

Disjoint set operationsIntroduction:-

Forests are used to represent sets.

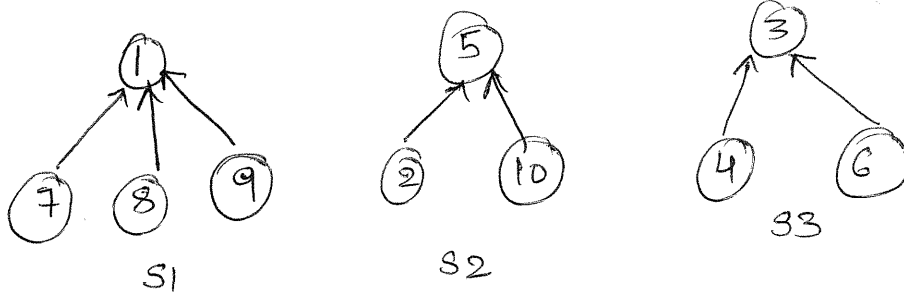
Assume the elements of the sets are the numbers $1, 2, 3, \dots, n$

Sets are pairwise disjoint. i.e. if S_i and S_j , $i \neq j$, are two sets, then there is no common element in both S_i and S_j .

Ex When $n=10$, the elements can be partitioned into 3 disjoint sets.

$$S_1 = \{1, 7, 8, 9\}, S_2 = \{5, 2, 10\}, \text{ \& } S_3 = \{3, 4, 6\}$$

Each set is represented by each tree.



links go from children to parent

The operation we perform on sets are.

1. Disjoint set union: - If S_i and S_j are two disjoint sets

then $S_i \cup S_j =$ all elts x such that x is in S_i or S_j .

$$S_1 \cup S_2 = \{1, 7, 8, 9, 2, 5, 10\}$$

2. Find(i) - Given the elt ' i ', find the set containing i .

$$\text{Find}(4) = S_3$$

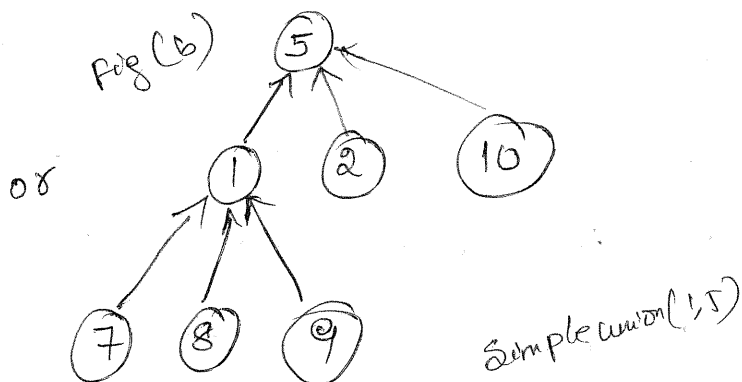
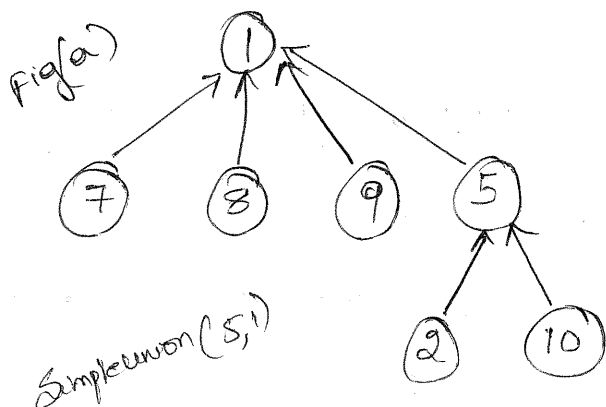
Union and Find Operations:-

Union:-

$$S_1 \cup S_2 = \{1, 7, 8, 9, 2, 5, 10\}$$

(4)

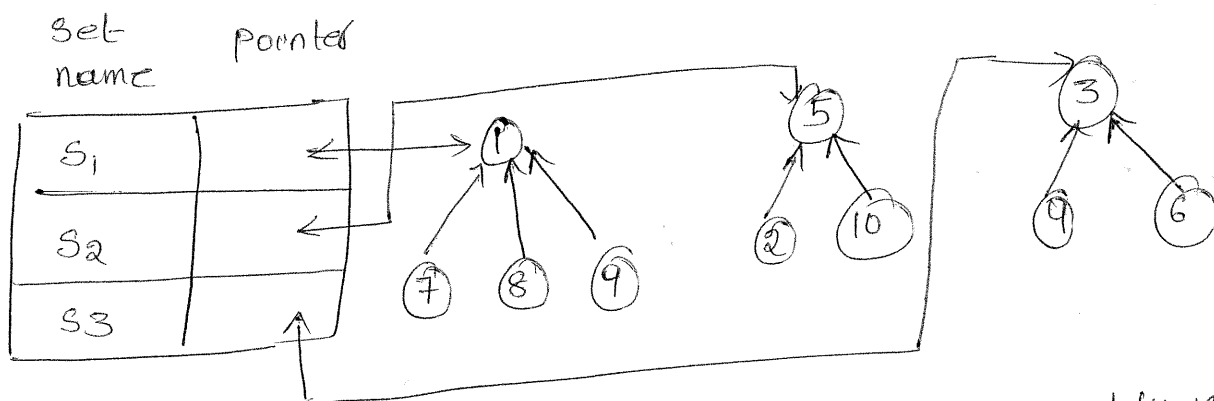
Make one of the tree as subtree of another



→ To obtain the union of two sets, set the parent field of one of the roots, to the other root.

Data representation for S_1 , S_2 and S_3

(2)



This determines which set an element is currently in, Follow parent links to the root of its tree and use the pointer to the set name.

Represent set elts 1 to n, by using an array $p[1:n]$
 ith element of this array represent the tree node that contains elt

i [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

p -1 5 -1 3 -1 3 1 1 1 5

Array representation for S_1, S_2, S_3 .

Find(i) is implemented, by following the indices, starting at 'i' until we reach a node with parent value -1. (2)

Find(6) - $P[6] = 3$
 $P[3] = -1$

→ once -1 is reached, stop & declare
 6's ^{tree's root} parent is 3.

Algorithm SimpleUnion(i, j)

{ $P[i] := j;$

}
 Algorithm Simple-Find(i)

{ while ($P[i] \neq 0$) do $i := P[i];$
 return i;

→ Draw DSG before union & after union

In Fig(b) after

SimpleUnion(1, 5)

Find(9)

i	$P[i]$	$P[i] \neq 0$	$P[i]$
1	5	1	5
2	0	0	0
3	0	0	0
4	0	0	0
5	-1	0	-1
6	3	1	3
7	0	0	0
8	0	0	0
9	1	1	1

$P[9] = 1$

$P[1] = 5$

~~$P[5] = -1$~~

return 5

Performance of simple_union & simple-find is not very good

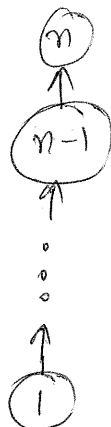
Ex:- If 'q' elts^{are} in a set $S_i = \{i\}$, ($1 \leq i \leq q$)

then initial configuration consist of a forest with 'q' nodes, &

$P[i] = 0, 0 \leq i \leq q$.

Let the sequence of operations performed on set are
 Union(1, 2), Union(2, 3), Union(3, 4), Union(4, 5), ..., Union(n-1, n)
 Find(1), Find(2), ..., Find(n), This seq results in degenerate

tree



- Time taken by Union is constant, then $n-1$ unions takes $O(n)$

- find requires following a seq of parent pointers from the elt to be found to the root.

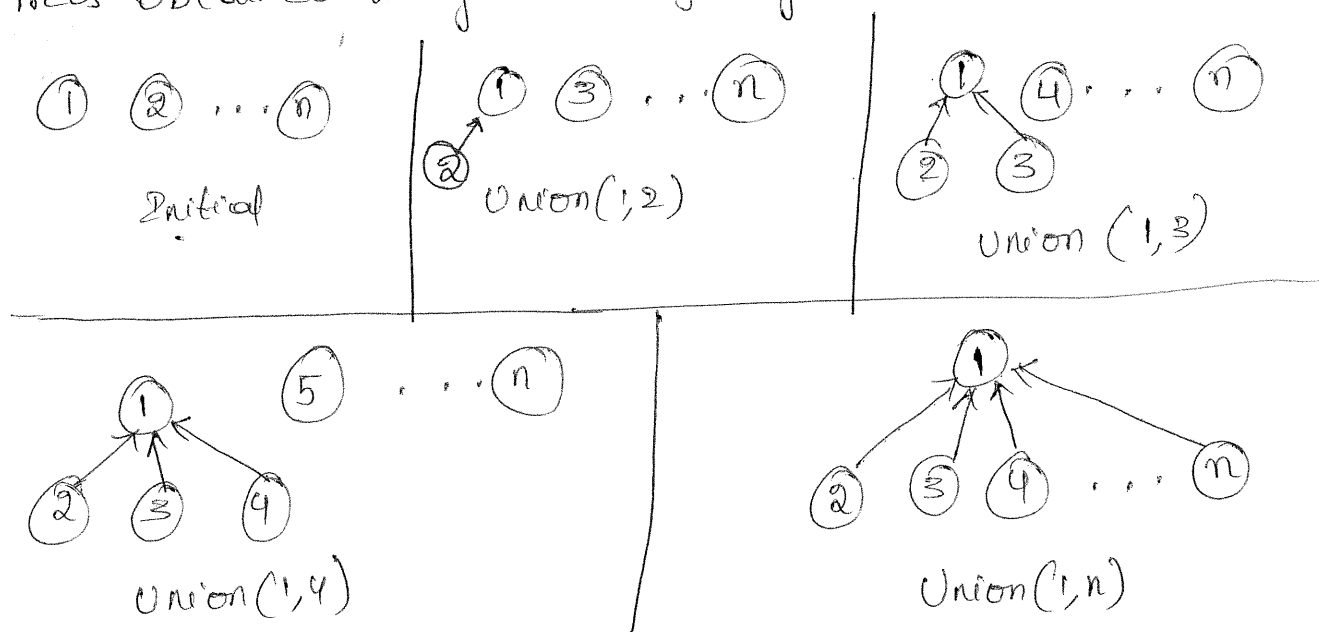
Since, the time reqd to process a find for an elt at level i of a tree is $O(i)$.

Total time needed to process the n finds is $O\left(\sum_{i=1}^n i\right) = O(n^2)$

- This time complexities can be improved by ~~creating~~ avoiding the creation of degenerate trees. To accomplish this, use weighting rule for $\text{Union}(i, j)$

Weighting rule for $\text{Union}(i, j)$ - If the no. of nodes in tree with root i is less than the number in the tree with root j , then make j the parent of i ; otherwise make i the parent of j .

Trees obtained using the weighting rule



→ To implement weighting rule, we maintain a count field in the root of every tree. If i is a root node, then $\text{count}[i]$ is no. of nodes in the tree.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	-1	-1	-1	-1	-1	-1	-1	-1
$\text{Union}(1, 2)$	-2	1						
$\text{Union}(1, 3)$	-3		1					
$\text{Union}(1, 4)$	-4			1				

Algorithm WeightedUnion (i, j)

(3)

/* Union sets with roots i and j, $i \neq j$, using the
weighing rule. $p[i] = -\text{count}[i]$ and $p[j] = -\text{count}[j]$ */

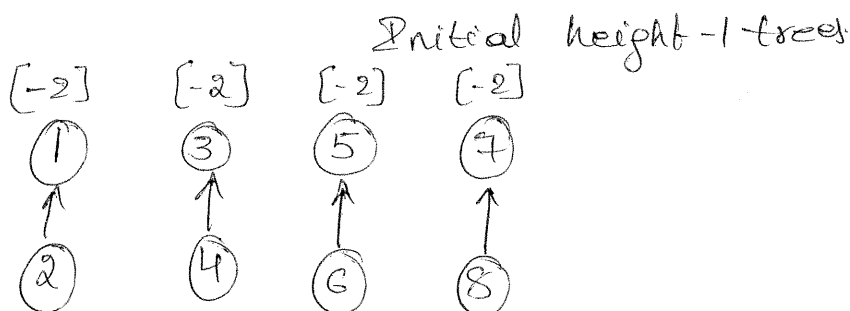
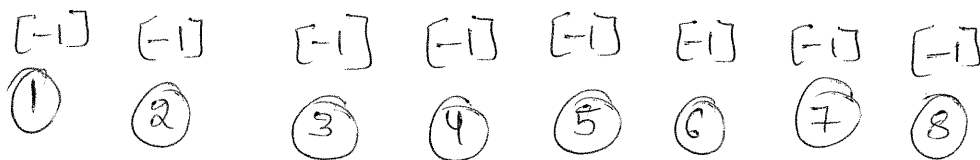
```
{
    temp := p[i] + p[j];
    if (p[i] > p[j]) then
        {
            // i has fewer nodes
            p[i] := j ; p[j] = temp;
        }
    else
        {
            j has fewer or equal nodes
            p[j] := i ; p[i] := temp;
        }
}
```

