



# SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

*Course*  
**Algorithm Design Strategies & Analysis**

*Topic*

**CIA 2 Topics**

**Shri B. Srinivasan**  
*Assistant Professor-III, School of Computing*  
*SASTRA Deemed To Be University*

# Topics

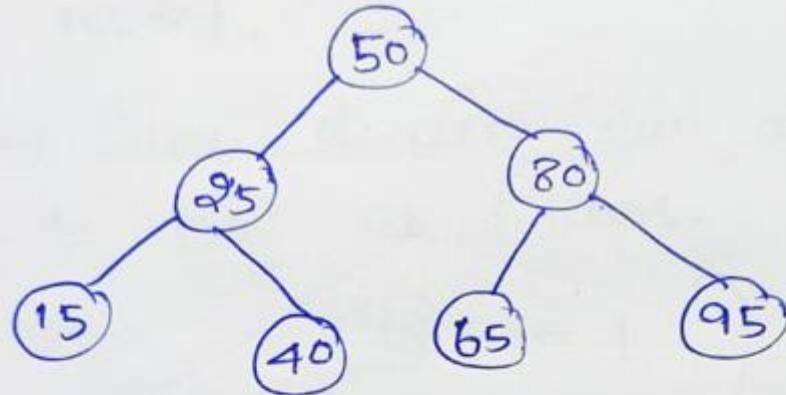
- ✓ **Dynamic Programming Method**
  - ✓ Optimal Binary Search Tree
  - ✓ String Editing Problem
  - ✓ 0/1 Knapsack Problem
- ✓ **Backtracking Strategy**
  - ✓ N-Queen Problem
  - ✓ Subset Sum Problem
  - ✓ Hamiltonian Cycles Problem
- ✓ **Graph – Basic Terminologies**
- ✓ **Traversal Algorithms**
  - ✓ Breadth First Search (BFS)
  - ✓ Depth First Search (DFS)
  - ✓ Topological Sort
- ✓ **Minimum Spanning Tree**
  - ✓ Prim's Algorithm
  - ✓ Kruskal's Algorithm

# **Dynamic Programming Approach**

## **Optimal Binary Search Tree**

## Optimal Binary Search Tree

### Binary Search Tree:



$$\text{Left}(R) \leq R < \text{Right}(R)$$

$$\text{Search Time : } O(\log_2 n) \Rightarrow O(h)$$

### Cost of Searching

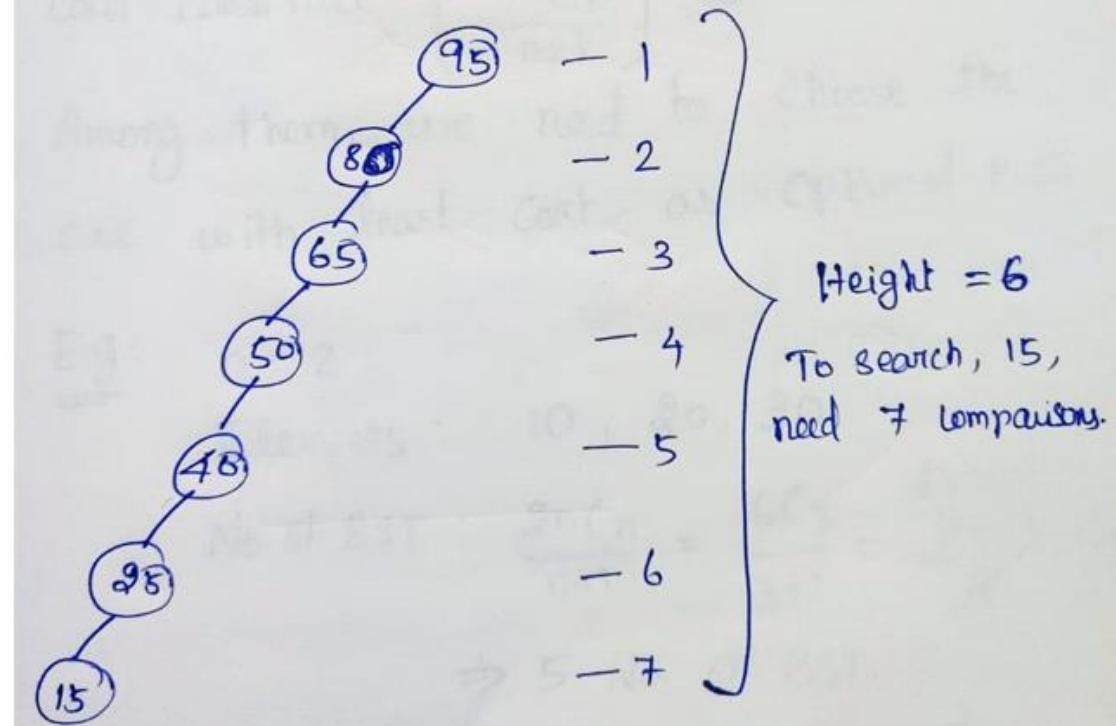
- No. of Comparisons needed for searching an element.
- It depending on the height of the tree.
- If the tree is balanced, then the height is  $\log_2 n$ .

→ So if tree is balanced, the cost will be minimum.

→ In above example,  $n = 7$  and  $h = 2$ . So to search an element, maximum of 3 comparisons are needed.



→ Same elements can also be added to BST as follows.





## Optimal BST:

It is BST constructed from a list of elements for which the effective cost of searching should be minimum.

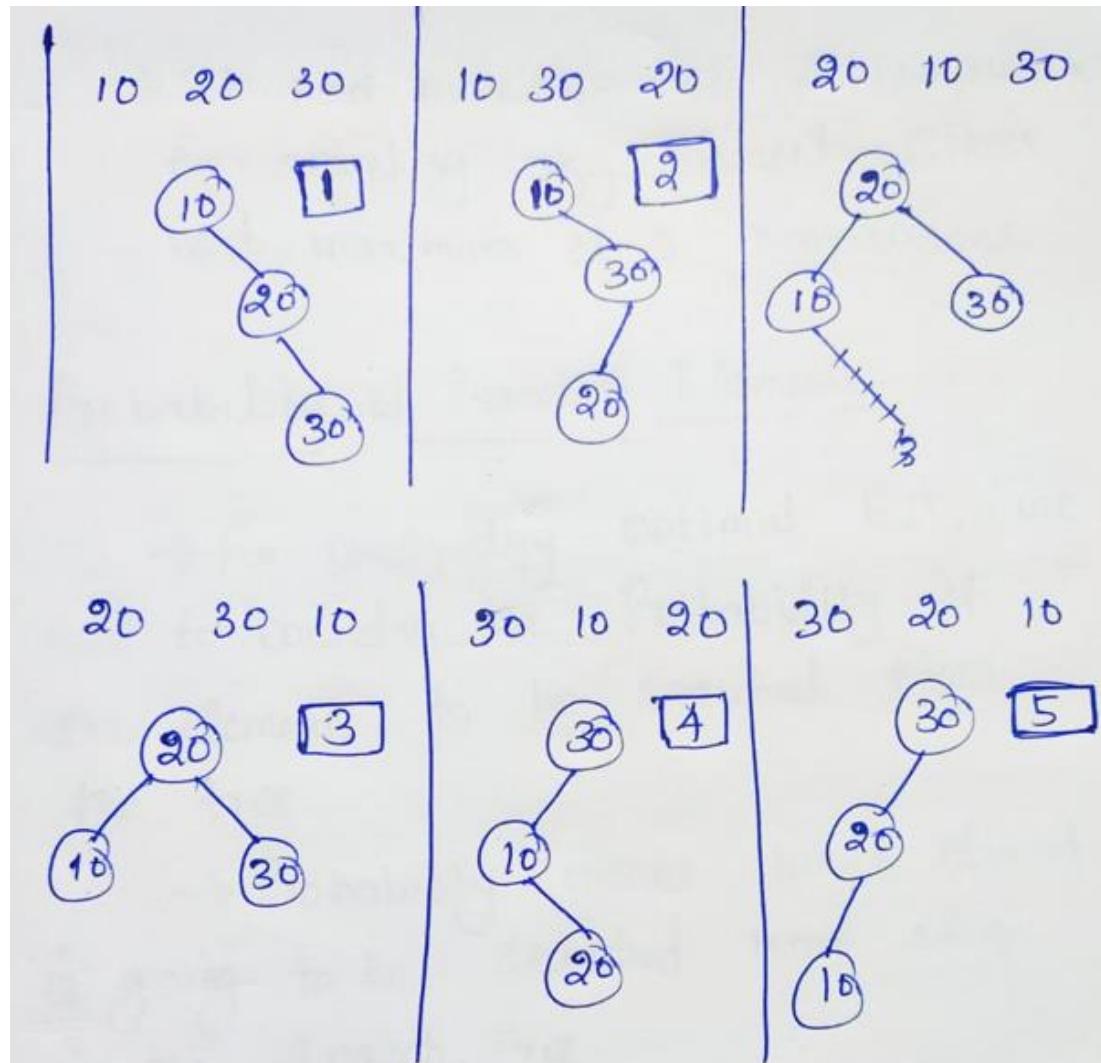
For a given 'n' elements, we can construct  $\binom{2^n C_n}{n+1}$  numbers of BST. Among them, we need to choose the one with least cost as Optimal BST.

E.g:  $n = 3$ :

Elements: 10, 20, 30

$$\text{No. of BST} : \frac{2^n C_n}{n+1} = \frac{6C_3}{3+1} = \frac{\frac{6 \times 5 \times 4}{3 \times 2 \times 1}}{4} = 5$$

$\Rightarrow 5$  No. of BST.



- We found 5 unique BSTs for the elements 10, 20, 30.
- Among these, the BST ③ will be the Optimal BST.
- It need maximum of 2 comparisons for searching any element. Others need maximum of 3 comparisons.

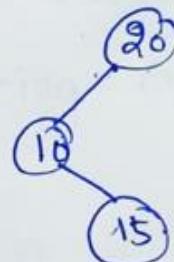


## Probability of Searching Elements:

→ For constructing optimal BST, we need to consider the Probability of the elements to be searched from the tree.

→ Probability means, which element is going to be searched more often in the search tree.

Ex: If the tree is:



→ Total number of search to be done is: 10



→ Among 10 searches,

3 times - 20

2 times - 15

1 time - 10

2 times - 30 (non existing)

3 times - 5 (non existing)

Then, the probability of the elements:

$$P(20) = 3/10$$

$$P(15) = 2/10$$

$$P(10) = 1/10$$

(successful search)

$$P(>20) = 2/10$$

$$P(<10) = 3/10$$

(unsuccessful search)

→ Probability of successful search as well as unsuccessful search will be given for constructing optimal BST.

→ The cost of searching will be computed from the given probabilities.

## Example: (Cost of Searching)

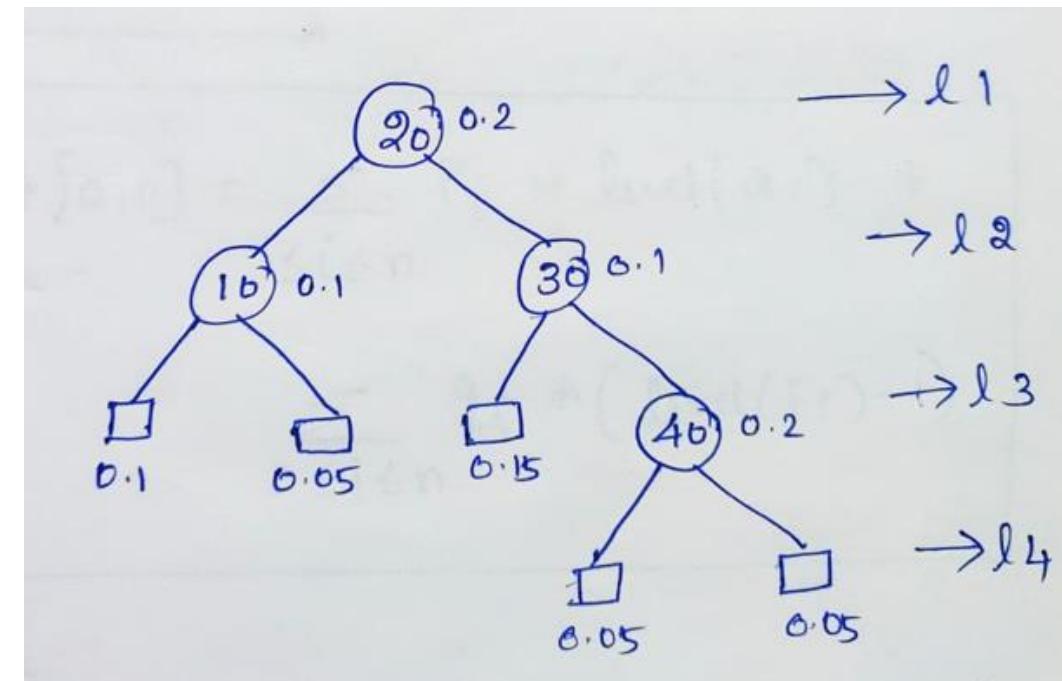
	1	2	3	4
(a) Keys :	10	20	30	40

•  $P_i$  : 0.1      0.2      0.1      0.2

$q_i$  : 0.1      0.05      0.15      0.05      0.05  
 0            1            2            3            4

$P_i \Rightarrow$  Probability of successful Search

$q_i \Rightarrow$  Probability of unsuccessful Search.





$$\begin{aligned} \text{Cost}[0,4] &= [0.1 * 2 + 0.2 * 1 + \\ &\quad 0.1 * 2 + 0.2 * 3] + \\ &\quad [0.1 * 2 + 0.05 * 2 + \\ &\quad 0.15 * 2 + 0.05 * 3 + 0.05 * 3] \\ &= 0.2 + 0.2 + 0.2 + 0.6 + 0.2 + 0.1 + \\ &\quad 0.3 + 0.15 + 0.15 \end{aligned}$$

$$\boxed{\text{Cost}[0,4] = 2.1}$$



$$\text{Cost}[0, n] = \sum_{1 \leq i \leq n} p_i * \text{level}(a_i) + \sum_{0 \leq i \leq n} q_i * (\text{level}(E_i) - 1)$$



## Dynamic Programming Approach (for optimal BST)

Problem:

- Input: 1. 'n' keys  
2. Successful Search Probabilities,  $p_i$   
3. UnSuccessful Search Probabilities,  $q_i$

Output: optimal BST



Formula:

$$C[i, j] = \min_{k=i+1, i+2, \dots, j} \left\{ C[i, k-1] + C[k, j] \right\} + w[i, j]$$

Where  $w[i, j] \Rightarrow \text{weight}$

$$w[i, j] = w[i, j-1] + p_j + q_j$$



$$\left. \begin{array}{l} w_{00} = q_0 \\ w_{11} = q_1 \\ w_{22} = q_2 \\ w_{33} = q_3 \end{array} \right\} \quad \begin{array}{l} w_{01} = q_0 + \underline{p_1 + q_1} \\ = w_{00} + \underline{p_1 + q_1} \\ \\ w_{02} = \underbrace{q_0 + p_1 + q_1}_{w_{01}} + \underline{p_2 + q_2} \\ w_{02} = w_{01} + p_2 + q_2 \end{array}$$

In general,

$$w_{ij} = w_{ij-1} + p_j + q_j$$



→ As per dynamic Programming algorithm, Smaller size Sub problems are going to be solved and the Solutions will be recorded in a table.

→ The cost for the problem with length = 1 will be determined first. From this other problems Solutions will be calculated.

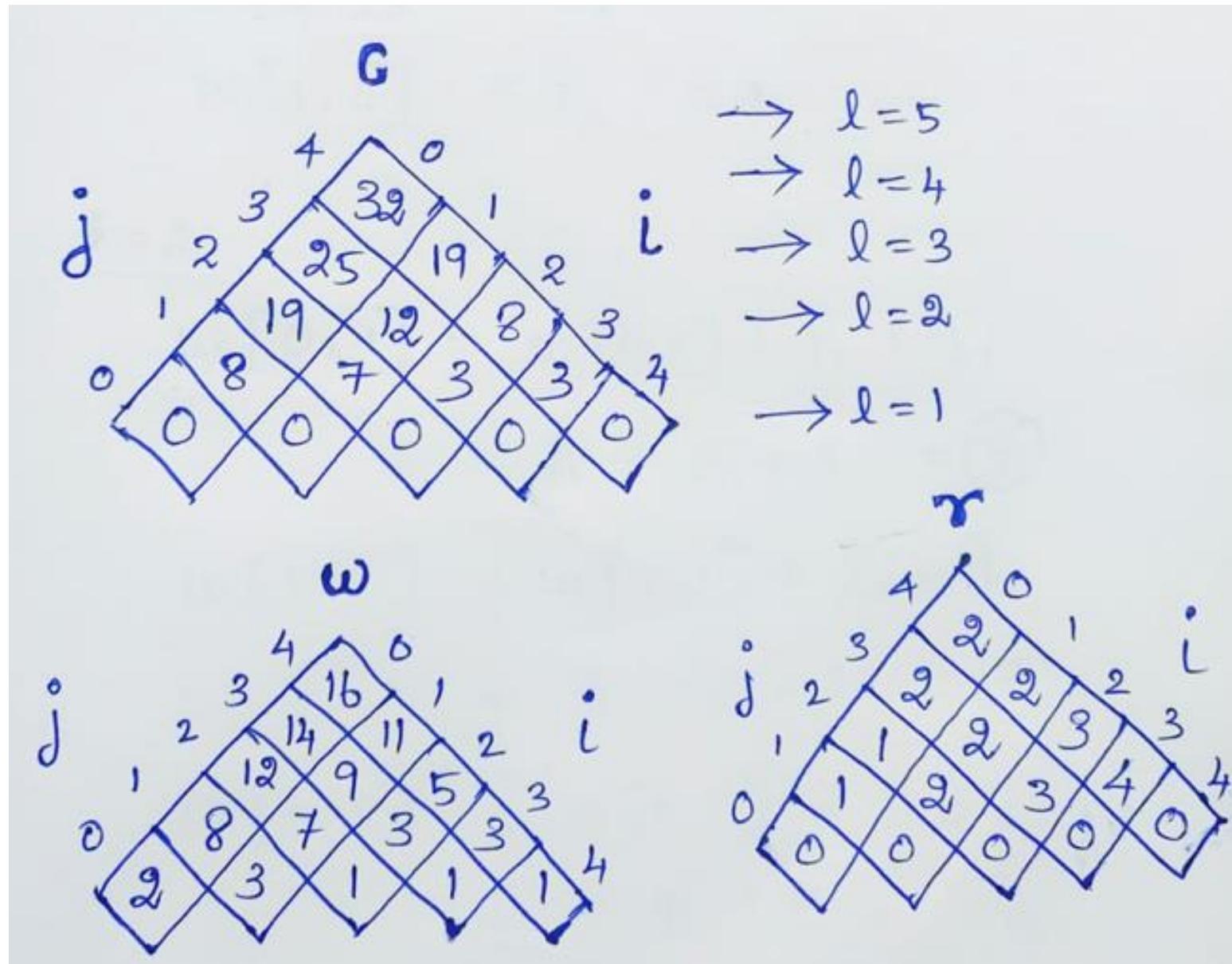


### Example For Optimal BST:

Keys : { 10, 20, 30, 40 }

$P_i$  : { 3, 3, 1, 1 }

$q_i$  : { 2, 3, 1, 1, 1 }





## Calculation of weight:

$l = 1$ :

$$w[0,0] = q_0 = 2$$

$$w[1,1] = q_1 = 3$$

$$w[2,2] = q_2 = 1$$

$$w[3,3] = q_3 = 1$$

$$w[4,4] = q_4 = 1$$

$l = 2$ :

$$\begin{aligned} w[0,1] &= w[0,0] + p_1 + q_1 \\ &= 2 + 3 + 3 = 8 \end{aligned}$$

$$\begin{aligned} w[1,2] &= w[1,1] + p_2 + q_2 \\ &= 3 + 3 + 1 = 7 \end{aligned}$$

$$\begin{aligned} w[2,3] &= w[2,2] + p_3 + q_3 \\ &= 1 + 1 + 1 = 3 \end{aligned}$$

$$\begin{aligned} w[3,4] &= w[3,3] + p_4 + q_4 \\ &= 1 + 1 + 1 = 3 \end{aligned}$$



$l=3:$

$$\begin{aligned}\omega[0,2] &= \omega[0,1] + p_2 + q_2 \\ &= 8 + 3 + 1 = 12\end{aligned}$$

$$\begin{aligned}\omega[1,3] &= \omega[1,2] + p_3 + q_3 \\ &= 7 + 1 + 1 = 9\end{aligned}$$

$$\begin{aligned}\omega[2,4] &= \omega[2,3] + p_4 + q_4 \\ &= 3 + 1 + 1 = 5\end{aligned}$$

$l=4:$

$$\begin{aligned}\omega[0,3] &= \omega[0,2] + p_3 + q_3 \\ &= 12 + 1 + 1 = 14\end{aligned}$$

$$\begin{aligned}\omega[1,4] &= \omega[1,3] + p_4 + q_4 \\ &= 9 + 1 + 1 = 11\end{aligned}$$

$l=5:$

$$\begin{aligned}\omega[0,4] &= \omega[0,3] + p_4 + q_4 \\ &= 14 + 1 + 1 = 16\end{aligned}$$



## Calculation of C & r

$l = 0, 1:$

$$c[i, i] = r[i, i] = 0$$

So,

$$\text{all } c[0,0], c[1,1] \dots c[4,4] = 0$$

$$r[0,0] \dots r[4,4] = 0$$

Formula:

$$c[i,j] = \min_{i < k \leq j} \left\{ c[i, k-1] + c[k, j] \right\} + w[i, j]$$

$d=2$ :  $\{ c[0,1], c[1,2], c[2,3], c[3,4] \}$

$$\begin{aligned}
 1. \quad c[0,1] &= \min_{k=1} \{ c[i, k-1] + c[k, j] \} + w[i, j] \\
 &= \min \{ c[0,0] + c[1,1] \} + w[0,1] \\
 &= \min \{ 0 + 0 \} + 8 \Rightarrow \boxed{c[0,1] = 8}
 \end{aligned}$$

$c[0,1] = 8$  obtained for  $k=1$ . So this  $k$  value needs to be recorded as corresponding root value.

$$r[0,1] = k = 1$$

$c[0,1] = 8$	$r[0,1] = 1$
--------------	--------------



2.  $c[1,2]$  :

$$c[1,2] = \min_{k=2} \left\{ c[i, k-1] + c[k, j] \right\} + w[i, j]$$

$$c[1,2] \xrightarrow{k=2} c[1,1] + c[2,2] + w[1,2]$$

` 0        +    0        + 7

$$\boxed{c[1,2] = 7} \quad \text{for } k=2, \text{ so } \boxed{r[1,2] = 2}$$

3.  $C[2,3]$ :

$$C[2,3] \xrightarrow{k=3} C[2,2] + C[3,3] + W[2,3]$$

$$0 + 0 + 3$$

$$\boxed{C[2,3] = 3} \quad \text{for } k=3, \boxed{W[2,3] = 3}$$

4.  $C[3,4]$

$$C[3,4] \xrightarrow{k=4} C[3,3] + C[4,4] + W[3,4]$$

$$0 + 0 + 3$$

$$\boxed{C[3,4] = 3} \quad \text{for } k=4, \boxed{W[3,4] = 3}$$

length = 3:  $\{c[0,2], c[1,3], c[2,4]\}$

$$\begin{array}{ccc}
 c[0,2] & \xrightarrow{\kappa=1} & c[0,0] + c[1,2] + w[0,2] \\
 & \min & 0 + 7 + 12 \Rightarrow 19
 \end{array}$$

$$\begin{array}{ccc}
 & \xrightarrow{\kappa=2} & c[0,1] + c[2,2] + w[0,2] \\
 & & 8 + 0 + 12 \Rightarrow 20
 \end{array}$$

$$\boxed{c[0,2] = 19} \text{ for } \kappa=1, \boxed{\gamma[0,2] = 1}$$

$$\begin{array}{ccc}
 c[1,3] & \xrightarrow{\kappa=2} & c[1,1] + c[2,3] + w[1,3] \\
 & \min & 0 + 3 + 9 = 12
 \end{array}$$

$$\begin{array}{ccc}
 & \xrightarrow{\kappa=3} & c[1,2] + c[3,3] + w[1,3] \\
 & & 7 + 0 + 9 = 16
 \end{array}$$

$$\boxed{c[1,3] = 12} \text{ for } \kappa=2, \boxed{\gamma[1,3] = 2}$$



$$C[2,4] \xrightarrow{\substack{K=3 \\ \min}} C[2,2] + C[3,4] + w[2,4] = \begin{array}{c} \min \\ (8) \end{array}$$

$$\quad \quad \quad 0 + 3 + 5$$

$$C[2,4] \xrightarrow{\substack{K=4 \\ \min}} C[2,3] + C[4,4] + w[2,4]$$

$$\quad \quad \quad 3 + 0 + 5 = 8$$

$$C[2,4] = 8 \quad \text{for } k=3 \Rightarrow R[2,4] = 3$$

length=4:  $\{ C[0,3], C[1,4] \}$

$$C[0,3] = 25 \quad \text{für } k=2 \Rightarrow r[0,3] = 2$$

$$\begin{aligned}
 C[1,4] &\xrightarrow{k=2} C[1,1] + C[2,4] + w[1,4] = \min \quad 19 \\
 &\xrightarrow{k=3} C[1,2] + C[3,4] + w[1,4] = 21 \\
 &\xrightarrow{k=4} C[1,3] + C[4,4] + w[1,4] \\
 &12 + 0 + 11 = 23 \\
 \boxed{C[1,4] = 19} \text{ for } k=2 &\Rightarrow \boxed{r[1,4] = 2}
 \end{aligned}$$

length = 5:  $\{ C[0,4] \}$

$$\begin{aligned}
 C[0,4] &\xrightarrow{k=1} C[0,0] + C[1,4] + w[0,4] = 35 \\
 &\xrightarrow{k=2} C[0,1] + C[2,4] + w[0,4] = \min \quad 32 \\
 &\xrightarrow{k=3} C[0,2] + C[3,4] + w[0,4] = 38 \\
 &\xrightarrow{k=4} C[0,3] + C[4,4] + w[0,4] \\
 &25 + 0 + 16 = 41
 \end{aligned}$$

$$\boxed{C[0,4] = 32} \text{ for } k=2 \Rightarrow \boxed{r[0,4] = 2}$$



Cost of BST:

$$c[0,4] = 32$$

Since the probability is taken as integer, the cost needs to be divided by total probability. Here the total probability is 16.

$$\text{So the cost } c[0,4] = 32/16 = 2$$

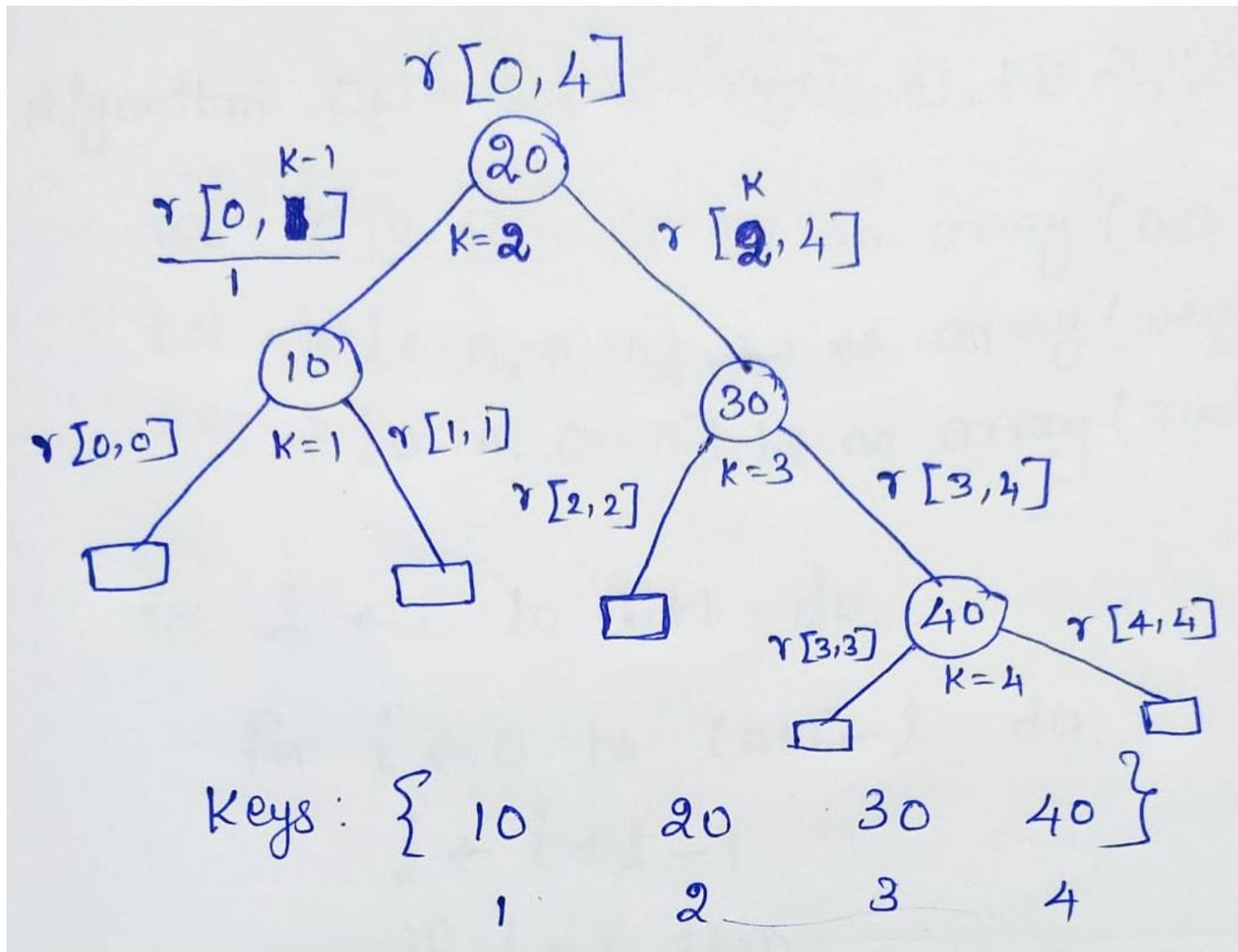
Minimum Cost of BST :  $c[0,4] = 2$



For Obtaining the BST, we  
need to consider the Root Table.

$$\tau[0,4] = 2$$

i.e., 2nd key is the root of BST.

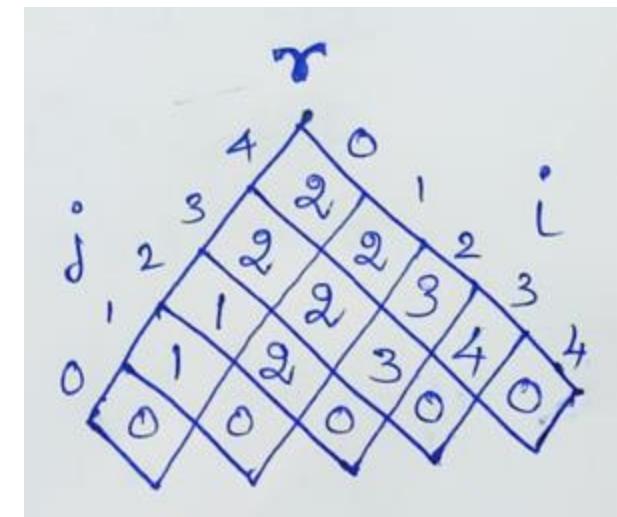


Keys : { 10 }

20

30

40





Algorithm Optimal BST ( $\text{keys}[1..n]$ ,  $P[1..n]$ ,  $Q[0..n]$ )

let  $C[0..n][0..n]$  be an array (cost)

Let  $W[0..n, 0..n]$  be an array (weight)

Let  $R[0..n, 0..n]$  be an array (root)

for  $l \leftarrow 1$  to  $n+1$  do

    for  $i \leftarrow 0$  to  $(n+1)-l$  do

$j \leftarrow i+l-1$

        if  $i=j$  then

$W[i,j] \leftarrow Q[i]$

        else

$W[i,j] \leftarrow W[i,j-1] + P[j]$   
             $+ Q[j]$

        end if

    end for

end for



```
For l ← 1 to (n+1) do
    for i ← 0 to (n+1)-l do
        j ← i + l - 1
        if i = j then
            c[i,j] ← r[i,j] ← 0
        else
            min ← ∞
            mink ← -1
            for k ← i+1 to j do
                sum ← c[i,k-1] + c[k,j] +
                    w[i,j];
                if sum < min then
                    min ← sum
                    mink ← k
                End if
            end for
            c[i,j] ← min
            r[i,j] ← mink
        end if
    end for
end for.
```

# **Dynamic Programming Approach**

## **String Editing Problem**



## String Editing Problem:

### Problem:

1. Given 2 Strings

$$X = x_1, x_2, \dots, x_n$$

$$Y = y_1, y_2, \dots, y_m$$

where  $x_i, 1 \leq i \leq n$  and

$y_i, 1 \leq i \leq m$ , are members

& Finite set of symbols known  
as alphabet.



2. The problem is transforming  $X$  into  $Y$  using a sequence of edit operations.
3. The permissible edit operations are insert, delete and replace.
4. The problem of string editing is to identify a minimum-cost sequence of edit operations that will transform  $X$  into  $Y$ .

