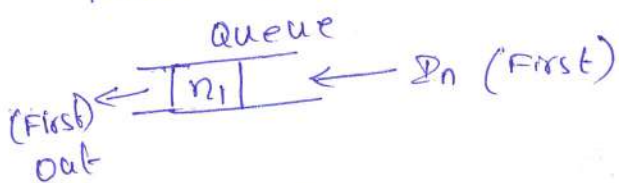


FIFO Branch-and-Bound solution:- for 0/1 knapsack.

$$n=4, (v_1, v_2, v_3, v_4) = (10, 10, 12, 18)$$

$$(w_1, w_2, w_3, w_4) = (2, 4, 6, 9) \text{ and } w=15.$$

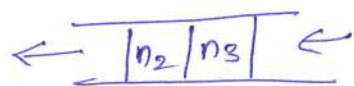
node 1 is root node $\&$ it is E-node.



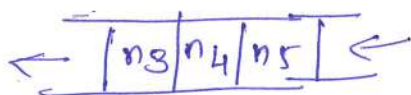
$$\hat{C}(s) > U(R) \text{ then kill node } s$$

$$u(1) = -32$$

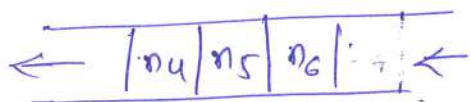
produce node 2 $\&$ nodes



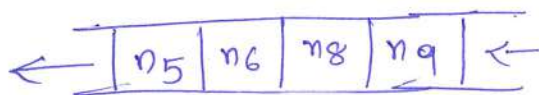
n_2 is E-node $\&$ produce 4 $\&$ 5



n_3 generates node 6 $\&$ 7, but node 7 is immediately killed b'coz $\hat{C}(7) > \text{upperbound}$.



n_4 generates 8 $\&$ 9.



$n_5 \& n_6$ are ^{not} expanded, but killed $\hat{C}() > \text{upper bound}$

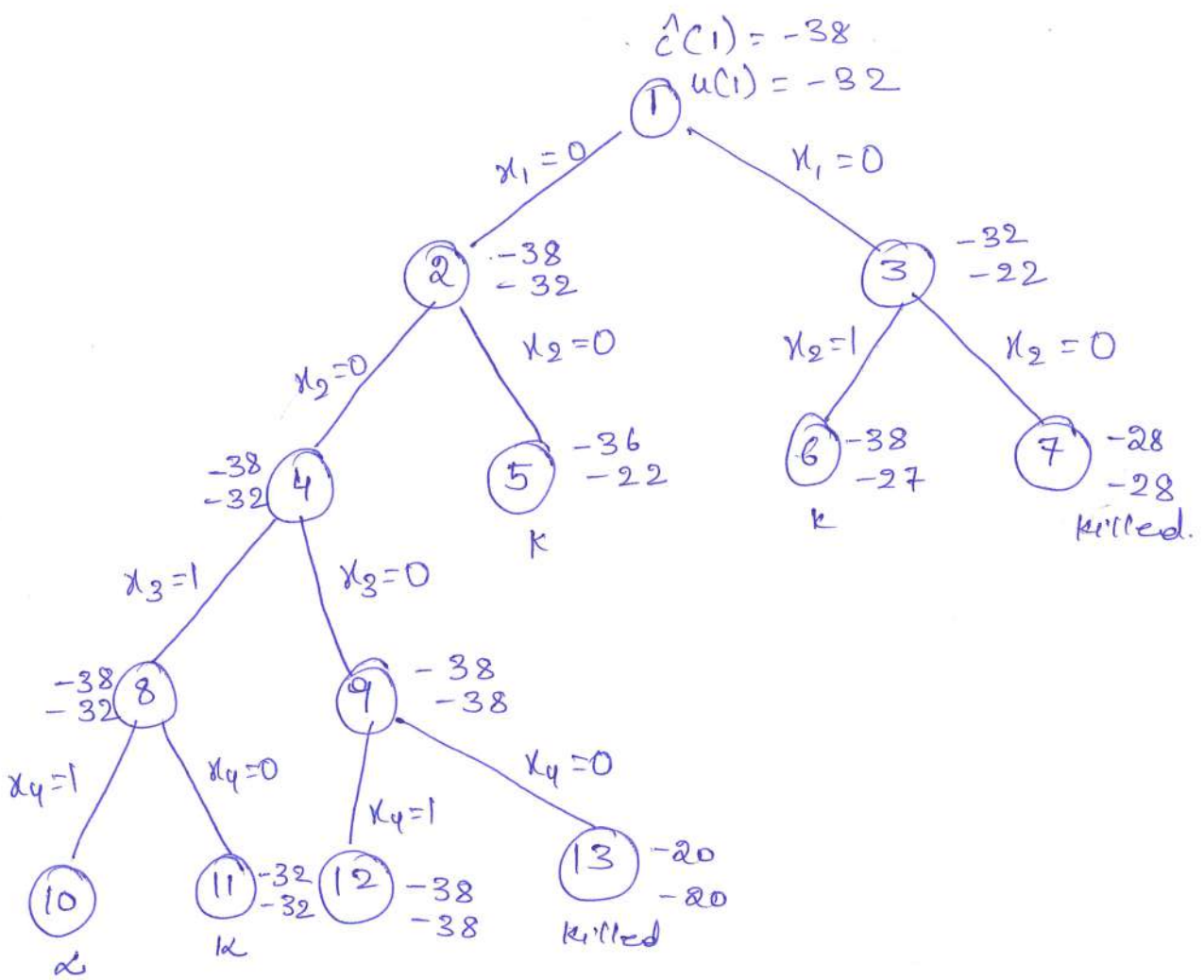
$\rightarrow$ hence n_8 is expanded, $\&$ $n_{10} \& n_{11}$ are killed immediately