→ Trace of algo
is in page (4)

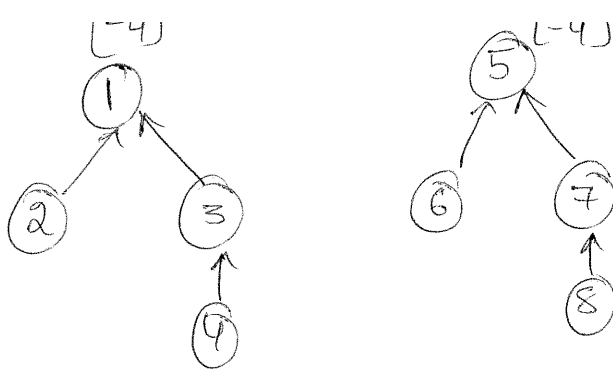
Time complexity of weightedUnion - $O(1)$

→ Height of each tree with m nodes is $\lfloor \log_2 m \rfloor + 1$

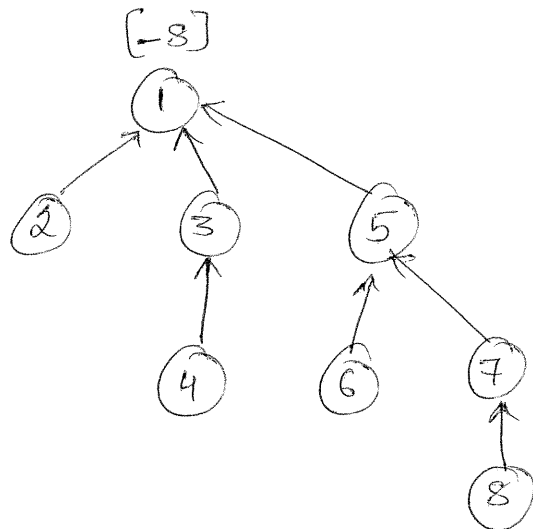
Ex. $n=8$ $\&$ $p[i] = -\text{count}[i] = -1, 1 \leq i \leq 8$. → This example is
Sequence to be taught along
with collapsing rule
Union(1,2) Union(3,4) Union(5,6), Union(7,8), Union(1,3) Union(5,7)
Union(1,5)



Height-2 trees following Union(1,2) (3,4), (5,6), (7,8)



Height-3 trees following Union (1,3), (5,7)



Height-4 tree following Union(1,5)

→ Now time to process a find is $O(\log m)$ if there are m elts in a tree.

If an intermixed sequence of 'u' union and 'f' find ops is to be processed, it takes $O(u + f \log u)$

Collapsing rule :- If 'j' is a node on the path from 'i' to its root and $p[i] \neq \text{root}[i]$, then set $p[i]$ to $\text{root}[i]$.

Algorithm CollapsingFind(i)

/* Find the root of the tree containing elt 'i'.

Use collapsing rule to collapse all nodes from i to the root.

```

{
  x := i;
  while (p[x] > 0) do x := p[x]; // find the root.
  while (i ≠ x) do
    { s := p[i]; p[i] := x; i := s;
  }
  return x;
}

```

→ Using disjoint-sets find the connected components in the undirected graph $G = (V, E)$, where vertices $V = \{a, b, c, d, e, f, g, h, i, j\}$ and edges $E = \{(a, c), (a, b), (e, f), (h, i), (e, g), (a, d), (e, d), (b, d), (e, a), (h, j)\}$. The edges are processed in the order given. List the vertices in each connected components after each step.

Ans:- To use union-find to find the connected component of a graph, we can perform the following steps:-

1. Initialize each vertex as a separate component.
2. For each edge in a graph, perform a union operation on the two components that edge connects.
3. After all edges have been processed, the connected components of the graph will be the disjoint sets in the union-find data structure.

- Assume graph is undirected always, if not specified as directed

→ Implementation $S = \{S_1, S_2, \dots, S_n\}$
 $S_1 = \{a, b, c\}, S_2 = \{d, e, f\}$

MAKE-SET(x) :- $S_i = \{x\}$, add S_i to S .

UNION(x, y) :- $x \in S_x, y \in S_y$, then $S = S - S_x - S_y \cup \{S_x \cup S_y\}$

FIND-SET(x) :- returns the representative of the set that contains x .

Algorithm

CONNECTED-COMPONENTS (G)

for each vertex $v \in G.V$

Make-SET (v)

for each $(u, v) \in G.E$

if FIND-SET(u) $\neq$ FIND-SET(v)

UNION (u, v).

Given
 $\rightarrow E = \{(a, c), (a, b), (e, f), (h, i), (e, g), (a, d), (e, d), (b, d), (e, a), (h, j)\}$

Edges	<u>connected components</u>									
—	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(a, c)	{a, c}	{b}		{d}	{e}	{f}	{g}	{h}	{i}	{j}
(a, b)	{a, b, c}			{d}	{e}	{f}	{g}	{h}	{i}	{j}
(e, f)	{a, b, c}			{d}	{e, f}		{g}	{h}	{i}	{j}
(h, i)	{a, b, c}			{d}	{e, f}	{g}		{h, i}		{j}
(e, g)	{a, b, c}			{d}	{e, f, g}			{h, i}		{j}
(a, d)	{a, b, c, d}				{e, f, g}			{h, i}		{j}
(e, d)	{a, b, c, d, e, f, g}							{h, i}		{j}
(b, d)	"							"		
(e, a)	"							"		
(h, j)	{a, b, c, d, e, f, g}							{h, i, j}		

$\Rightarrow$ Two connected components exist in the graph given
1) {a, b, c, d, e, f, g} 2) {h, i, j}

Follow the example in previous page to process

$\text{Find}(8), \text{Find}(8), \dots, \text{Find}(8)$ (8 find operations on element 8)

In SimpleFind , $\text{Find}(8)$ require going up ^{three} parent link fields for a total of 24 moves to process all 8 finds.

In collapsingFind , each of the remaining seven finds req going up only one link field. ; total cost is now only 13 moves.

→ weighted union(i,j) Trace.

i	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
P(i)	-1	-1	-1	-1	-1	-1	-1	-1
	-2	-1	-2	3	-2	5	-2	7
	-4		1		-4		5	
	-8				1			

→ After $\text{union}(1,2), \text{union}(3,4)$
" $(5,6), \text{union}(7,8)$

→ After $\text{union}(1,3)$
" $(5,7)$

→ After $\text{union}(1,5)$

⇒ finally

-8	1	1	3	1	5	5	7
----	---	---	---	---	---	---	---

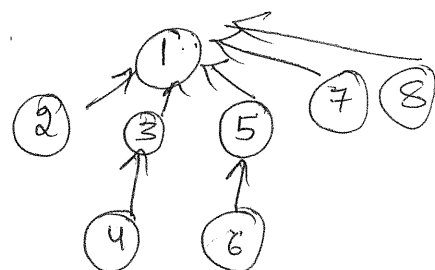
→ collapsing find(i) Trace.

First while loop		Second while loop			
i=8	x	P[x] > 0			
8		7 > 0	i ≠ x	s = P(i)	P[i] = 8 x
7		5 > 0	8 ≠ 7	s = 7	P[8] = 1
5		1 > 0	7 ≠ 1	s = 5	P[7] = 1
1		-8 ≠ 0	5 ≠ 1	s = 1	P[5] = 1
⇒ x = 1					

$1 \neq 1$ false
return 1

⇒ P[i] =

-8	1	1	3	1	5	1	1
----	---	---	---	---	---	---	---



After collapse find

Backtracking

In greedy & dyn programming techniques, we use brute force approach. We evaluate all possible solns, among which, we select one soln as optimal soln.

In backtracking, we get same optimal soln with a less no. of steps.

→ Backtracking method's soln is in the form n -tuple $(x_1, \dots, x_n)$ where x_i is chosen from finite set S_i . The algo calls for finding the vector that maximizes (or minimizes or satisfies) a criterion function $P(x_1, \dots, x_n)$.

Every criterion function is made up of implicit & explicit constraints.

Explicit constraints:- Rules which restrict each x_i to take on values only from given set.

Ex 1. In knapsack problem, explicit constraints are

i) $x_i = 0$ or 1

ii) $0 \leq x_i < 1$

2. In n -queens problem, n -queens ^{can be} placed in $n \times n$ chessboard in $n!$ ways.

Implicit constraints:- Rules that determine which tuples in soln space satisfy the criterion func.

Ex:- In n -queens, the implicit constraints are no two queens can be on the same ~~column~~ col, row & same diagonal.

Soln space:- All tuples that satisfy the explicit constraints define a possible soln space for a particular instance 'I' of the problem.

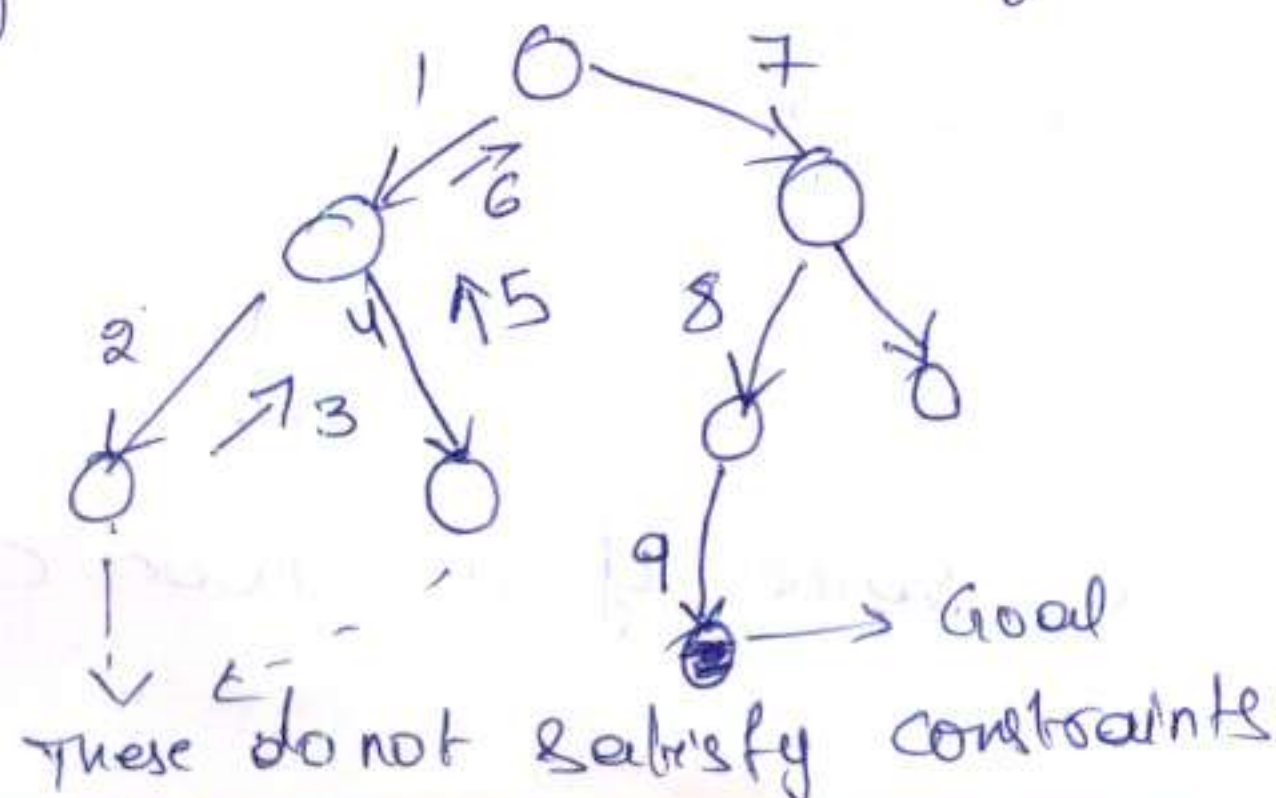
Backtracking

Backtracking is an intelligent way of gradually building the soln. It can be applied to constraint satisfaction problems like. Sudoku, Crossword, 8-queens puzzles, Chess etc.

Backtracking builds the soln incrementally, partial solns that do not satisfy the constraints are abandoned.

- Backtracking is a recursive approach.
- Soln in backtracking is expressed as a n -tuple $(x_1, x_2, \dots, x_n)$ where x_i is chosen from the finite set of choices, S_i . Elements in soln tuple are chosen such that they maximize or minimize the given criterion func $C(x_1, x_2, \dots, x_n)$, by selecting or not selecting item x_i .
- Backtracking systematically searches the set of all feasible solns, called solution space, to solve the given problem.
- Each choice leads to a new set of partial solns. Partial solns are explored in DFS (Depth first search) order, if the partial solns satisfy the bounding func.
- If the partial soln does not satisfy the constraint, it will not be explored further. The algo backtracks from that point & explores the next possible candidate.
- Processing is represented using state space tree.

E-x



→ While solving the problem, the backtracking method imposes two constraints.

- 1) Explicit constraints: - It restricts the selection of x_i from S_i .
These are problem dependent. i.e. x_i values should be taken only from given set.
- 2) Implicit constraints: - Determine which tuple in solution space satisfies the criterion function.