# Dynamic Programming Approach

## (For String Editing Problem)

Example:

$$X = adceg \quad n=5$$

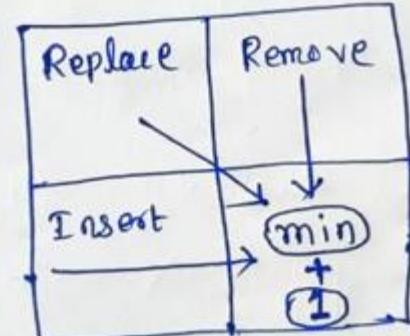
$$Y = abc Fg \quad m=5$$

Symbols:

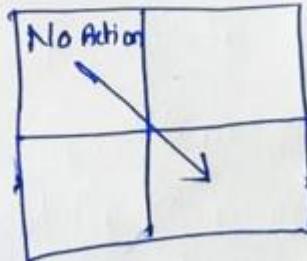
- Insert
- ↓ Remove
- Replace

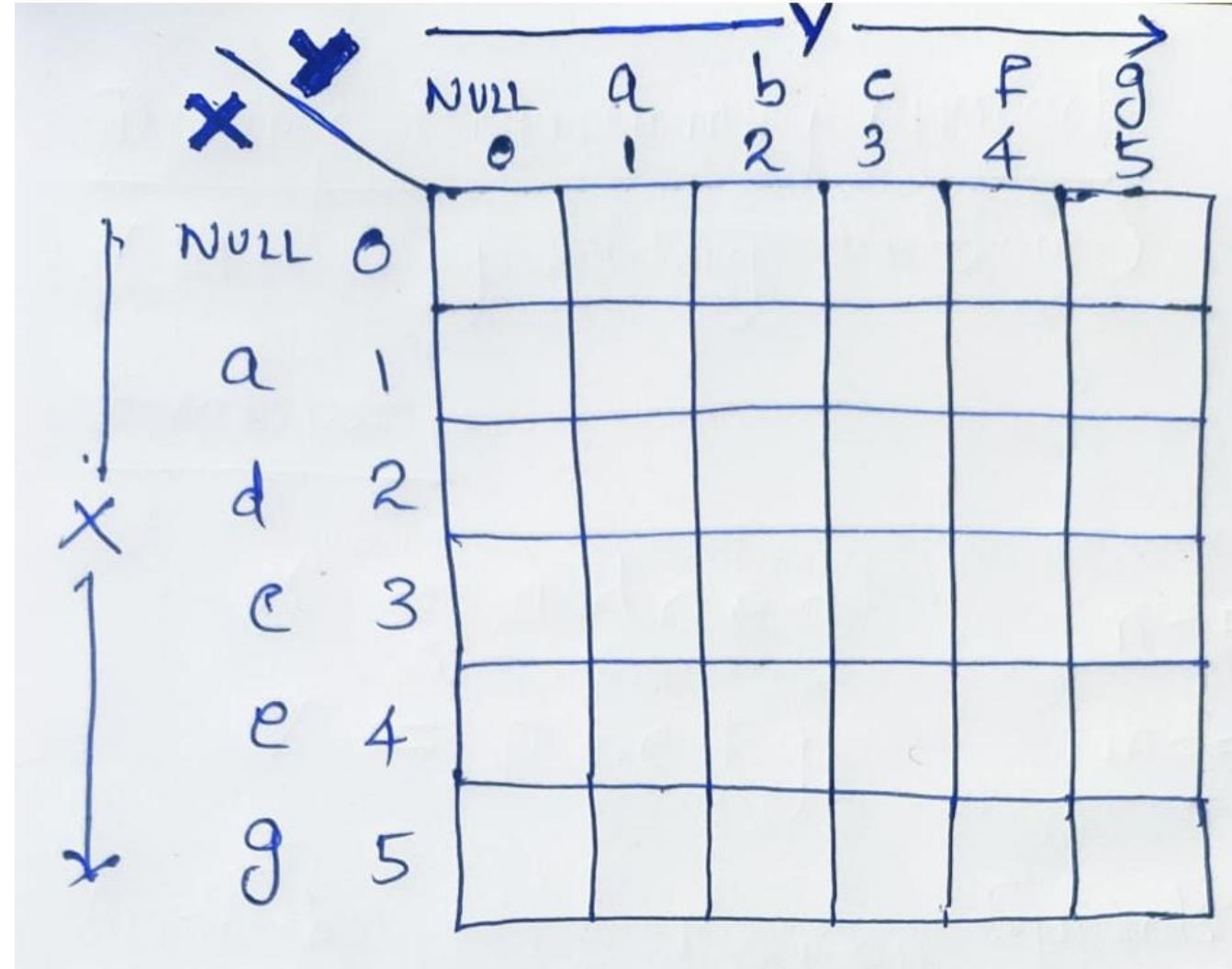
Approach:

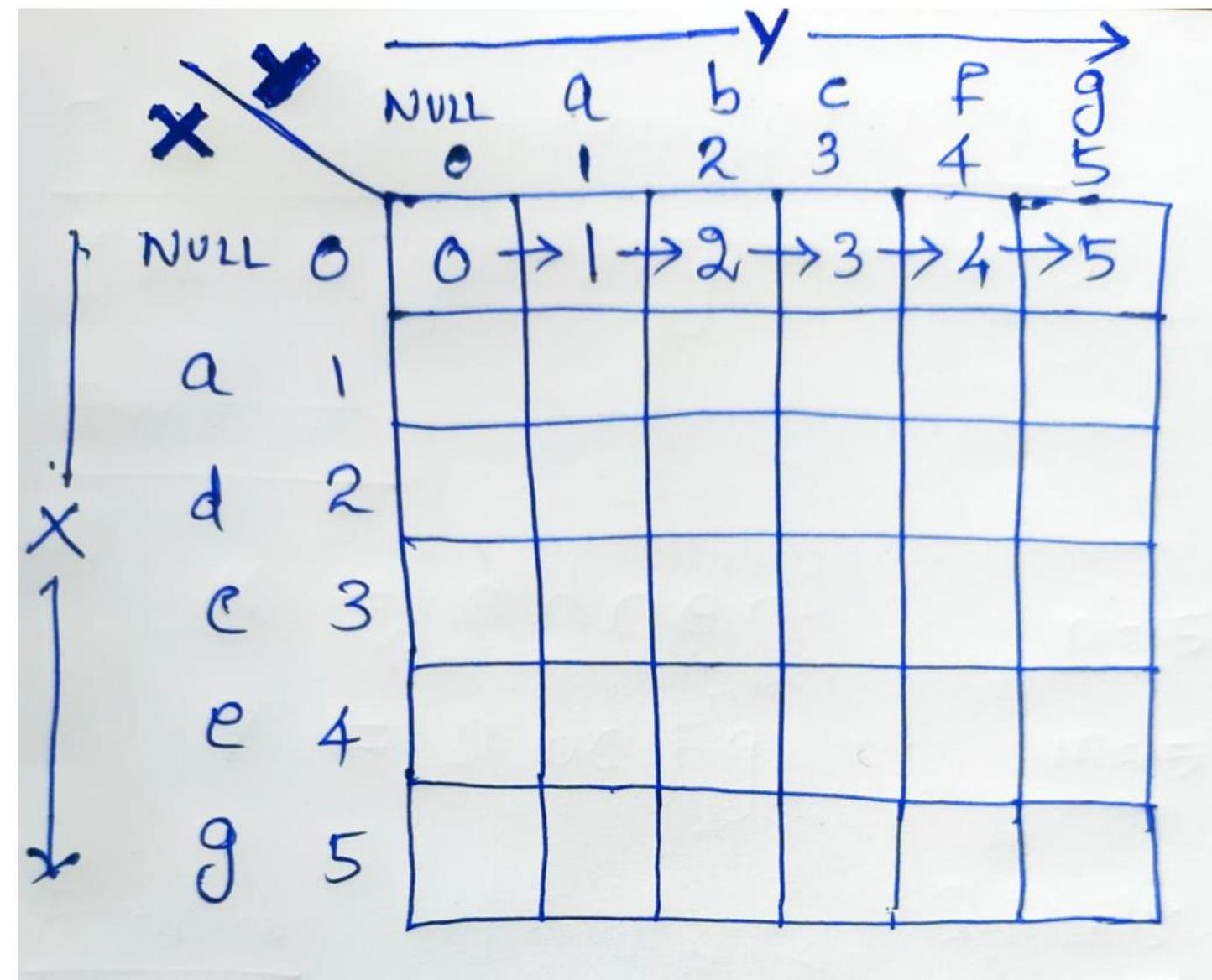
1. If  $\gamma \neq c$

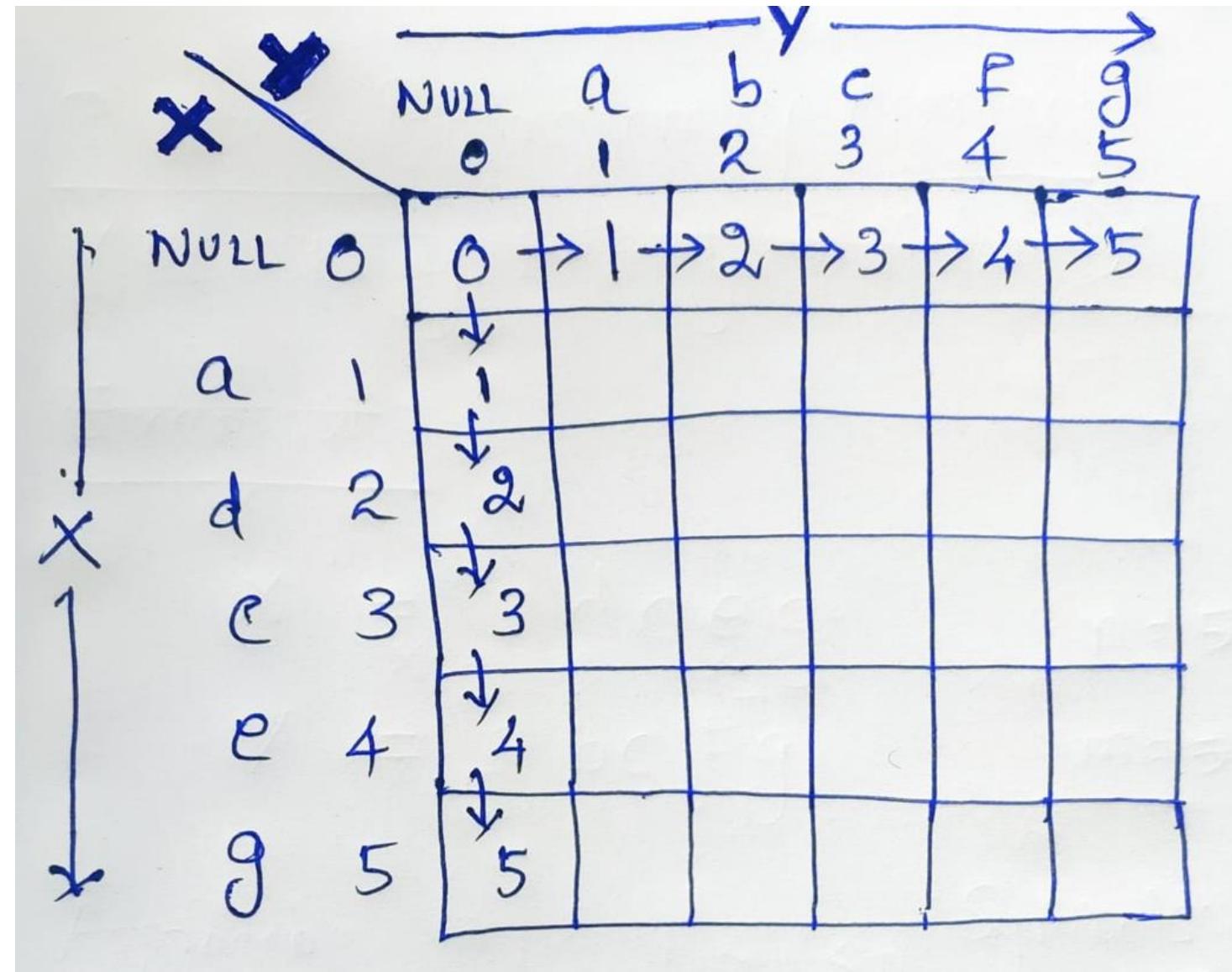


2. If  $\gamma = c$









$m[1,1]$ :  $\tau = 'a'$   $c = 'a'$

2.  $\tau = c$ : No Action Required.

$m[1,2]$ :  $\tau = 'a'$   $c = 'b'$

1.  $\tau \neq c \Rightarrow \min(Replace, Delete, Insert)$

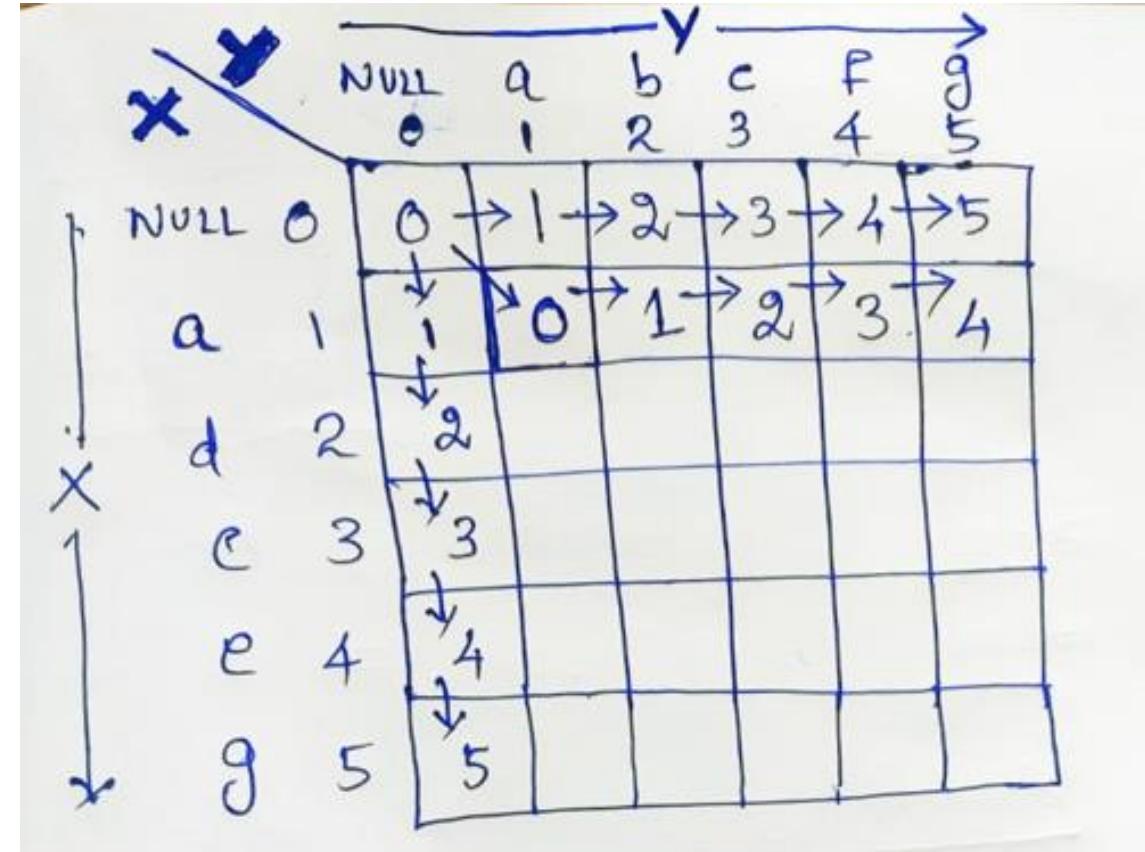
$+1$	$1$	$2$	$0$
------	-----	-----	-----

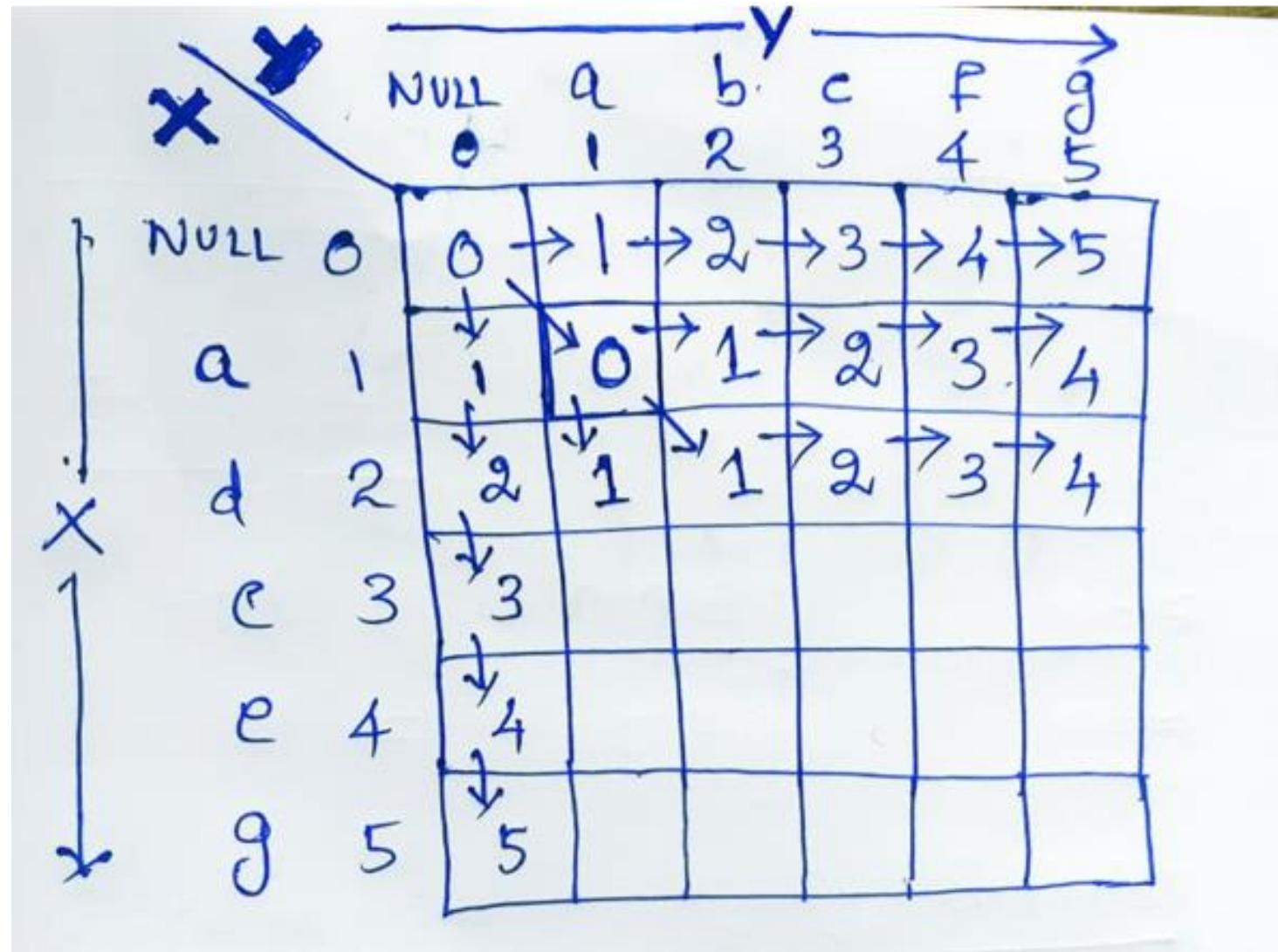
$$\Rightarrow 0 + 1 = 1 \\ (\text{Insert})$$

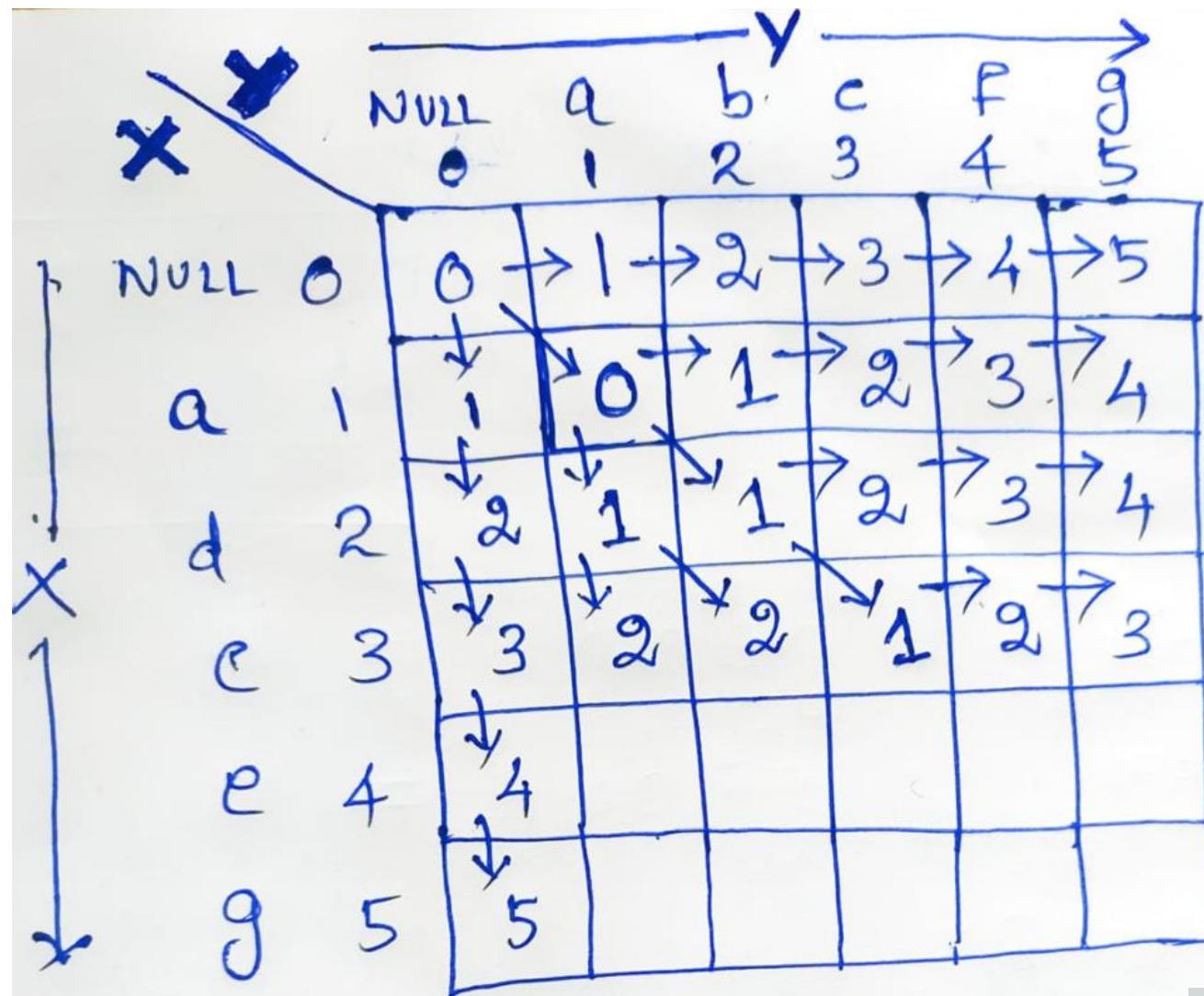
$m[1,3]$ :  $\tau = 'a'$   $c = 'c'$

$\tau \neq c$ :  $\min(I, R, D) + 1$

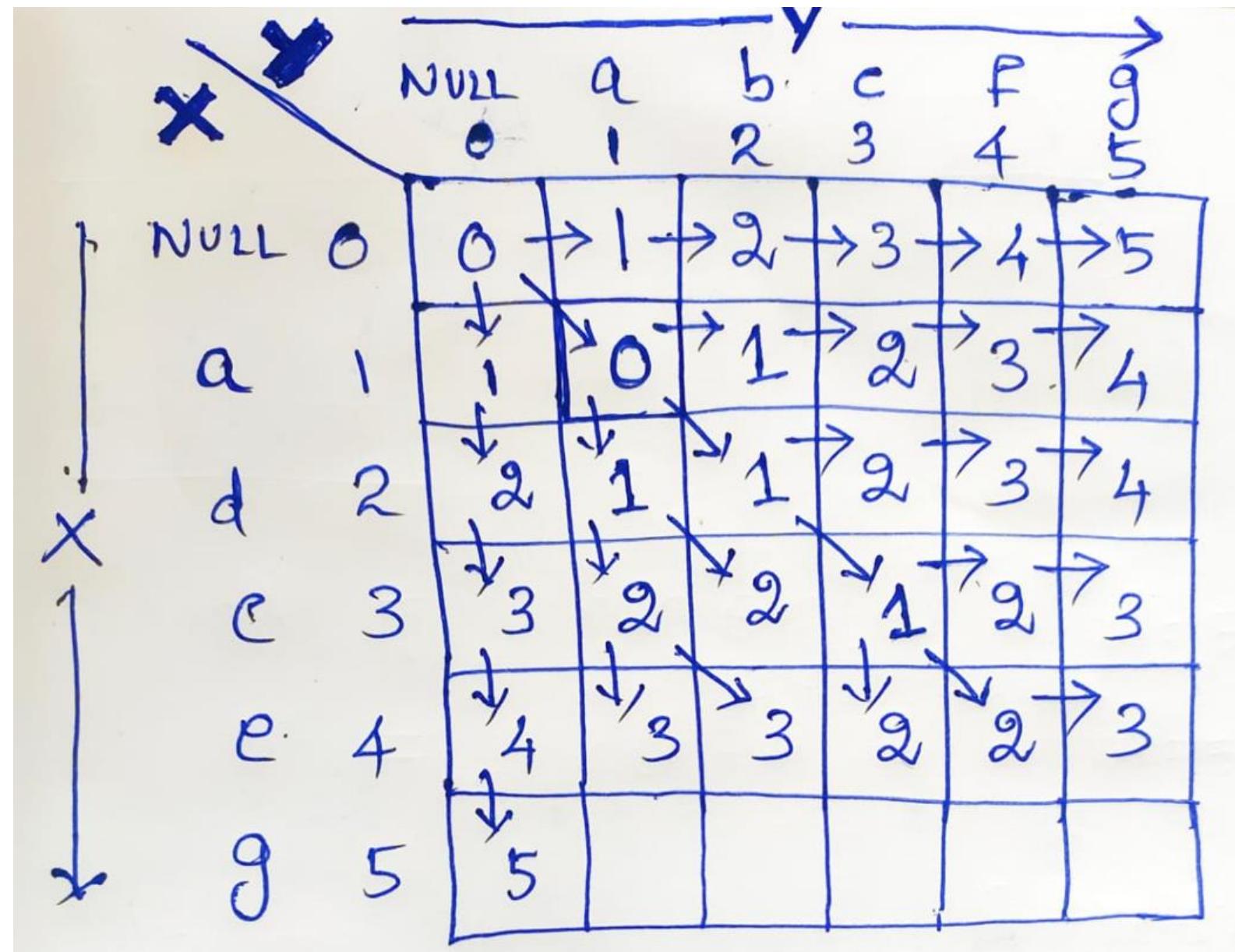
$$\frac{1}{(I)} + 1 \Rightarrow 2$$

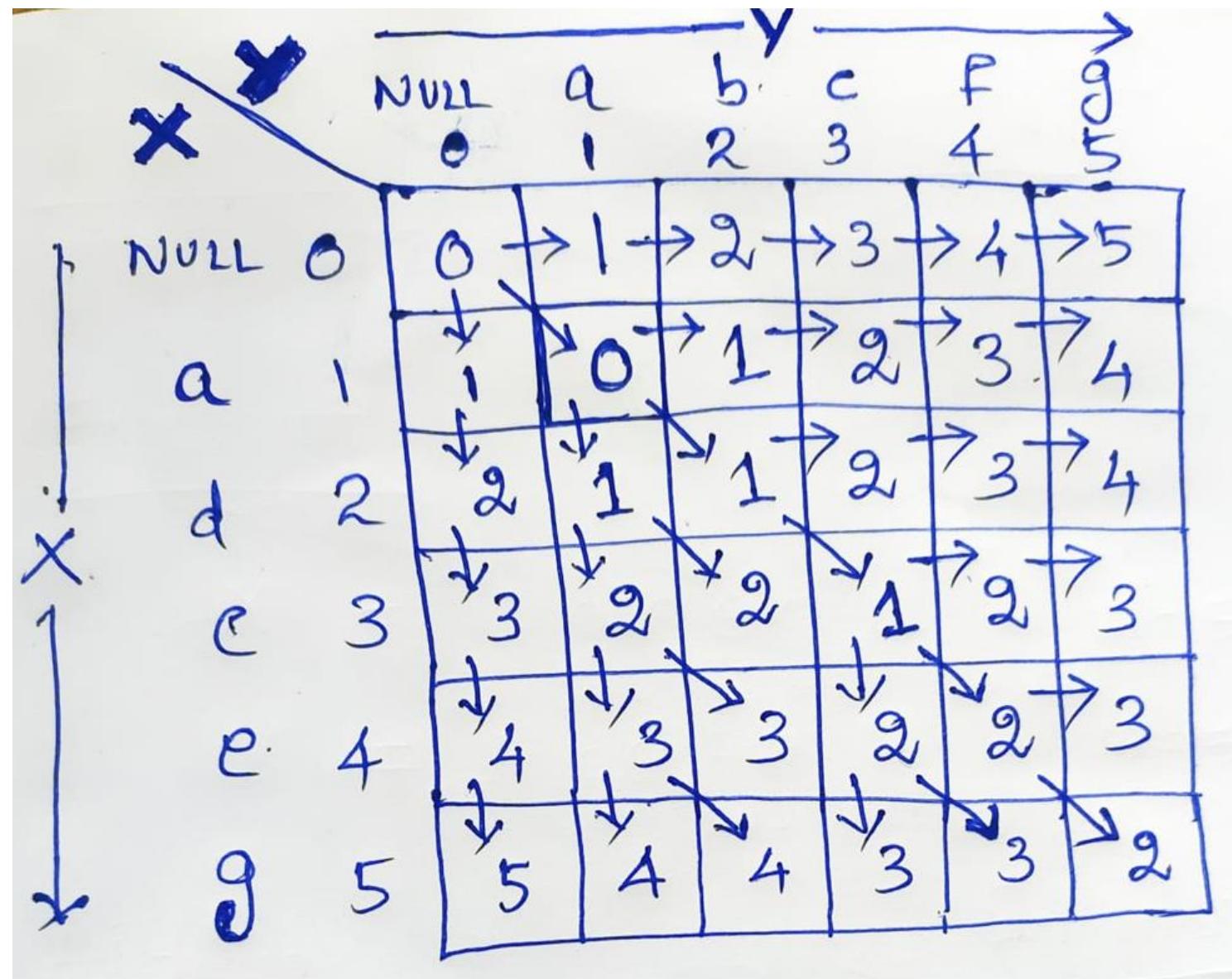






$m[3,3]$  :  $\tau = 'c'$      $e = 'c'$   
 $\tau = c$  : No Action Required.







$$\boxed{\text{Minimum No. of Edit Operations Required}} = m[5,5] = 2$$

Operations Required:

1. Replace 'e' with 'f'
2. Replace 'd' with 'b'

# **Dynamic Programming Approach**

## **0/1 Knapsack Problem**



## 0/1 Knapsack Problem:

### Dynamic Programming Approach:

#### Problem:

Given: 1. Items with Profit & weight.  
2. A bag (knapsack) capable of carrying some amount of weight say,  $x$  kg.

#### Objective:

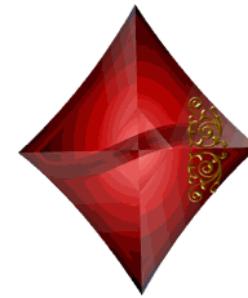
Need to carry maximum no. of items with maximum profit in a bag. without exceeding its capacity.



**Wt. = 5  
Value = 10**



**Wt. = 3  
Value = 20**



**Wt. = 8  
Value = 25**



**Wt. = 4  
Value = 8**



→ **Maximum wt. = 13**



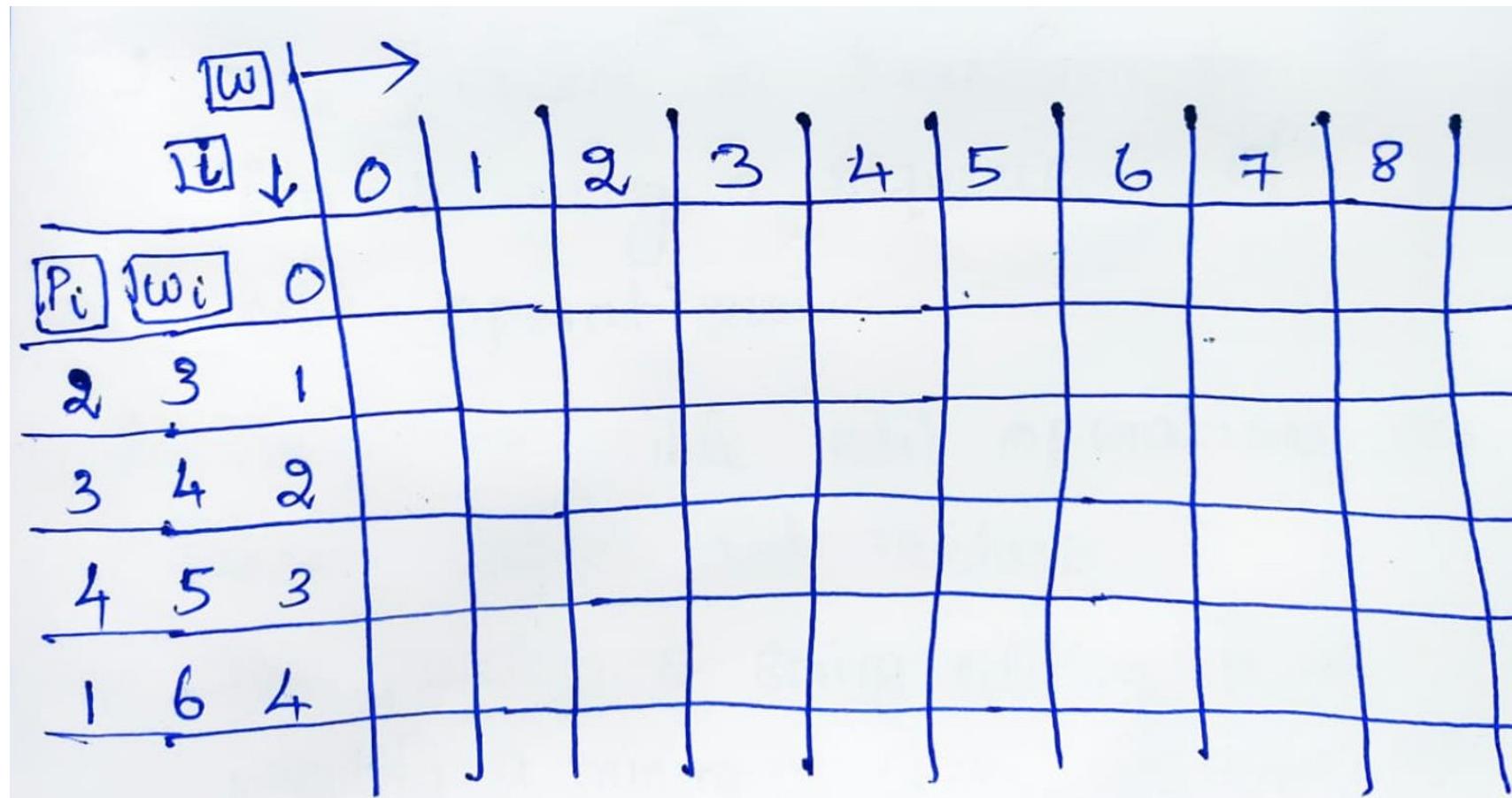
Example:

weights : { 3, 4, 6, 5 }

Profits : { 2, 3, 1, 4 }

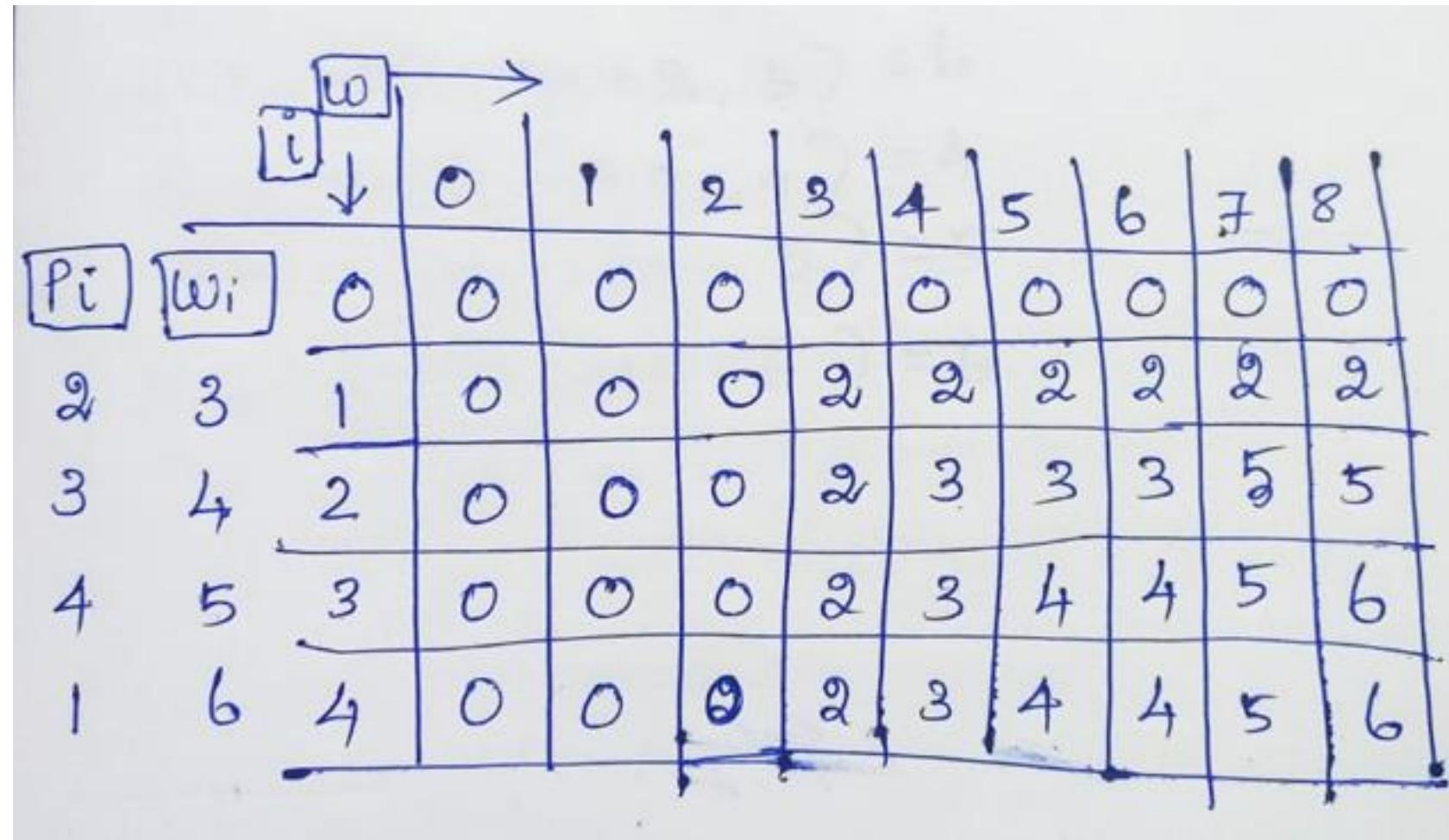
w : 8 kg (Bag capacity)

n = 4



$w_i$	0	1	2	3	4	5	6	7	8
$P_i$	$w_i$	0	0	0	0	0	0	0	0
2	3	1	0						
3	4	2	0						
4	5	3	0						
1	6	4	0						

$P_i$	$w_i$	0	1	2	3	4	5	6	7	8
2	3	1	0	0	0	2	2	2	2	2
3	4	2	0	0	0	2	3	3	3	5
4	5	3	0	0	0	2	3	4	4	5
1	6	4	0	0	0	2	3	4	4	6



$$i, w \quad \max(P_i + m[i-1, w] - w_i, m[i-1, w])$$



$$2,4 \quad \max(3+0, 2) = 3$$

$$2,5 \quad \max(3+0, 2) = 3$$

$$2,6 \quad \max(3+0, 2) = 3$$

$$2,7 \quad \max(3+2, 2) = 5$$

$$2,8 \quad \max(3+2, 2) = 5$$

$$3,5 \quad \max(4+0, 3) = 4$$

$$3,6 \quad \max(4+0, 3) = 4$$

$$3,7 \quad \max(4+0, 5) = 5$$

$$(3,8) \quad \max(4+2, 5) = 6$$

$$4,6 \quad \max(1+0, 4) = 4$$

$$4,7 \quad \max(1+0, 5) = 5$$

$$4,8 \quad \max(1+0, 6) = 6$$



Algorithm ZeroOne Knapsack DP( $wt[1..n]$ ,  
 $p[1..n]$ ,  
 $w, n$ )

Let  $m[0..n, 0..w]$  be a matrix to maintain the maximum profit.