node 10 is not feasible $\&$ so killed, node 11 - $\hat{C}(11) > \text{upper}$ so killed.

n_9 ^{has to} produce $n_{12} \& n_{13}$. n_{13} is killed as $\hat{C}(13) > \text{upper}$

so, only live node in queue is node 12 and search terminates. declaring node 12 as answer node.

$$\Rightarrow \text{Soln is } x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$$



NP-HARD $\subseteq$ NP-Complete Problems

15

Basic concepts:-

The best algorithms have computing times that cluster into two groups:

1. Problems whose soln times are bounded by polynomials of small degree.

Ex. Ordered search i.e. binary search $O(\log n)$

Polynomial evaluation $- O(n)$

Sorting - (Merge, quick) $- O(n \log n)$

String editing $- O(mn)$

2. Problems whose best-known algos are nonpolynomial.

Ex. TSP $- O(n^2 2^n)$

Knapsack $- O(2^{n/2})$

→ Algos of group 1 are more efficient, No one has been able to develop a polynomial time algo for any problem in second group

→ A problem that is NP-complete has the property that it can be solved in polynomial time iff all other NP-problems can be solved " " " " " "

→ A ~~problem~~

If an NP-hard problem can be solved in polynomial time, then an NP-complete problem can be solved " " " "

→ All NP-complete problems are NP-hard, but some NP-hard problems are not known to be NP-complete.

→ No NP-complete or NP-hard problem is polynomially solvable.

Non-deterministic Algorithms

If the ~~result~~ of every operation is uniquely defined, in the algo, those type of algos are called deterministic algos.

→ Nondeterminism allow algos with operations whose outcomes are not uniquely defined, but are limited to specified sets of possibilities. 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 non-det algos use 3 funcs, all with $O(1)$ computing time.

1. Choice(S) - arbitrarily choose one of the elems of set S .
2. Failure() - signals an unsuccessful completion.
3. Success() - " " successful " "

$x := \text{Choice}[1, n]$ → assigns any one of the integers $[1, n]$ to x .

Whenever there is a set of choices that leads to a successful completion, then one such set of choices is always made and the algo terminates successfully.

- A nondeterministic algo terminates unsuccessfully if & only if there exists no set of choices leading to a success signal.

→ A machine capable of executing a nondet algo in this way is called non-det machine.

- Non-det machines do not exist in practice.

Ex: 1 Non-det search.

- 1 $j := \text{Choice}(1, n);$
- 2 if $A[j] = x$ then { write (j); Success(); }
- 3 write (0); Failure();

Complexity $O(1)$, but deterministic search - $\Omega(n)$

Ex. 2. Nondet Sorting:-

Algo NSort(A, n)
 // Sort n +ve numbers

```

{
  for  $i := 1$  to  $n$  do  $B[i] := 0$  // Initialize
  for  $i := 1$  to  $n$  do
    {
       $j := \text{Choice}(1, n)$ ;
      if  $B[j] \neq 0$  then Failure();
       $B[j] := A[i]$ ;
    }
  for  $i := 1$  to  $n-1$  do // verify order
    if  $B[i] > B[i+1]$  then Failure();
  write( $B[1:n]$ );
  Success();
}

```

$\rightarrow O(n)$,

but det sorting algo takes $\Omega(n \log n)$
 $\rightarrow$ A deterministic interpretation of a nondet algo can be made by allowing unbounded parallelism in computation. Each time a choice is to be made, the algo makes several copies of itself. One copy is made for each of the possible choices. Thus, many copies are executing at same time. The first copy that succeeds, terminates all other computations. If the copy reaches a failure completion, then only that copy of the algo terminates.