Terminology:-

1. Soln space: - The collection of all feasible solns is called soln space.
2. State space: - All the paths in the tree from root to other nodes ~~from~~ ^{form} the state space of given problem.
3. Problem state: - Each node in the state space tree, rep the
4. Answer state: - is the leaf in the state space tree which satisfy the implicit constraints.
5. Soln state: - State S for which the path from the root to S represents a tuple in the soln space.
6. Live node: - A node in the state space tree, that is generated but its children are yet to be generated.
7. E-node: - live node whose children are currently being explored. E - Expansion.
8. Dead nodes: - A node that is generated and either its all children are generated or node will not be expanded further due to a violation of criterion func.
9. Bounding func: - This kills the live node without exploring its children if the bound value of live node crosses the bound limits.

10. static tree:- If a tree is independent of an instance of the problem being solved, it is called static tree. (4b)

11. Dynamic tree:- If the tree depends on an instance of the problem being solved, it is called dynamic tree.

N-Queens Problem

Given $N \times N$ chess board, arrange N queens in such a way that no two queens attack each other by being in same row, column or diagonal.

$N=1 \rightarrow$ trivial case

$N=2$ and $N=3$, soln is not possible.

4-queen problem:-

Explicit constraint:- only 4 rows & cols are possible.
Hence solution space must be $S_i = \{1, 2, 3, 4\}$.

Implicit constraint

- No two queens can be in same row
- " " " col
- " " " diagonal

Problem: 1 2 3 4. No two x_i can be same.

1				
2				
3				
4				

Put Q_i in i th row.

1. Q_1 in $(1,1) \Rightarrow$ Partial solution = $\langle 1 \rangle$

2. Q_2 cannot be placed in $(2,1)$ or $(2,2)$

Q_2 in $(2,3) \Rightarrow P.S = \langle 1, 3 \rangle$

3. Q_3 cannot be placed in $(3,1), (3,2), (3,3), (3,4)$

Go back to previous, to readjust Q_2 from $(2,3)$

to $(2,4) \Rightarrow P.S = \langle 1, 4 \rangle$

4. Q_3 can be placed at $(3,2) \Rightarrow P.S = \langle 1, 4, 3 \rangle$

5. Q_4 does not fit in row 4. Backtrack to readjust Q_3 .

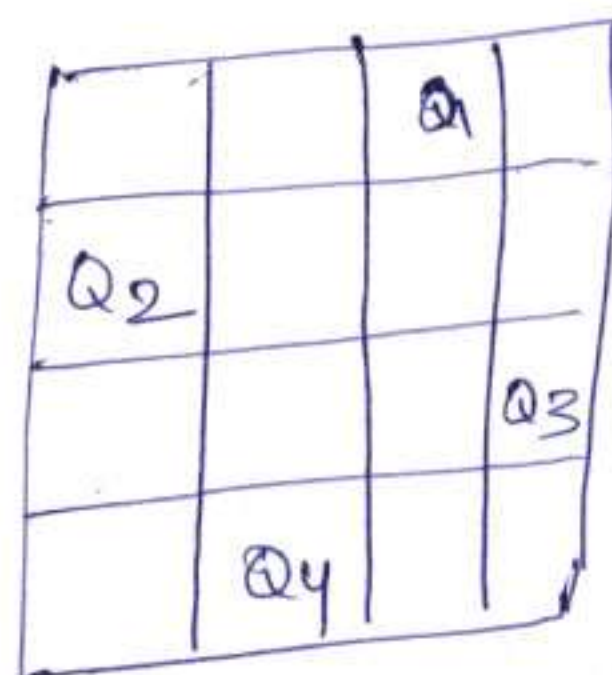
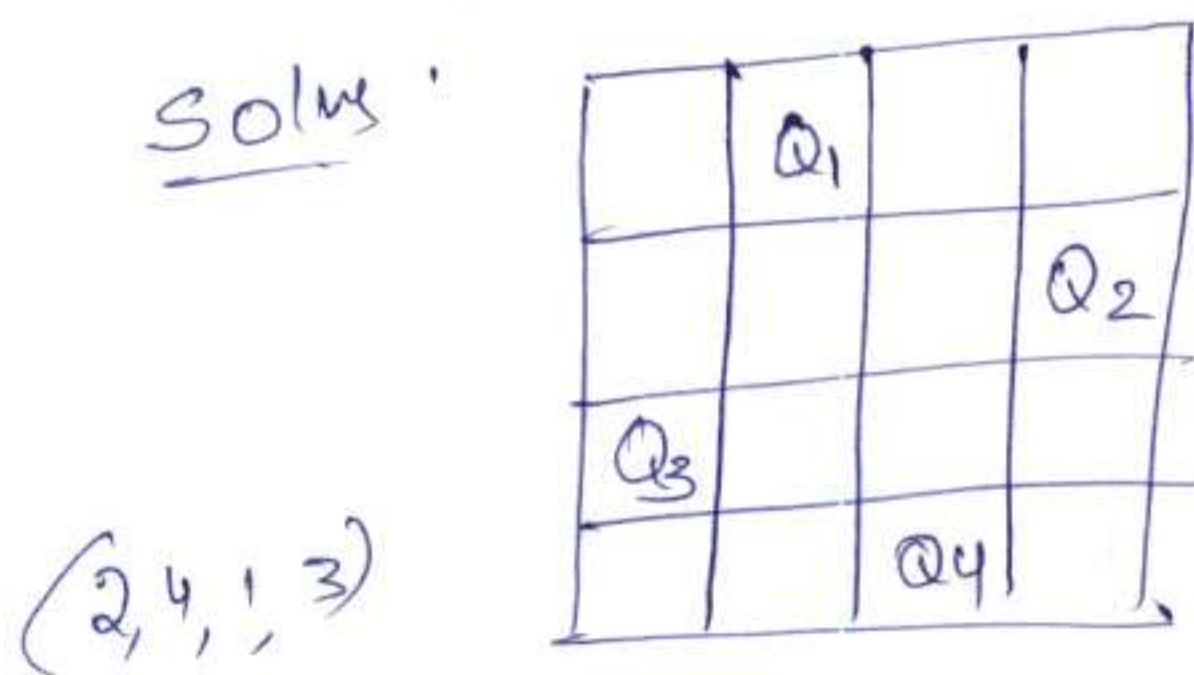
But Q_3 doesn't fit in $(3,3)$ or $(3,4)$.

So, backtrack even further.

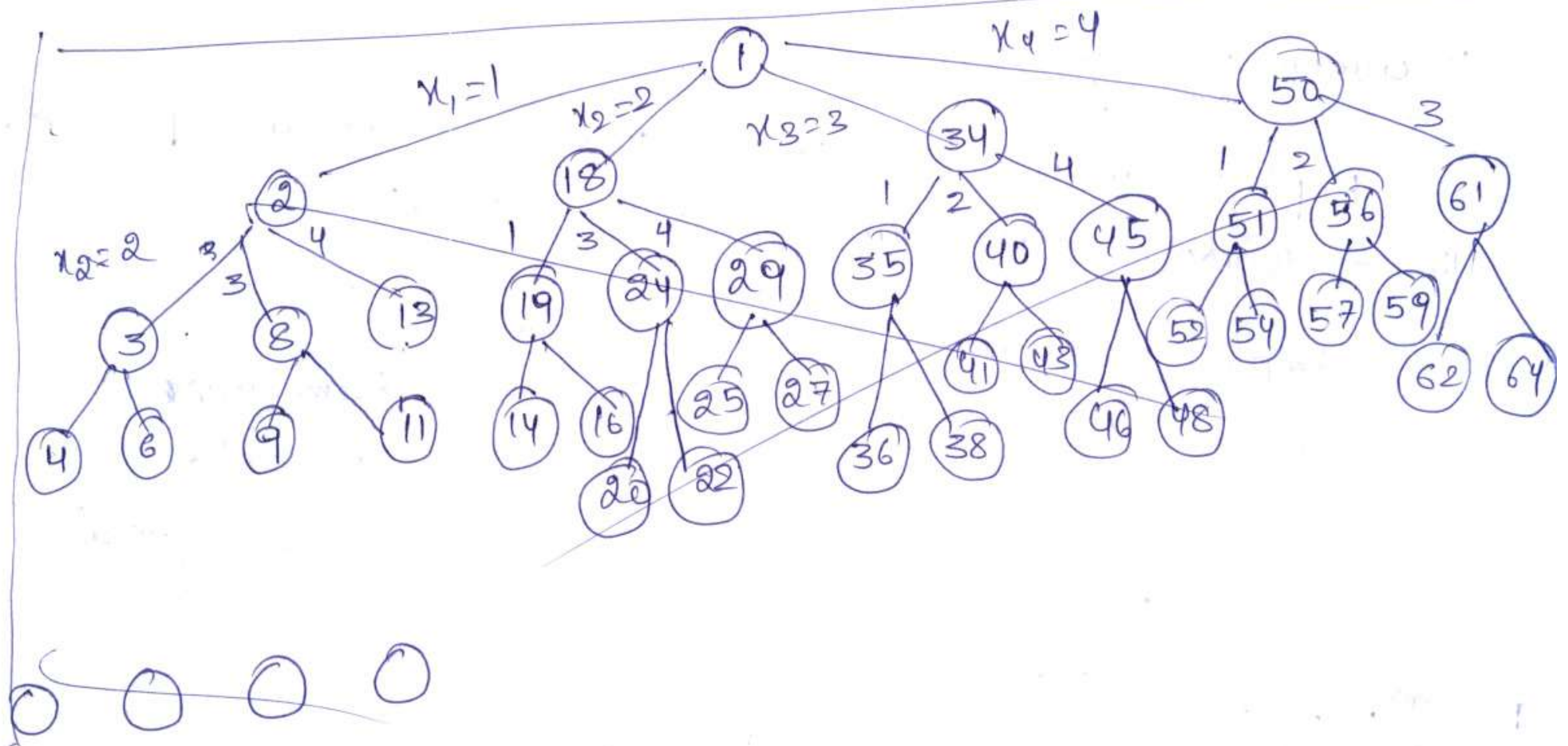
Q_2 all choices explored. Go back to PS $\langle 1 \rangle$

Move Q_1 from $(1,1)$ to $(1,2)$. Continue the process.

Soln:



$(3, 1, 4, 2)$



Applications: -

5

N-Queens problem

Consider an $n \times n$ chessboard. Let there are 'n' queens. These n queens are to be placed on the board so that no two queens are on same column, same row or same diagonal, i.e. they do not ~~collide~~ ^{attack} each other.

4-Queens Problem:-

Explicit constraints:- 4 queens are to be placed on 4×4 chessboard in 4^4 ways.

Implicit " : - No two queens are in the same row or col or diagonal.

Apply backtracking

Begin with root node and generate other nodes.

- A node which has been generated and all of whose children have not yet been generated is called a live node.
- The live node whose children are currently being generated is called the E-node (node being expanded)
- Dead node - is a generated node which is not to be expanded further or all of whose children have been generated.

In both methods, of generating, we have a list of live nodes.

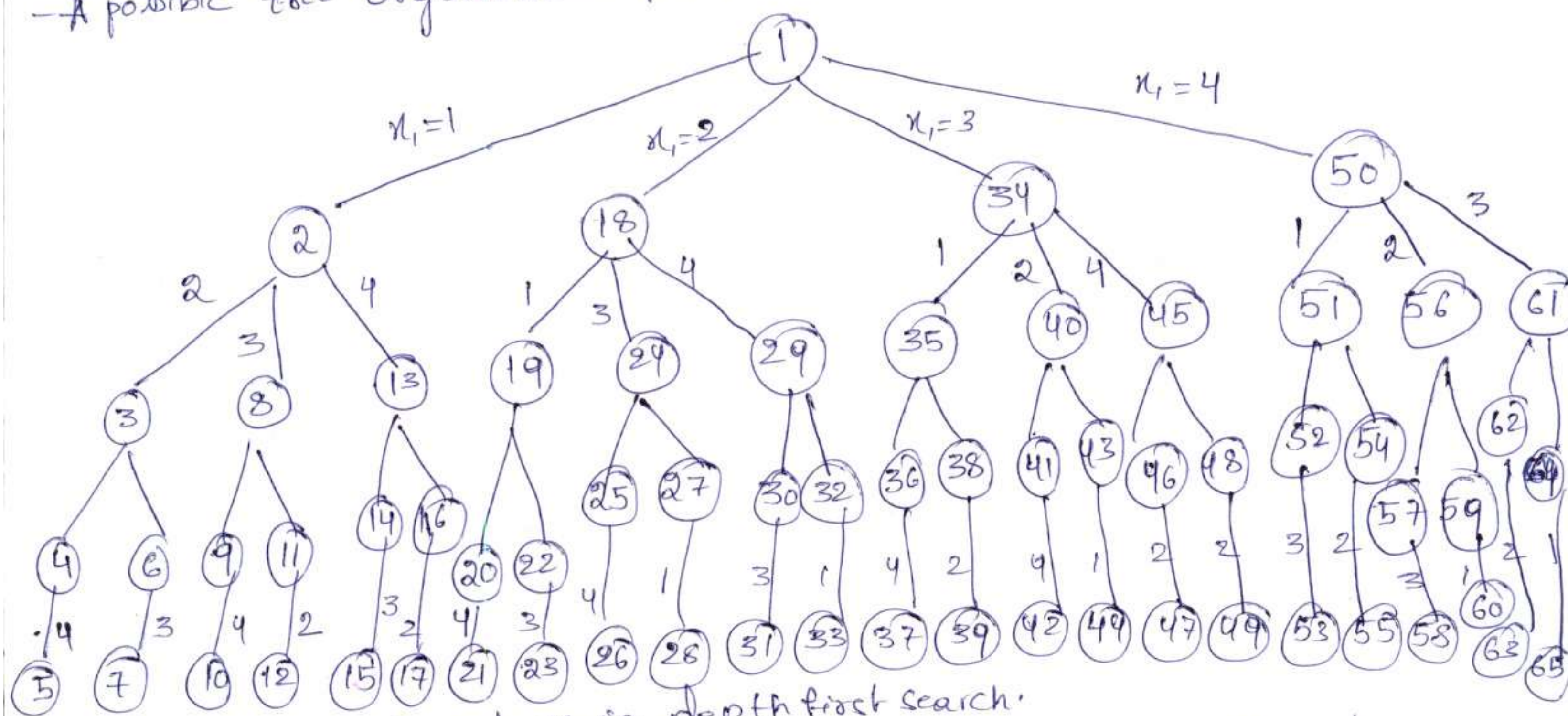
Method 1:- As soon as a new node's child C of the current E-node R is generated, this child will become a new E-node. Then R will become the E-node again when the subtree C has been fully explored. This corresponds to a depth first generation of the problem states.