For  $w \leftarrow 0$  to  $w$  do

$m[0, w] \leftarrow 0$

end for

for  $i \leftarrow 0$  to  $n$  do

$m[i, 0] \leftarrow 0$

end for



For  $i \leftarrow 1$  to  $n$  do

for  $w \leftarrow 1$  to  $wt[i]-1$  do

$m[i, w] \leftarrow m[i-1, w]$

end for.

for ~~w~~  $w \leftarrow wt[i]$  to  $W$  do

if  $m[i-1, w] > p[i] + m[i-1, W-wt[i]]$   
then

$m[i, w] \leftarrow m[i-1, w]$

else

$m[i, w] \leftarrow p[i] + m[i-1, W-wt[i]]$

end if

end for

end ZeroOne Knapsack DP

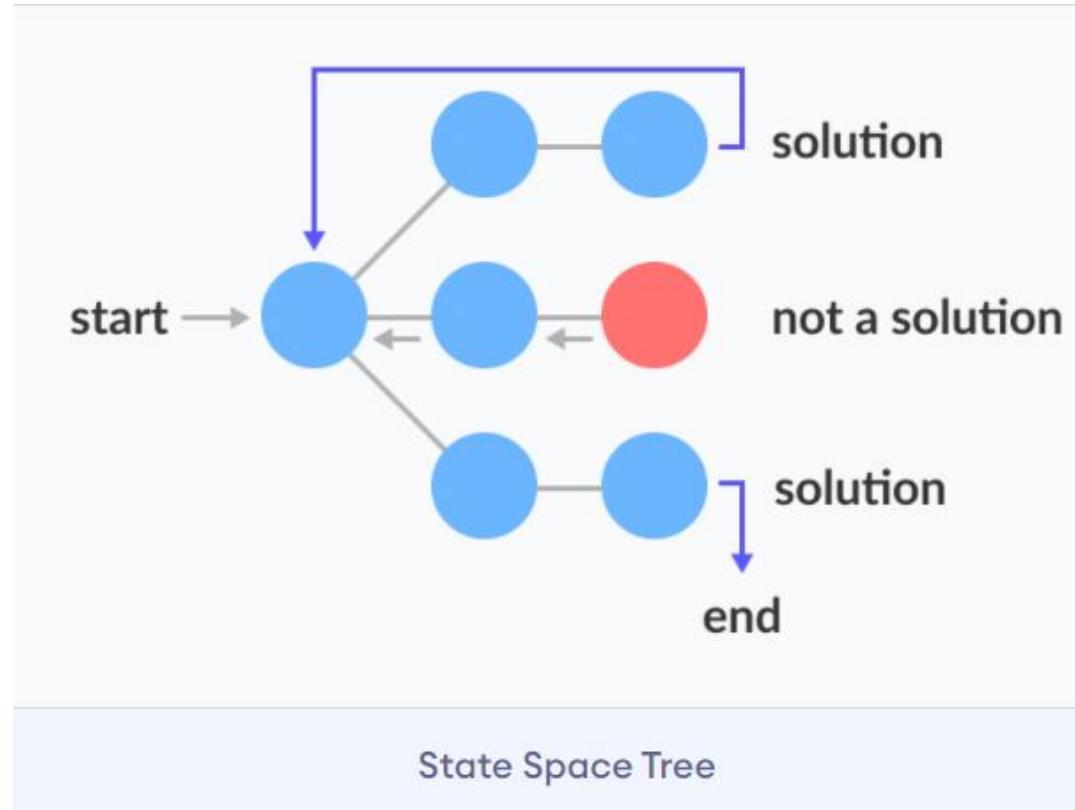
# Backtracking Approach

# Backtracking - Introduction

- ✓ A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output
- ✓ The Brute force approach tries out all the possible solutions and chooses the desired/best solutions.
- ✓ The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions.
- ✓ This approach is used to solve problems that have multiple solutions. If you want an optimal solution, you must go for dynamic programming or greedy.

# State Space Tree - *Representation of the solutions*

- ✓ A state space tree is a tree representing all the possible states (solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state.



# Backtracking – Example Problem

- ✓ **Problem:** You want to find all the possible ways of arranging 2 boys and 1 girl on 3 benches.
- ✓ **Constraint:** Girl should not be on the middle bench.
- ✓ **Solution:**

There are a total of  $3! = 6$  possibilities.

We will try all the possibilities and get the possible solutions.

Need to choose the solutions those satisfying the given conditions

Need to use recursion for generating all solutions.

Let B1 – Boy 1, B2 – Boy 2 and G - Girl

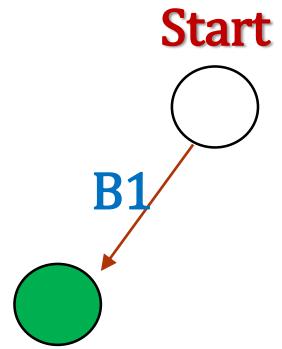
## Step-1

Start



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

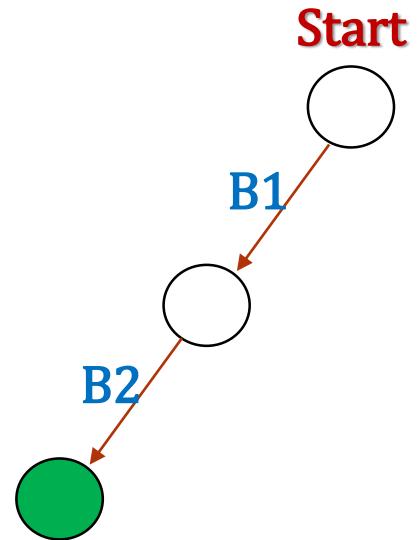
## Step-2



## Step-3



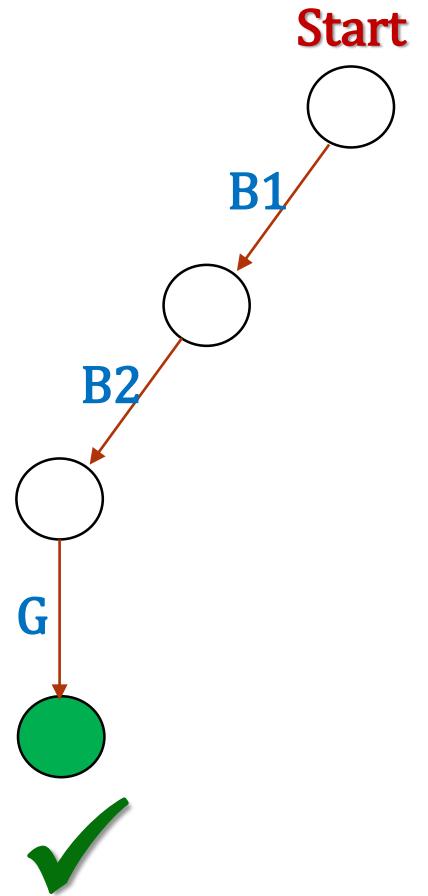
**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



## Step-4

Solutions

B1-B2-G

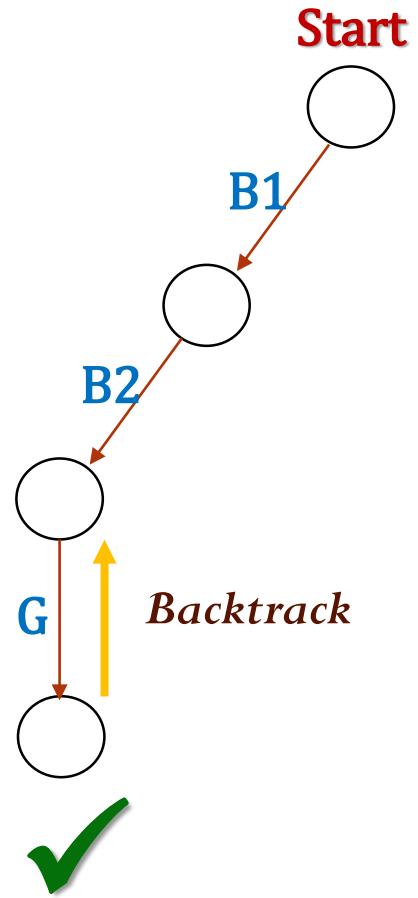


**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

## Step-5

Solutions

B1-B2-G

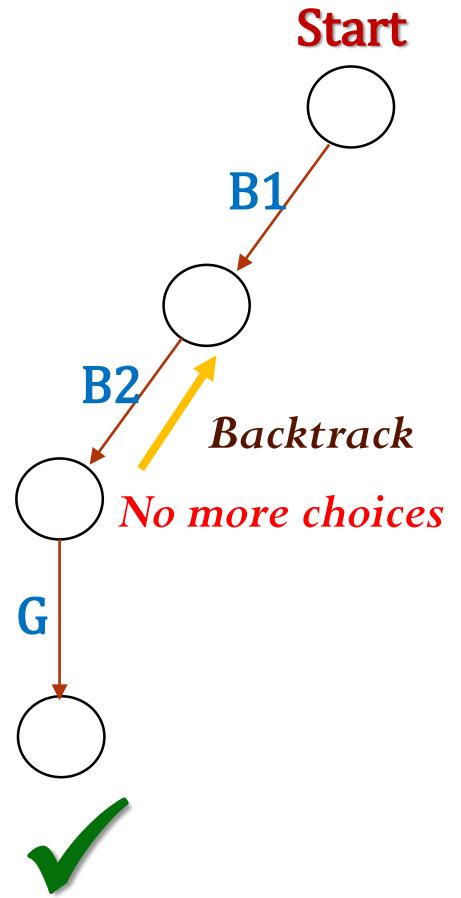


**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

## Step-6

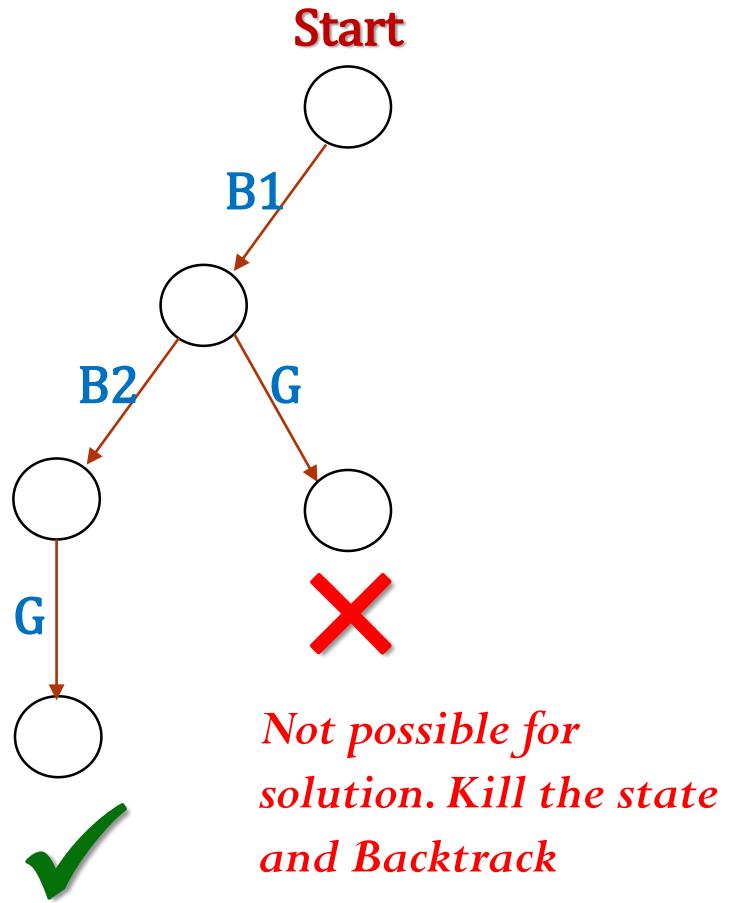
Solutions

B1-B2-G



### Solutions

B1-B2-G



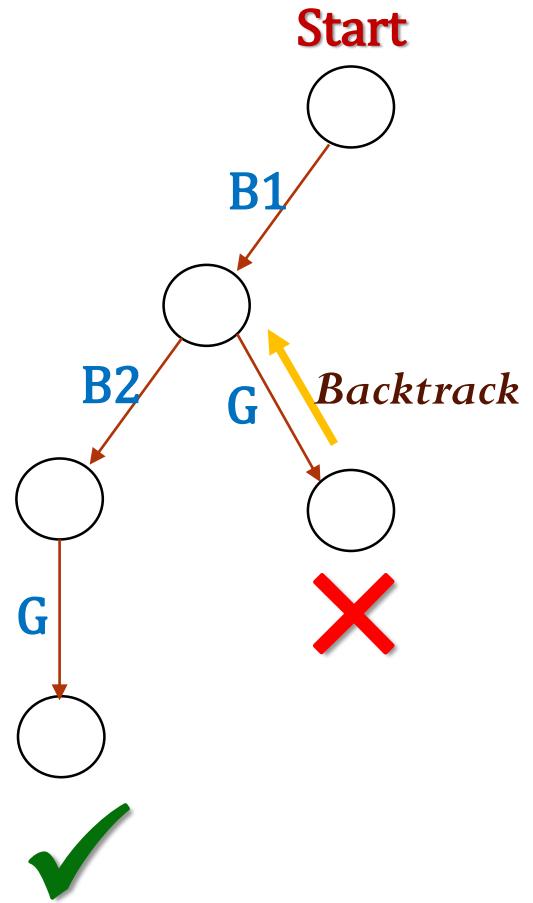
*Not possible for  
solution. Kill the state  
and Backtrack*