$\rightarrow$ Any problem for which the answer is either zero or one is called a decision problem. Algorithm for decision problem is called decision algo.

→ Any problem that involves the identification of the optimal (max or min) value of a given cost func is known as an optimization problem.

An optimisation algo is used to solve an optimization problem.

Ex:-3 Max clique

A max complete subgraph of a graph $G = (V, E)$ is a clique. The size of the clique is the no. of vertices in it. The max clique problem is an optimization problem that has to determine the size of a largest clique in G .

Let $DClique(G, k)$ be a det decision algo for the clique decision problem; determines whether G has a clique of size atleast k .
Use $DClique$ for each ' k ', $k = n, n-1, n-2, \dots$, until the o/p from $DClique$ is 1.

If the time complexity of DClique is $f(n)$, then the size of a max clique can be found in time $\leq n \cdot f(n)$.

i.e. Max clique problem can be solved in poly time if & only if the clique problem

Ex:- 4 0/1 knapsack.

The knapsack decision problem is to determine whether there is 0/1 assignment of values to x_i , $1 \leq i \leq n$, such that $\sum p_i x_i \geq r$ and $\sum w_i x_i \leq m$. The r is the given number.

If the knapsack decision problem cannot be solved in deterministic polynomial time, then the optimization problem cannot either.

polynomial time, then the optimization problem cannot either.

Input size : — Input 10 is rep as bin '1010' i.e. with 4 bits. i.e. any number k 's rep by $\lceil \log_2 k \rceil + 1$ bits in binary. problem can be provided as

Input size :- Input to is $\log_2 k$ bits.

1. Maxclique :- IP of max clique decision problem can be provided as a seq. of edges $\in$ an integer k . Each edge in $E(n)$ is a pair of numbers (i, j) .

The i/p size of any instance is

(17)

$$n = \sum_{\substack{(i,j) \in E(G) \\ i < j}} (\lfloor \log_2 i \rfloor + \lfloor \log_2 j \rfloor + 2) + \lfloor \log_2 k \rfloor + 1$$

If G has only one connected component, then $n \geq |V|$.

If this decision problem cannot be solved by an algo of complexity $P(n)$ for some polynomial $P()$, then it cannot be solved by an algo of complexity $P(|V|)$.

2. 0/1 knapsack

$$q = \sum_{1 \leq i \leq n} (\lfloor \log_2 p_i \rfloor + \lfloor \log_2 w_i \rfloor) + 2n + \lfloor \log_2 m \rfloor + \lfloor \log_2 r \rfloor + 2$$

$q > n$

Unary - 1111 for 4 (dec)
- 11111 for 5 etc

0/1 - unary i/p size

$$S = \sum p_i + \sum w_i + m + r$$

knapsack decision and optimization problems can be solved in-time $p(s)$ for some polynomial $p()$

Defn:-
The time reqd by a nondet algo performing on any given i/p is the min no. of steps needed to reach a successful completion, $O(f(n))$

If no successful completion is possible, time is $O(1)$

Ex. p.Deterministic sum of subsets:

Algorithm Sum(A, n, m)

1 $s := 1$; // s is an (m+1)-bit word,

for $i := 1$ to n do

$s := s$ OR Shift($s, A[i]$);

if the m^{th} bit is s is 1 then

write ("A subset sums to m.");

else
write ("No subset sums to m");

}
→ It uses $(m+1)$ bit word 's'. The i th bit in 's' is zero if and only if no subset of the integers $A[j]$, $1 \leq j \leq n$, sums to i .
Bit '0' of 's' is always 1, and the bits are numbered $0, 1, 2, \dots, m$ right to left.

shift() - shifts the bits in 's' to the left by $A[i]$ bits.