Method 2:- E-node remains the E-node until it is dead.

- In both methods, bounding functions are used to kill live nodes without generating all their children. At the end of the process one answer node is always generated or all answer nodes are generated if problem requires us to find all solutions.
- Depth first node generation with bounding functions is called backtracking.
- state generation methods in which the E-node remains the E-node until it is dead lead to branch-and-bound ~~function~~ methods.

Terminology:-

1. Each node of the tree defines a problem state.
 2. All paths from the root to other nodes define the state space of the problem.
 3. Soln states are those problem states 's' for which the path from root to 's' defines a tuple in the soln space.
- A possible tree organization for the case $n=4$. i.e. Permutation tree

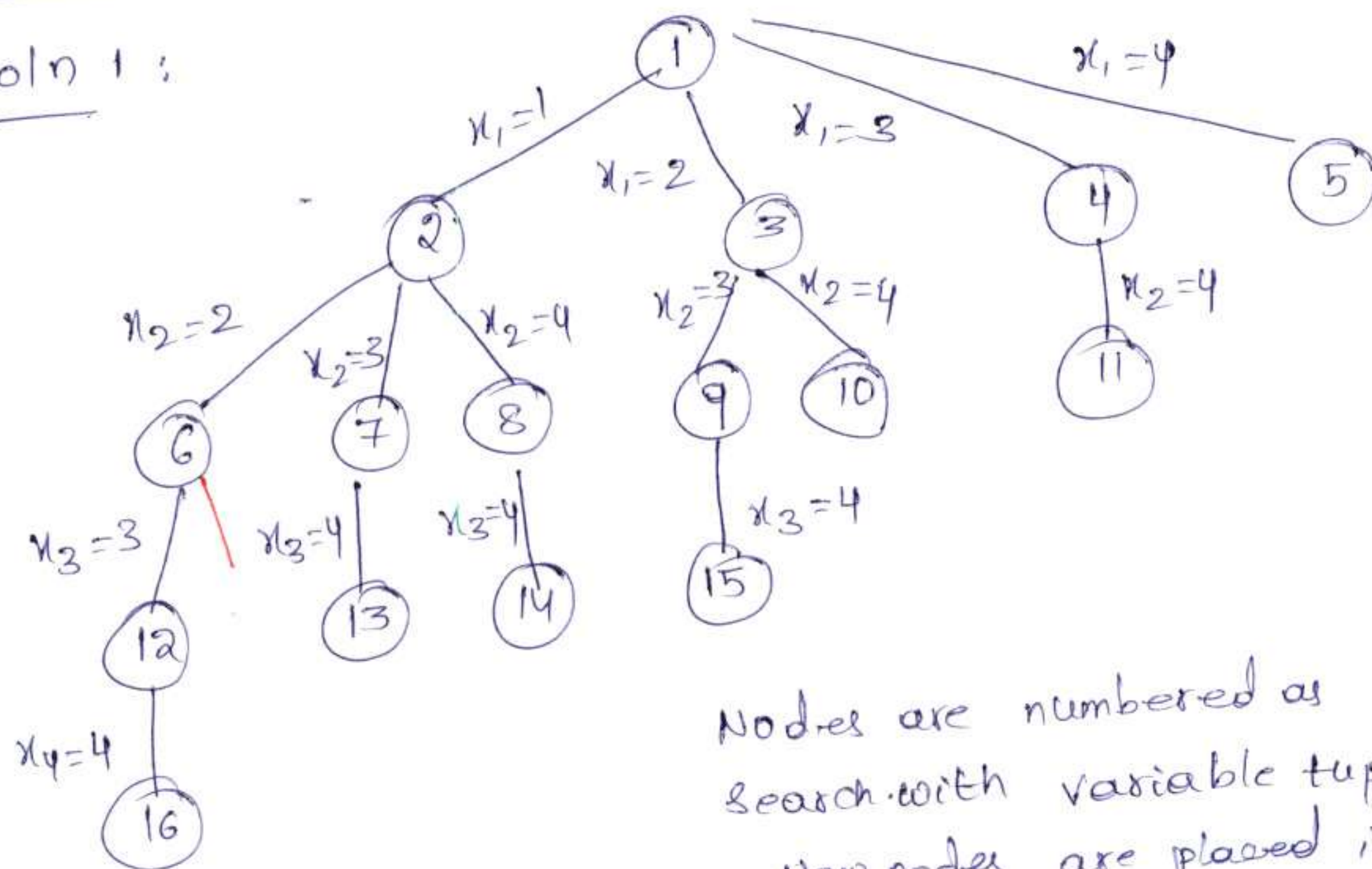


- Nodes are numbered as in depth first search.
- Edges from level i to level $i+1$ are labeled with the values of x_i .
- The soln space is defined by all the paths from the root node to a leaf node. There are $4! = 24$ leaf nodes in the tree.

Sum of subsets: ~~problem~~ for 4 queens problem.

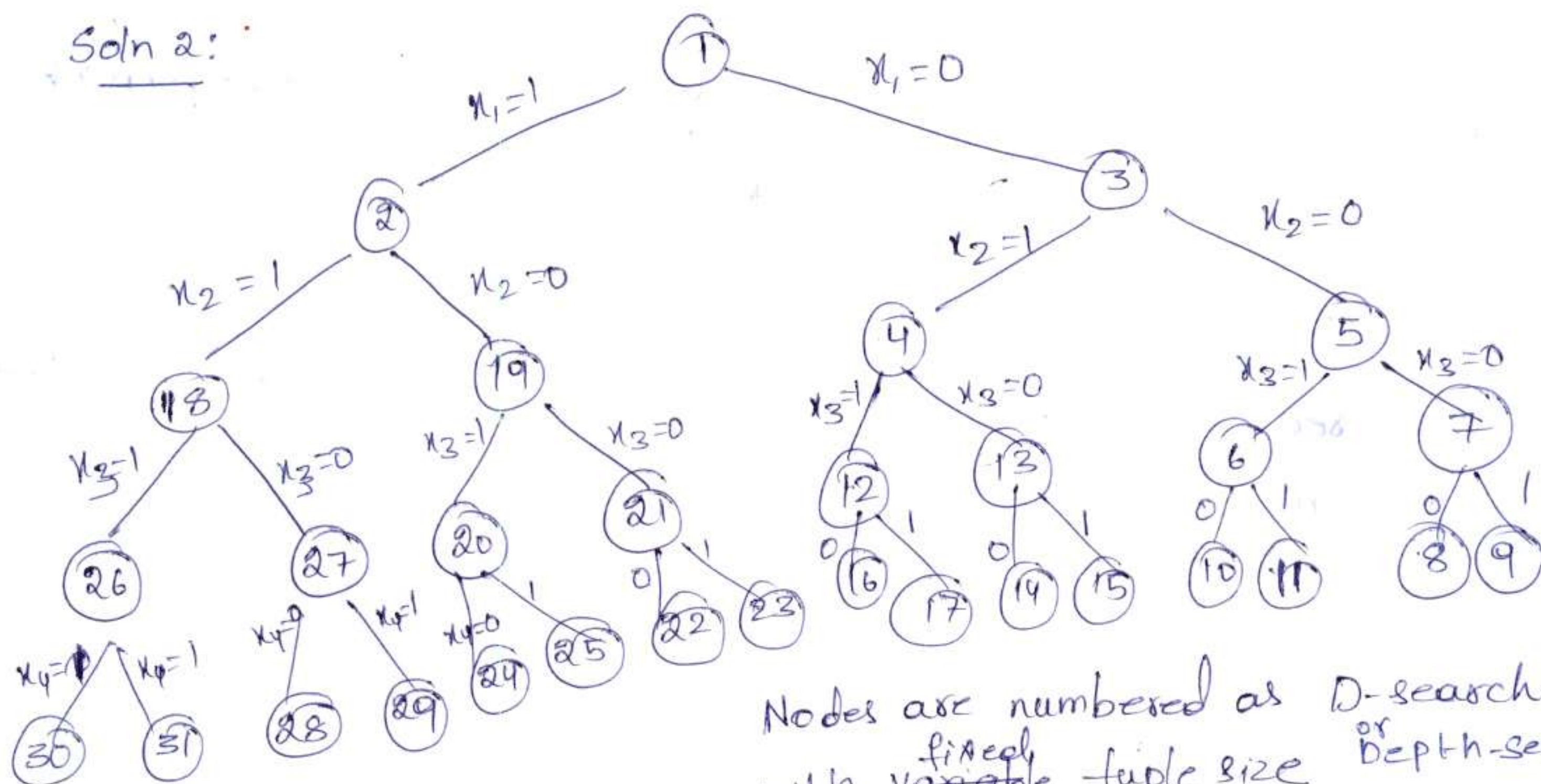
(6)

Soln 1:



Nodes are numbered as breadth first search with variable tuple size formulation. New nodes are placed into queue.

Soln 2:



Nodes are numbered as D-search or Depth-search with fixed tuple size. New nodes are placed into stack.

Soln 1

At each node soln space is partitioned into sub soln space. The soln space is defined by all paths from the root node to any node in the tree, since any such path corresponds to a subset satisfying the explicit constraints. Possible paths are $\{ \}$, $\{1\}$, $\{1,2\}$, $\{1,2,3\}$, $\{1,2,3,4\}$, $\{1,3,4\}$, $\{2\}$, $\{2,3\}$ and soon.

Soln 2 The left subtree of the root defines all subsets containing w_1
 " right " " " not " w_1
 These are 2^4 leaf nodes which represent 16 possible tuples.

In soln 1 tree, all nodes are soln states.

In soln 2 " , only leaf nodes are soln states.

Terminology (contd)

4. Answer states are those soln states 's' for which the path from the root to 's' defines a tuple that is a member of the set of solns (i.e. it satisfies the explicit constraints) of the problem.

5. The tree organization of the soln space is referred to as state space tree.

- soln space is partitioned into disjoint sub soln space, at each internal node.

Every elt of the soln space is represented by atleast one node in the state space tree.

6. Soln tree 2 is a static tree. Tree organizations are independent of the problem instance being solved.

7. Soln tree 1 is a dynamic tree.

→ soln 2 tree are numbered according to two generation methods

In which E-node remains the E-node until it is dead.