**Bounding Condition:** Girl should not be on the middle bench

## Step-8

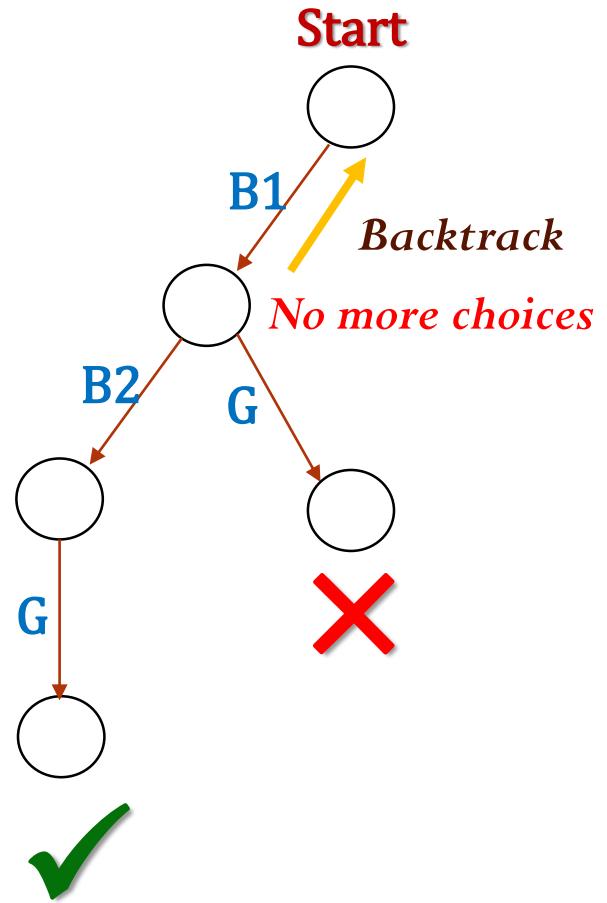
Solutions

B1-B2-G



Solutions

B1-B2-G



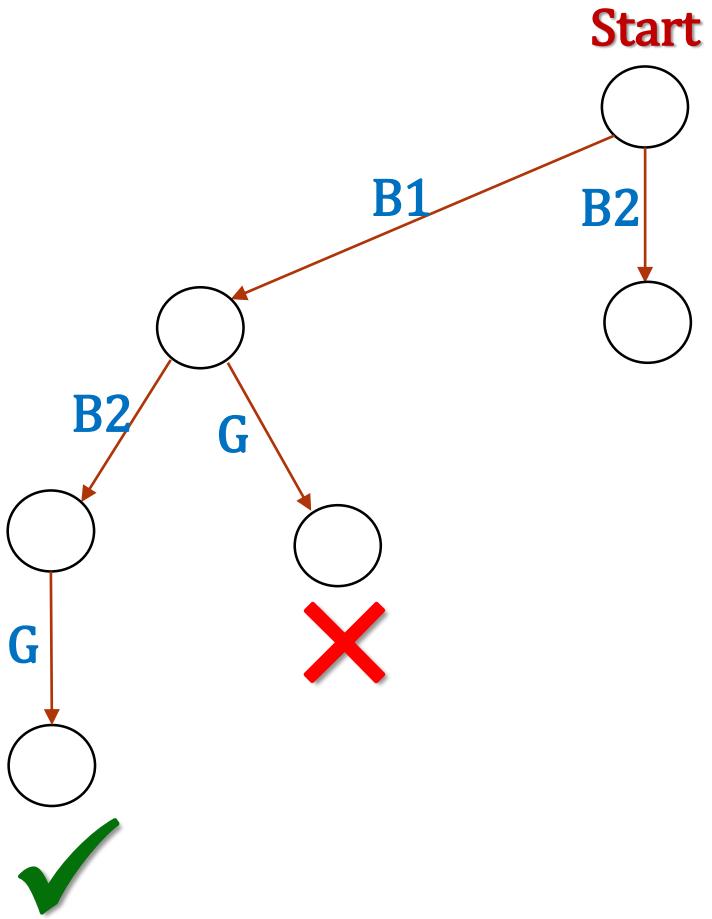
## Step-10



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

### Solutions

B1-B2-G



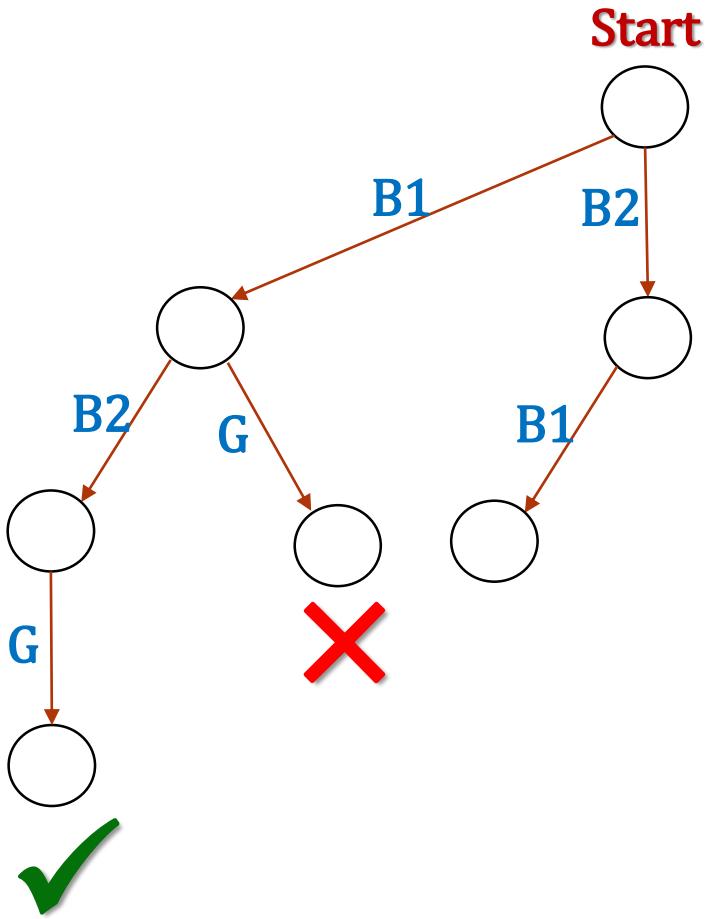
## Step-11



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

### Solutions

B1-B2-G



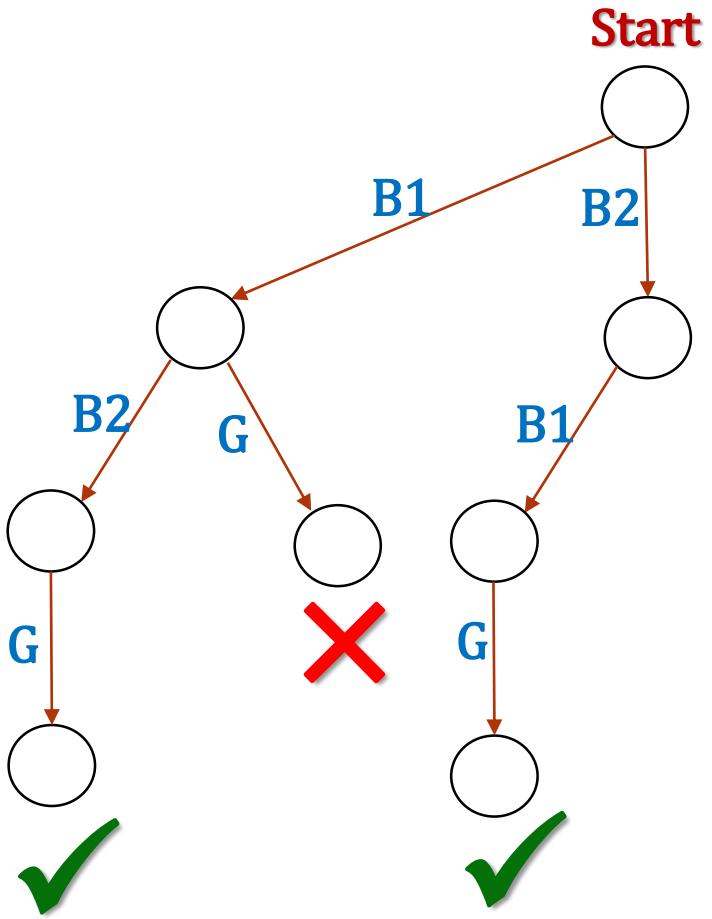
## Step-12



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

### Solutions

B1-B2-G  
B2-B1-G



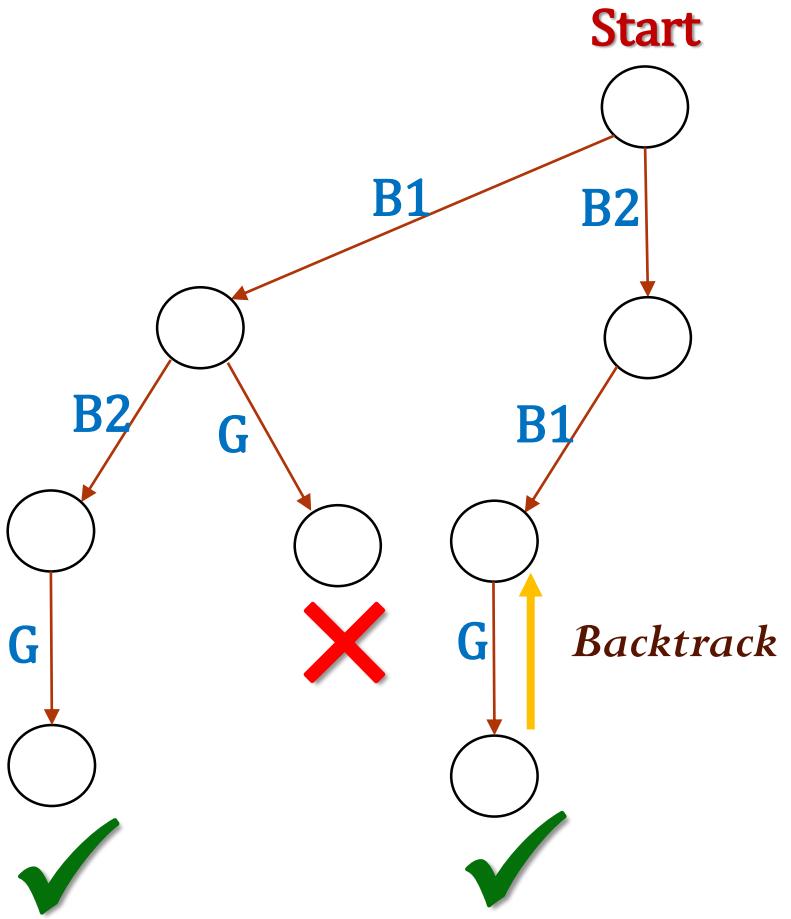
## Step-13



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

### Solutions

B1-B2-G  
B2-B1-G

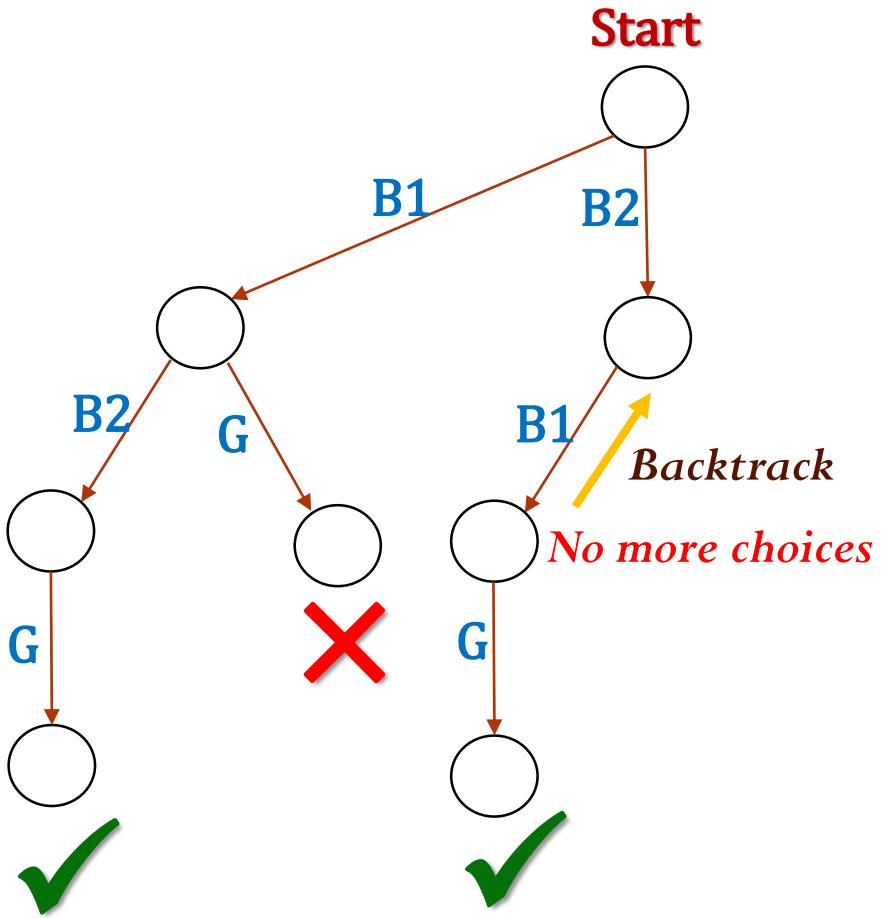


## Step-14



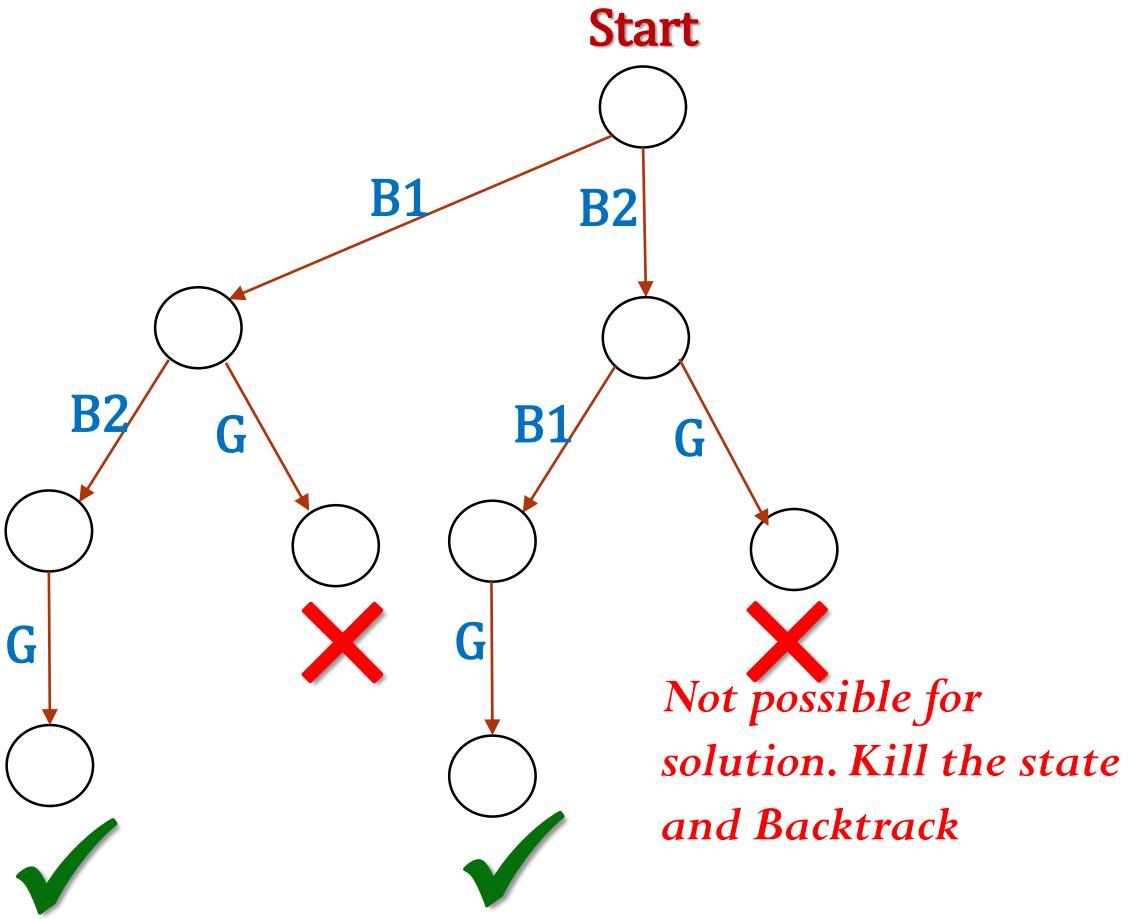
### Solutions

B1-B2-G  
B2-B1-G



### Solutions

B1-B2-G  
B2-B1-G



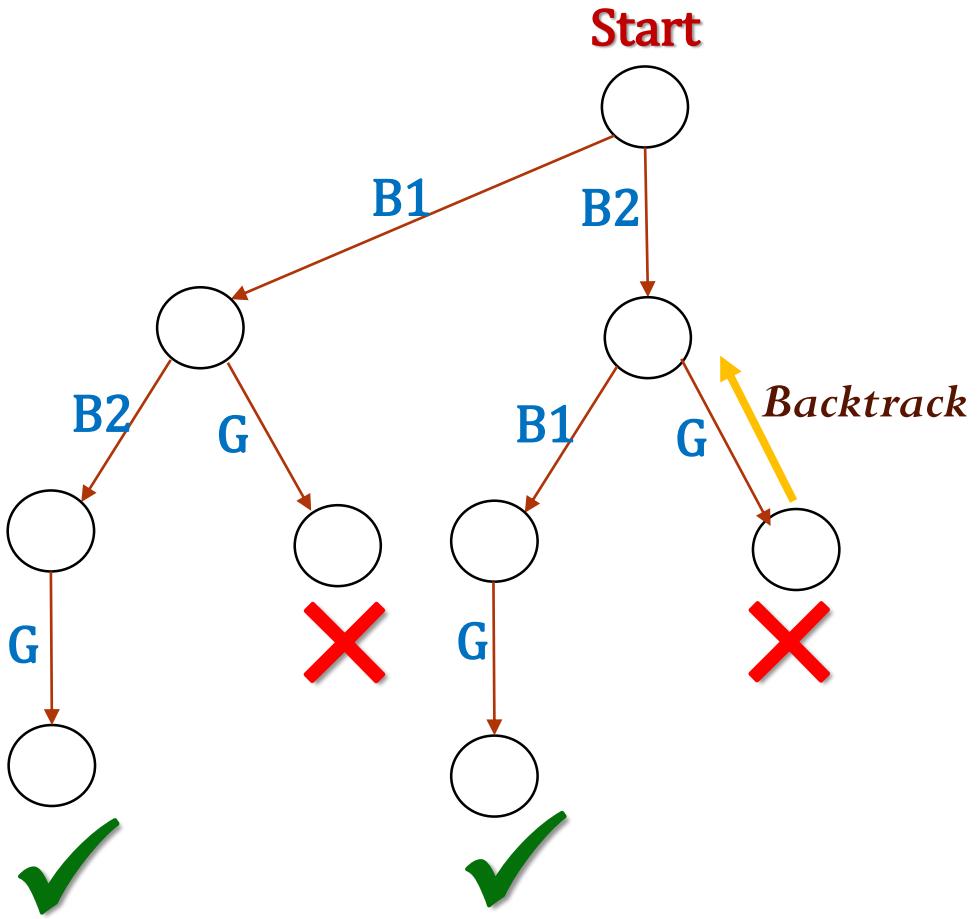
## Step-16



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

### Solutions

B1-B2-G  
B2-B1-G

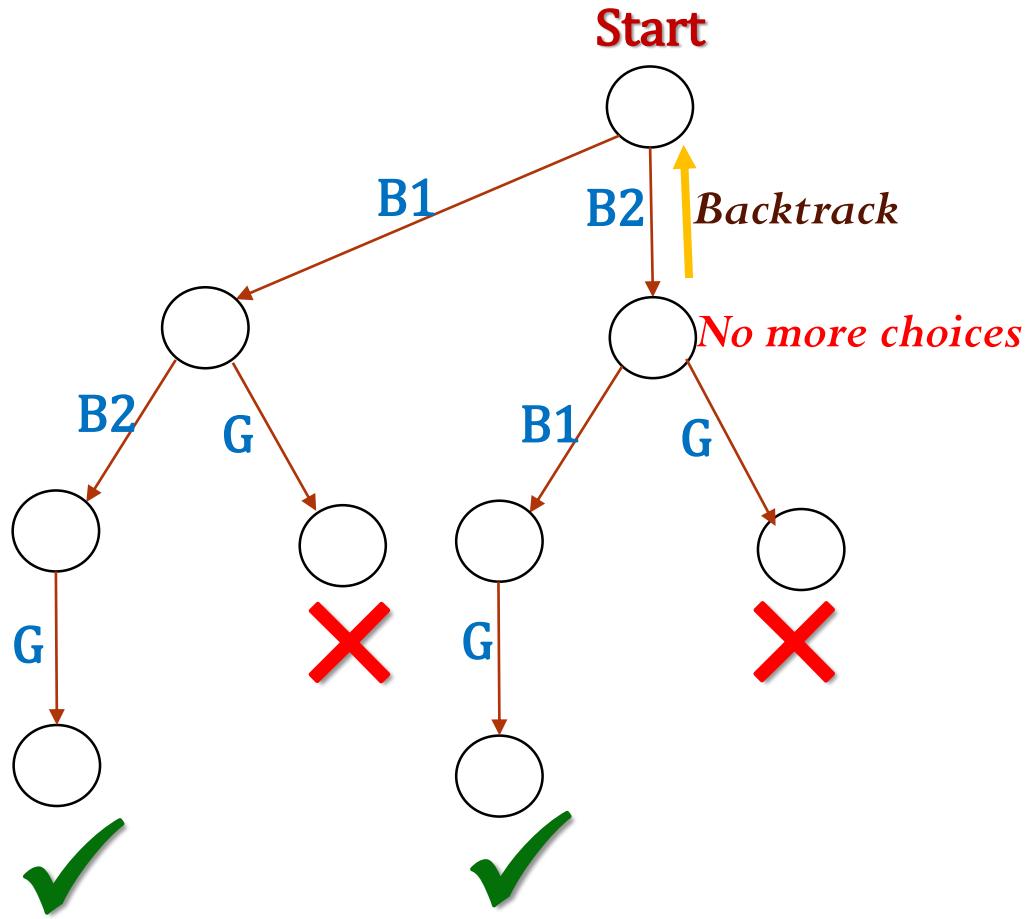


## Step-17



### Solutions

B1-B2-G  
B2-B1-G

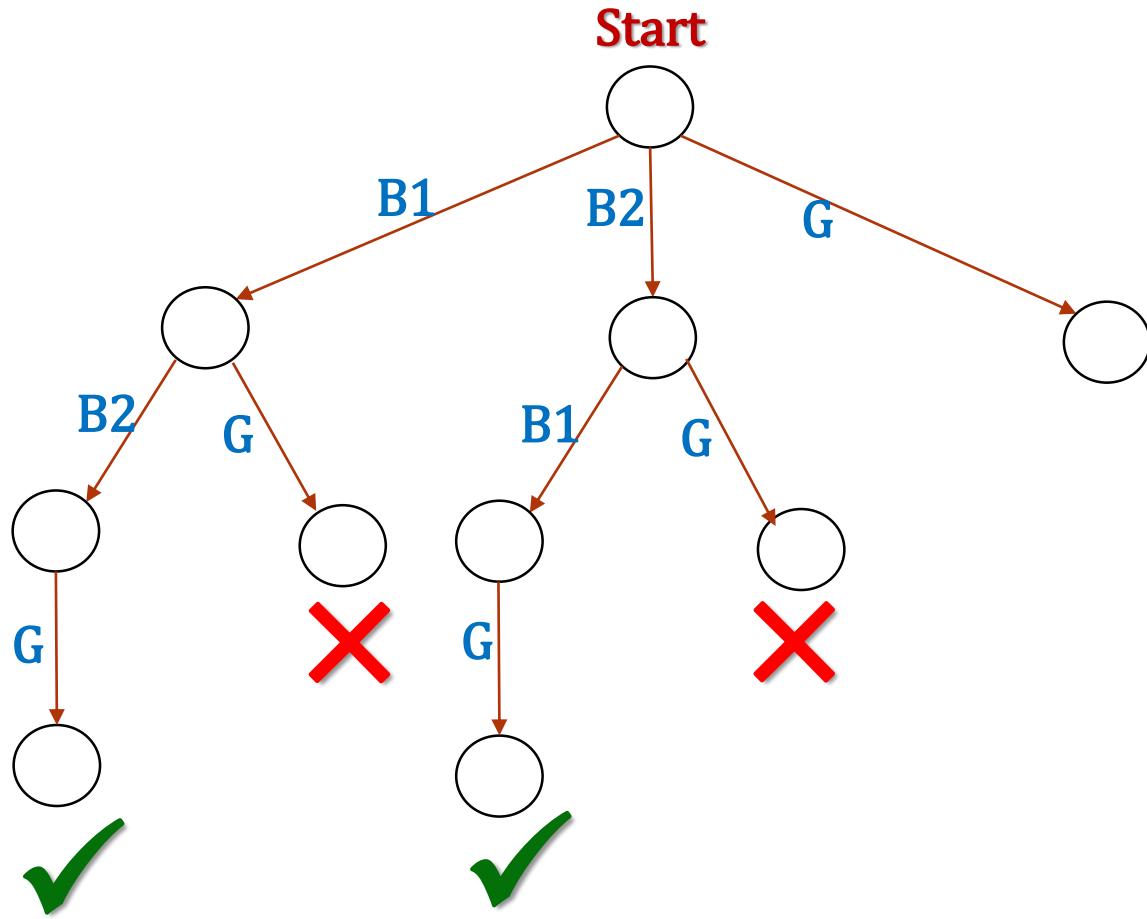


## Step-18



### Solutions

B1-B2-G  
B2-B1-G



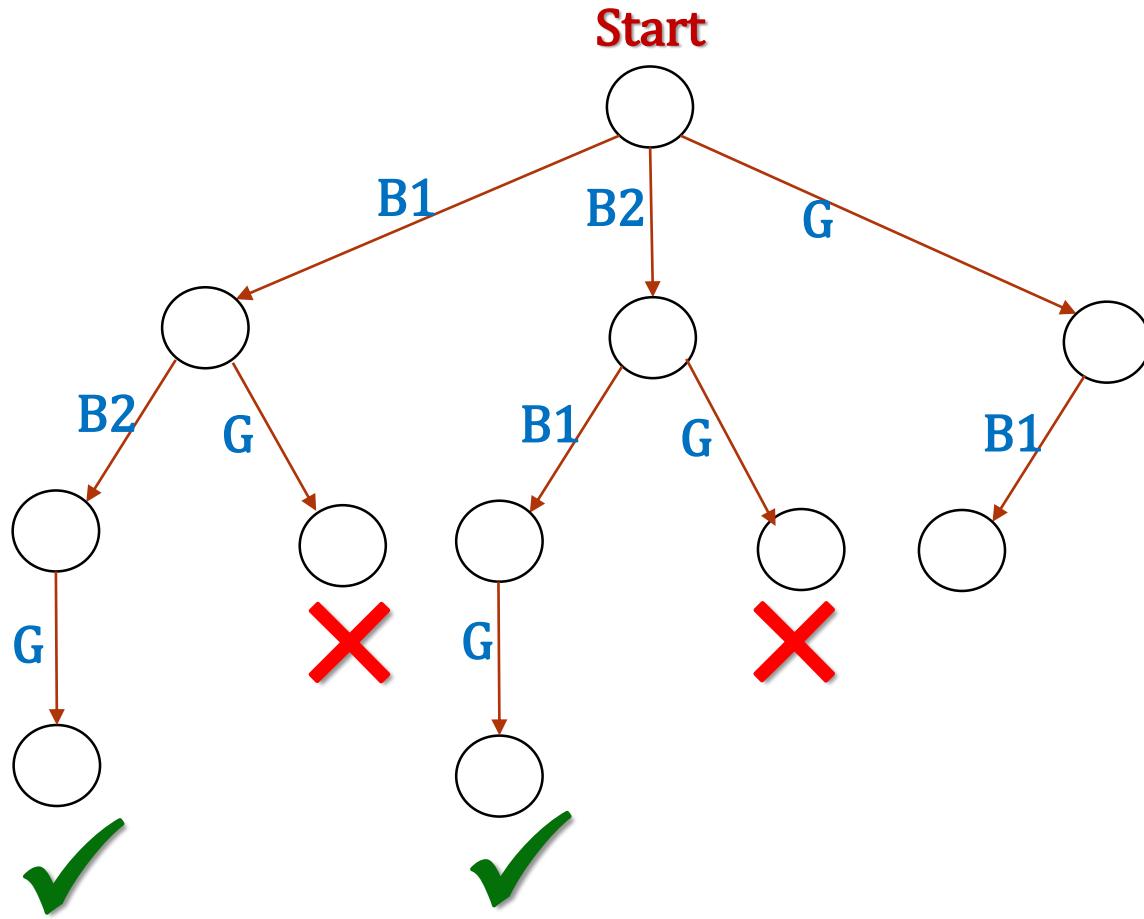
## Step-19



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

### Solutions

B1-B2-G  
B2-B1-G

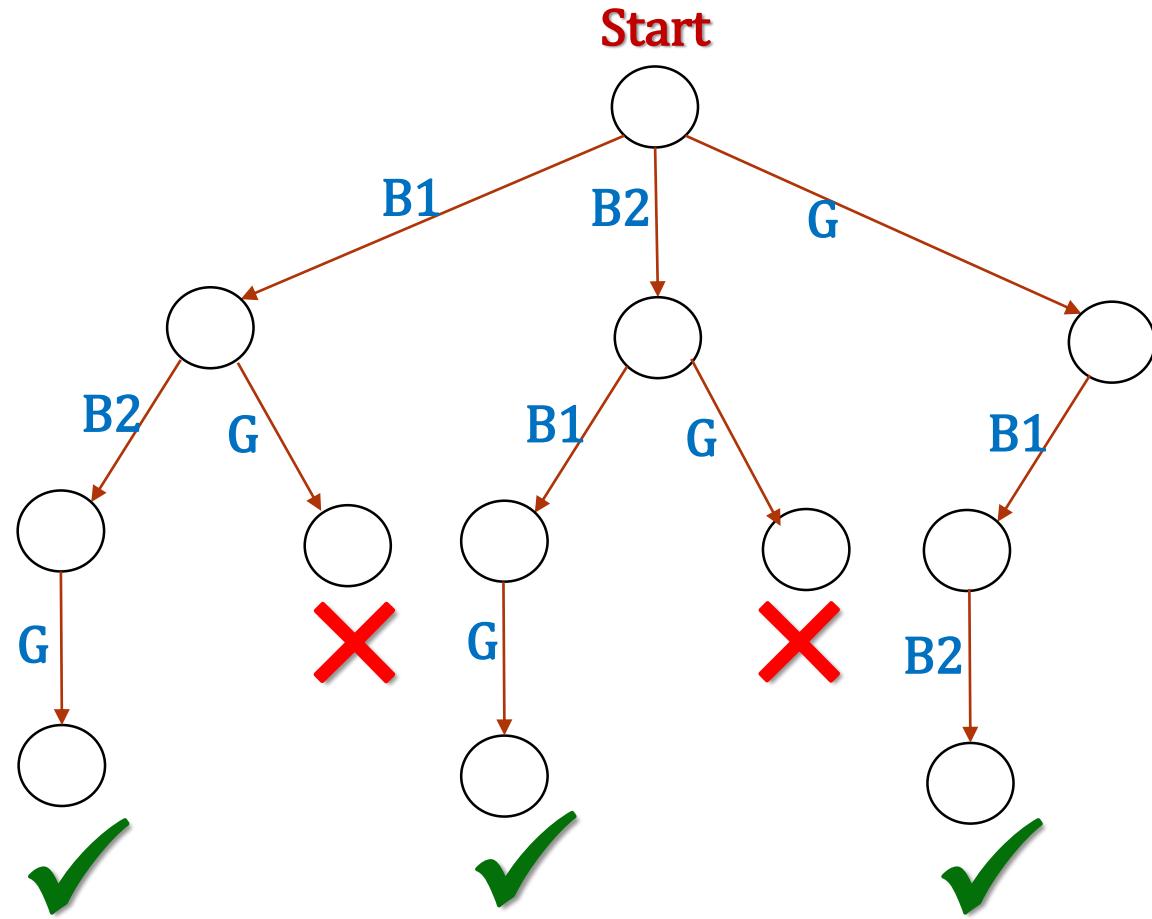


## Step-20



### Solutions

B1-B2-G  
B2-B1-G  
G-B1-B2



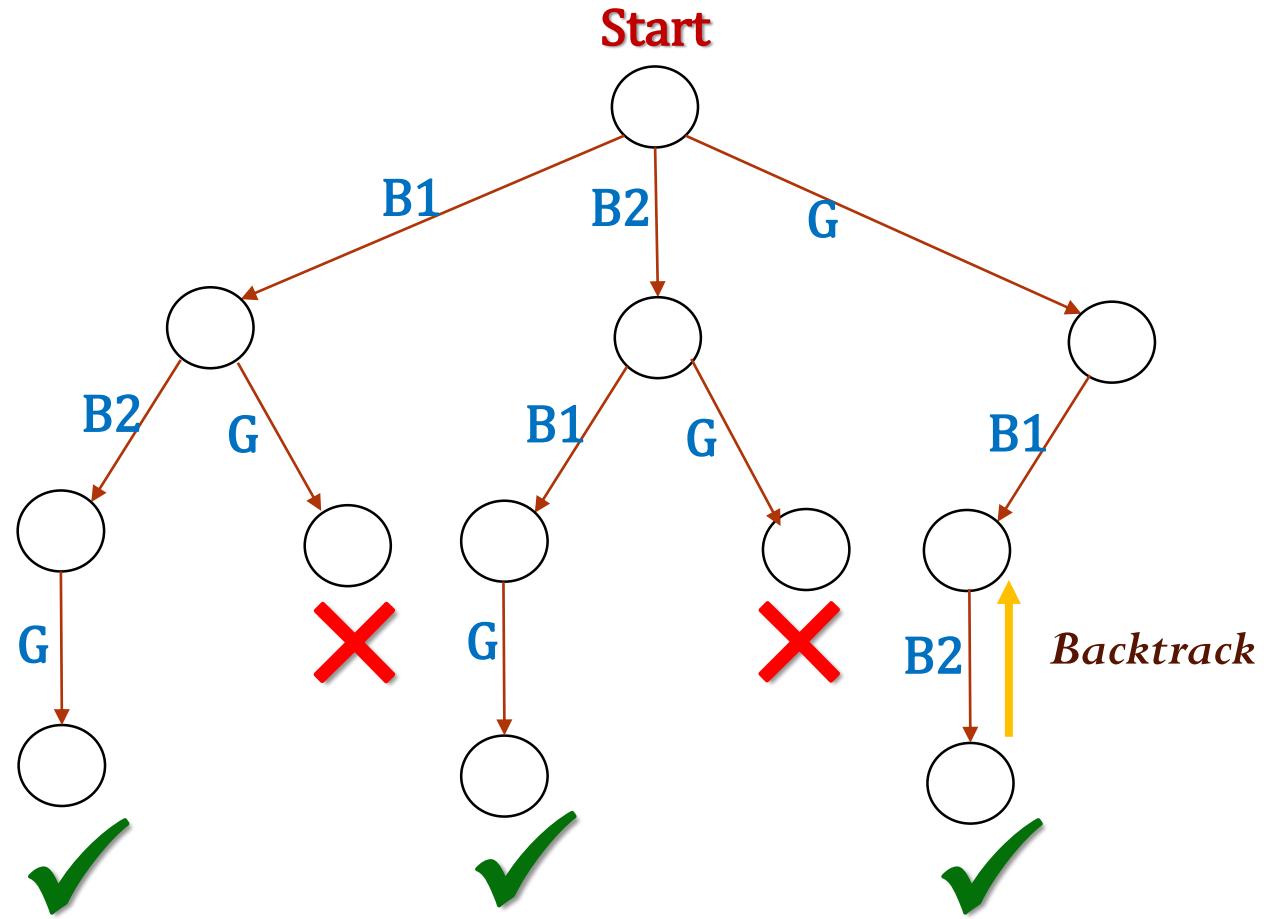
## Step-21



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

### Solutions

B1-B2-G  
B2-B1-G  
G-B1-B2

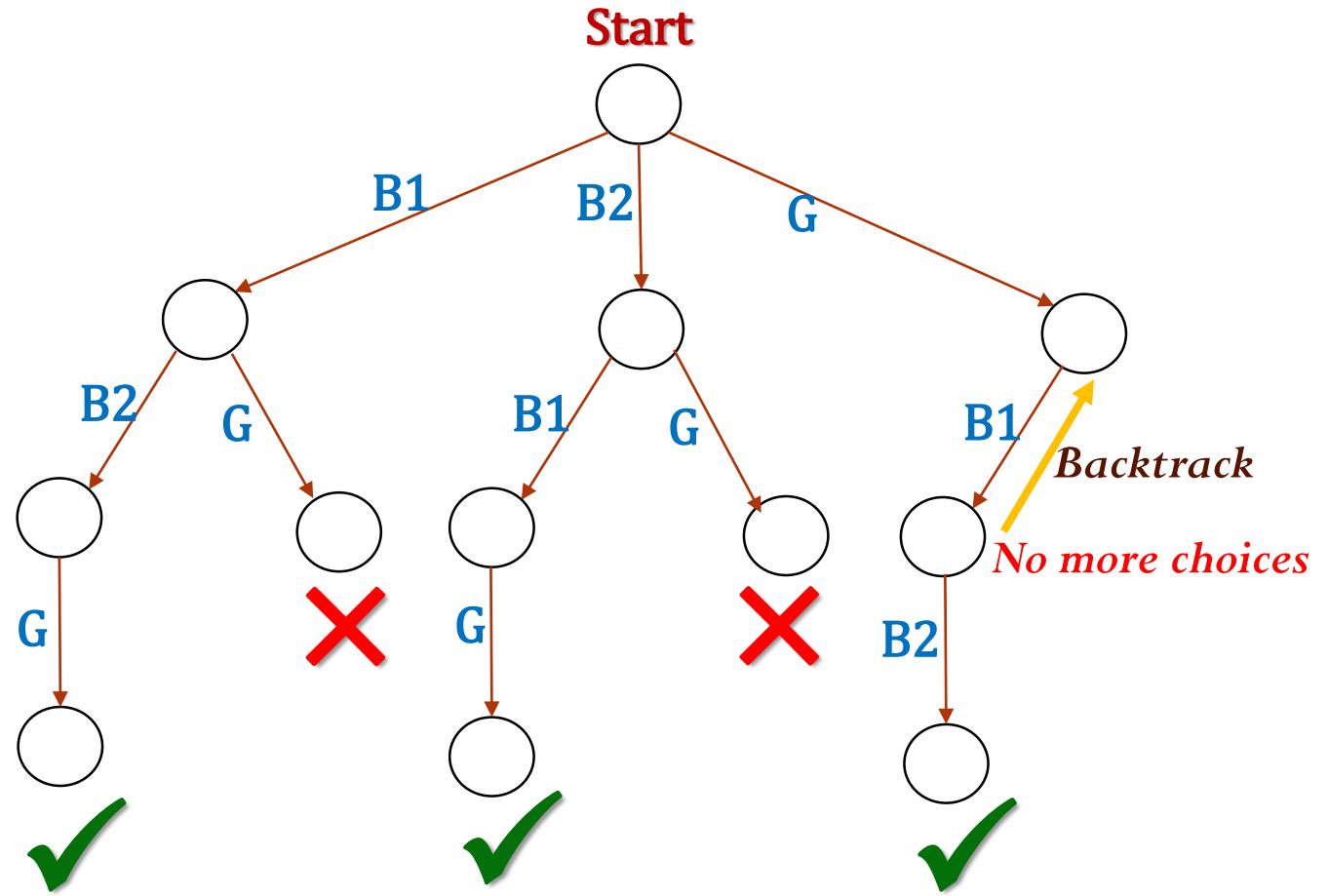


## Step-22



### Solutions

B1-B2-G  
B2-B1-G  
G-B1-B2



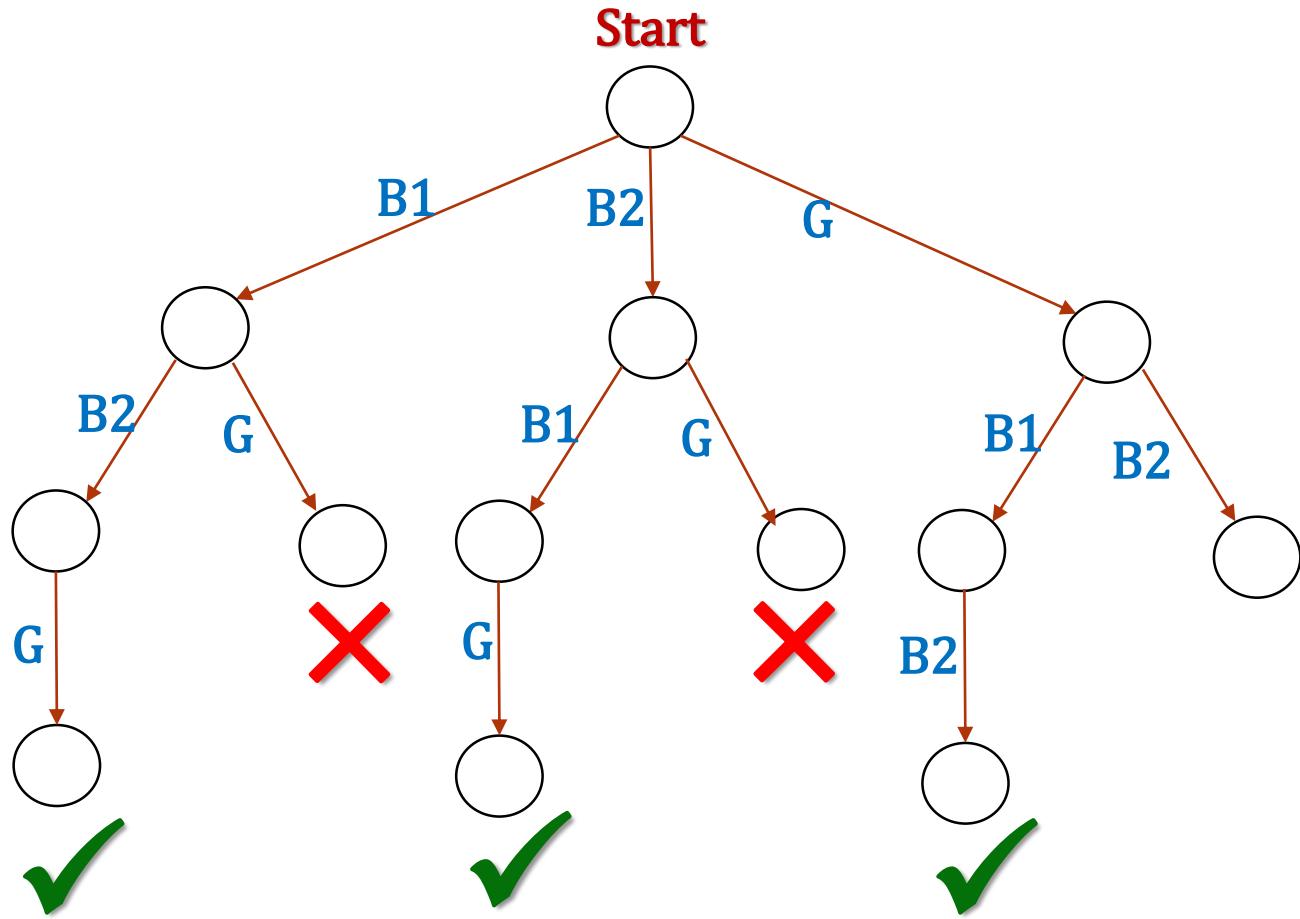
## **Step-23**

## Solutions

B1-B2-G

B2-B1-G

G-B1-B2



## Step-24

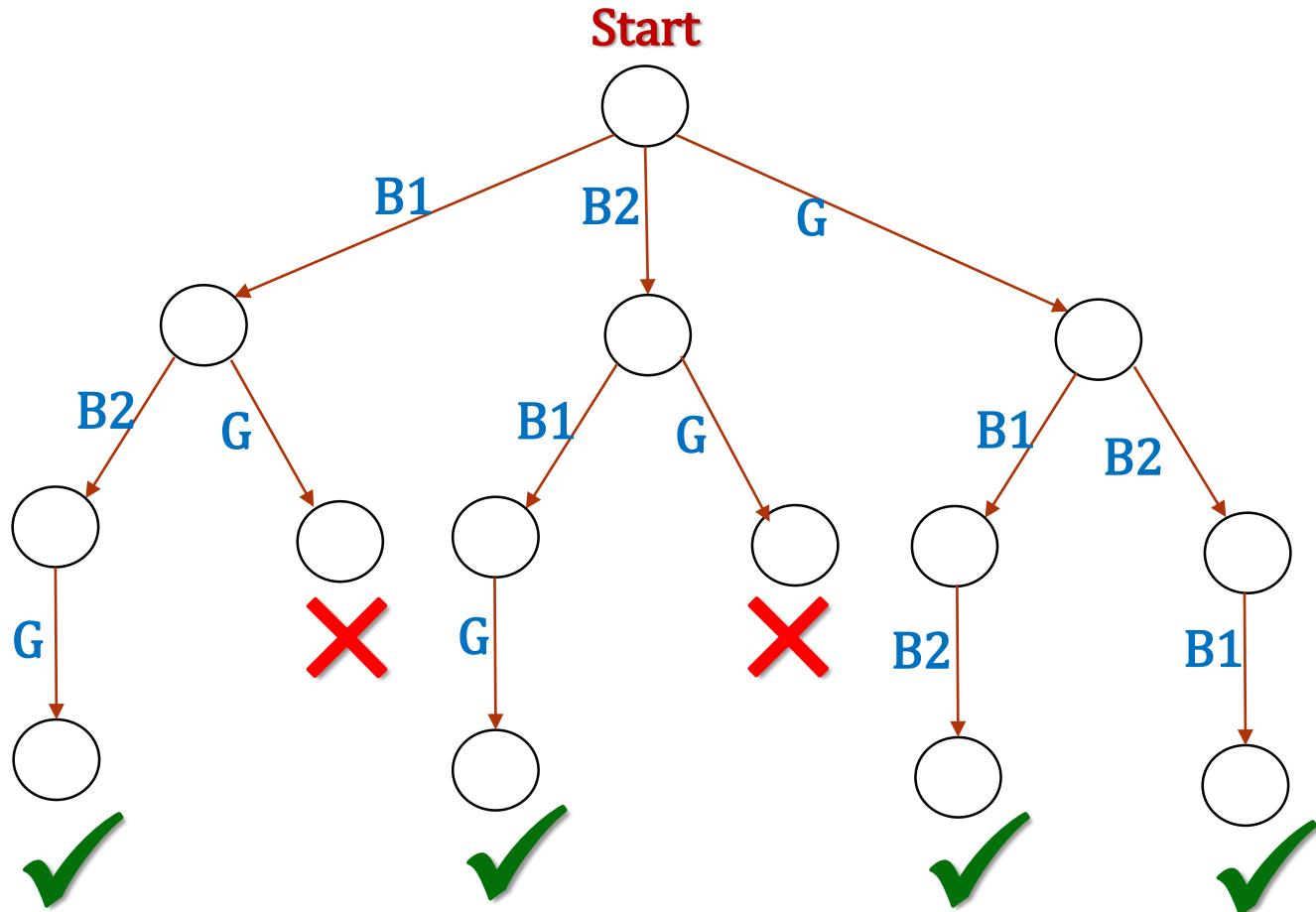
## Solutions

B1-B2-G

B2-B1-G

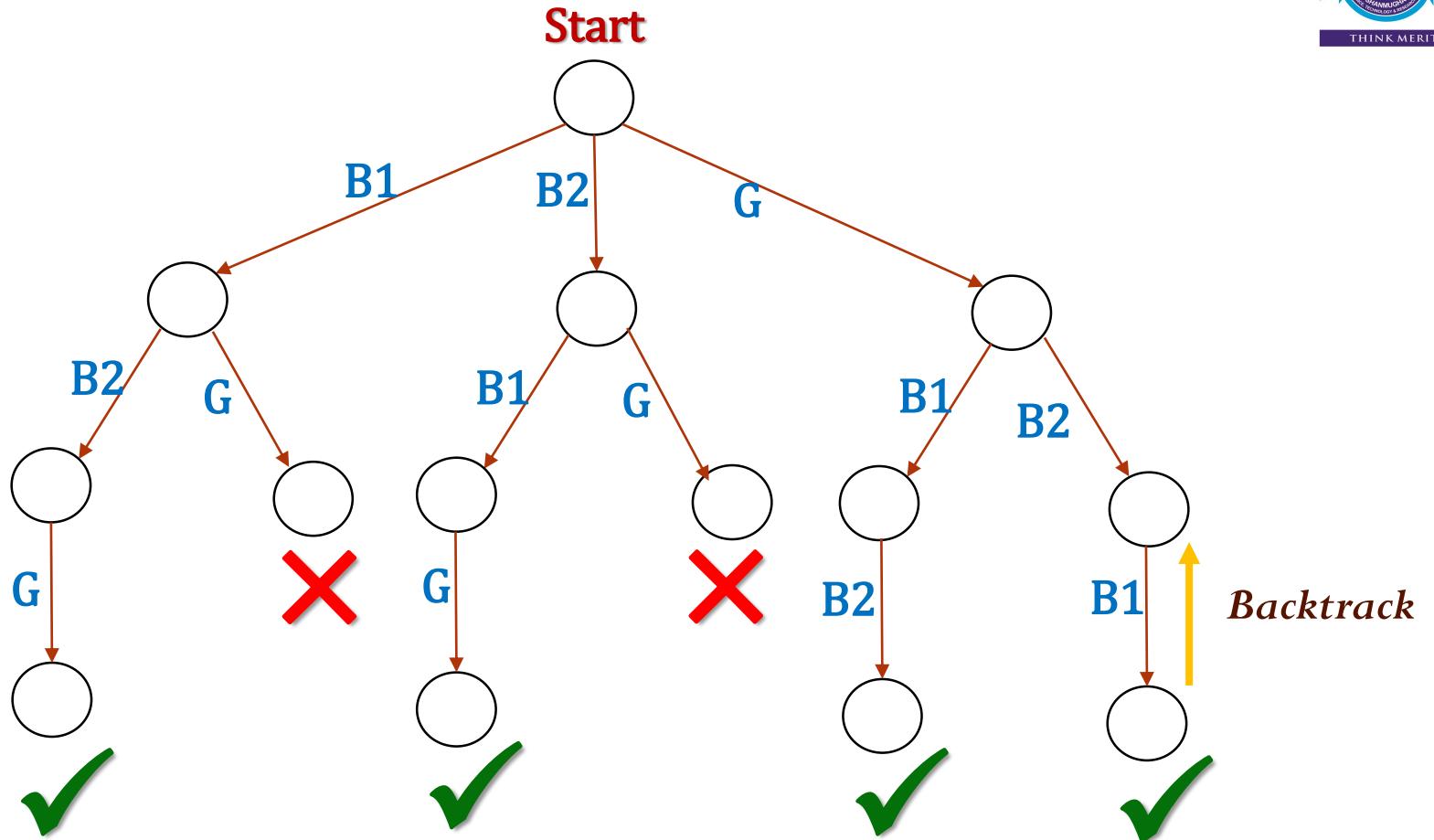
G-B1-B2

G-B2-B1



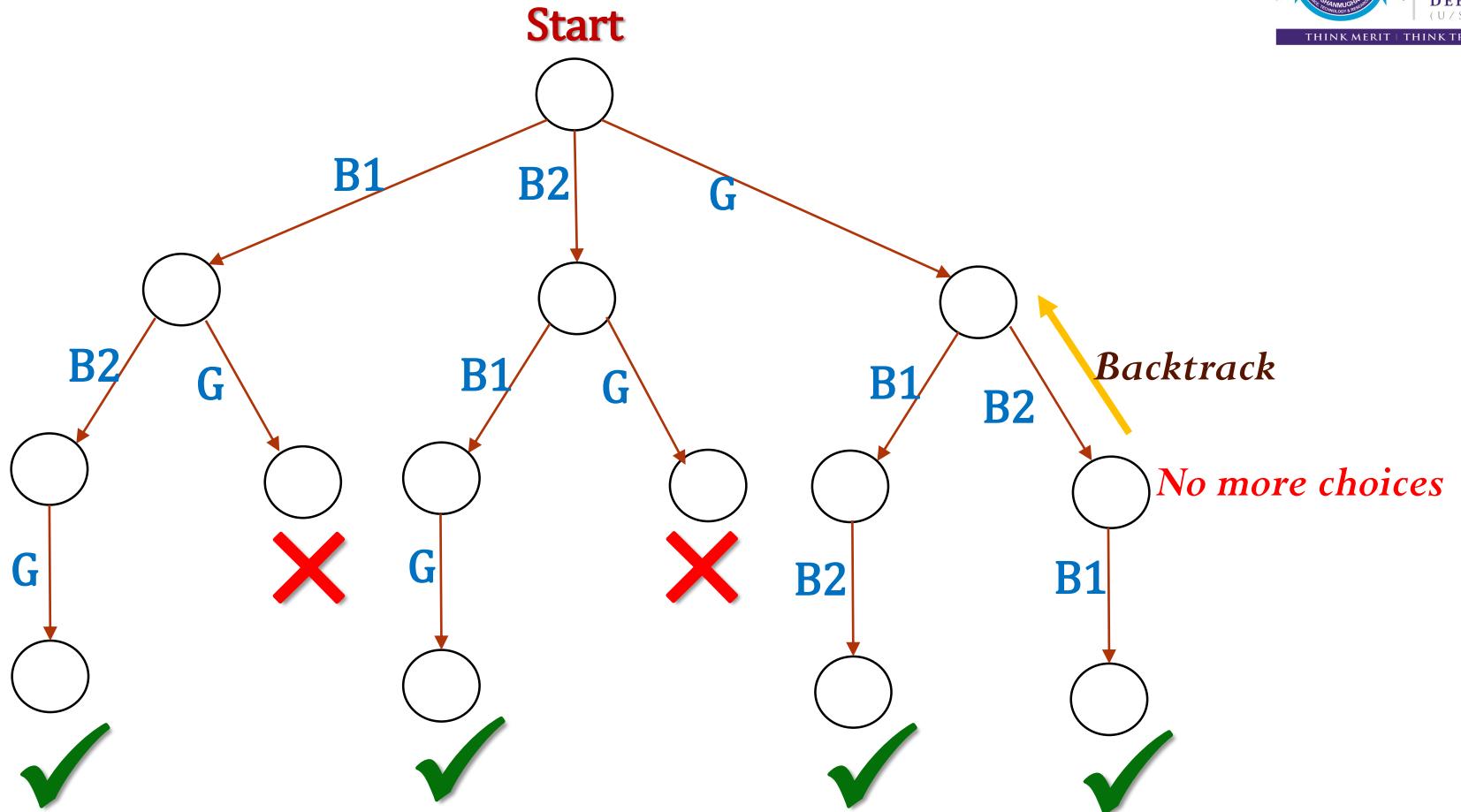
Solutions

B1-B2-G  
 B2-B1-G  
 G-B1-B2  
 G-B2-B1



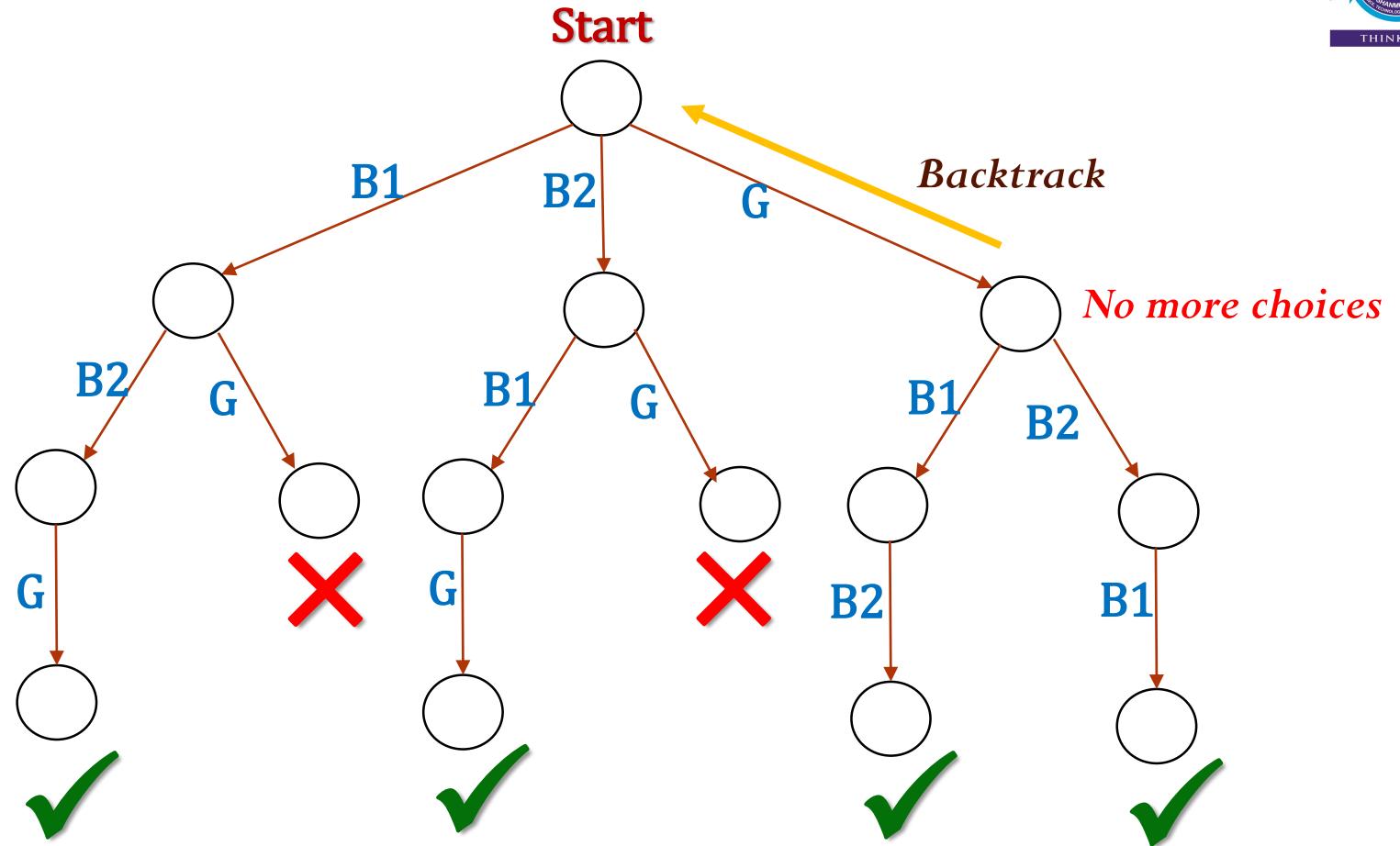
Solutions