→ Total no. of steps is $O(n)$

Each step moves $m+1$ bits of data i.e. $O(m)$

Assume one unit of time is needed for each basic operation for a fixed word size, $\Rightarrow$ time complexity $O(nm)$

Ex: - 2 :- knapsack

Non-deterministic knapsack algo

Algo DKP (P, w, n, m, r, x)

{
 $w := 0, P := 0;$

 for $i := 1$ to n do

 {
 $x[i] := \text{Choice}(0, 1);$

$w := w + x[i]; P := P + x[i] * P[i];$

 }

 if $((w > m) \text{ or } (P < r))$ then Failure(); //check if assignment is feasible?

 else Success();

 }

A successful termination is possible iff the answer to the decision problem is yes.

The time complexity is $O(n)$

If q is the i/p length using a bin rep, the time is $O(q)$.

Ex: 3 Nondeterministic Clique pseudocode,

(18)

Algo DCK (G, n, k)

$\{$ $S := \emptyset$ // S is an initially empty set.

for $i := 1$ to k do

$\{$ $t := \text{Choice}(1, n)$

if $t \in S$ then Failure();

$S := S \cup \{t\}$ // Add t to set S .

$\}$

// At this point S contains k distinct vertex indices.

for all pairs (i, j) such that $i \in S, j \in S$, and $i \neq j$ do

if (i, j) is not an edge of G then Failure();

Success();

$\}$

Algo begins by trying to form a set of k distinct vertices. Then it tests to see whether these vertices form a complete subgraph. If G is given by its adjacency matrix and $|V| = n$, the i/p length m

is $n^2 + \lfloor \log_2 k \rfloor + \lfloor \log_2 n \rfloor + 2$.

overall nondet-time = $O(n + k^2) = O(n^2) = O(m)$.

There is no known polynomial time det-algo for this problem.

Ex: 4 [Satisfiability]

Let $x_1, x_2, \dots$ denote bool vars, let $\bar{x}_i$ denote the negation of x_i .

A literal is either a var or its negation.

A formula in a propositional calculus is an expr that can be constructed using literals and the ops 'and' and 'or'.

Ex: $(x_1 \wedge x_2) \vee (x_3 \wedge \bar{x}_4)$

— A formula is in conjunctive normal form (CNF) if and only if it is represented as $\bigwedge_{i=1}^k C_i$, where the C_i are clauses each represented as

$\forall i: \bigvee l_{ij}$. The l_{ij} are literals.

— A formula is in disjunctive normal form (DNF) iff it is represented as $\bigvee_{i=1}^k C_i$ and each clause C_i is represented as $\bigwedge l_{ij}$.

Thus $(x_1 \wedge x_2) \vee (x_3 \wedge \bar{x}_4)$ is DNF

$(x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_2)$ is CNF.

— The satisfiability problem is to determine whether a formula is true for some assignment of truth values to the variables.

CNF-satisfiability is the satisfiability problem for CNF formula.
Nondet satisfiability algo:-

Algorithm Eval (E, n)

* Determine whether the propositional formula E is satisfiable.

The vars are $x_1, x_2, \dots, x_n$ */

{ for $i := 1$ to n do // choose a truth val assignment

$x_i := \text{Choose}(\text{false}, \text{true});$

if $E(x_1, \dots, x_n)$ then Success();

else Failure();

}

Nondet time reqd $\approx O(n)$ to choose value $(x_1, \dots, x_n)$ +
time needed to deterministically evaluate
 E for that assignment.

The classes NP-hard $\approx$ NP-complete:-

An algo A is of polynomial complexity if there exists a polynomial $P(n)$ such that the computing time of A is $O(P(n))$ for every ip of size n .

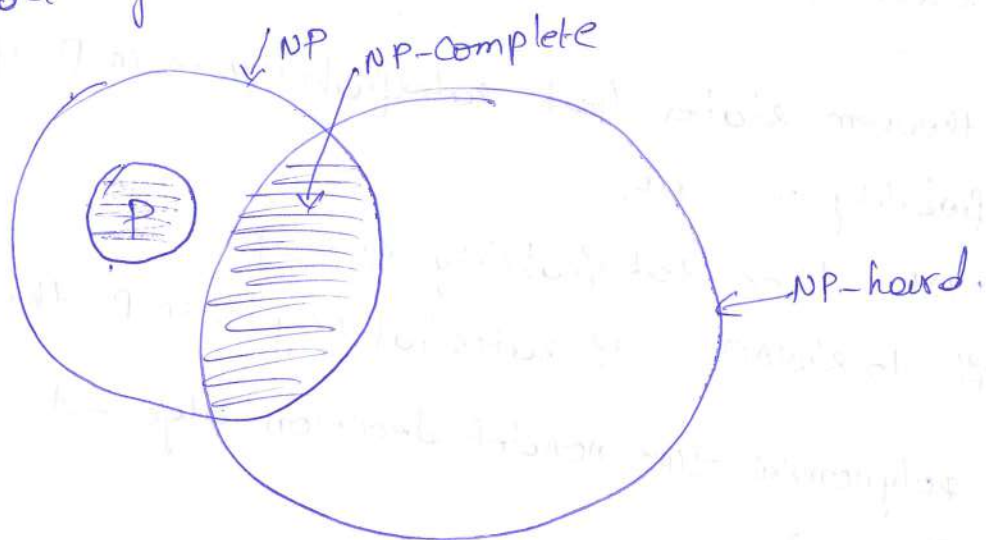
Defn:- P is a set of all decision problems solvable by deterministic algos in polynomial time. NP is set solvable by nondet algos in polynomial time. (19)