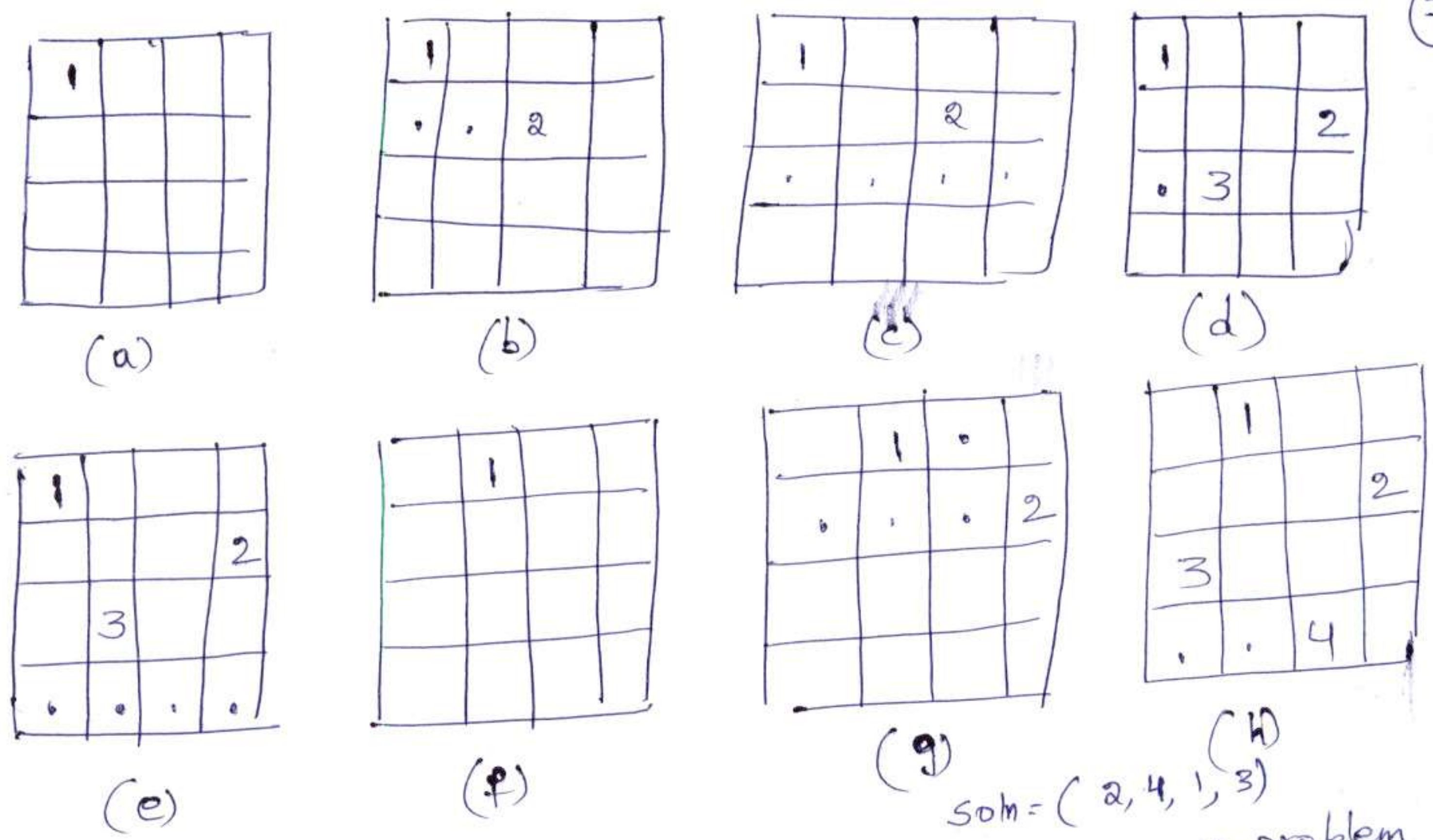
→ New nodes are placed into a stack ~~instead~~. i.e. uses Depth-Search.

→ soln 1 tree each new node is placed into a queue. When all the children of the current E-node have been generated, the next node at the front of the queue becomes the new E-node.

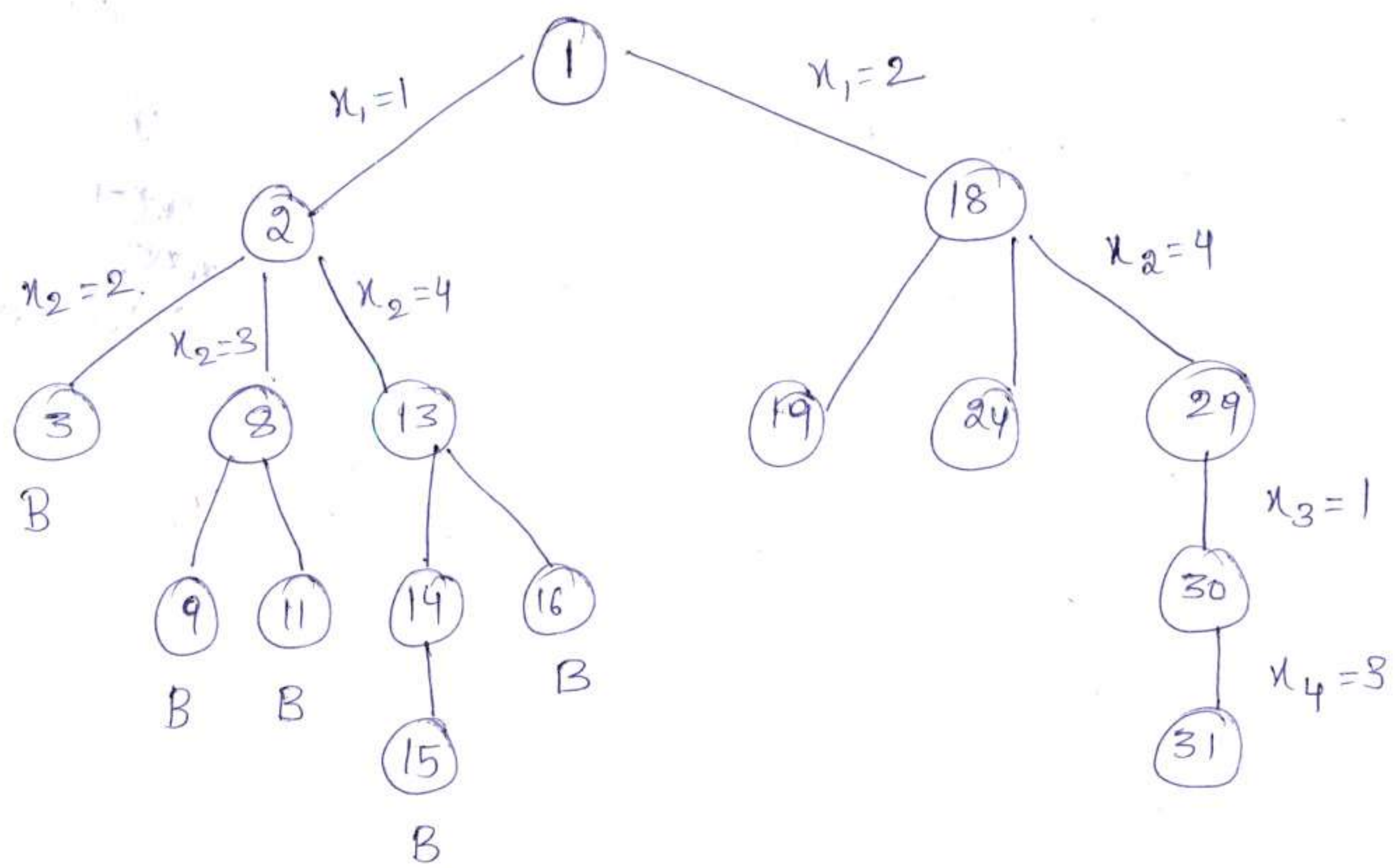
Backtracking soln for soln tree 1:-

Bounding func, criteria is if $(x_1, x_2, \dots, x_i)$ is the path to the

current E-node; then all children nodes with parent-child labelings x_{i+1} are such that $(x_1, \dots, x_{i+1})$ represents a chessboard configuration in which no two queens are attacking.



Example of a backtrack solution to the 4-queens problem



Portion of the tree that is generated during backtracking.

This tree is the part of solution 1 tree.

B - Node that gets killed as a result of the bounding func.

→ Assume $(x_1, x_2, \dots, x_i)$ be the set of all possible values for x_{i+1} such that $(x_1, x_2, \dots, x_{i+1})$ is also a path to a problem state.

- Assume the existence of bounding func B_{i+1} , such that if $B_{i+1}(x_1, x_2, \dots, x_{i+1})$ is false for a path $(x_1, x_2, \dots, x_{i+1})$ from the root node to a problem state, then the path cannot be extended to reach an answer node. Thus the candidates for position $i+1$ of the soln vector $(x_1, \dots, x_n)$ are those vals which are generated by T and satisfy B_{i+1} .

- Follows a postorder traversal of a tree.

→ Recursion is invoked by `Backtrack(i);`

- Algo Backtrack (n)

- Algo Backtrack(k)
 1. This schema describes the backtracking process using recursion.
 on entering, the first $k-1$ values $x[1], x[2], \dots, x[k-1]$ of the soln
 vector $x[1:n]$ have been assigned. $x[]$ and n are global

$$x_i = 1 \begin{array}{c} \textcircled{1} \\ \textcircled{2} \end{array}$$

2 for each $x[k] \in T(x[i], \dots, x[k-1])$ do

2 if $(B_k(x[1], x[2], \dots, x[k]) \neq 0)$ then

if $(x[1], x[2], \dots, x[k])$ is a path to an answer node

then write $(x [k: k])$,

if ($k < n$) then Backtrack($k+1$);

2

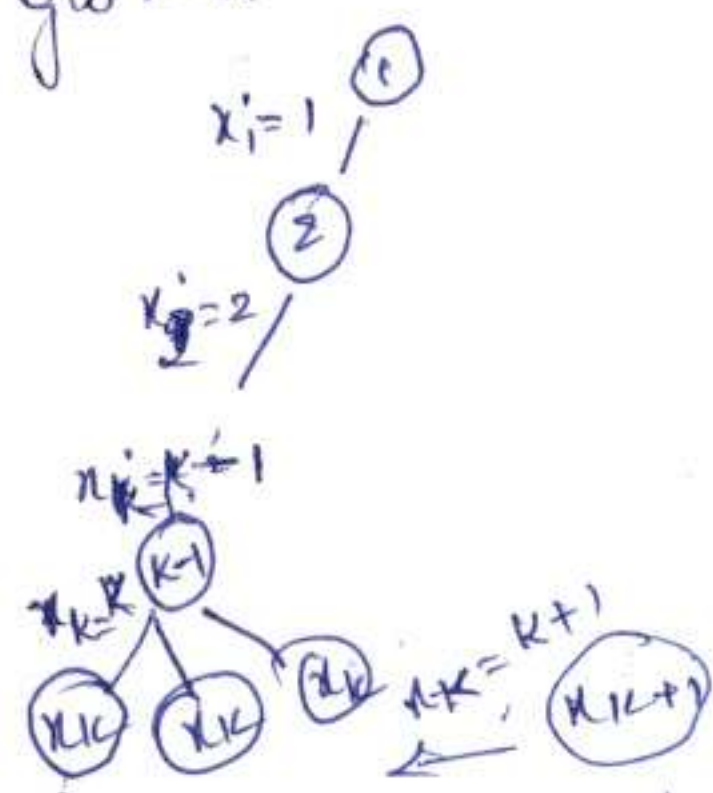
2

Iterative version:

Iterative version:
- Algo $\mathbb{P}$ Backtrack(n)

✓ This schema describes the backtracking process

All solutions are generated in $x[1:n]$ and printed as soon as they are determined $\pm /$

 $k=1, 2$ 

while ($k \neq 0$) do

(8)

{ if (there remains an untried $x[k] \in T(x[1], x[2], \dots, x[k-1])$
and $B_k(x[1], \dots, x[k])$ is true then

{ if $(x[1], \dots, x[k])$ is a path to an answer node
then write $(x[1], k)$;

$k := k + 1$; // consider the next set

}

else

$k := k - 1$; // Backtrack to previous set.

}

}

8-queens problem:-

→ column

	1	2	3	4	5	6	7	8
1				Q ₁				
2						Q ₂		
3								Q ₃
4		Q						
5							Q	
6	Q							
7			Q					
8					Q			

$Q_1 = 4$

$Q_2 = 6$

4 6 8 2 7 1 3 5

Suppose two queens are placed at positions (i, j) and (k, l)

If they are on the same diagonal only if

$$i - j = k - l$$

or

$$i + j = k + l$$

E.g. If a is at $(4, 2)$, diagonal
can be $a(3, 1), a(5, 3), a(6, 4)$
etc.

Algorithm place(k,i)

/* Returns true if a queen can be placed in the k^{th} row and i^{th} column. Otherwise it returns false. $x[]$ is a global array whose first $(k-1)$ values have been set. $\text{Abs}(x)$ returns the absolute value of x */

```

{ for j=1 to k-1 do
  if ((x[j] == i) // two in the same column
    or (Abs(x[j] - i) == Abs(j - k)))
    // or in the same diagonal

```

then return false;

return true;

}

Algorithm NQueens(k,n)

/* Using backtracking, this procedure prints all possible placements of 'n' queens on an $n \times n$ chessboard so that they are nonbacktracking */

```

{ for i:=1 to n do

```

```

  { if place(k,i) then

```

```

    { x[k] := i;

```

```

    if (k == n) then write (x[1:n]);

```

```

    else NQueens(k+1,n);

```

```

  }

```

```

}

```

Other solns to 8-queens

	1						
				2			
3							
						4	
5							
		6					
							7
				8			

Total no. of nodes in the 8-queens state space tree is

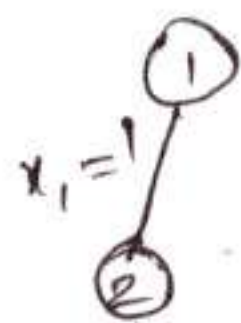
$$1 + \sum_{j=0}^7 \left[\prod_{i=0}^j (8-i) \right] = 69,281$$

NQueens (1, 4) Trace 4-Queens problem.