B1-B2-G  
 B2-B1-G  
 G-B1-B2  
 G-B2-B1



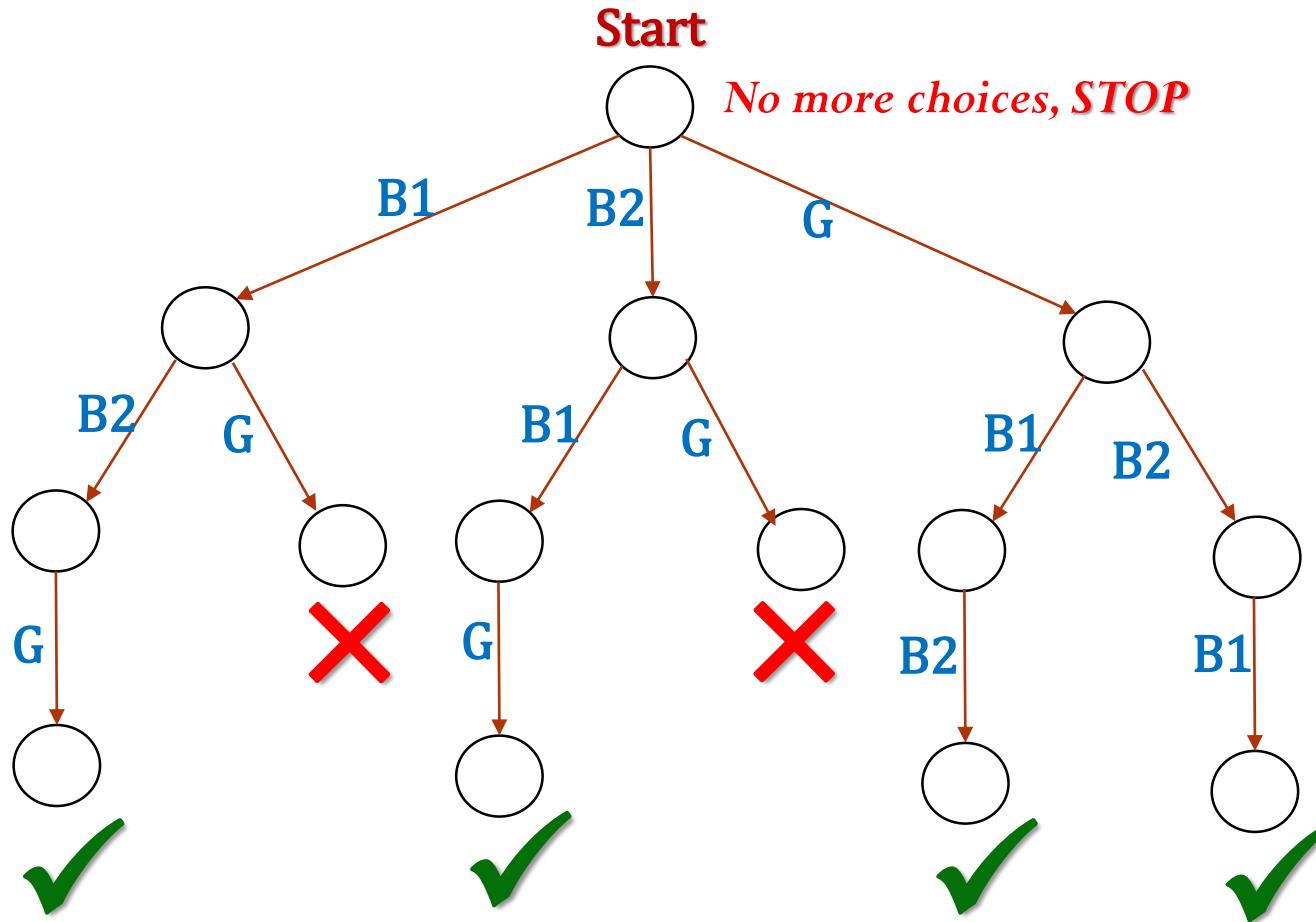
Solutions

B1-B2-G  
 B2-B1-G  
 G-B1-B2  
 G-B2-B1



Solutions

B1-B2-G  
 B2-B1-G  
 G-B1-B2  
 G-B2-B1



# **Backtracking Approach**

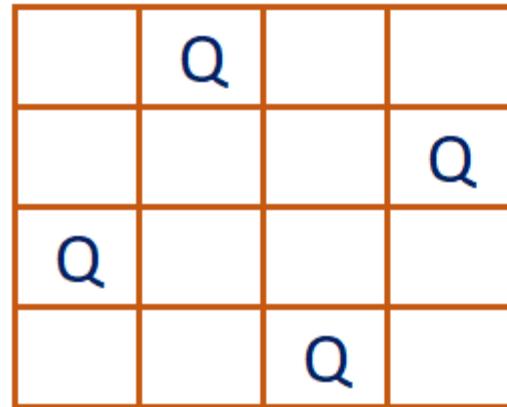
## **n-Queen Problem**

# Problem

- ✓ The N Queen is the problem of placing N chess queens on an NxN chessboard so that no two queens attack each other.
- ✓ The chess queens can attack in any direction as horizontal, vertical, horizontal and diagonal way.
- ✓ A binary matrix is used to display the positions of N Queens, where no queens can attack other queens.
- ✓ Example: 4-Queens Problem – Placing 4 Queens in 4x4 Chess board  
8-Queens Problem – Placing 8 Queens in 8x8 Chess board

# Input & Output – For Feasible Solution

- ✓ **Input:** N – Number of queens to be placed
- ✓ **Output:** N x N binary (0/1) matrix – Represents chess board whose values represents the placement of queens.
- ✓ Example: For 4-Queens Problem, N=4



Feasible Solution

0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0

Solution Matrix

# Input & Output – For All Possible Solutions

- ✓ **Input:** N – Number of queens to be placed
- ✓ **Output:** M x N matrix – Represents all possible 'M' numbers of solutions.  
Each row contains the position of queens in rows of solution matrices.
- ✓ Example: For 4-Queens Problem, N=4, the resultant 2 x 4 matrix is given below. i.e. there are 2 solutions for 4-Queens Problem (i.e., M=2).

	Q		
			Q
Q			
		Q	

**Solution-1**

2	4	1	3
3	1	4	2

**Solution Matrix**

		Q	
	Q		
			Q

**Solution-2**



## For Feasible Solution – Dynamic Programming

## For All Possible Solutions - Backtracking Algorithm

# Backtracking Algorithm

- ✓ The idea is to place queens one by one in different rows, starting from the first row.
- ✓ When we place a queen in a column, we check for clashes with already placed queens.
- ✓ In the current row, if we find a column for which there is no clash, we mark this row and column as part of the solution.
- ✓ If we do not find such a column due to clashes then we backtrack and return false.