$$P \subseteq NP.$$



Theorem:- [Cook] Satisfiability is in P if and only if $P=NP$

Defn:- Let L_1 and L_2 be problems. Problem L_1 reduces to L_2 if and only if there is a way to solve L_1 by a det poly time algo using a det algo that solves L_2 in poly time.



Defn:- A problem L is NP-hard if and only if Satisfiability reduces to L . A problem L is NP-complete if and only if

$$L \text{ is NP-hard } \wedge L \in NP.$$

Ex:- 1 NP-hard decision problem that is not NP-complete.

The halting problem is to determine for an arbitrary det algo A and an i/p $\mathcal{I}$ whether algo A with i/p $\mathcal{I}$ ever terminates (or enters an infinite loop). This is undecidable. No algo exists to solve this prob so it is not NP.

To show satisfiability is halting problem, construct an algo A whose i/p is a propositional formula x . A halts on i/p x iff x is satisfiable. $\Rightarrow$ Halting problem is an NP-hard problem that is not in NP.

Defn:- Two problems L_1 and L_2 are said to be polynomially equivalent iff $L_1 \leq L_2$ and $L_2 \leq L_1$.

To show a prob L_2 is NP-hard, it is adequate to show $L_1 \leq L_2$ where L_1 is some problem already known to be NP-hard.

$\leq$ is transitive.

satisfiability $\leq L_1$ & $L_1 \leq L_2 \Rightarrow$ Satisfiability $\leq L_2$.

Cook's Theorem: (*)

Cook's theorem states that satisfiability is in P iff $P = NP$.

Proof - Satisfiability is in NP.

If $P = NP$, then satisfiability is in P.

only left to show is, if satisfiability is in P, then $P = NP$.

Let P be polynomial time nondet decision algo - A

Input $\mathcal{D}$

$Q(A, \mathcal{D})$ - Q is satisfiable iff A has a successful

termination with i/p $\mathcal{D}$.

If length of $\mathcal{D}$ is n , time complexity A is $p(n)$ for some

polynomial $p()$, then length of Q is $O(p^3(n) \log n) = O(p^4(n))$

Assumptions

1. A machine on which A is to be executed is word oriented. Each word is w bits long. Add, mul, sub ... b/w one word numbers take one unit of time.

2. A simple expression is an expression that contains at most one operation and all operands as simple vars (ie no array vars are used).
Some sample expressions - B , $B+C$, D or E , and F .

Assume assignments in A

a) $\langle \text{simple-var} \rangle := \langle \text{simple-expr} \rangle$

b) $\langle \text{array-var} \rangle := \langle \text{simple-expr} \rangle$

c) $\langle \text{simple-var} \rangle := \langle \text{arr-var} \rangle$

d) $\langle \text{simple-var} \rangle := \text{Choice}(S)$ where S is a finite set $\{s_1, s_2, \dots, s_n\}$ or l, u . In the later case func chooses an integer in the range $[l:u]$.

Indexing within an array is done using a simple integer var

- all index vals are positive.
- only 1D arrs are allowed.

3. All vars in A are of type integer or boolean

4. Algo A contains no read or write stmts. The only i/o to A is via its pars. At the time A is involved, all vars have value zero

5. Algo A contains no constants. All constants in any algo can be replaced by new vars.

6. A is allowed to contain only the following types of stmts.

a) The statement goto k , where ' k ' is an instruction number.

b) The stmt if c then goto a , ^{where} c is a bool var.
 a - ins number.

c) Success(), Failure()

d) Algo A may contain type declaration and dimension stmts.