(8a)

$$\frac{k}{1} \quad \frac{i}{1}$$

place(1, 1)
k, i



(x_1, x_2, x_3, x_4)
(1,

$\frac{j}{NIL} \rightarrow$ return true

place(k, i) checks if k^{th} queen can be placed in column i

2
3
4 } $\rightarrow$ done when backtracked

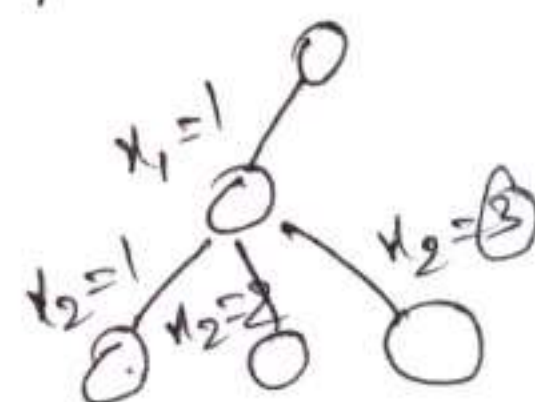
NQueens (2, 4)
k, n

Q1			1
.	.	Q2	

$$\frac{k}{2} \quad \frac{i}{1}$$

place(2, 1)

$\frac{j}{1} \quad x[i] == 1 \rightarrow$ return false



2

place(2, 2)
k, i

$\frac{j}{1} \quad x[i] \neq 2$ or $abs(x[i] - 2) = Abs(1 - 2)$
 $1 - 2 = 1 - 2$
 $\rightarrow$ return false.

3

place(2, 3)

$\frac{j}{1} \quad x[i] \neq 3$ or $abs(x[i] - 3) = Abs(1 - 3)$
 $1 - 3 \neq 1 - 2$
 $\rightarrow$ return true.

Q2 cannot be placed in 3rd coln.

4 $\rightarrow$ done when backtracked.

NQueens (3, 4)
k, n

$$\frac{k}{3} \quad \frac{i}{1}$$

place(3, 1)

$$\frac{j}{1}$$

$x[i] = 1$
 $1 = 1$

$\rightarrow$ return false

$$\frac{i}{2}$$

place(3,2)

$$\frac{j}{1}$$

$$x[1] = 2$$

$$1 \neq 2$$

$$\text{or } \text{Abs}(x[1] - 2) = \text{Abs}(1 - 2)$$

$$1 \neq 2$$

$$2$$

$$x[2] \neq 2$$

$$3$$

$$\text{or } \text{Abs}(x[2] - 2) = \text{Abs}(2 - 3)$$

$$3 - 2 = 1$$

return false

$$3$$

place(3,3)

$$\frac{j}{1}$$

$$x[1] \neq 3$$

$$\text{or } \text{Abs}(x[1] - 3) = \text{Abs}(1 - 3)$$

return false

$$2 \times$$

$$4$$

place(3,4)

$$\frac{j}{1}$$

$$x[1] \neq 4$$

$$\text{or } \text{Abs}(1 - 4) \neq \text{Abs}(1 - 3)$$

$$2$$

$$x[2] \neq 4$$

$$3$$

$$\text{or } \text{Abs}(3 - 4) = \text{Abs}(2 - 3)$$

$$1 = 1$$

return false

Backtrack to NQueens(2,4)

$$\frac{i}{4}$$

$$\frac{k}{2}$$

place(2,4)

$$\frac{j}{1}$$

$$x[1] \neq 4$$

$$\text{or } \text{Abs}(1 - 4) \neq \text{Abs}(1 - 2)$$

return true

NQueens(3,4)

$$i=1$$

place(3,1)

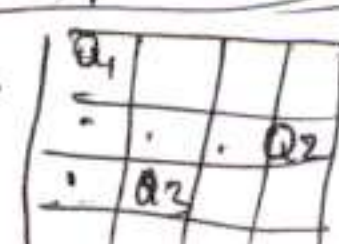
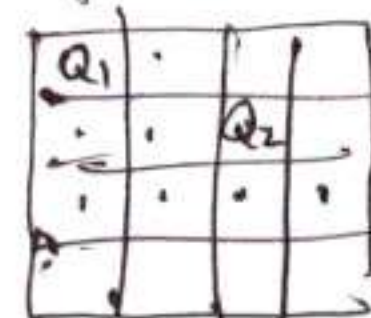
$$\frac{j}{1}$$

$$x[1] = 1$$

$$1 = 1$$

return false

$$2 \times$$



Sum of subsets problem

90

Given positive numbers w_i $1 \leq i \leq n$, and m , find all subsets of the w_i whose sum are m .

If $n=4$, $(w_1, w_2, w_3, w_4) = (1, 13, 24, 7)$ & $m=31$ then

desired subsets are $(1, 13, 7)$ & $(24, 7)$

Indexes of solns $(1, 2, 4)$ & $(3, 4)$ are answer.

Formulation 1:-

All solns are k -tuples $(x_1, x_2, \dots, x_n)$, $1 \leq k \leq n$, and diff solns may have diff-sized tuples. Soln1 - $(1, 2, 4)$, Soln2 - $(3, 4)$

Explicit constraints, $x_i \in \{s \mid s \text{ is an integer } \& 1 \leq s \leq n\}$

Implicit " " $\Rightarrow$ no two be the same and sum of the corresponding

w_i 's be m . Ex $(1, 2, 4)$ & $(1, 4, 2)$ represent same

subset.

a) $x_i < x_{i+1}$, $1 \leq i < k$.

Formulation 2:

Each soln subset is represented by an n -tuple $(x_1, x_2, \dots, x_n)$

such that $x_i \in \{0, 1\}$, $1 \leq i \leq n$. $x_i = 0$ if w_i is not chosen

$= 1$ if " chosen.

$\Rightarrow$ soln of above instance is $(1, 1, 0, 1)$ & $(0, 0, 1, 1)$

This formulation expresses all solns using a fixed-sized tuple.

$\Rightarrow$ In both the formulations, the soln space consists of 2^n distinct tuples.

Backtracking algo determine problem solns by systematically searching the soln space for the given problem instance.

Search is done by using a tree organization for the soln space. For a given soln space many tree organizations may be possible.

Ex: 1 [n-queens] The n-queens problem is a generalization of the 8x8 queens problem. n-queens are to be placed on an $n \times n$ chess board so that no two attack i.e. no two queens are on the same row, column, or diagonal.

Soln space consists of all $n!$ permutations of the n -tuple, $(1, 2, \dots, n)$

Follow Fig in page 5 backside

→ once a state space tree has been conceived of for any problem, this problem can be solved by generating the problem state, determining which soln states are answer states.

2 Methods to generate problem states.

- Both begin with root node & generate other nodes.

- A node which has been generated and all whose children have not yet been generated is called a live node.

- The live node whose children are currently being generated is called

E-node.

- A dead node is a generated node which is not to be expanded further or all of whose children have generated.

- In both methods, we have list of live nodes.

Method 1: As soon as a new child 'C' of the current E-node 'R' is generated this child will become the new E-node. Then R will become the E-node again when the subtree C has been fully explored. — Depth first

2. E-node remains the E-node until it is dead.

- Breadth first
- In both methods bounding funes kill live nodes without generating child.

Depth first node gen, with bounding funes is called backtracking
State gen methods, E-node remains E-node until it is dead leads to branch-&-bound methods

Backtracking soln to Sum of Subsets problem : -

9a

Suppose 'n' distinct +ve numbers (usually called weights), find all the combinations of these numbers whose sums are 'm'. This is called the sum of subsets problem.

Consider a backtracking soln using the fixed tuple size strategy. Element x_i of the soln vector is either one or zero depending on whether the weight w_i is included or not.

For a node at level 'i' the left child corresponds to $x_i = 1$ and the right to $x_i = 0$.

A simple choice for the bounding func is $B_k(x_1, \dots, x_k) = \text{true}$ iff

$$\sum_{i=1}^k w_i x_i + \sum_{i=k+1}^n w_i \geq m$$

$x_1, \dots, x_k$ cannot lead to an answer node if this condn is not satisfied.

The bounding func can be strengthened if we assume the w_i 's are initially in nondecreasing order.

and $B_k(x_1, \dots, x_k) = \text{true}$ if

$$\sum_{i=1}^k w_i x_i + \sum_{i=k+1}^n w_i \geq m \text{ and } \sum_{i=1}^k w_i x_i + w_{k+1} \leq m.$$

Ex. $n=6, m=30, \{w[1:6]\} = \{5, 10, 12, 13, 15, 18\}$.

Rectangular nodes list values of s, k, r on each call of SumofSub

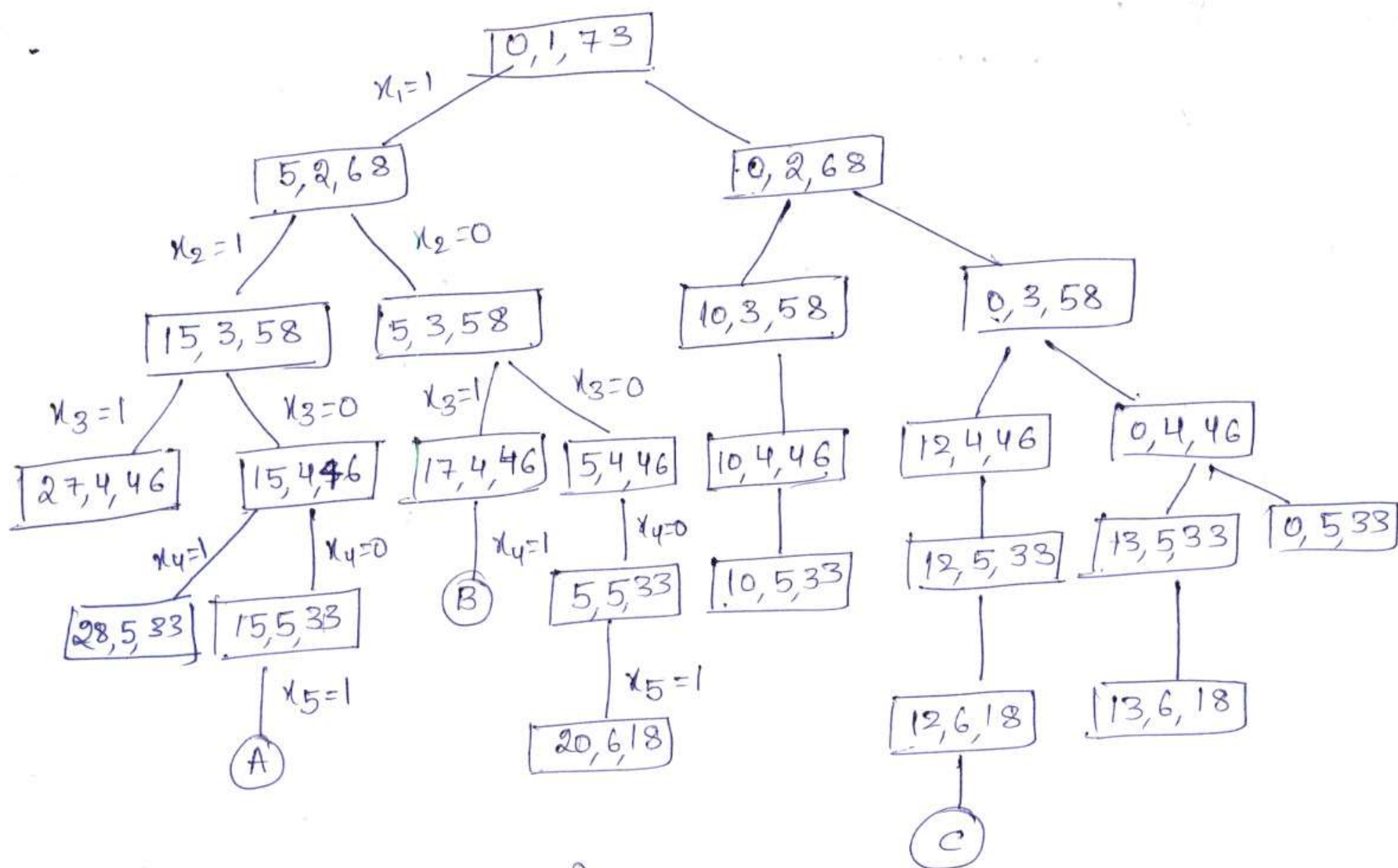
Circular nodes represent points at which sums m are printed out.

Node A o/p's $(1, 1, 0, 0, 1)$

B " $(1, 0, 1, 1)$

C " $(0, 0, 1, 0, 0, 1)$

Full state space tree for $n=6$ contains $2^6 - 1 = 63$ nodes (neglecting all leaf nodes 64)



Algorithm SumofSub(s, k, r)

Find all subsets of $w[1:n]$ that sum to m . The values of $x[j]$, $1 \leq j < k$, have already been determined. $S = \sum_{j=1}^{k-1} w[j] * x[j]$ and $r = \sum_{j=k}^n w[j]$. The $w[j]$'s are in nondecreasing order.