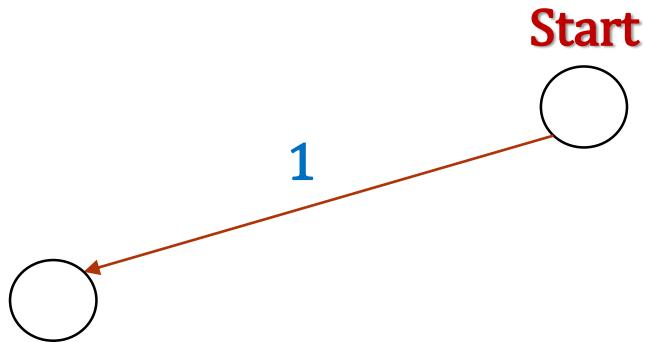
	1	2	3	4
1				
2				
3				
4				

Start

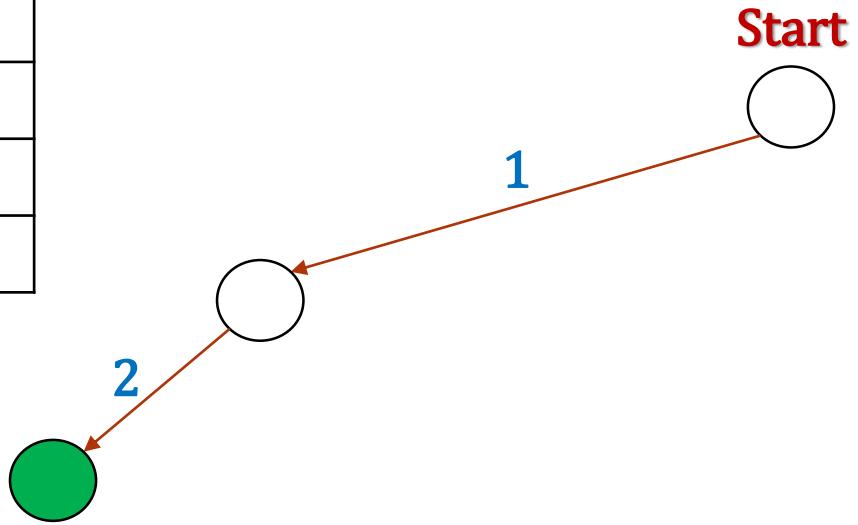


**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

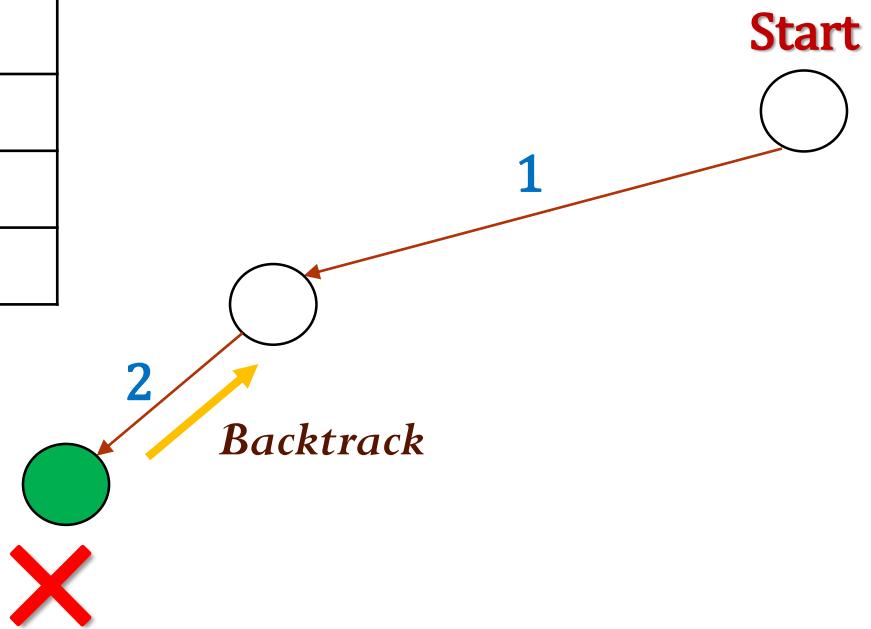
	1	2	3	4
1	Q			
2				
3				
4				



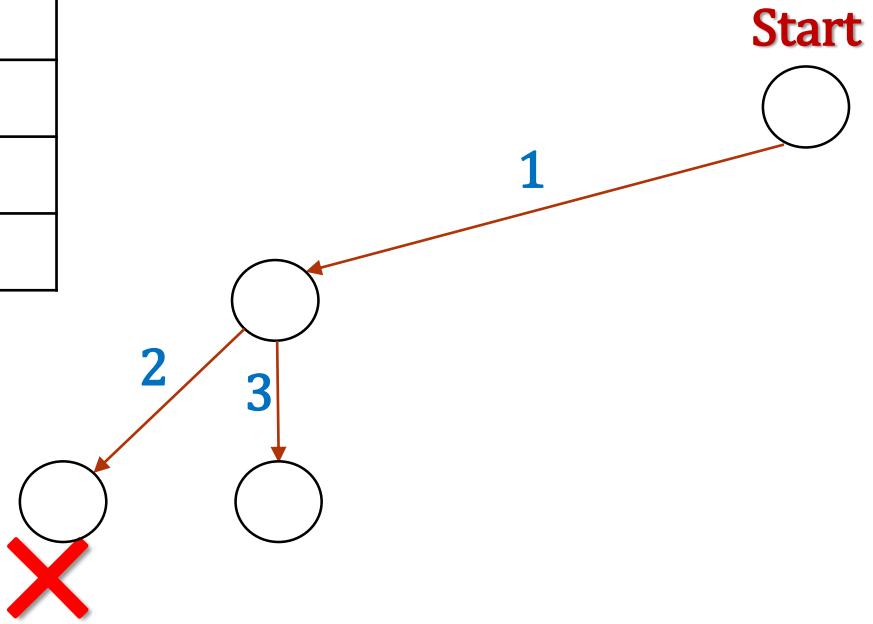
	1	2	3	4
1	Q			
2		Q		
3				
4				



	1	2	3	4
1	Q			
2				
3				
4				

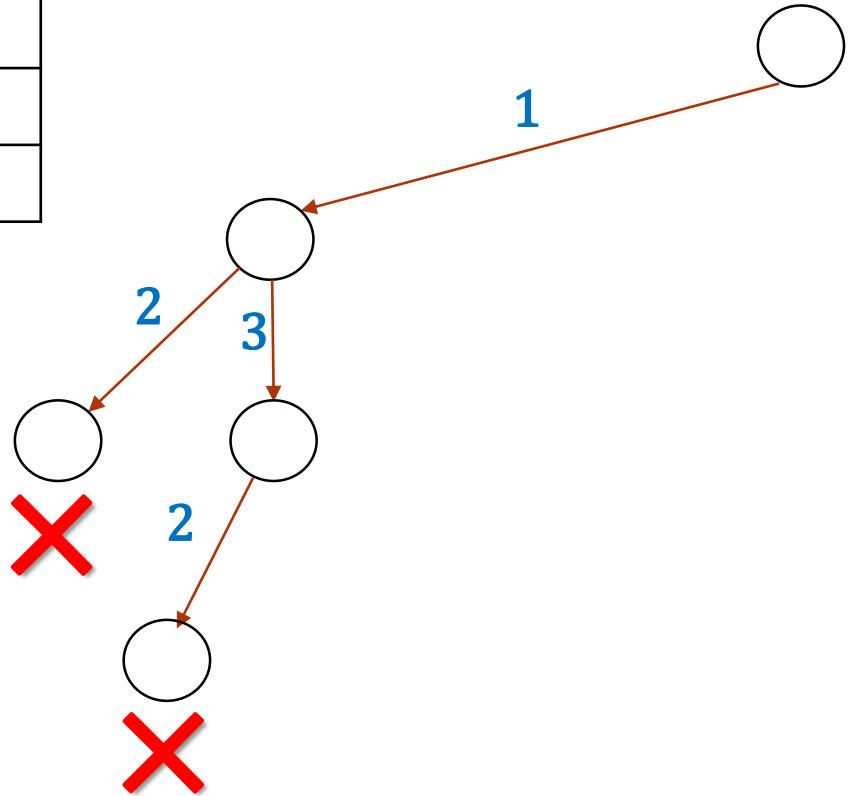


	1	2	3	4
1	Q			
2			Q	
3				
4				

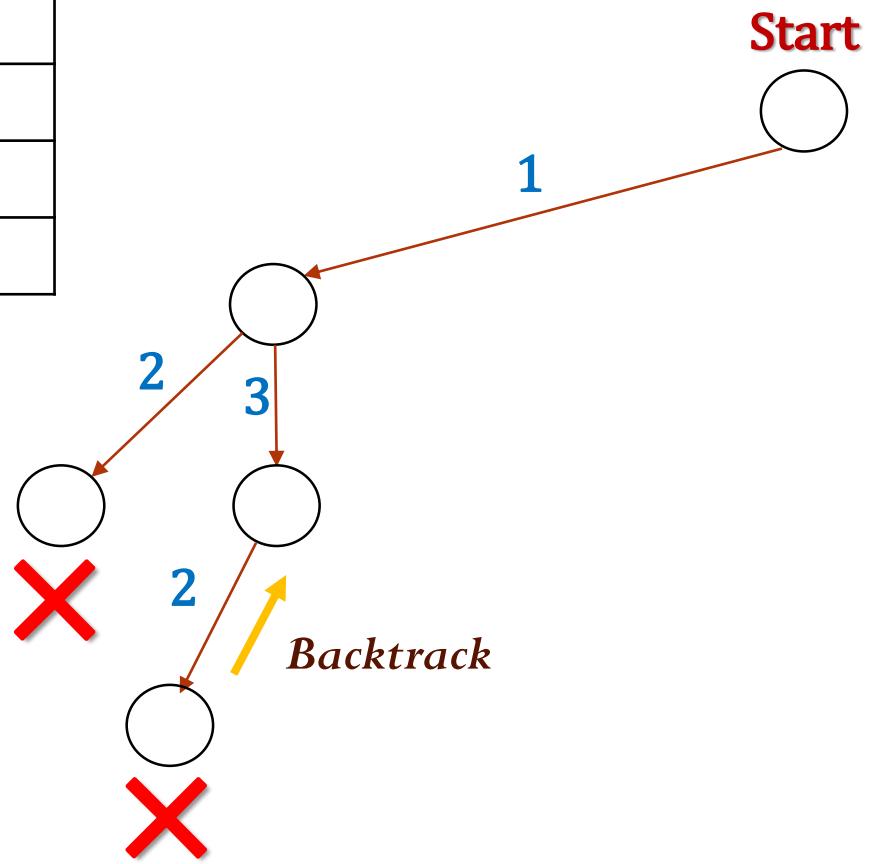


	1	2	3	4
1	Q			
2			Q	
3		Q		
4				

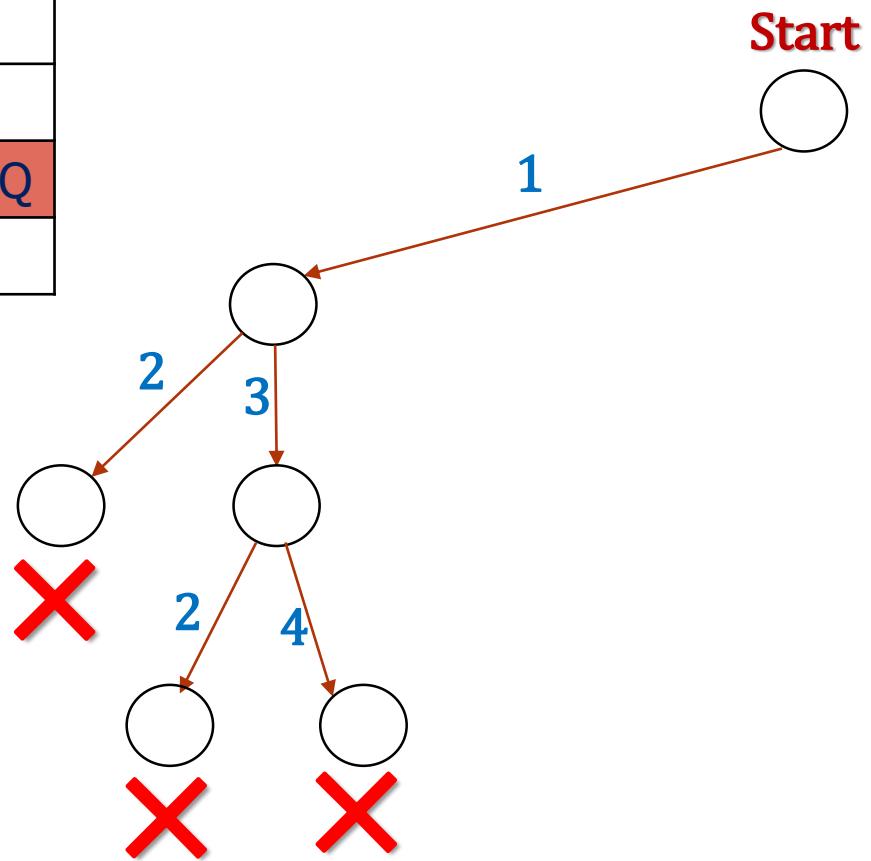
Start



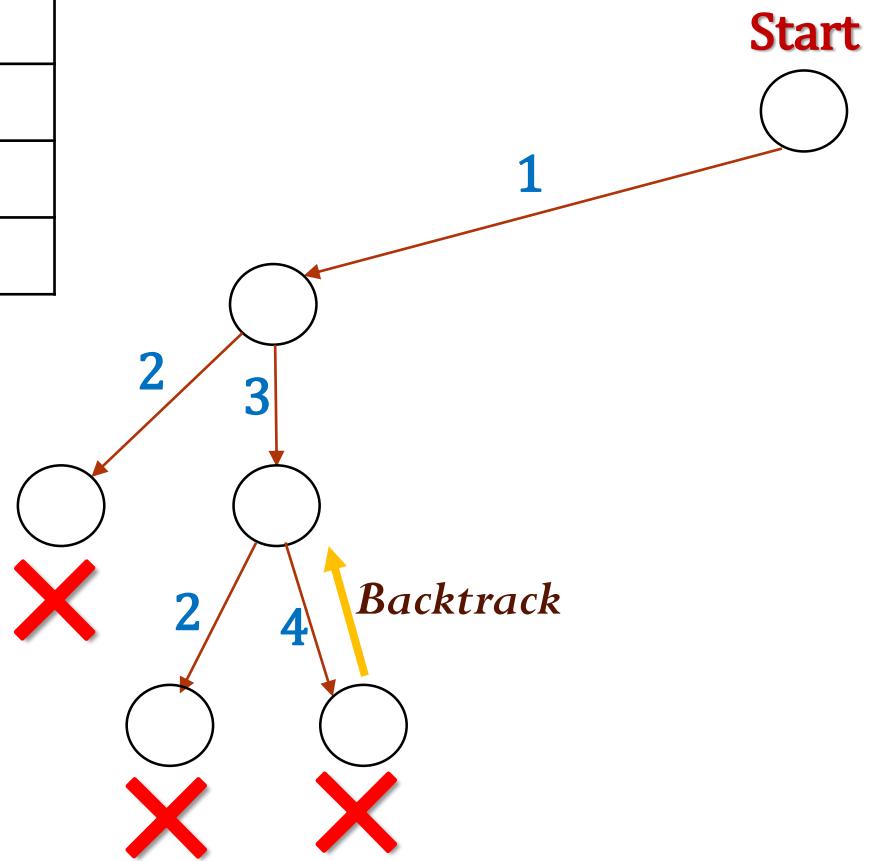
	1	2	3	4
1	Q			
2			Q	
3				
4				



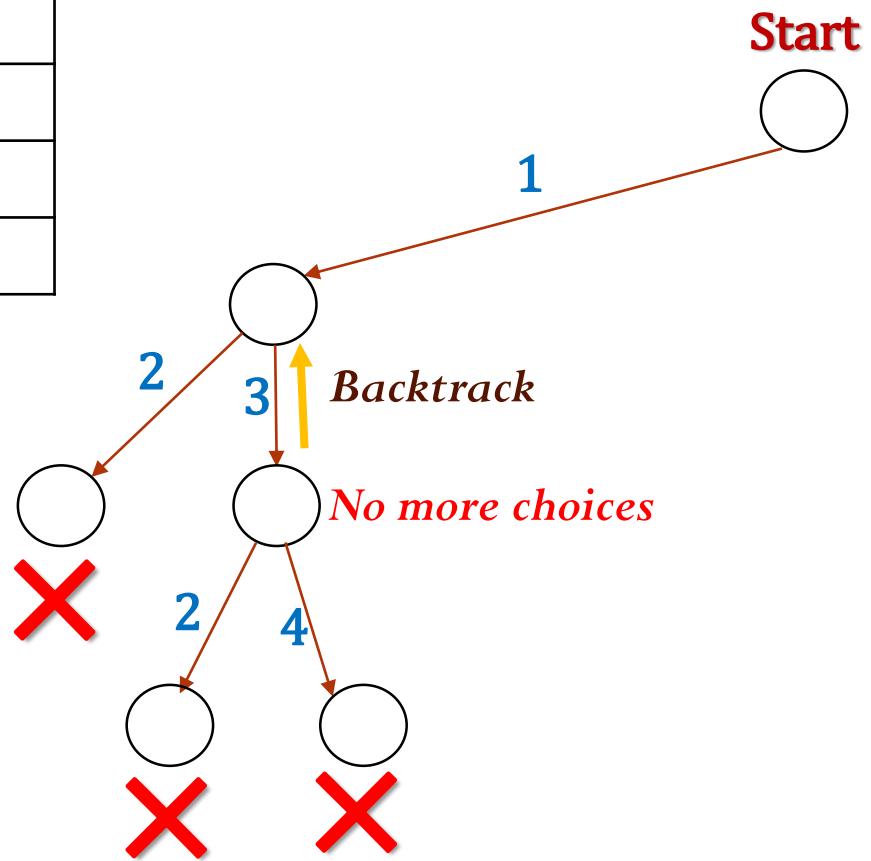
	1	2	3	4
1	Q			
2			Q	
3				Q
4				



	1	2	3	4
1	Q			
2			Q	
3				
4				

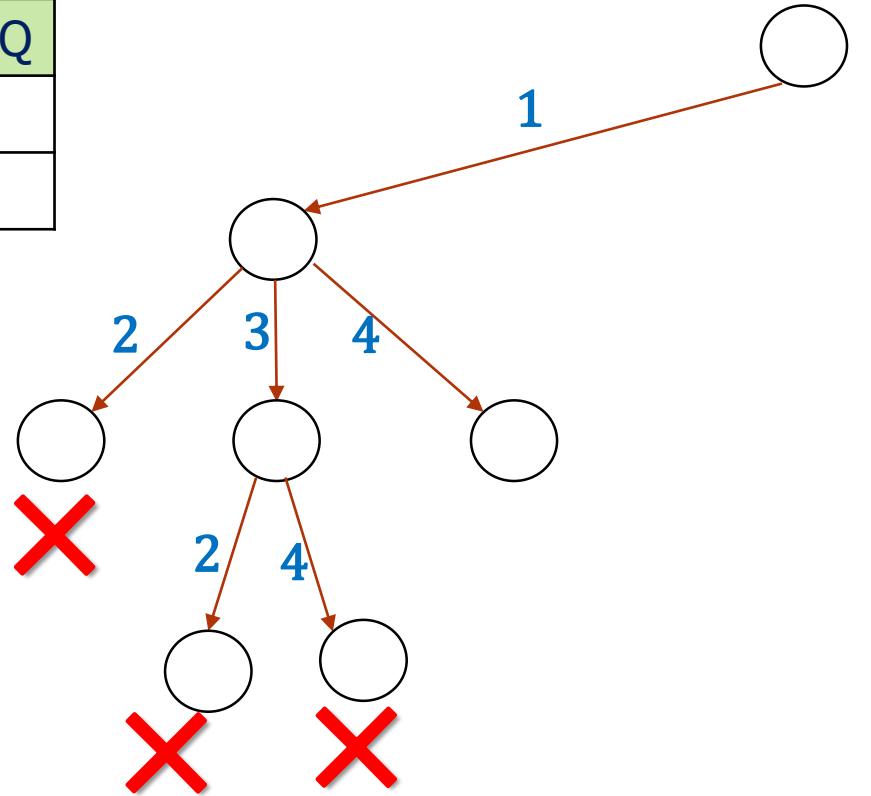


	1	2	3	4
1	Q			
2				
3				
4				



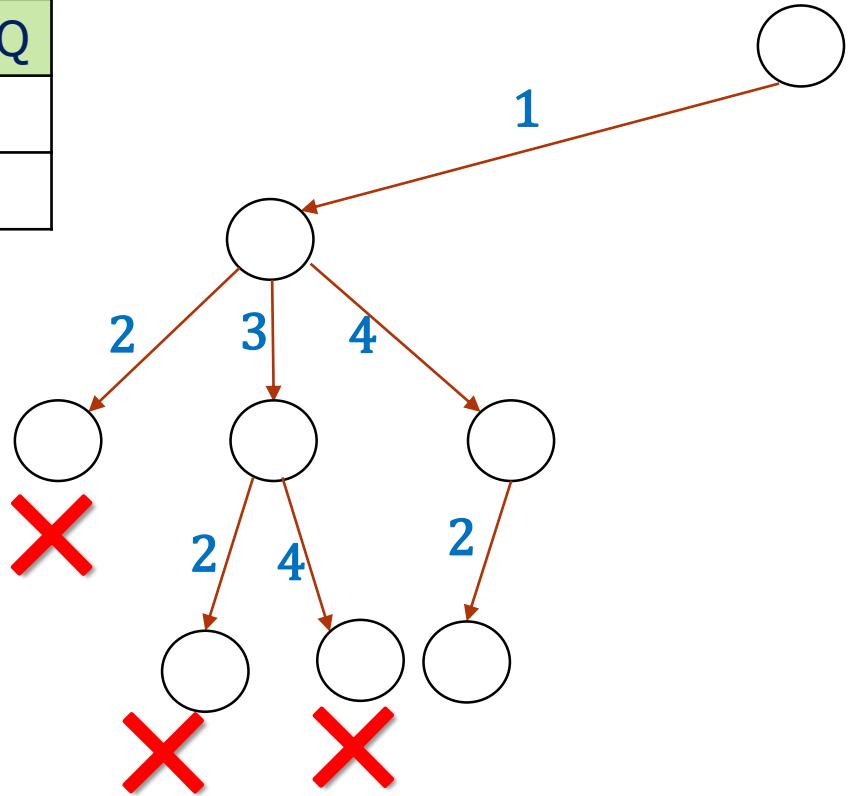
	1	2	3	4
1	Q			
2				Q
3				
4				

Start

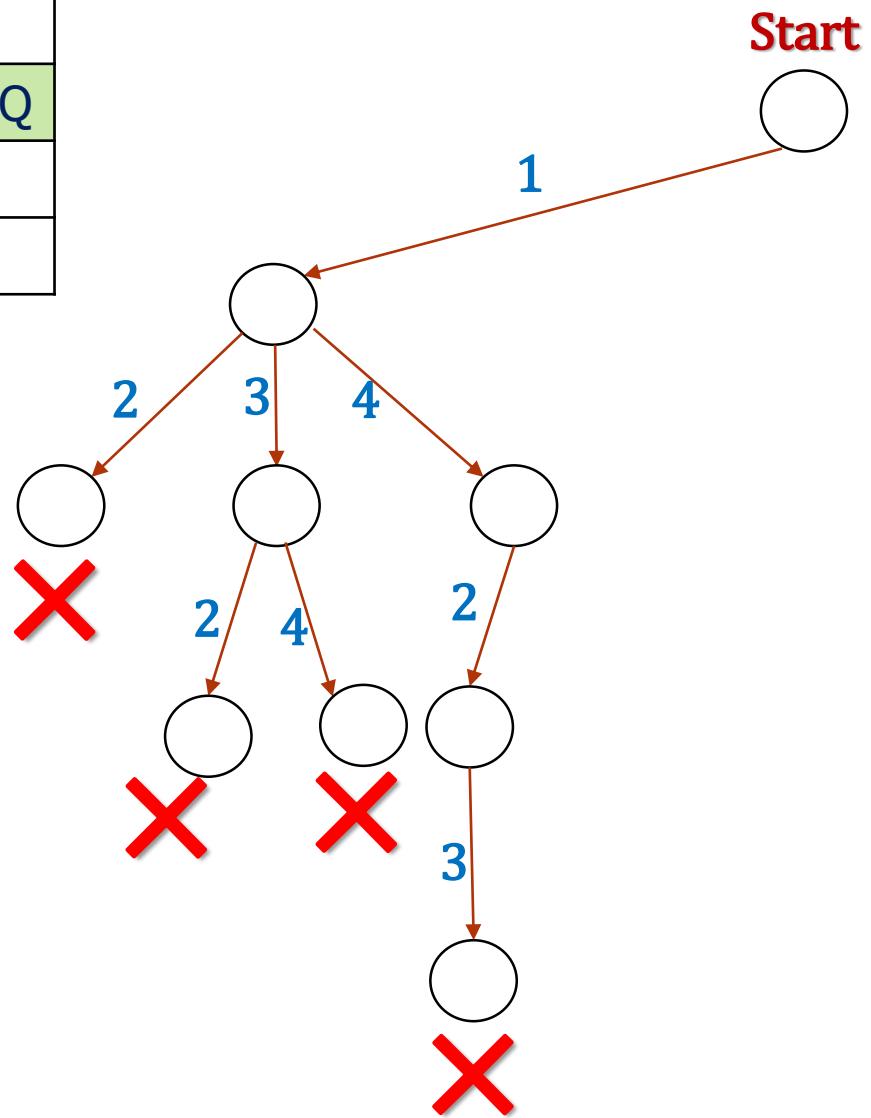


	1	2	3	4
1	Q			
2				Q
3		Q		
4				

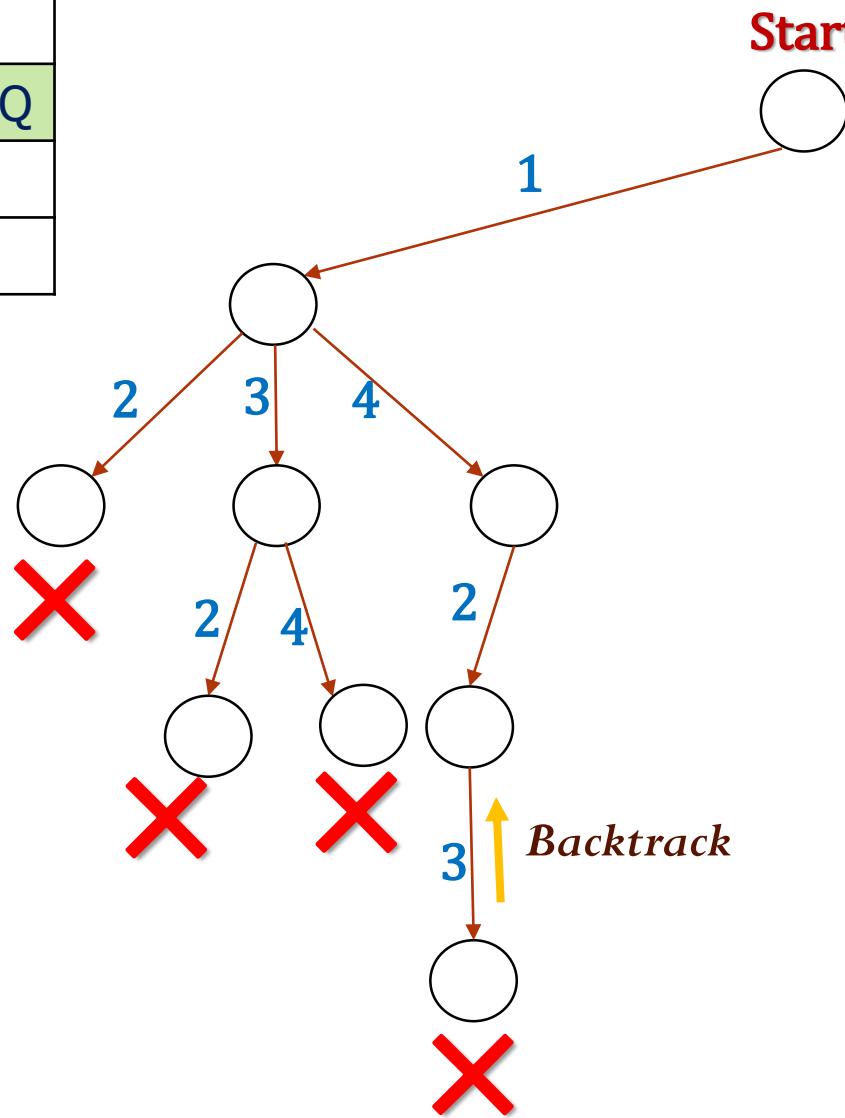
Start



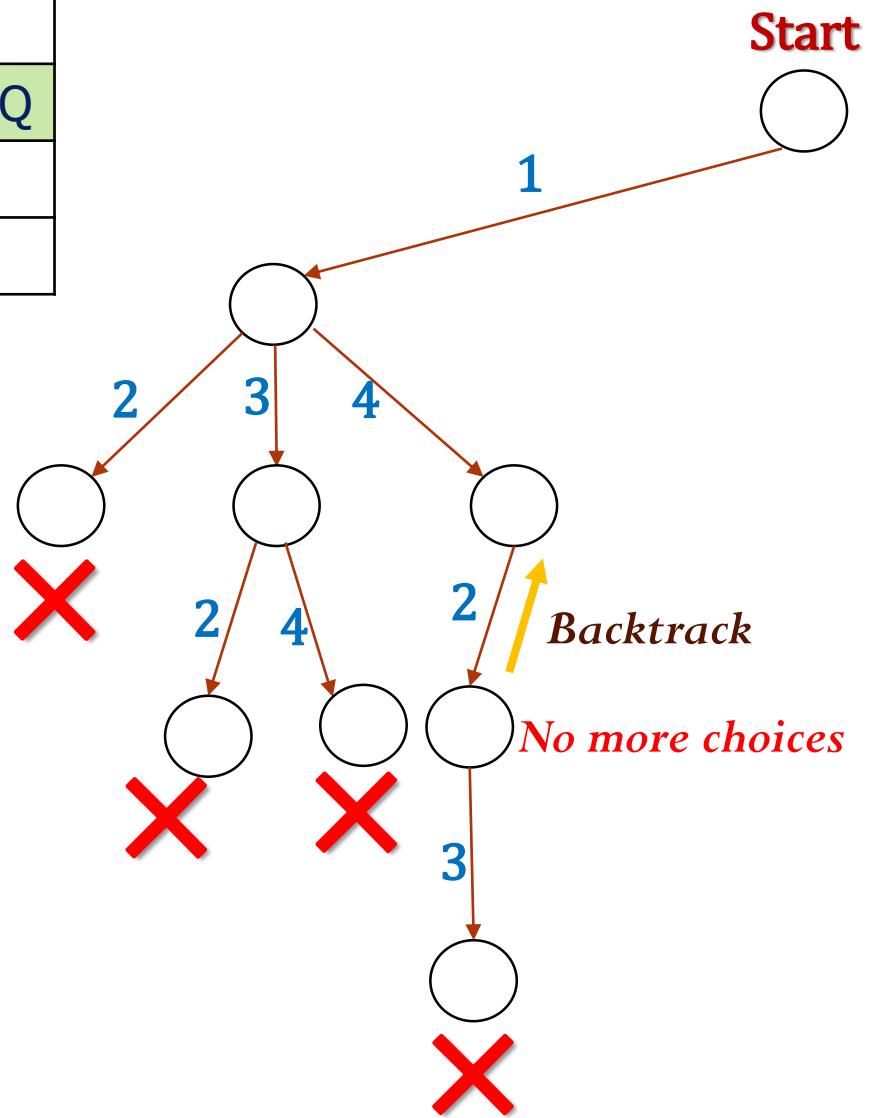
	1	2	3	4
1	Q			
2				Q
3		Q		
4			Q	



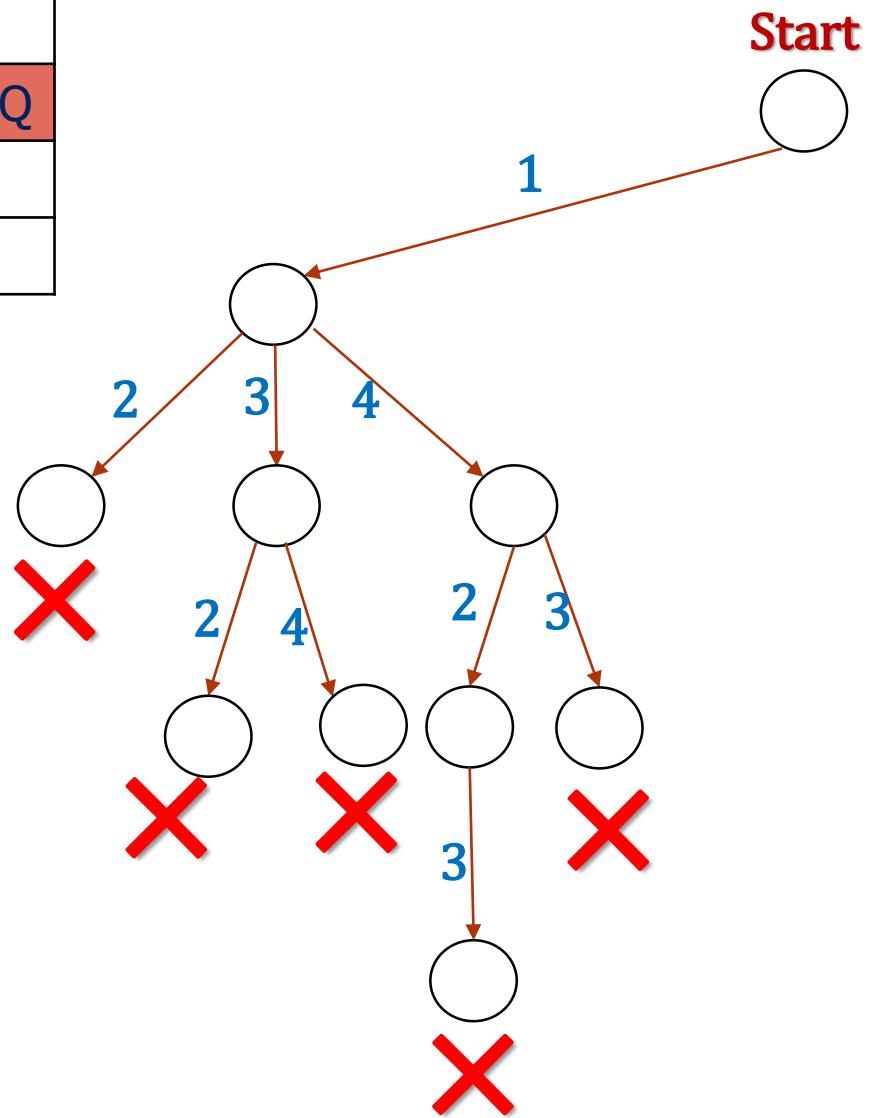
	1	2	3	4
1	Q			
2				Q
3		Q		
4				



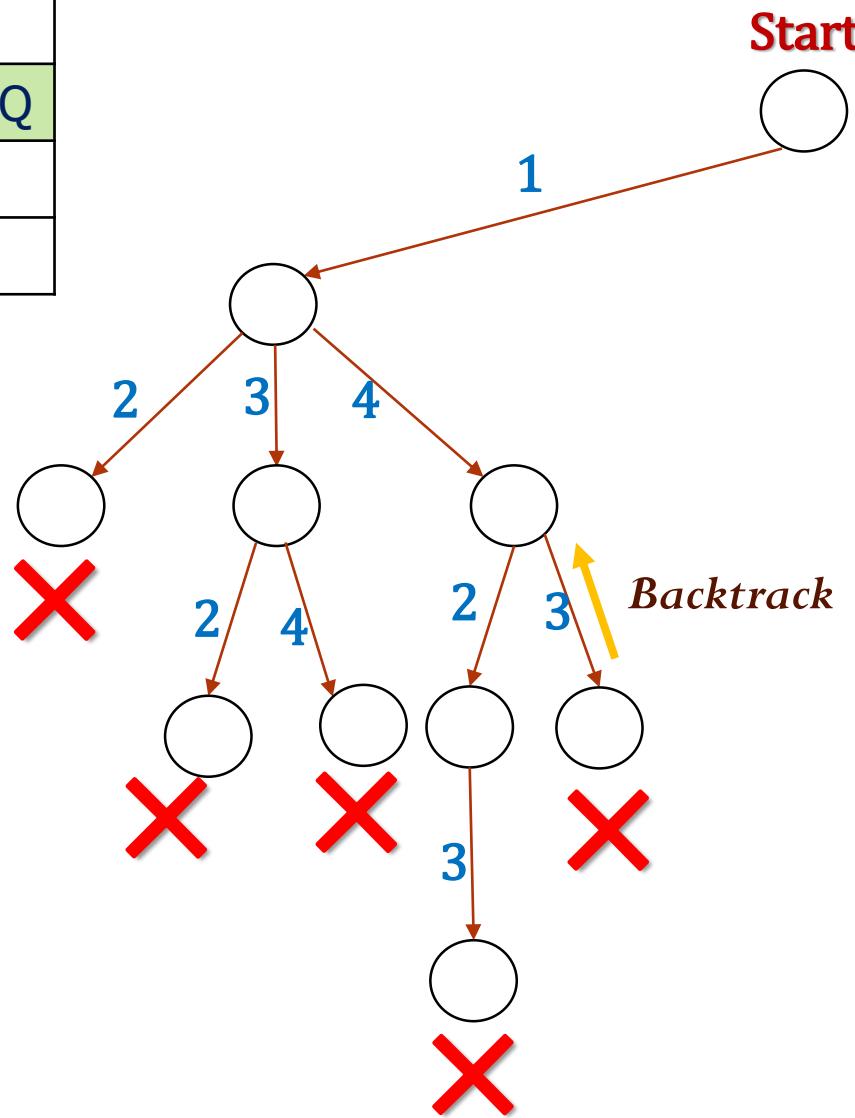
	1	2	3	4
1	Q			
2				Q
3				
4				



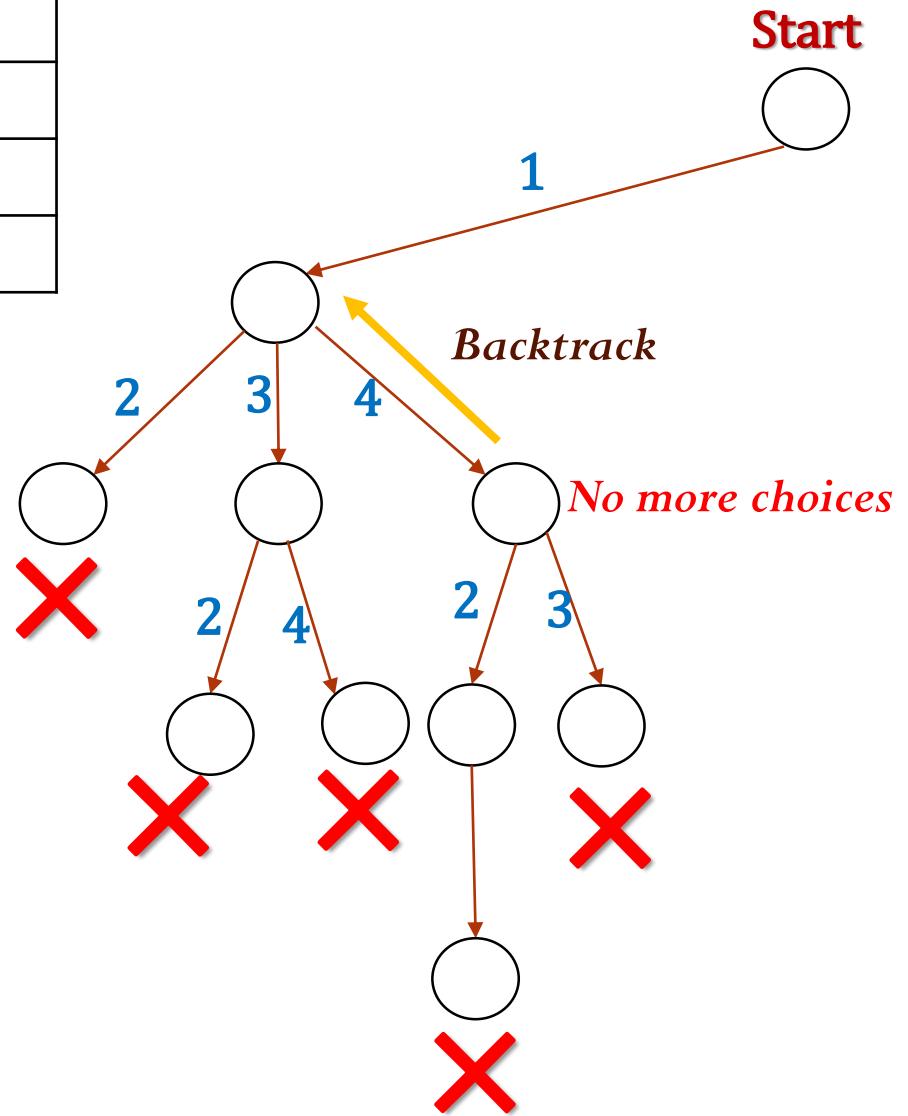
	1	2	3	4
1	Q			
2				Q
3			Q	
4				



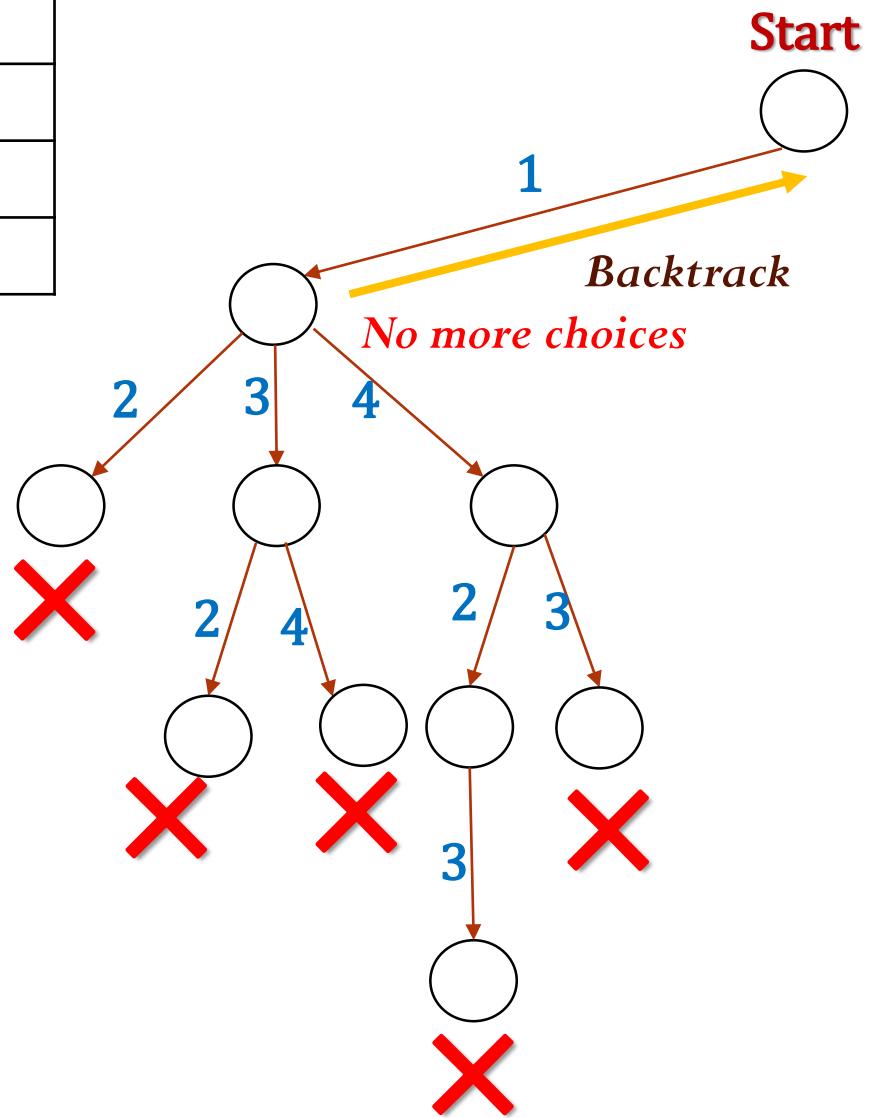
	1	2	3	4
1	Q			
2				Q
3				
4				