These are not used during exec of A and so need not be used during execution of A and so need not be translated into Q . The dimension info is used to allocate array space.

- Assume that ins in A are numbered sequentially from 1 to l . Every stmt in A has a number. The goto ins in (a) & (b) use this numbering scheme to effect a branch.

goto k can be replaced by stmt 'if true then goto k '.

7. Let $p(n)$ be a polynomial such that A takes no more than $P(n)$ time units on any i/p of length n .

→ Define Bounding function.

(21)

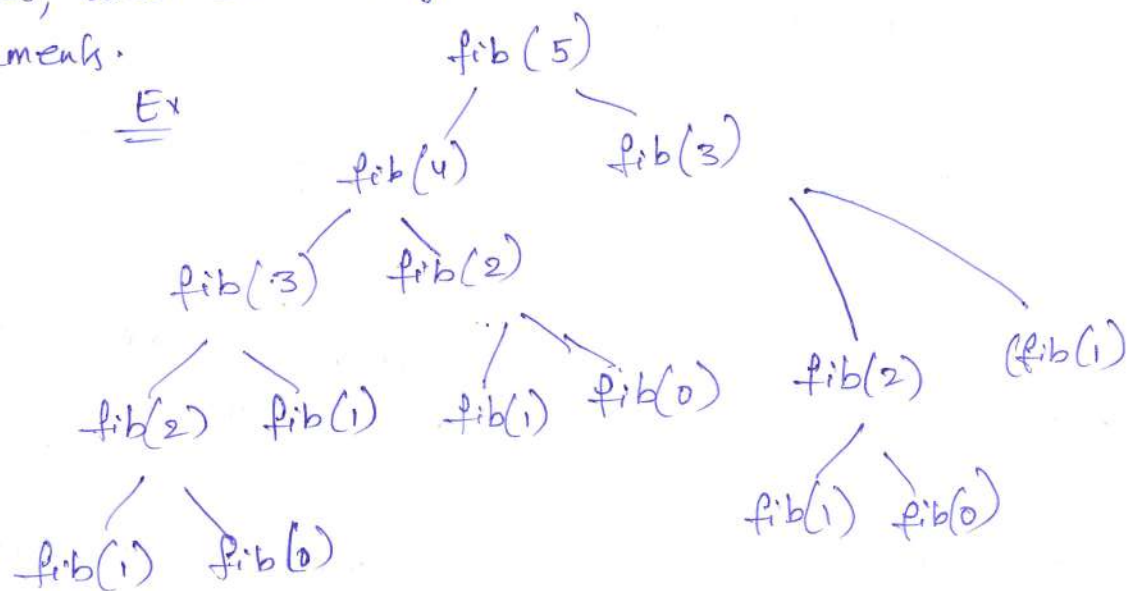
A branch and bound method searches a state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E-node.

Bounding function helps to avoid generating subtrees that do not contain an answer node.

A good bounding helps to prune the tree efficiently, leading to a faster exploration of solution space.

→ Define overlapping subproblems.

Overlapping subproblems mean the recursive function will be called multiple times with identical arguments. Draw breadth-first tree of the recursive calls your function makes, and see if you notice multiple calls with the same arguments.



UNIT-5

→ Compare & contrast deterministic & non-deterministic algorithms

Deterministic algo

1. A deterministic algo, is one where, given a particular input, the algo will always produce the same o/p and follow the same sequence of states.
2. Execution path:-
Follow a single, clear path through execution
3. Randomness:-
No randomness is possible
4. Testing & Verification:-
Easier to test & verify due to predictability.
5. Complexity.
Generally simpler and more straightforward
6. Optimization:-
Easier to optimize as the execution path is predictable.

Non-deterministic algo

1. A non-deterministic algo is one where the same i/p can lead to multiple possible outcomes.
2. Explore multiple paths
3. Often involves randomness or probability.
4. Testing can be more challenging due to variability in outcomes.
5. Tend to be more complex due to multiple possible paths
6. Optimization is more challenging but offers exploration of multiple solutions.

7. Applications: -

:-

Sorting, searching,
mathematical calculations,
data compression,
cryptography

Advantages:

8. Consistency, ease of
straightforward
optimization, simplicity
in implementation.

Machine learning, heuristic
search, randomized
algorithms, certain
cryptographic protocols

8. Handling ambiguity,
exposing multiple solutions
potentially faster in
certain tasks, adaptability,
creativity.