It is assumed that $w[1] \leq m$ and $\sum_{i=1}^n w[i] \geq m$

// Generate left child: Note, $s + w[k] \leq m$ since B_{k-1} is true.

$x[k] = 1$;
if $(s + w[k] = m)$ then write $x[1:k]$, ~~$\leq m$ since B_{k-1} is true.~~ // subset found

// There is no recursive call here as $w[j] > 0, 1 \leq j \leq n$.

else if $(s + w[k] + w[k+1] \leq m)$

then SumofSub($s + w[k], k+1, r - w[k]$);

// Generate right child and evaluate B_k .

if $((s + r - w[k] \geq m) \text{ and } (s + w[k+1] \leq m))$ then

$x[k] := 0$;

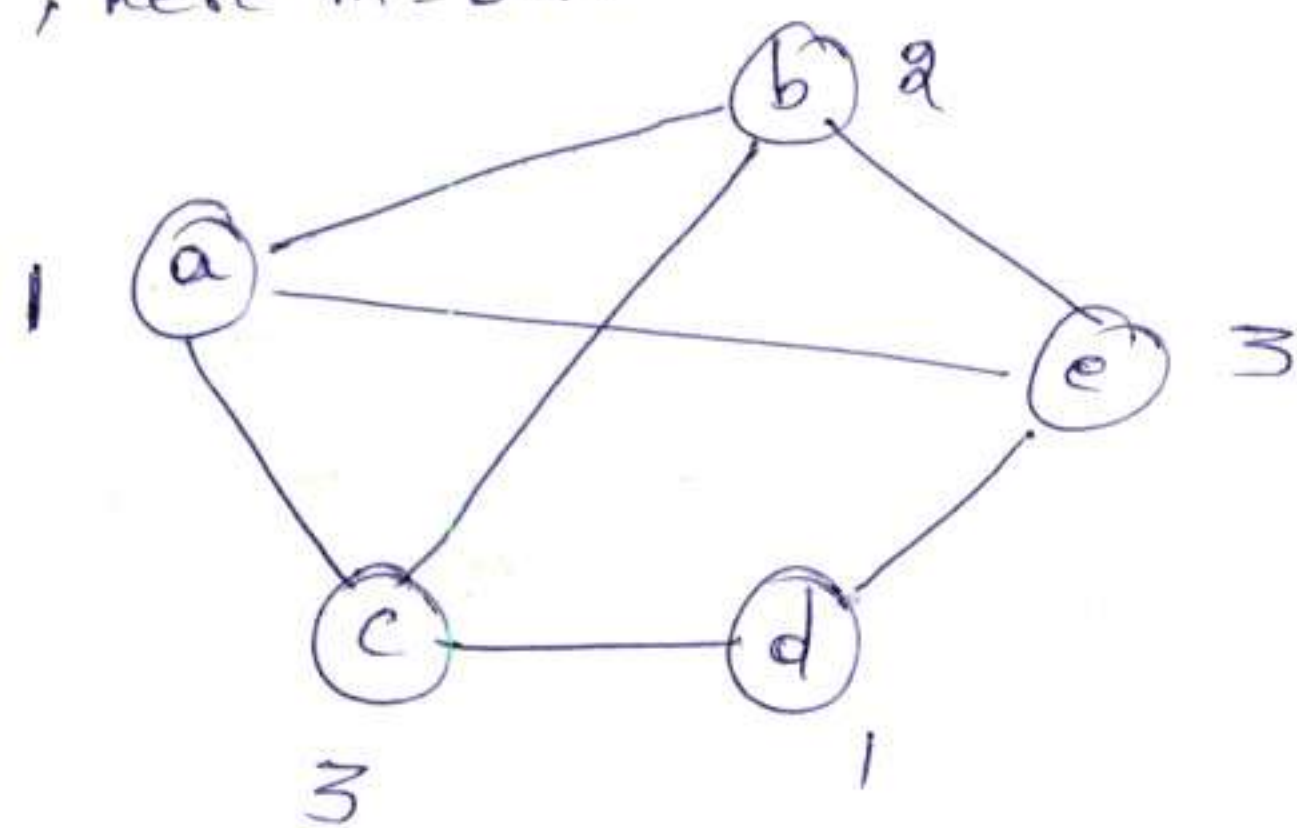
SumofSub($s, k+1, r - w[k]$);

end if

Graph Coloring:-

10

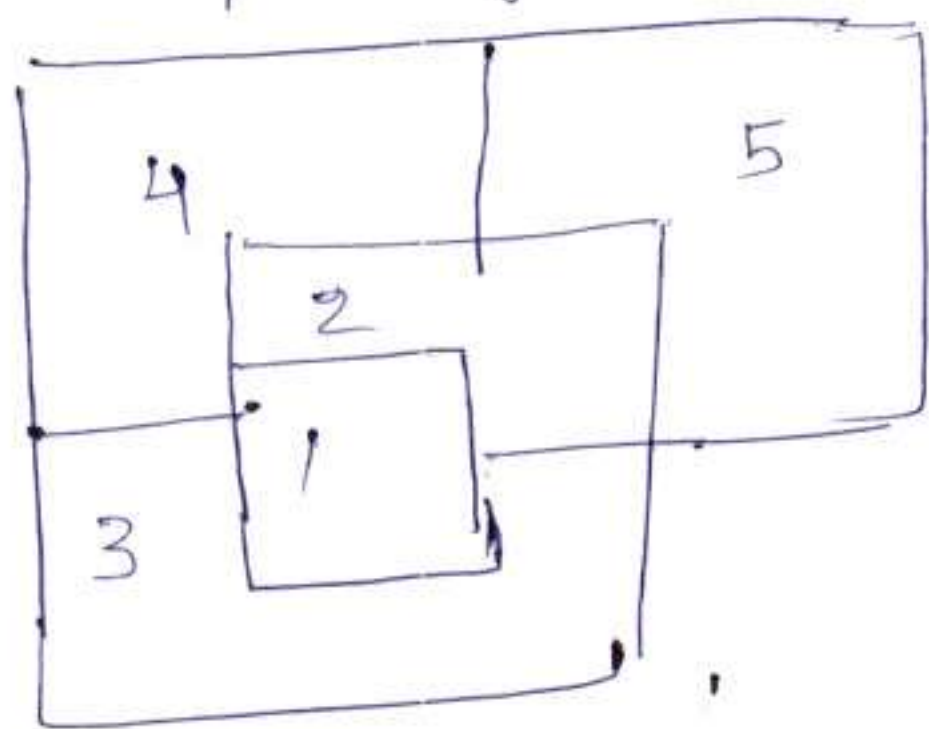
Let G be a graph and m be a given +ve number. Discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only ' m ' colors are used. This is called m -colorability decision problem, m is called chromatic number. If d is the degree of the given graph, then it can be colored with $d+1$ colors, here $m = d+1$.



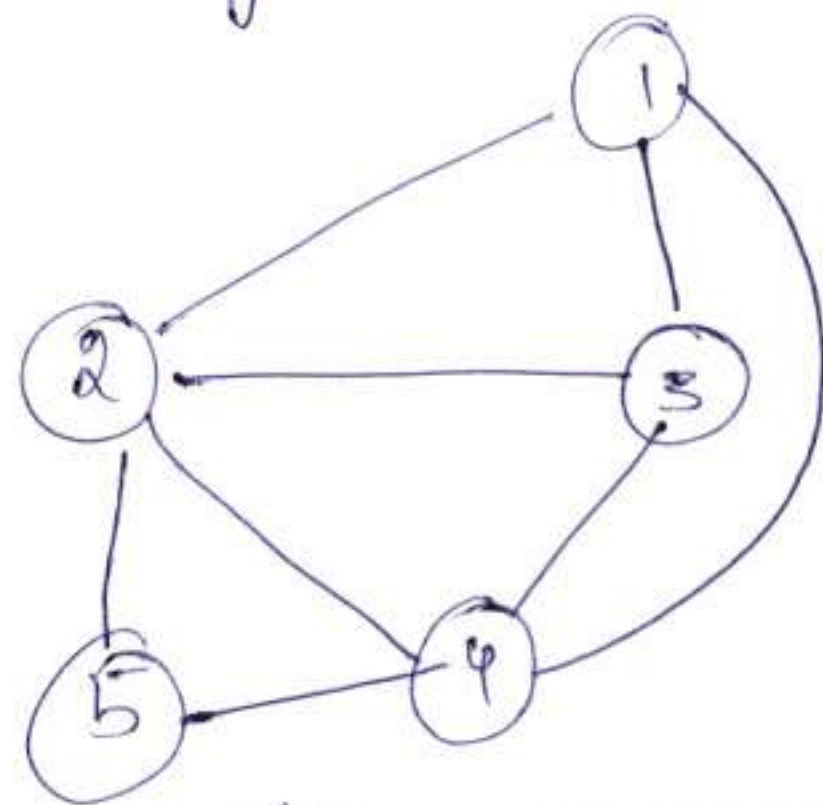
chromatic number = 3

A graph is said to be planar iff it can be drawn in a plane in such a way no two edges cross each other. Famous spl case of m -colorability decision problem is the 4-color problem for planar graphs.

Any map can be easily transformed into a graph. Each region of the map becomes a node, and if two regions are adjacent then the corresponding nodes are joined by an edge.



Map with 5 regions



It's corresponding graph.

This map requires 4 colors.

* For any size of map size, five colors are sufficient

→ Suppose we represent a graph by its adjacency matrix $G[1:n, 1:n]$ where $G[i, j] = 1$, if (i, j) is an edge of G & $G[i, j] = 0$ otherwise.

The colors can be represented by the integers $1, 2, \dots, m$ and solns are given by the tuple $(x_1, \dots, x_n)$ where x_i is the color of node i .

Algorithm mColoring(k)

/* This algo was formed using the recursive backtracking schema. The graph is represented by its boolean adjacency matrix $G[1:n, 1:n]$. All assignments of $1, 2, \dots, m$ to the vertices of the graph such that adjacent vertices are assigned distinct integers are printed. k is the index of the next vertex to color.

{ repeat

{ // Generate all legal assignments for $x[k]$.

Next-Value(k); // Assign to $x[k]$ a legal color.

if ($x[k] = 0$) then return; // No new color possible

if ($k = n$) then // At most ' m ' colors have been used to color the n vertices.

write ($x[1:n]$);

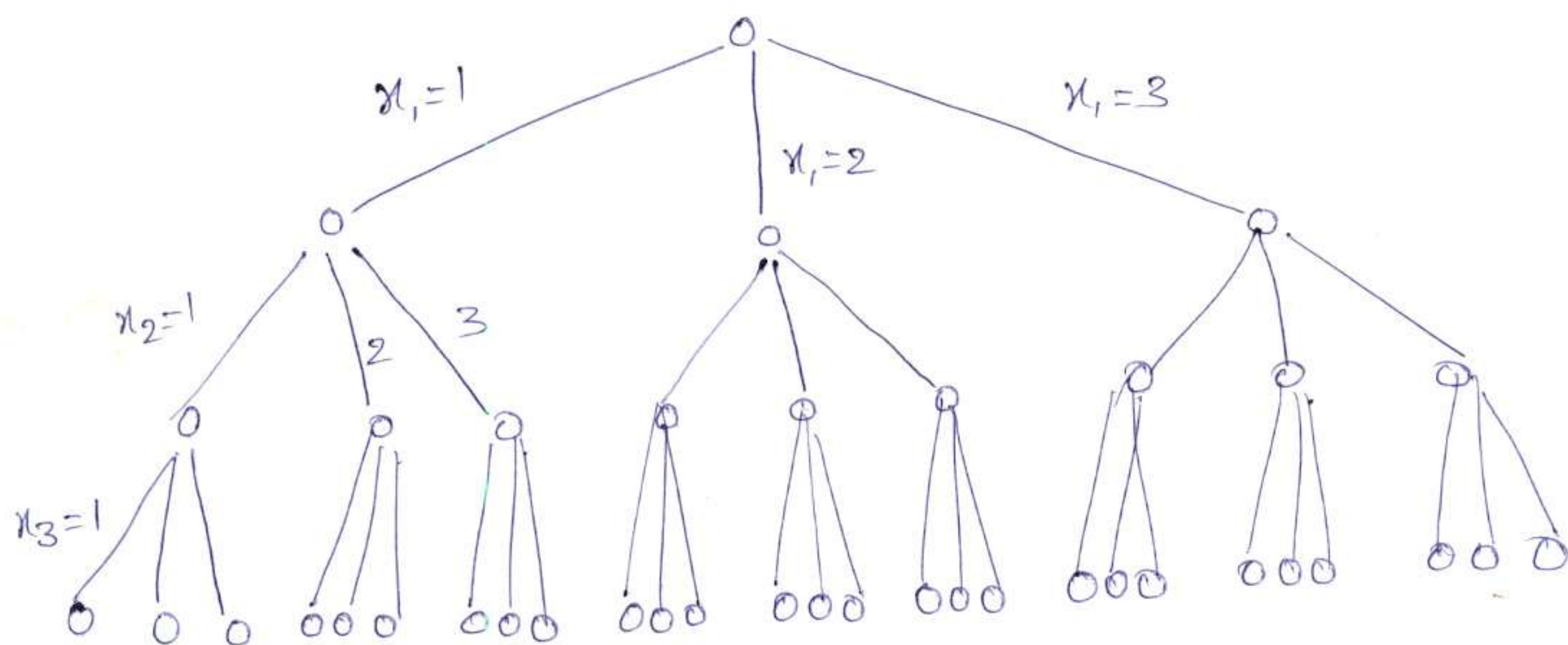
else mColoring($k+1$);

} until (false);

}

→ The underlying state space tree used is a tree of degree m and height $n+1$. Each node at level i has m children corresponding to the m possible assignments to x_i , $1 \leq i \leq n$. Nodes at level $n+1$ are leaf nodes.