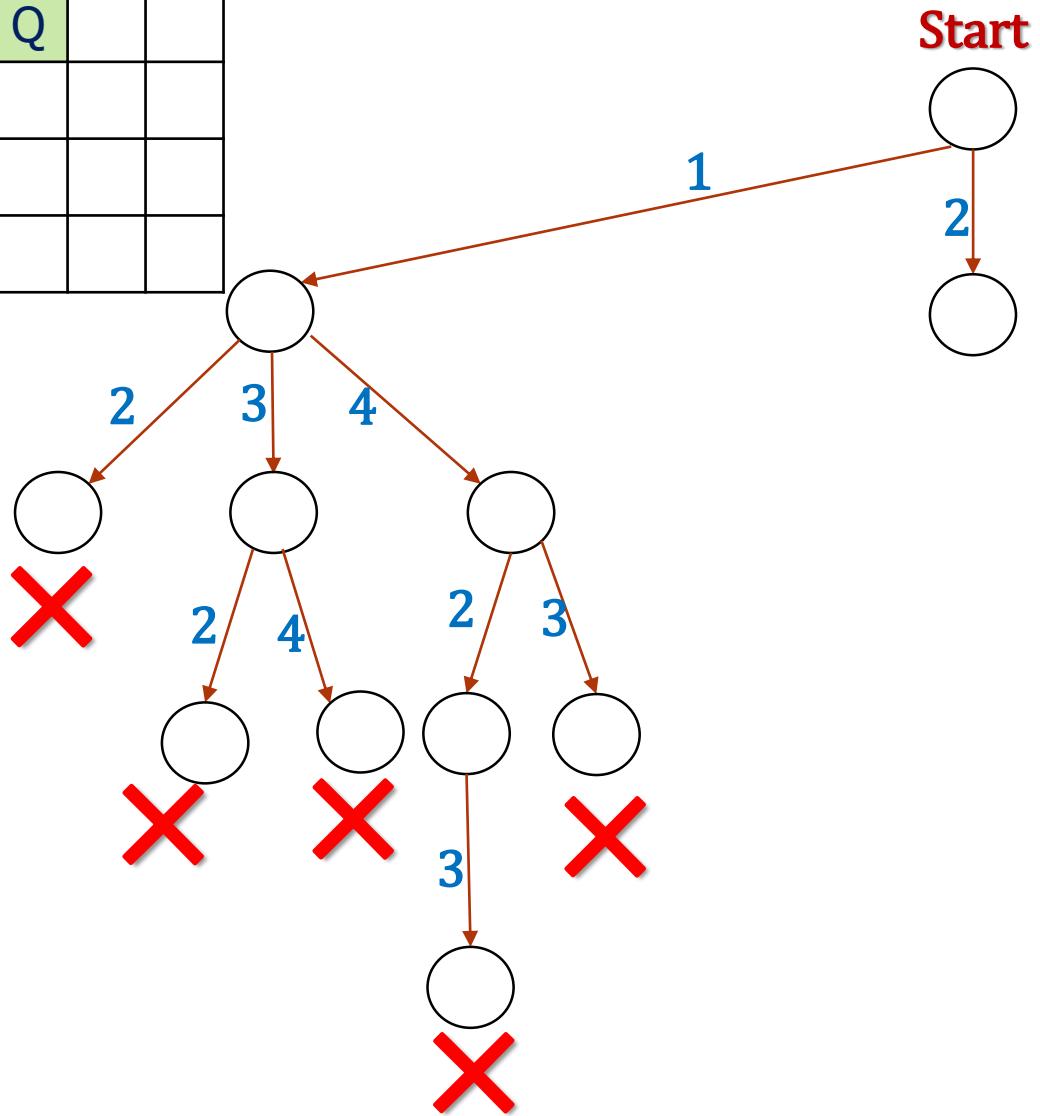
	1	2	3	4
1	Q			
2				
3				
4				



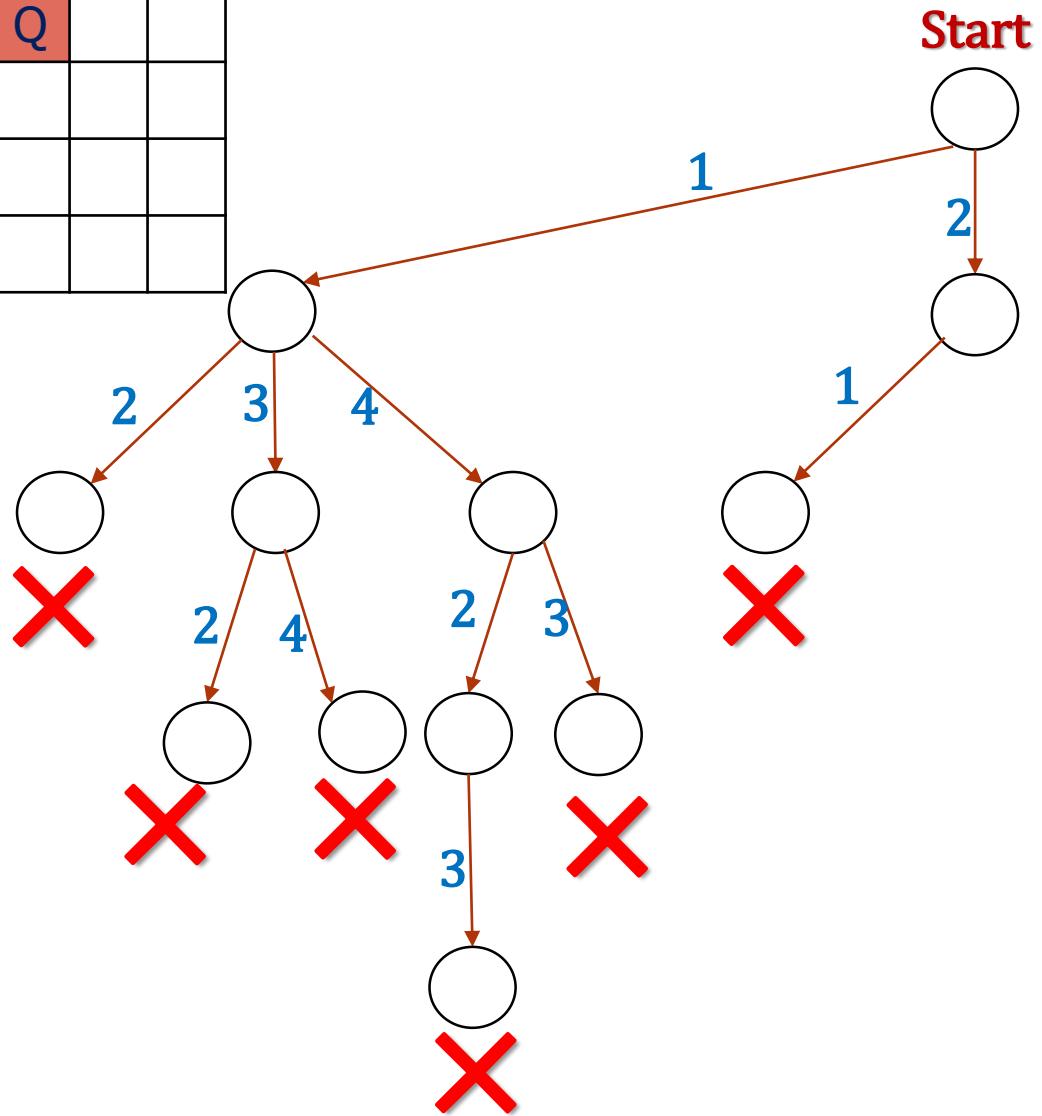
	1	2	3	4
1				
2				
3				
4				



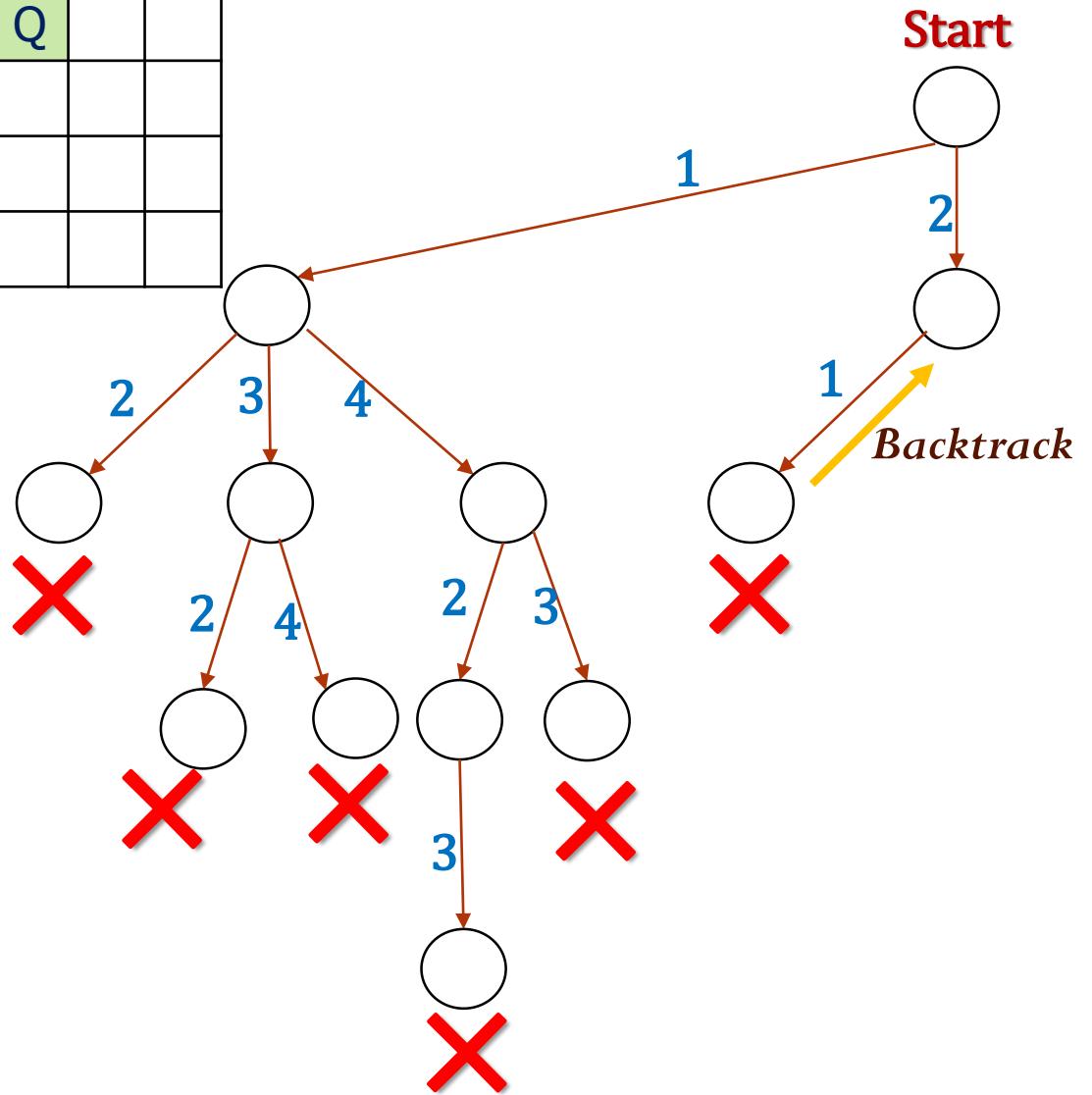
	1	2	3	4
1		Q		
2				
3				
4				



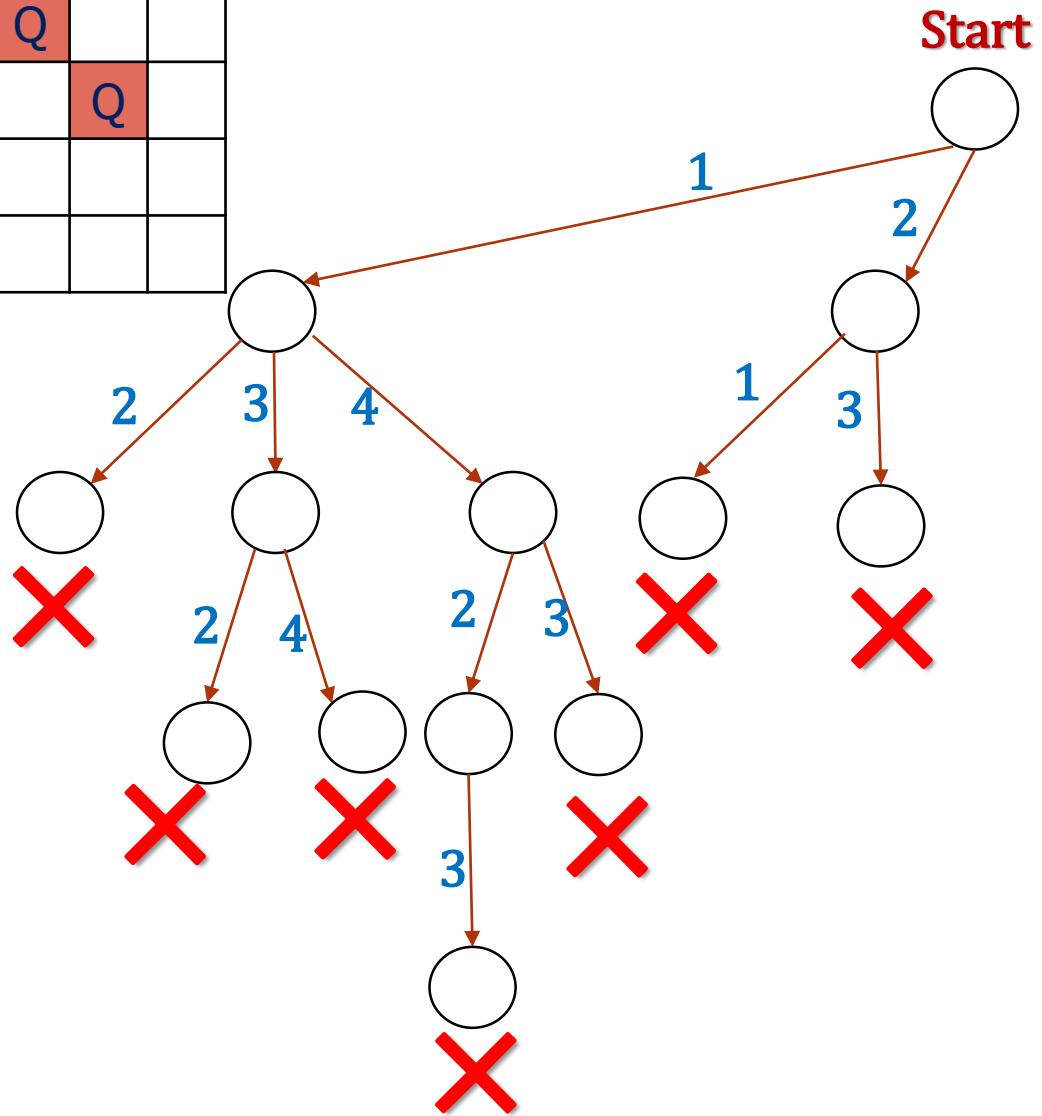
	1	2	3	4
1		Q		
2	Q			
3				
4				



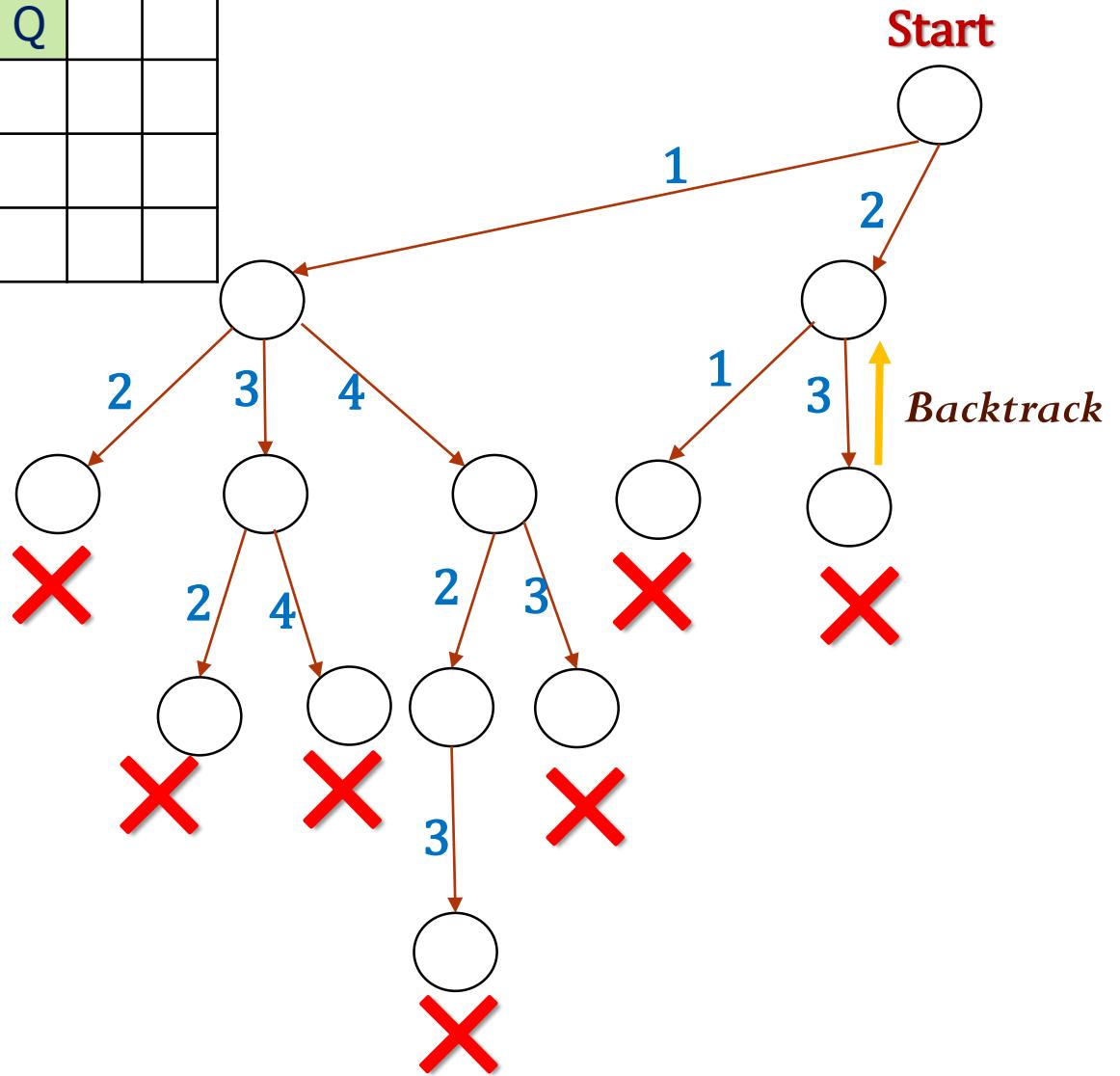
	1	2	3	4
1		Q		
2				
3				
4				



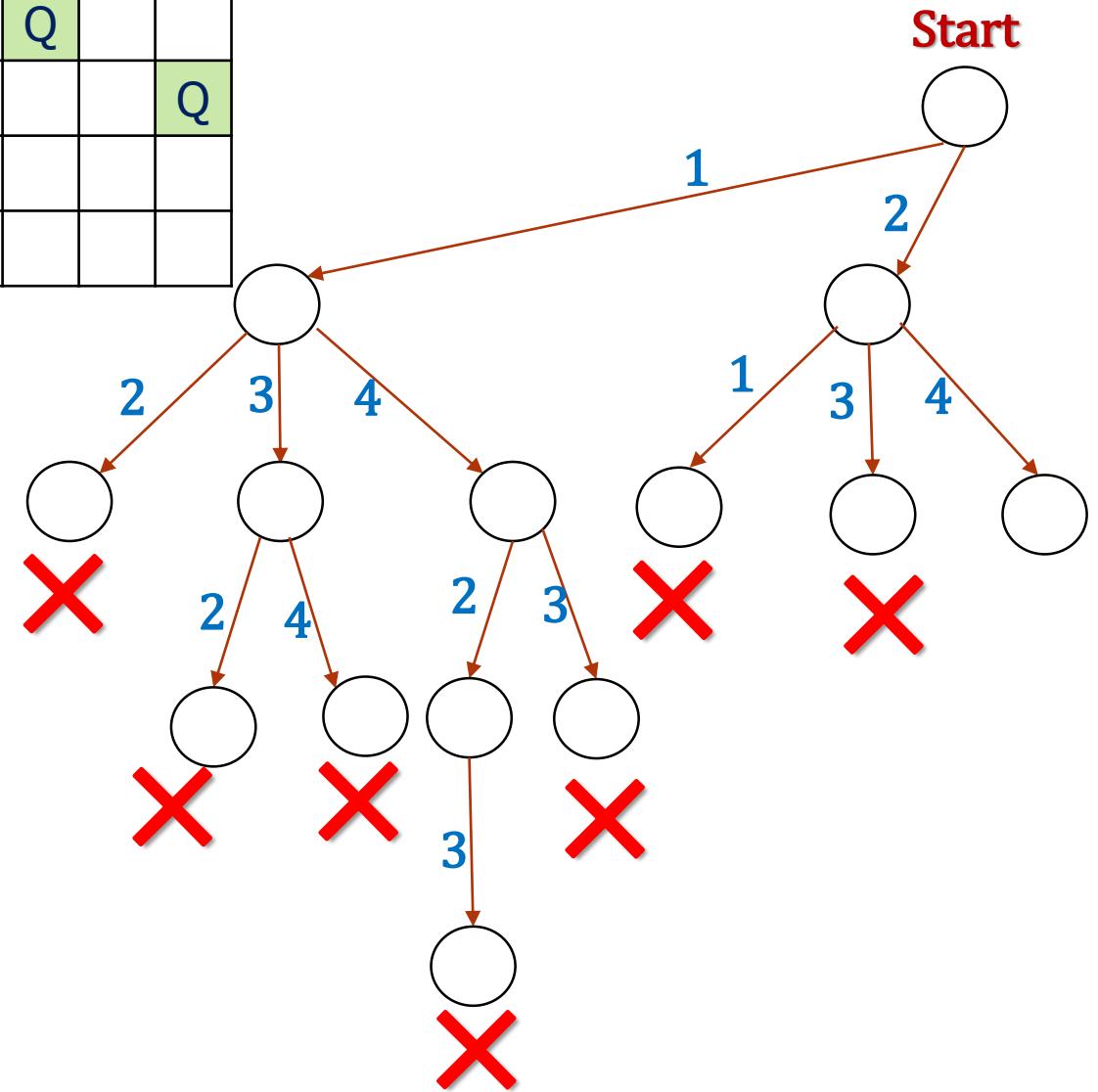
	1	2	3	4
1		Q		
2			Q	
3				
4				



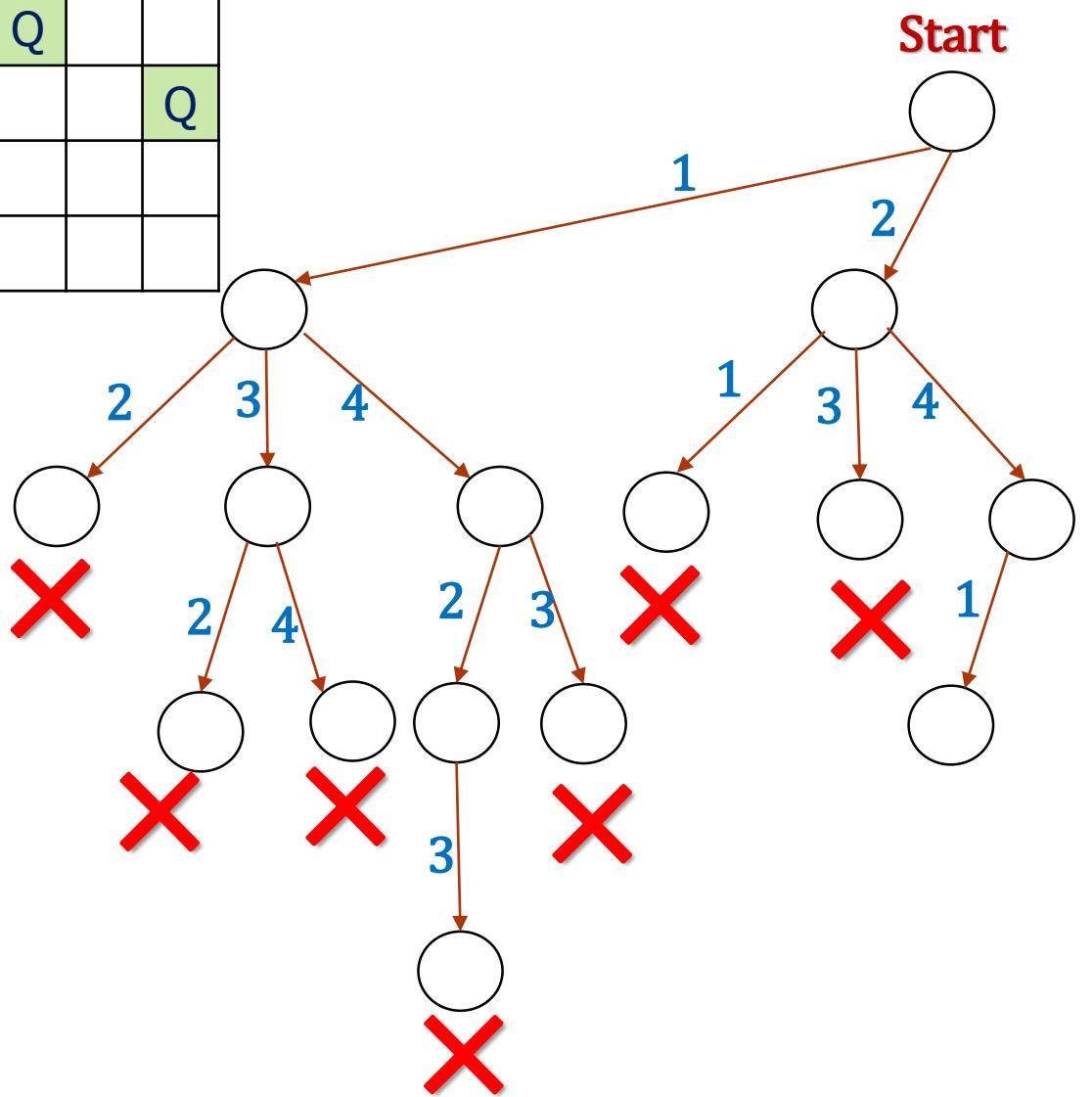
	1	2	3	4
1		Q		
2				
3				
4				



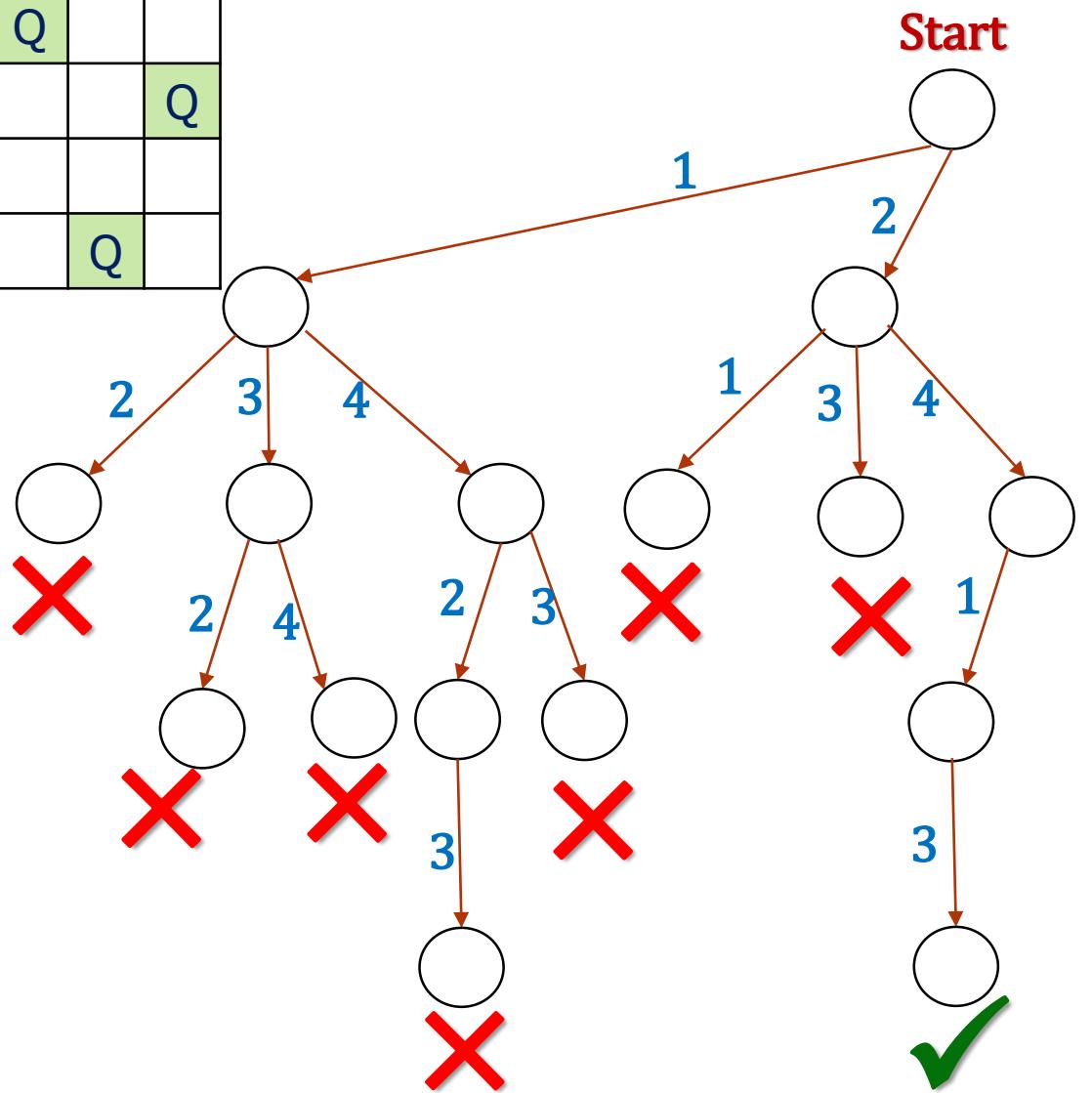
	1	2	3	4
1		Q		
2				Q
3				
4				



	1	2	3	4
1		Q		
2				Q
3	Q			
4				



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

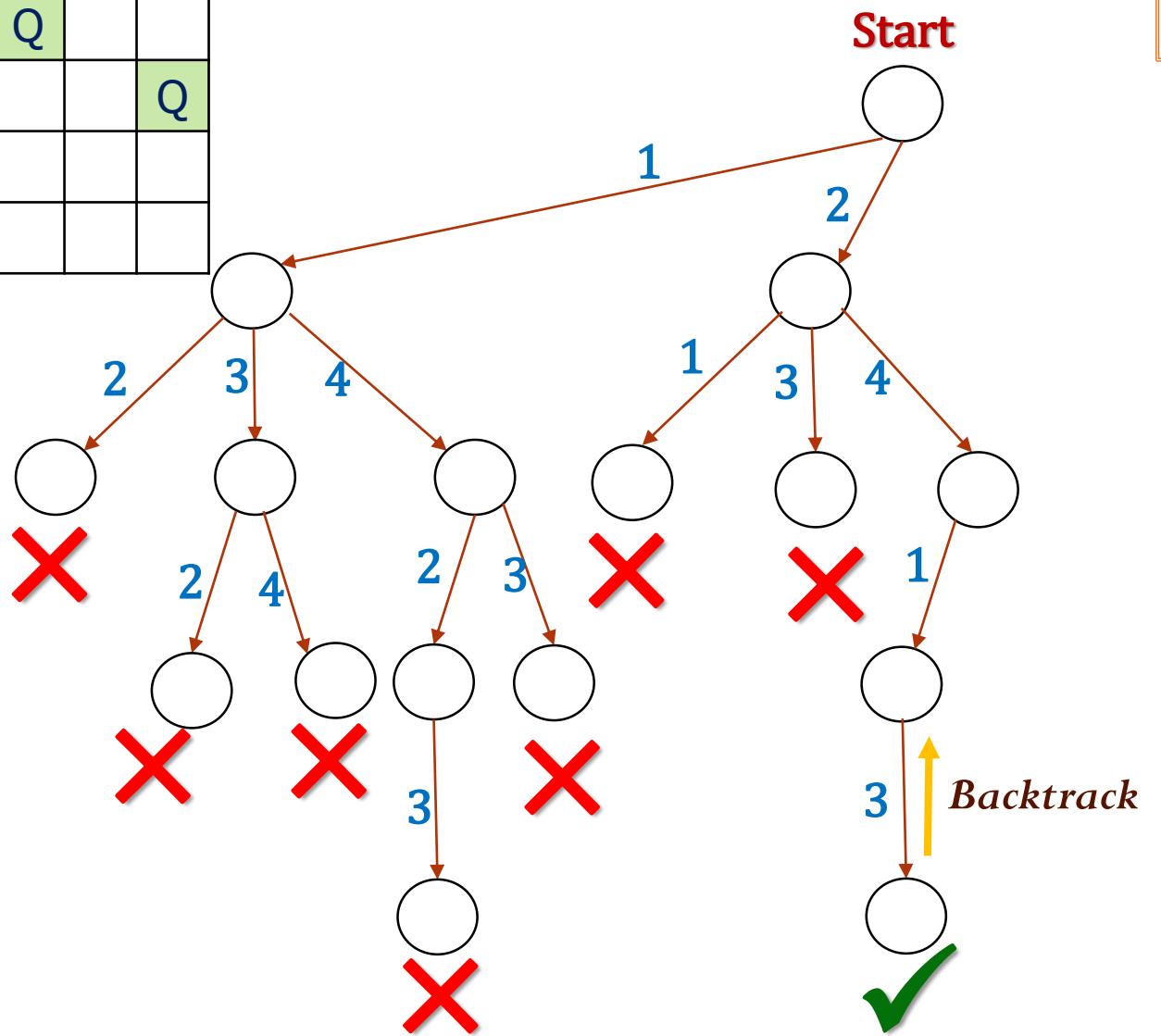


### Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				Q
3	Q			
4				

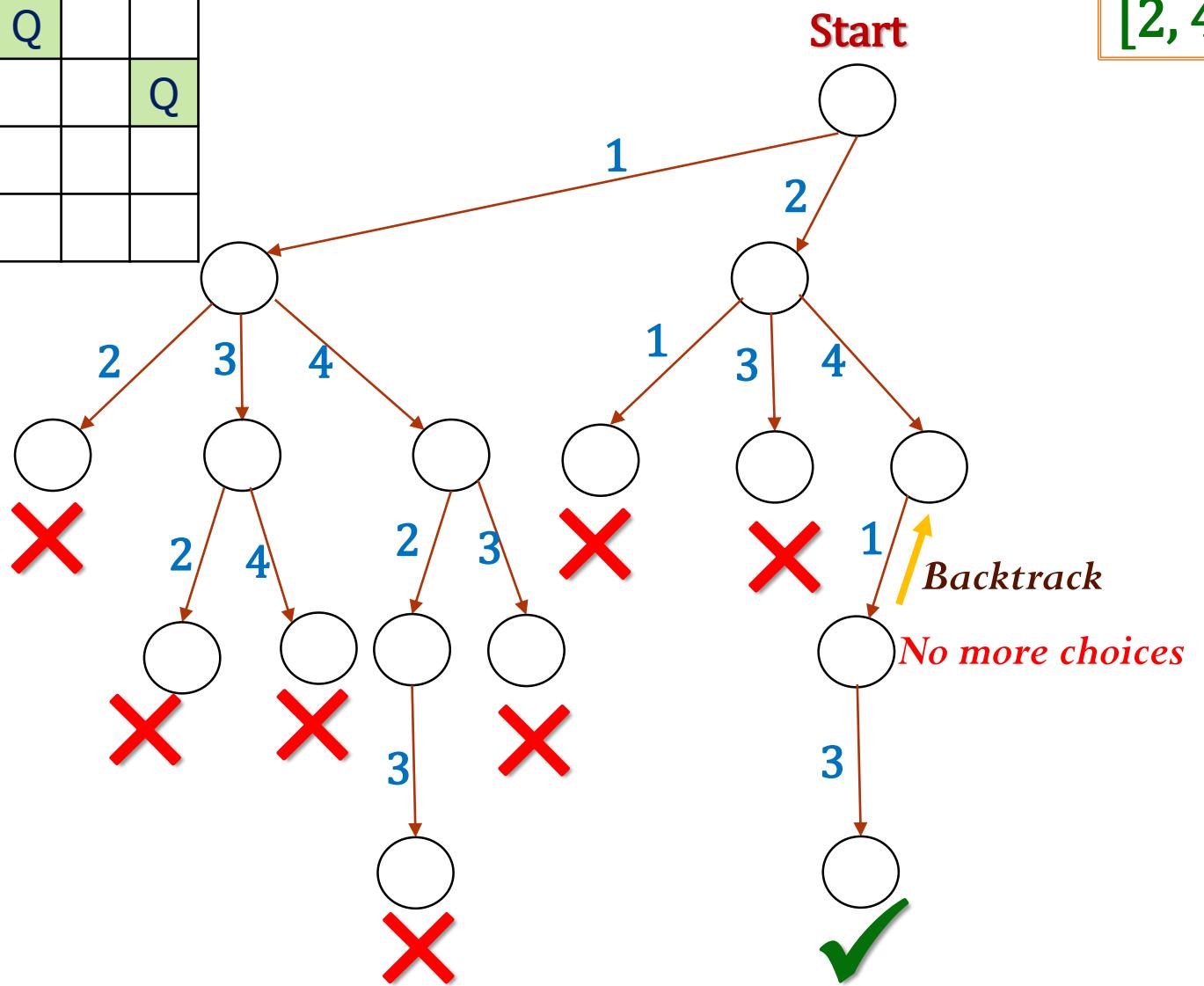


### Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				Q
3				
4				

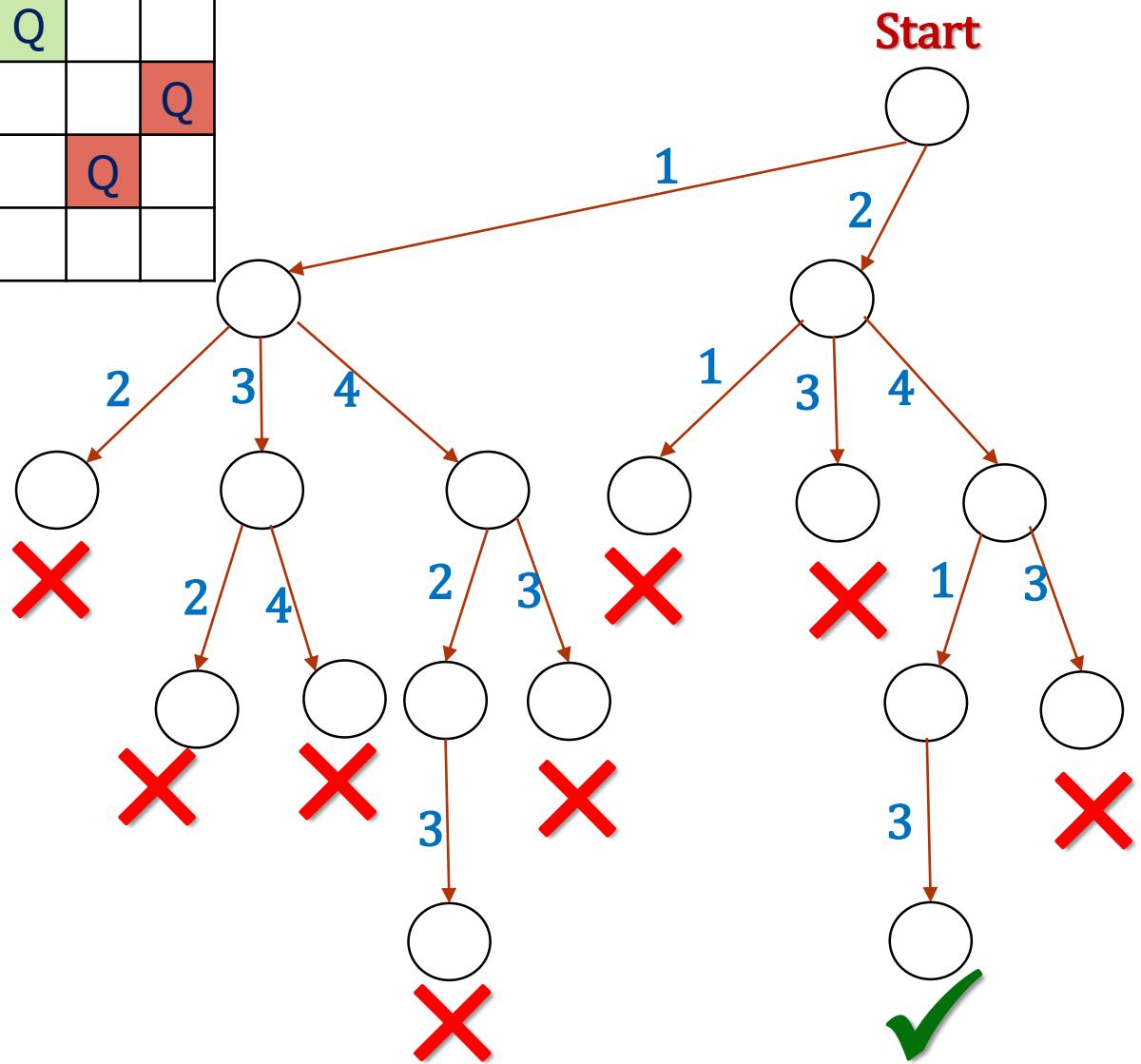


### Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				Q
3			Q	
4				

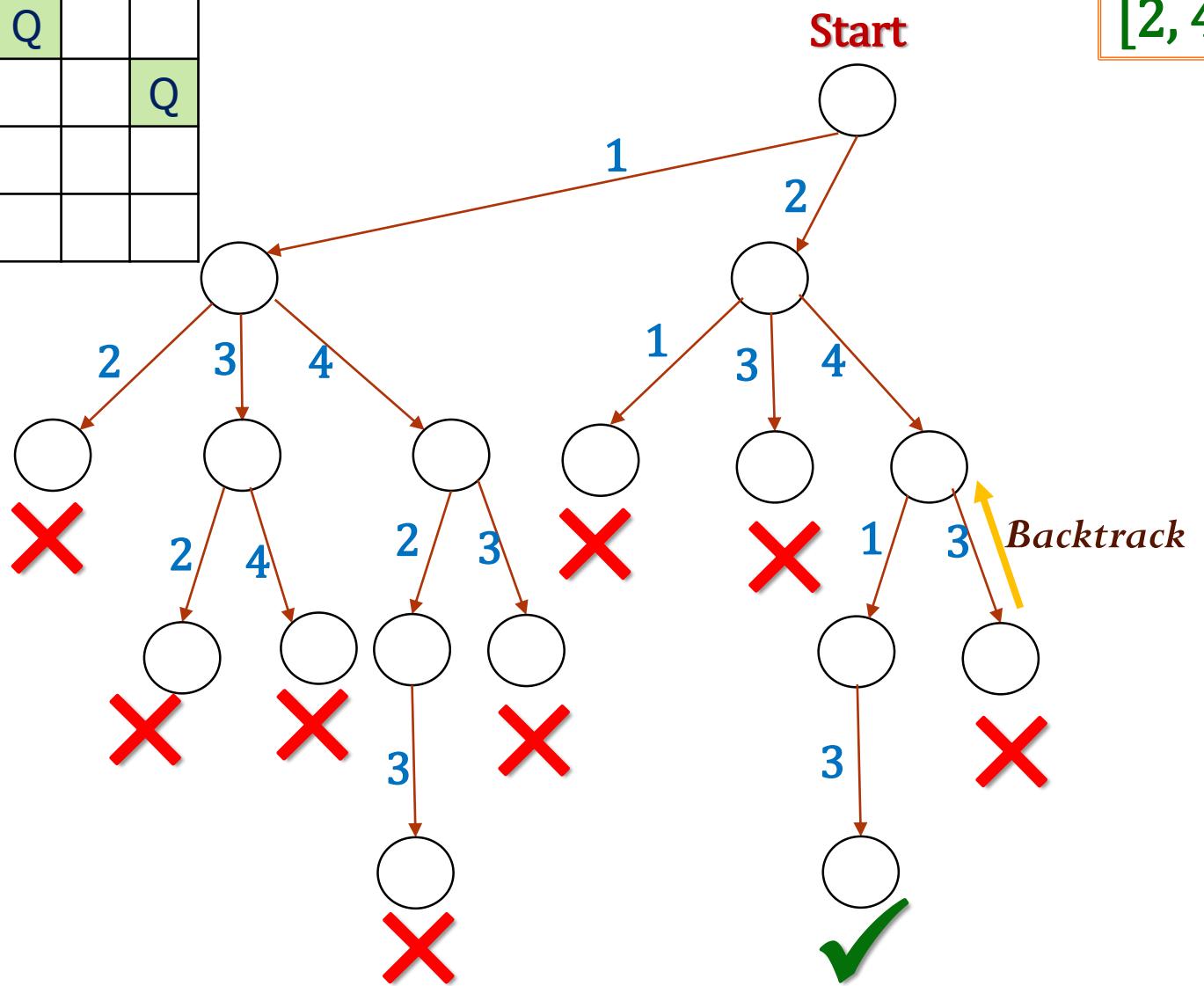


### Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				Q
3				
4				

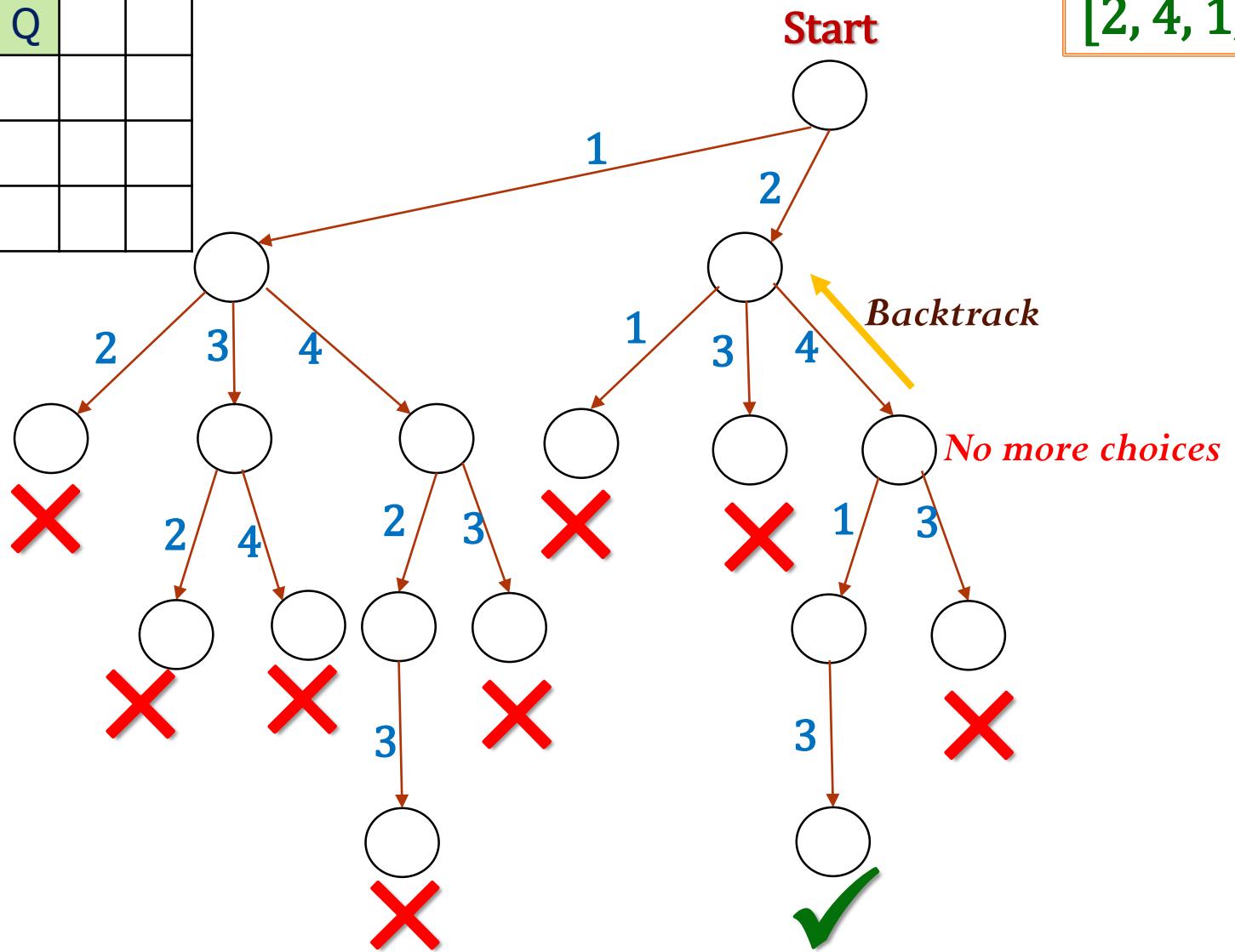


### Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				
3				
4				

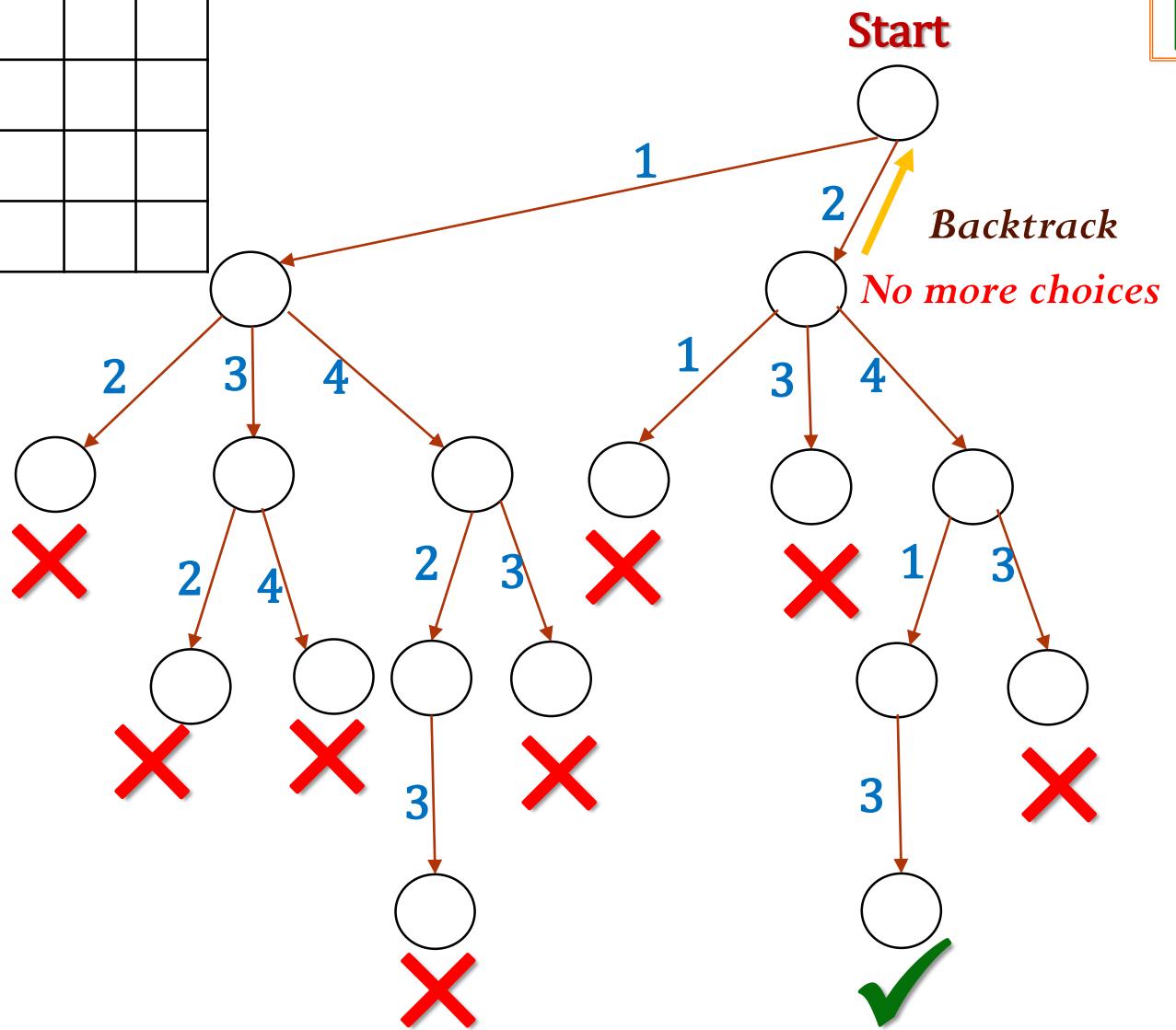


Solutions

[2, 4, 1, 3]



	1	2	3	4
1				
2				
3				
4				

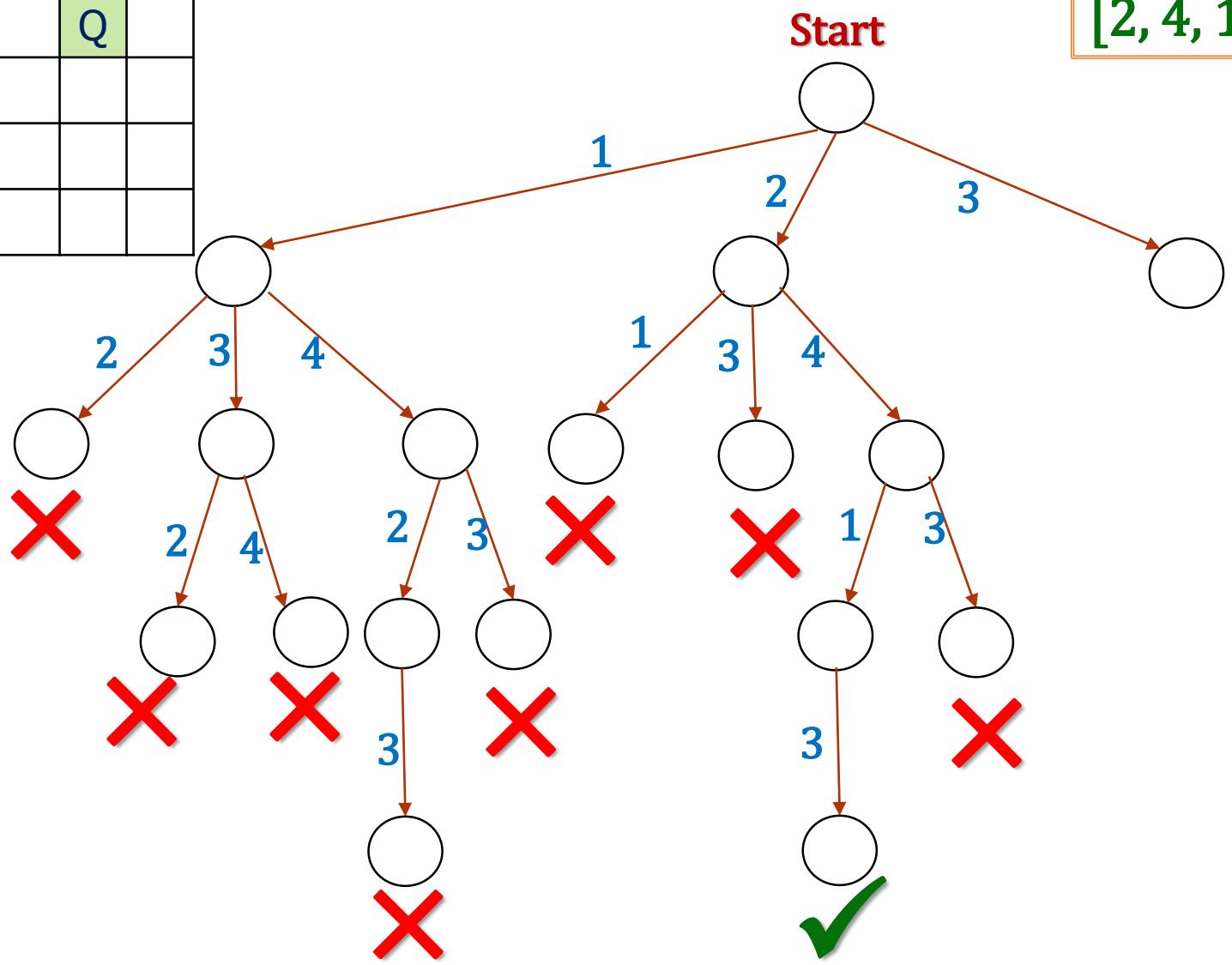


### Solutions

[2, 4, 1, 3]