Statespace tree when $n=3$ and $m=3$



Each path to a leaf represents a coloring using atmost three colors.

only 12 solns exist with 3 colors:

computing time = $O(nm^n)$.

Algorithm Nextvalue(k)

/* $x[1], \dots, x[k-1]$ have been assigned integer values in the range $[1, m]$ such that adjacent vertices have distinct integers. A value for $x[k]$ is determined in the range $[0, m]$. $x[k]$ is assigned the next highest numbered color while maintaining distinctness from the adjacent vertices of vertex k . If no such color exists, then $x[k]$ is 0 */

Σ repeat

Σ $x[k] := (x[k] + 1) \bmod (m+1)$; // Next highest color.

if ($x[k] = 0$) then return; // All colors have been used

for $s := 1$ to n do

Σ // check if this color is distinct from adjacent colors

if ($(G[k, s] \neq 0 \text{ and } (x[k] = x[s]))$ and (k, s) is an edge & if adj vertices have same color

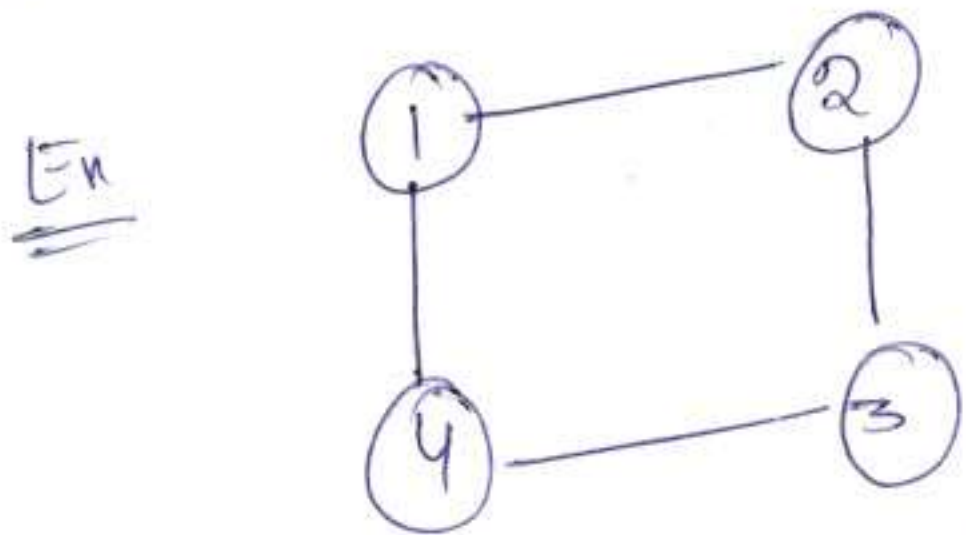
then break;

}

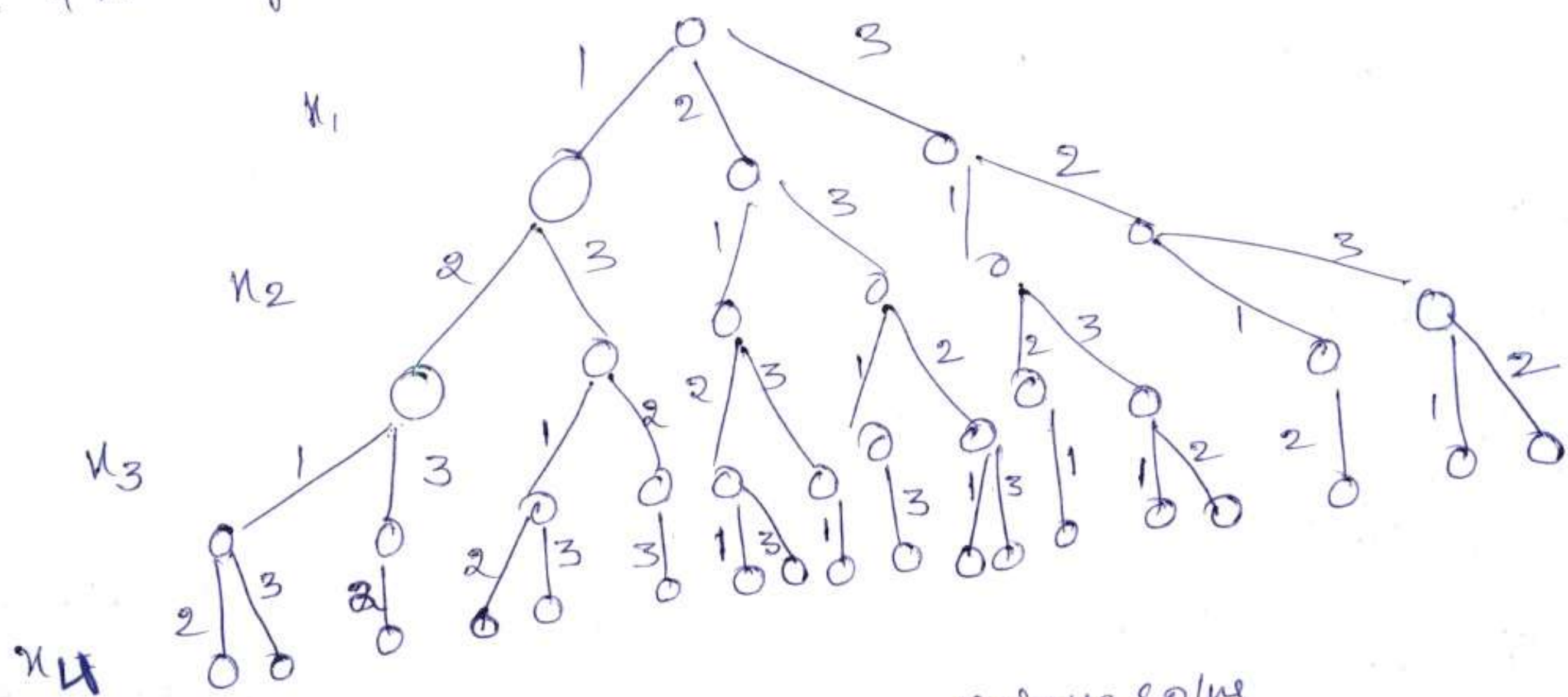
if($j = n + 1$) then return; // New color found

{until (false);

↓



A 4 node graph and all possible 3-colorings



N-Queen
Board

1x1

2x2

3x3

4x4

5x5

6x6

7x7

8x8

9x9

10x10

Total solns

1

0

0

2

10

4

40

92

362

724

Unique solns

1

0

0

1

2

1

6

12

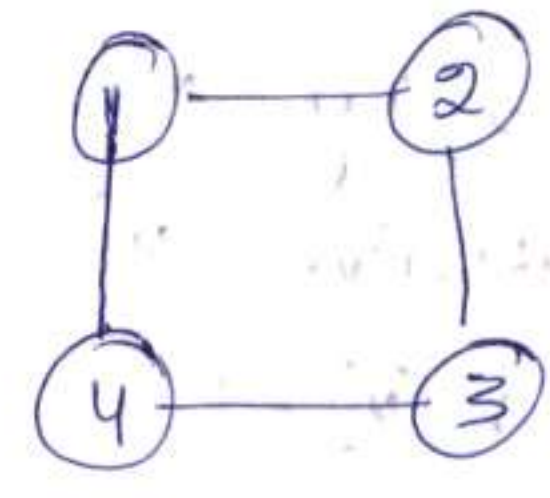
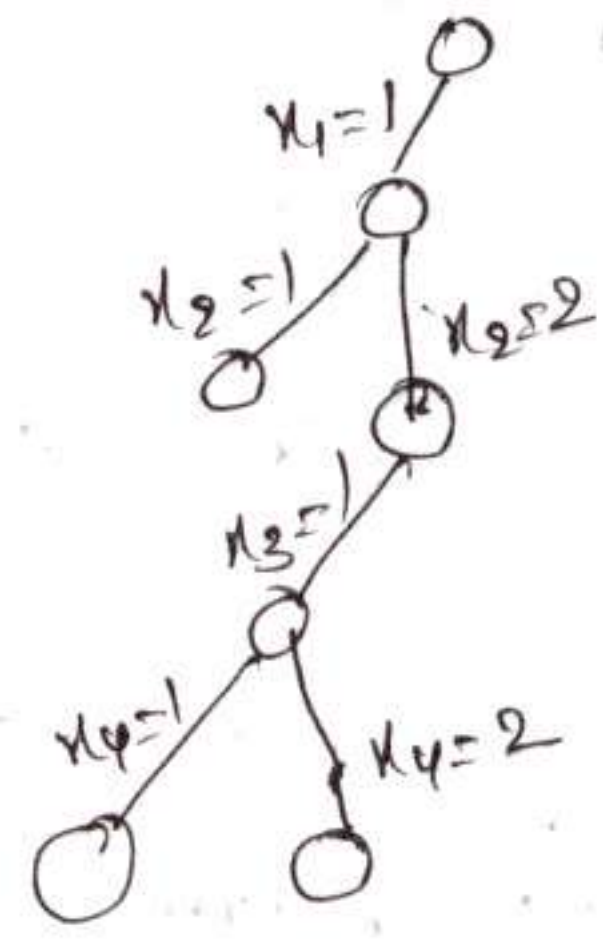
46

92

MColoring(1) $n=4, m=3$, $(x_1, x_2, x_3, x_4) = (0, 0, 0, 0)$
NextValue(1)

$k=1$ $x[k] = (0+1) \bmod (3+1)$
 $x[1] = 1 \bmod 4 = 1$

- j
1 $G(1,1) \neq 0$
 $0 \neq 0 \rightarrow \text{False}$
- 2 $G(1,2) \neq 0$ $x[k] = x[j]$
 $1 \neq 0 \rightarrow T$ $x[1] = x[2]$
 $1 = 0 \rightarrow F$
- 3 $G(1,3) \neq 0$
 $0 \neq 0 \rightarrow F$
- 4 $G(1,4) \neq 0$ $x[1] = x[4]$
 $1 \neq 0 \rightarrow T$ $1 \neq 0 \rightarrow F$
- 5
if(5 = 4+1) $\rightarrow$ return



MColoring(2)
Next-value(2)
 $k=2$ $x[2] = (0+1) \bmod 4$
 $= 1$

- j
1 $G(2,1) \neq 0$
 $1 \neq 0 \rightarrow T$
 $\hookrightarrow$ break

2 X
3 X
4 X

$x[2] = 2 \bmod 4$
 $= 2$

- j
1 $G(2,1) \neq 0$
 $1 \neq 0 \rightarrow T$
- 2 $G(2,2) \neq 0$
 $0 \neq 0 \rightarrow F$
- 3 $G(2,3) \neq 0 \rightarrow T$
- 4 $G(2,4) \neq 0 \rightarrow F$
- 5

$x[2] = x[1]$
 $1 = 1 \rightarrow T$

$x[k] = x[j]$
 $x[2] = x[1]$
 $2 = 1 \rightarrow F$

$x[2] = x[3] \rightarrow F$

$j = n+1$ return

$n=4$

	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	0
4	1	0	0	0

mcoloring (3)

Nextvalue (3)

k=3 $n[3] = 1$

j

1 $G[3,1] \neq 0 \rightarrow F$

2 $G[3,2] \neq 0 \rightarrow T$ $\&$ $n[3] = n[2] \rightarrow F$

3 $G[3,3] \neq 0$
 $0 \neq 0 \rightarrow F$

4 $G[3,4] \neq 0 \rightarrow F$

5 $j = n+1$ return

mcoloring (4)

Nextvalue (4)

$n[4] = 1$

k=4

j

1 $G[4,1] \neq 0$ and $n[4] = n[1]$

$1 \neq 0 \rightarrow T$

$1 = 1 \rightarrow T$

break

2 $\times$

3 $\times$

4 $\times$

$n[4] = 2$

j

1 $G[4,1] \neq 0 \rightarrow T$ and $n[4] = n[1]$

$2 = 1 \rightarrow F$

2 $G[4,2] \neq 0$

$0 \neq 0 \rightarrow F$

3 $G[4,3] \neq 0 \rightarrow T$

$\&$ $n[4] = n[3]$

$2 = 1 \rightarrow F$

4 $G[4,4] \neq 0 \rightarrow F$

5 $j = n+1$ then return

if (4=4) write $(n_1, n_2, n_3, n_4) = (1, 2, 1, 2)$