	1	2	3	4
1			Q	
2				
3				
4				

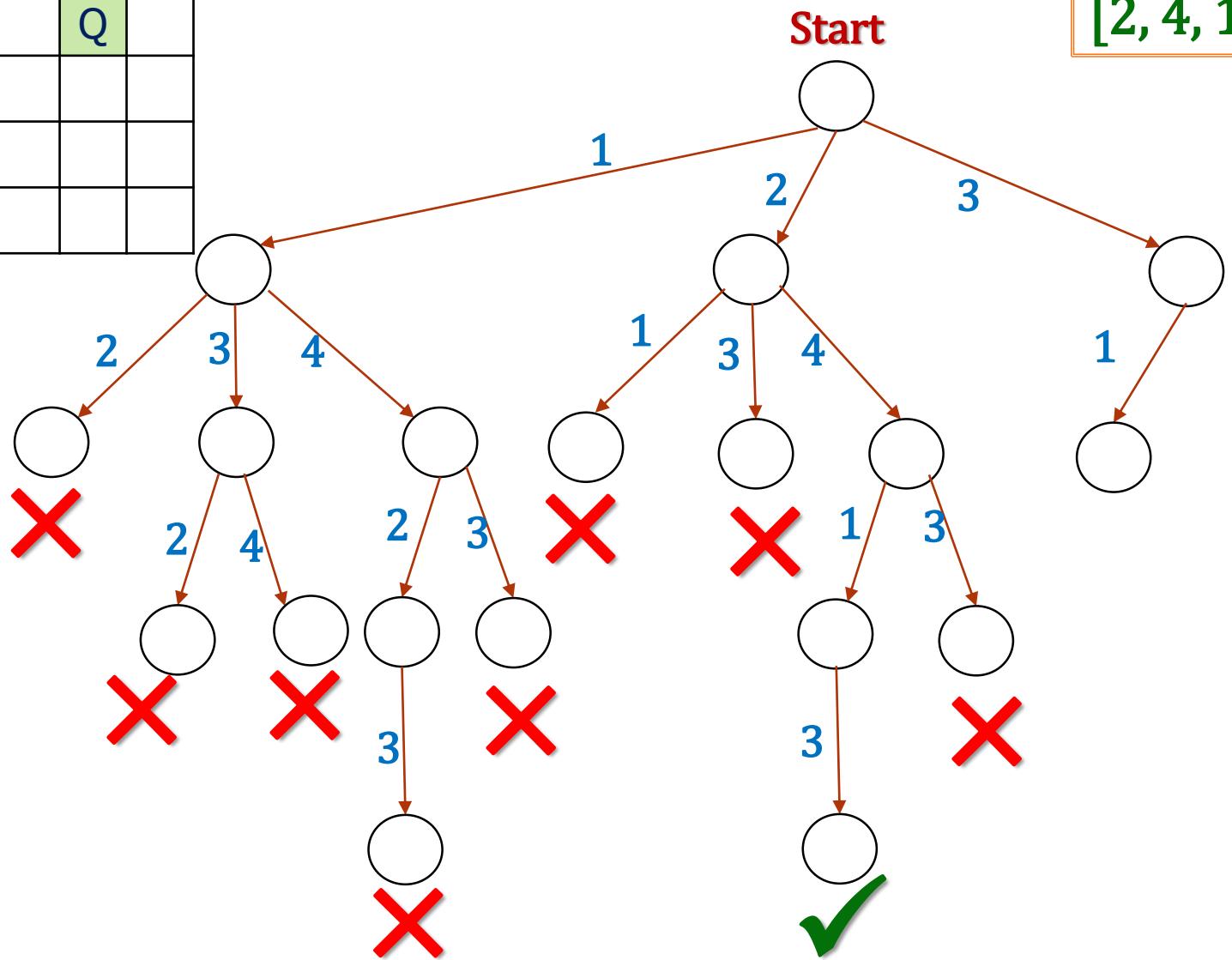


## Solutions

[2, 4, 1, 3]



	1	2	3	4
1			Q	
2	Q			
3				
4				



## Solutions

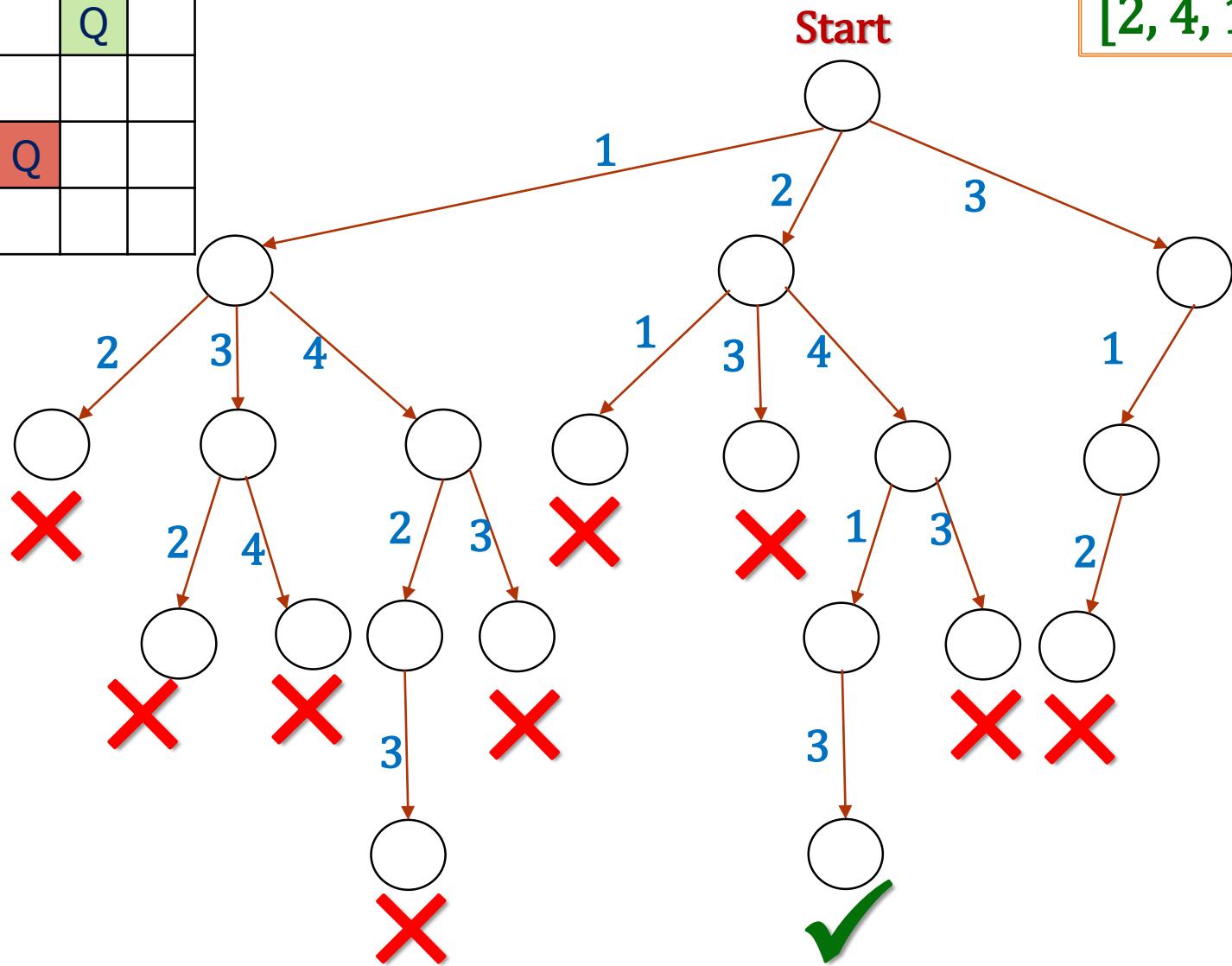
[2, 4, 1, 3]



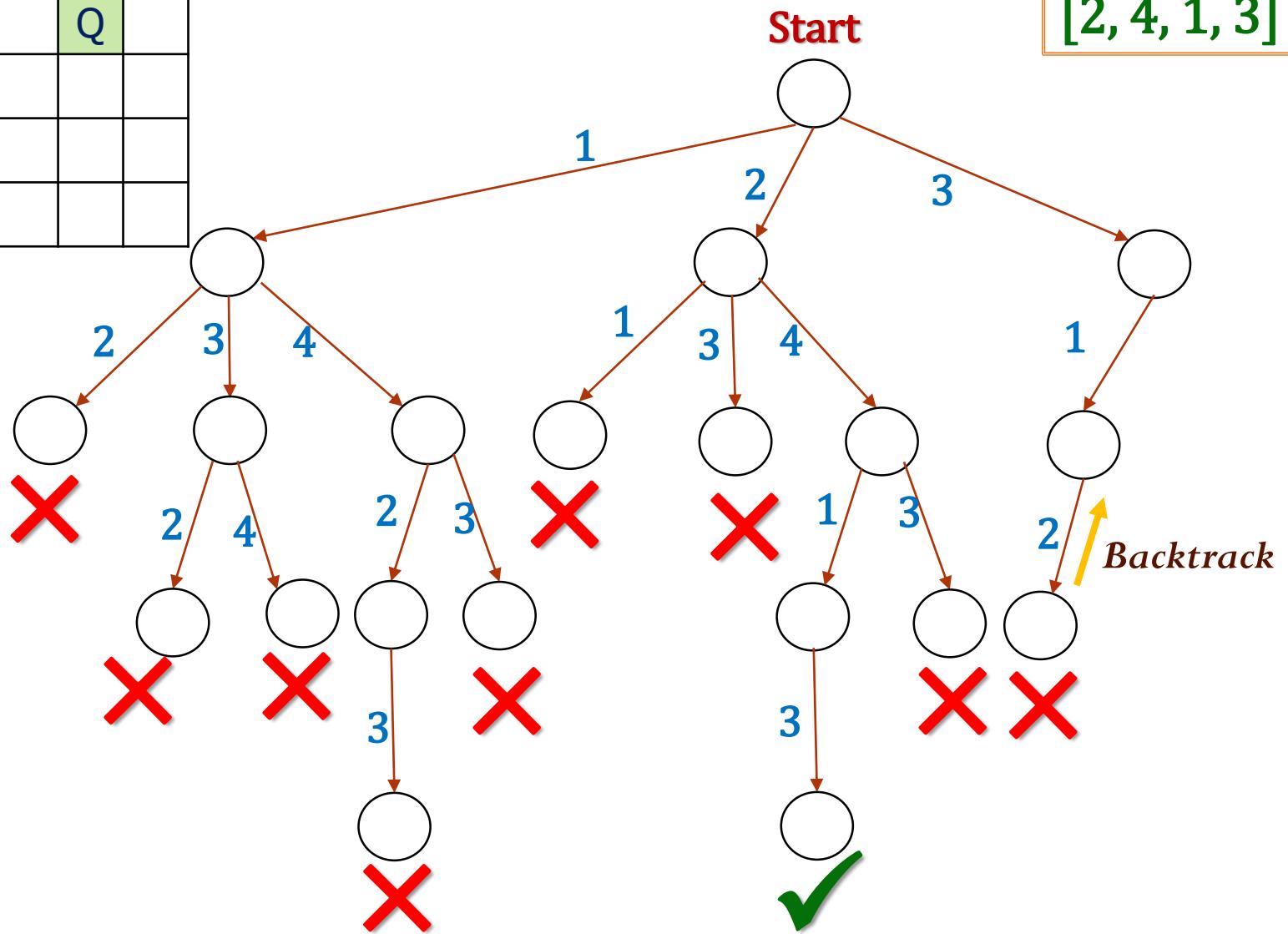
	1	2	3	4
1				Q
2	Q			
3		Q		
4				

### Solutions

[2, 4, 1, 3]



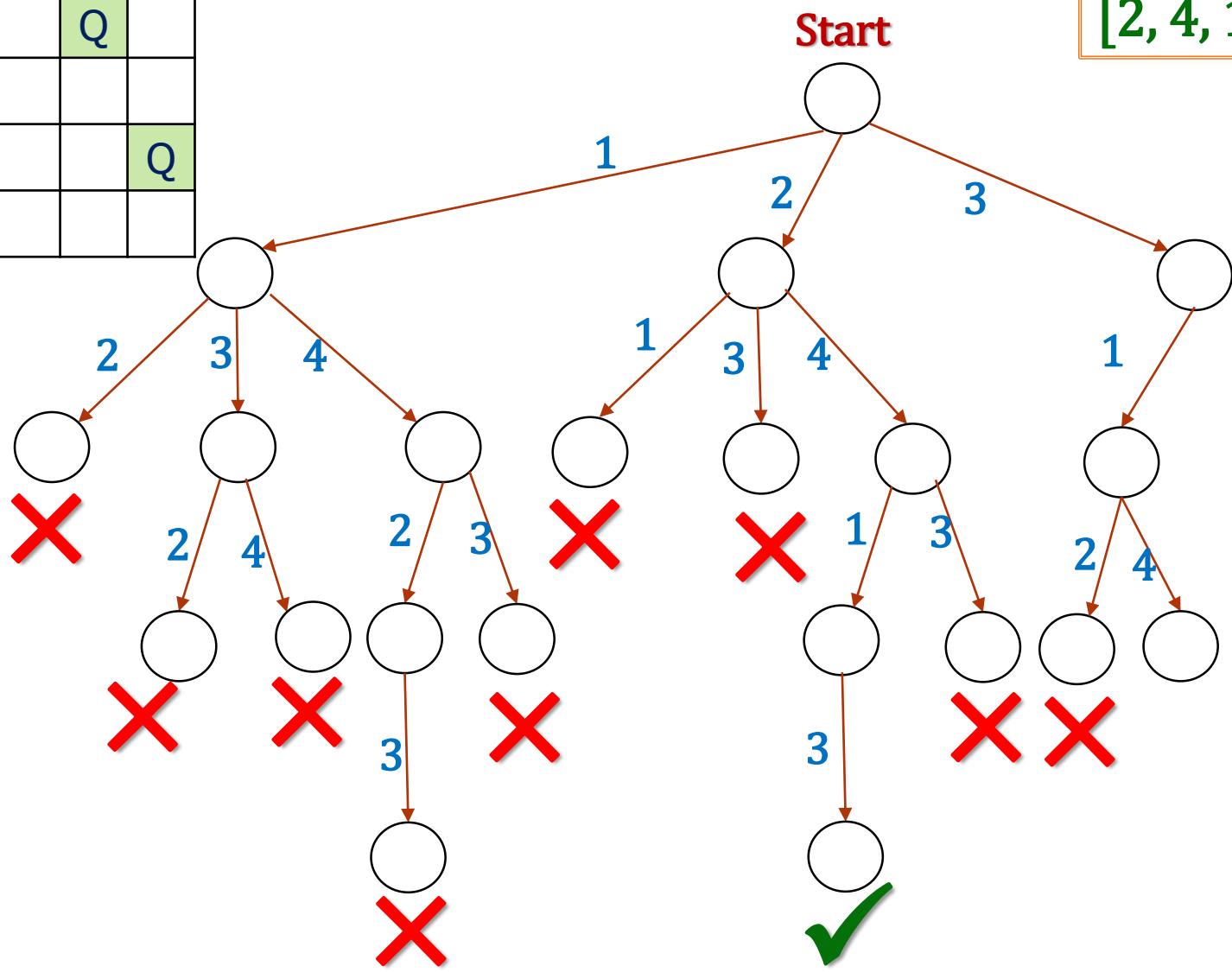
	1	2	3	4
1			Q	
2	Q			
3				
4				



### Solutions

[2, 4, 1, 3]

	1	2	3	4
1			Q	
2	Q			
3				Q
4				

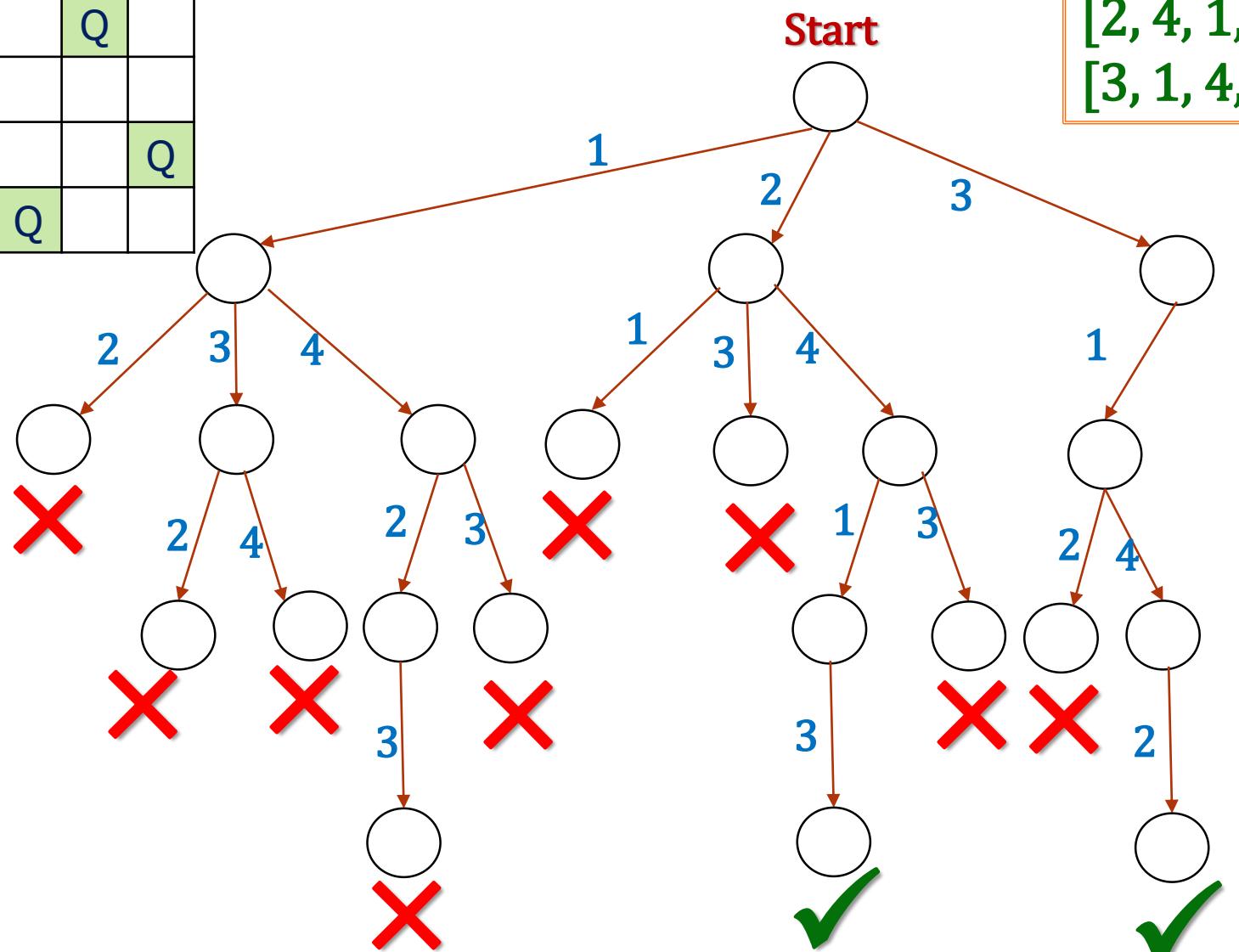


## Solutions

[2, 4, 1, 3]



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

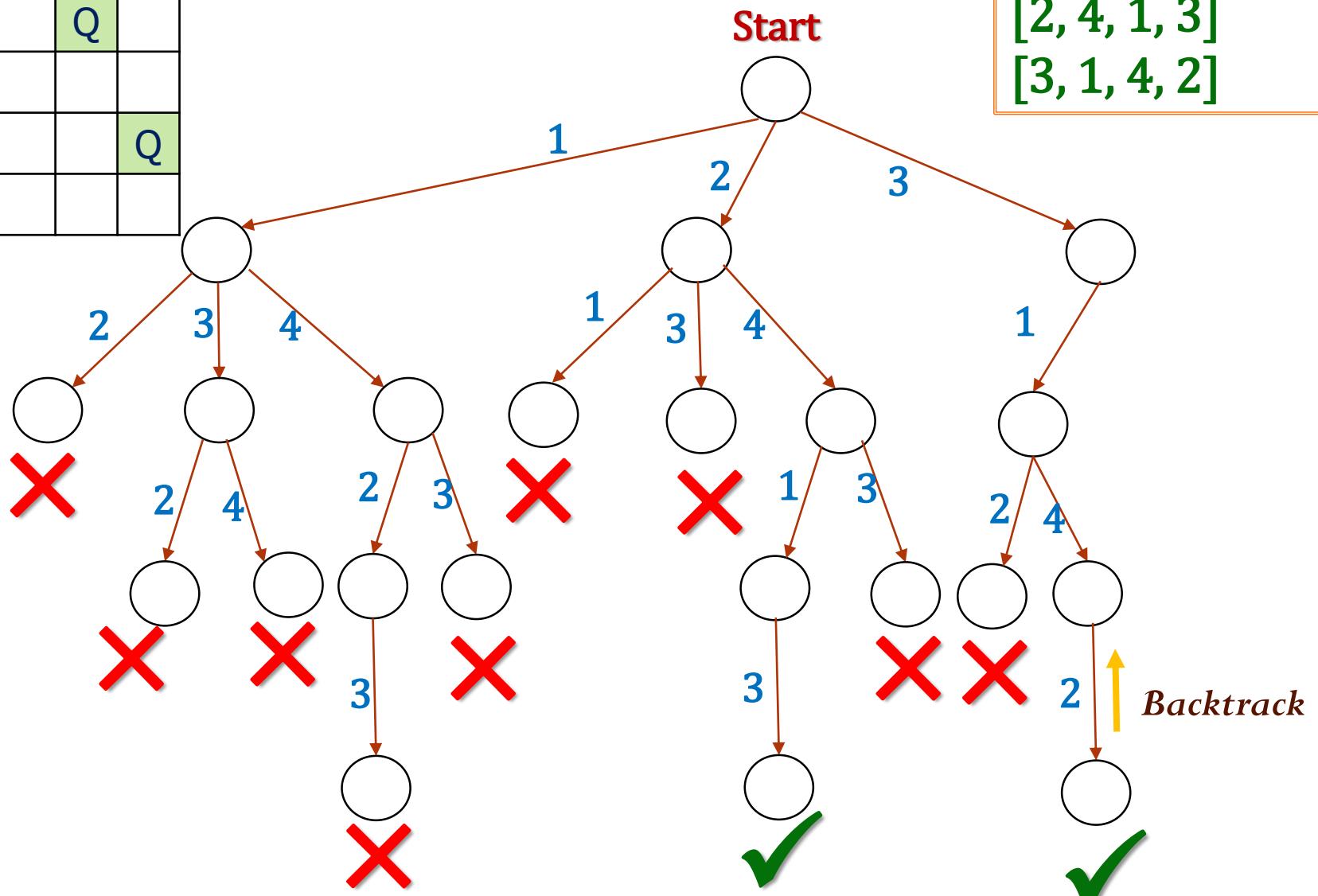


## Solutions

$[2, 4, 1, 3]$   
 $[3, 1, 4, 2]$



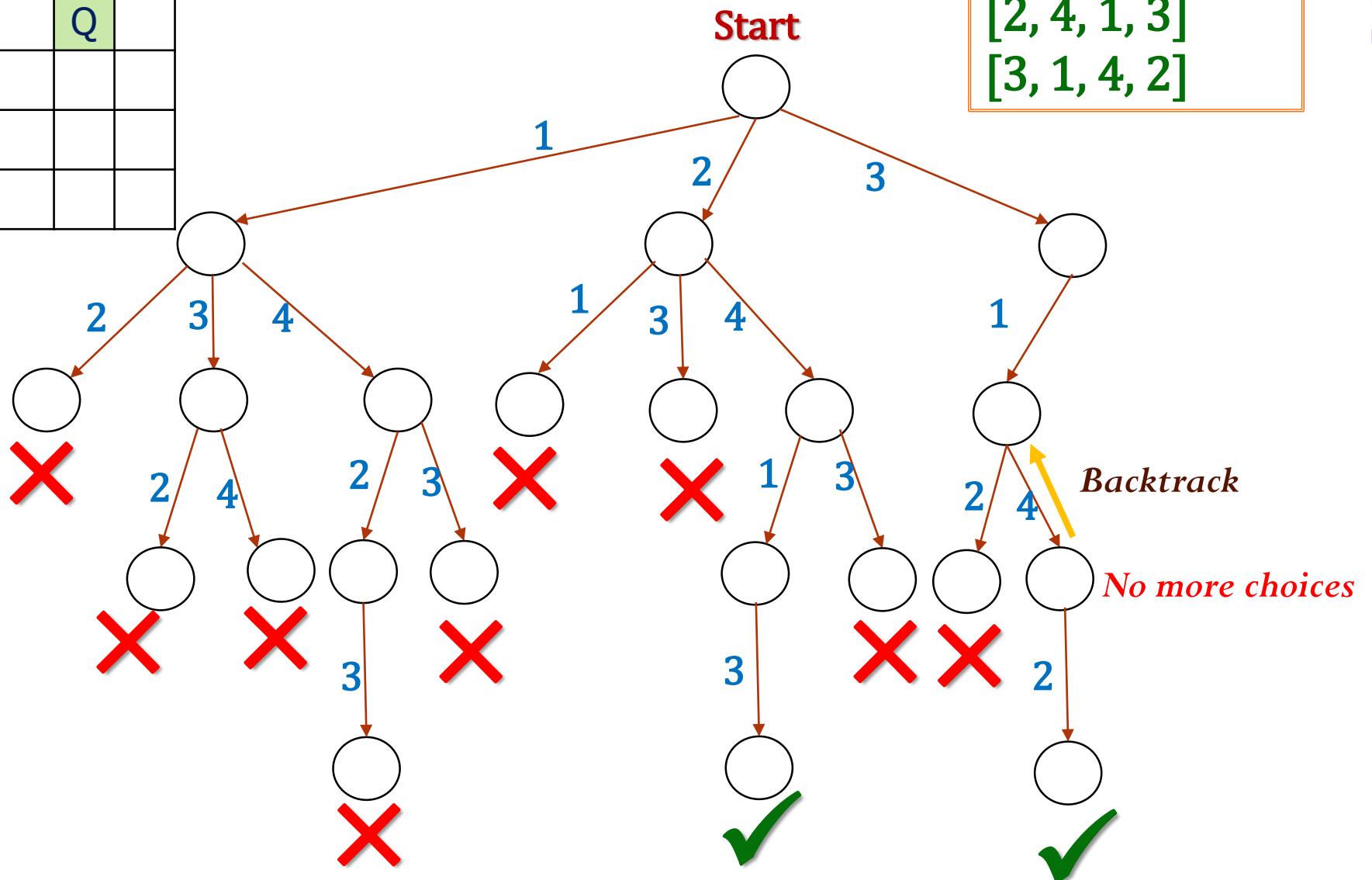
	1	2	3	4
1			Q	
2	Q			
3				Q
4				



### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]

	1	2	3	4
1			Q	
2	Q			
3				
4				



### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]

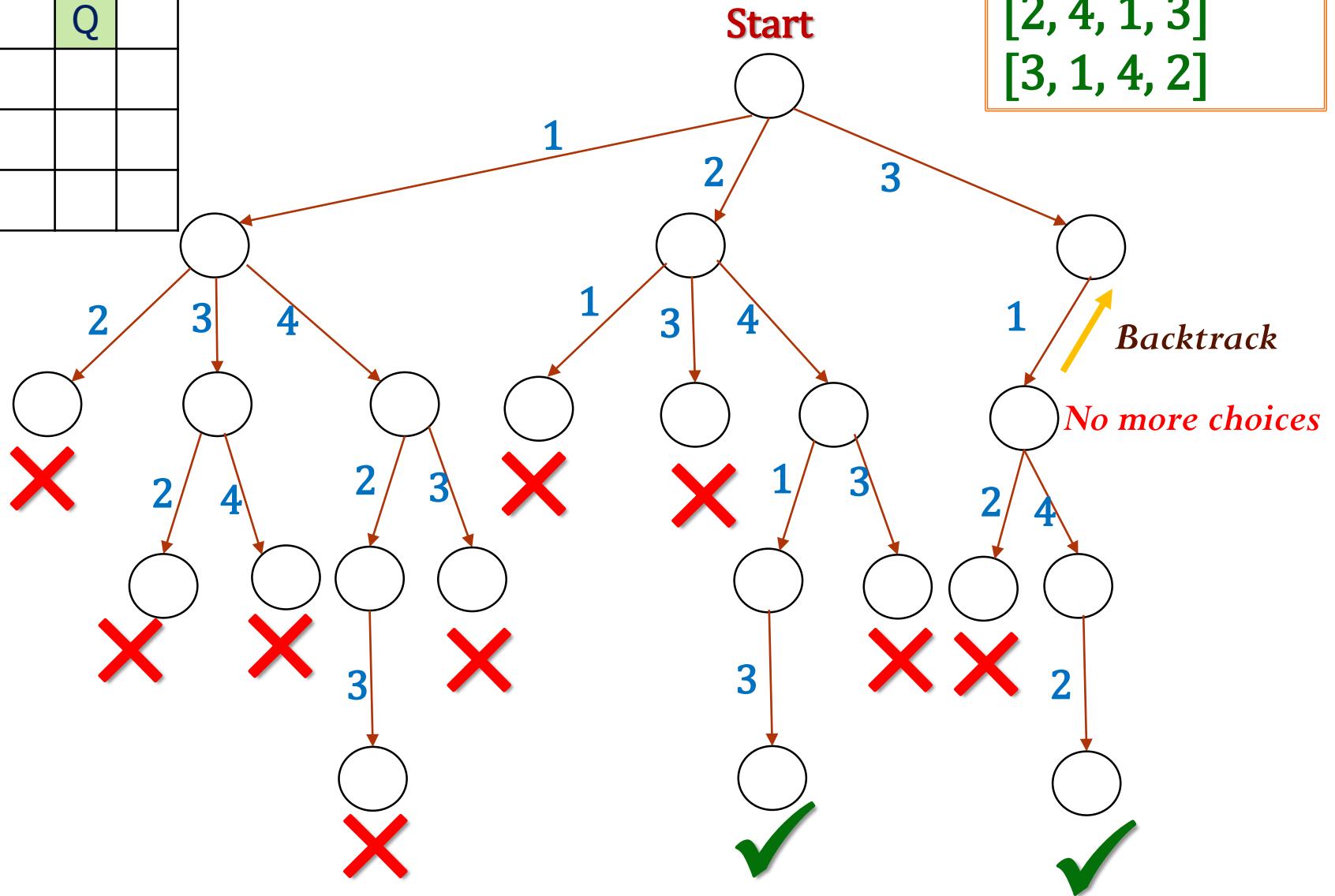


	1	2	3	4
1			Q	
2				
3				
4				

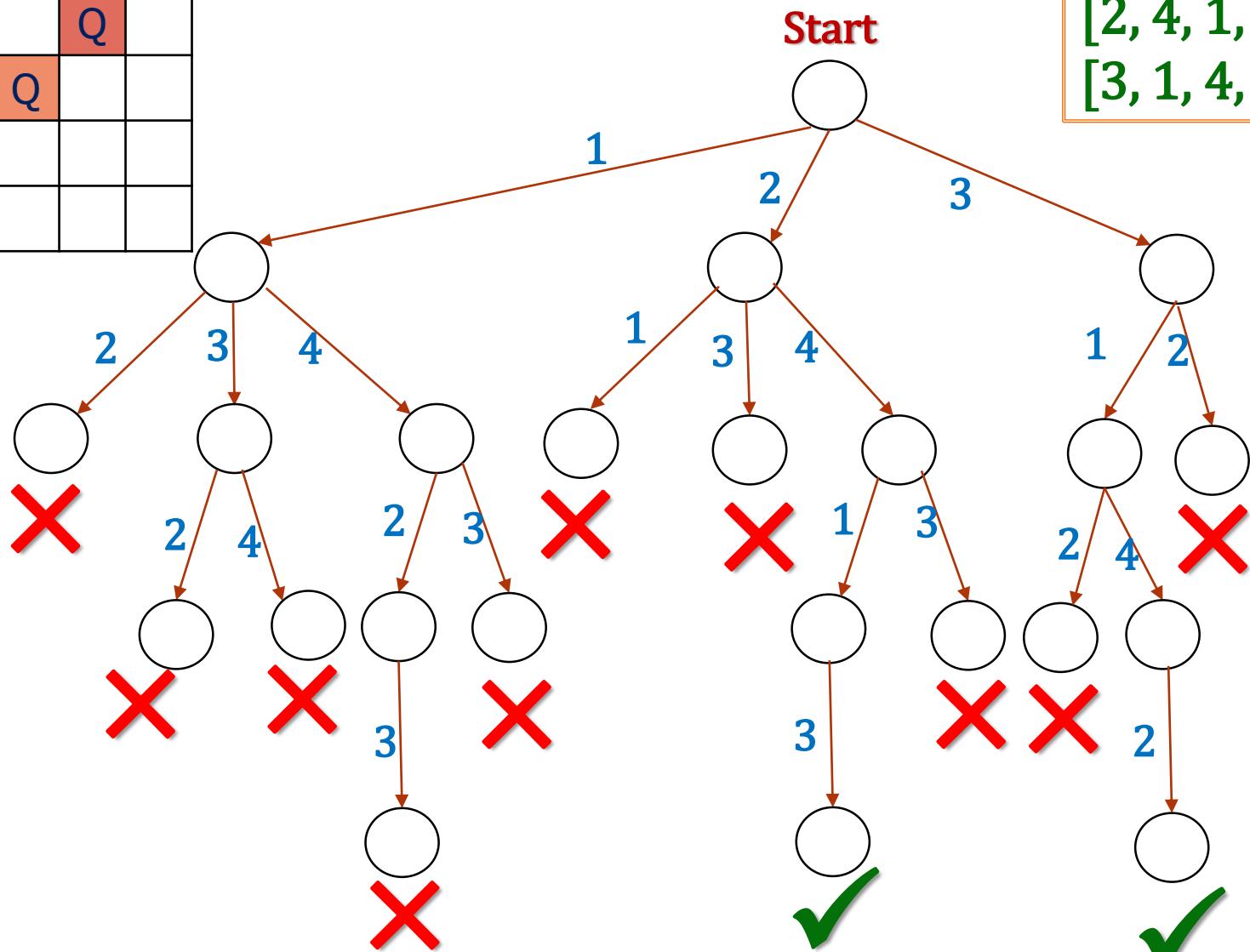
Start

### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1			Q	
2		Q		
3				
4				

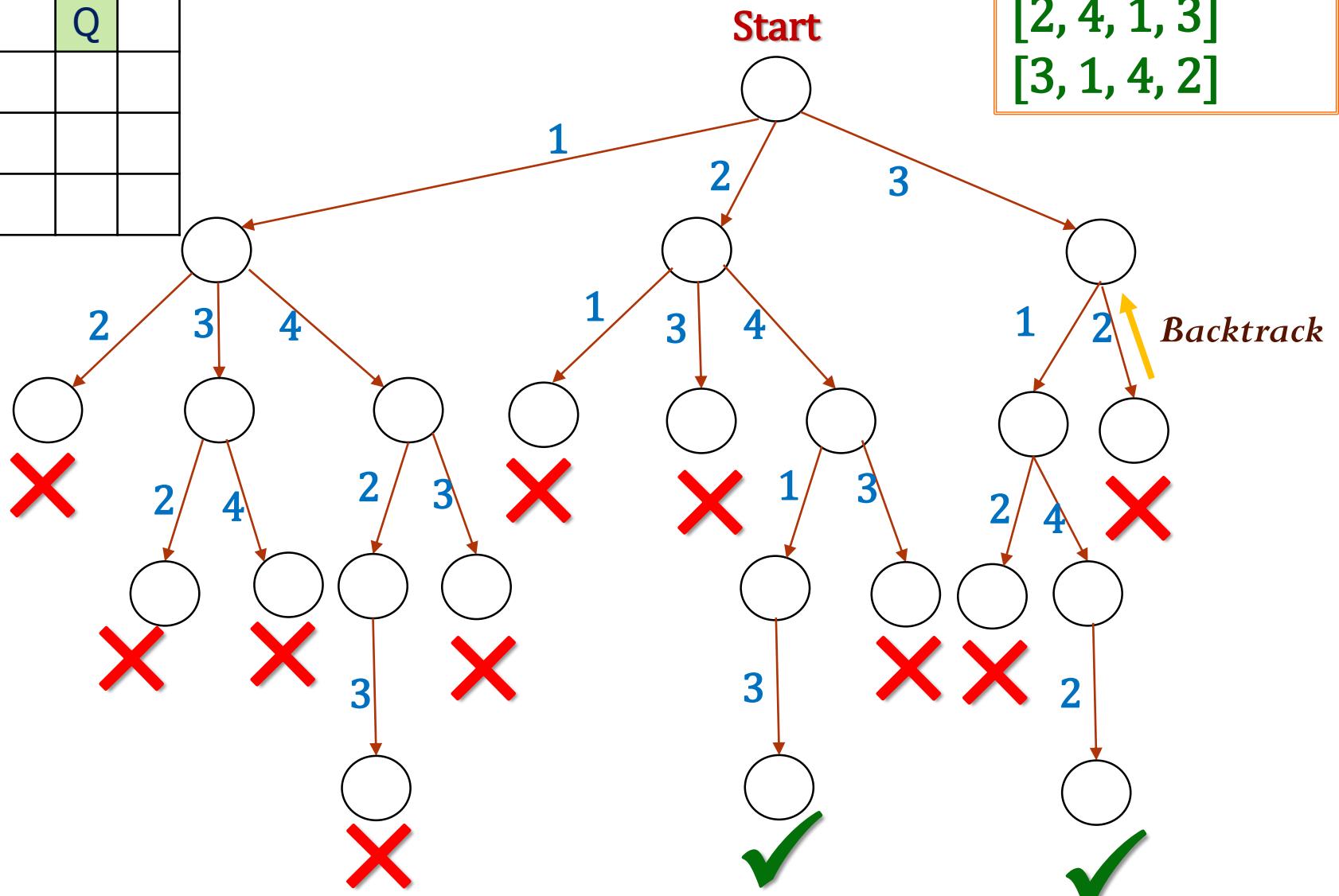


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



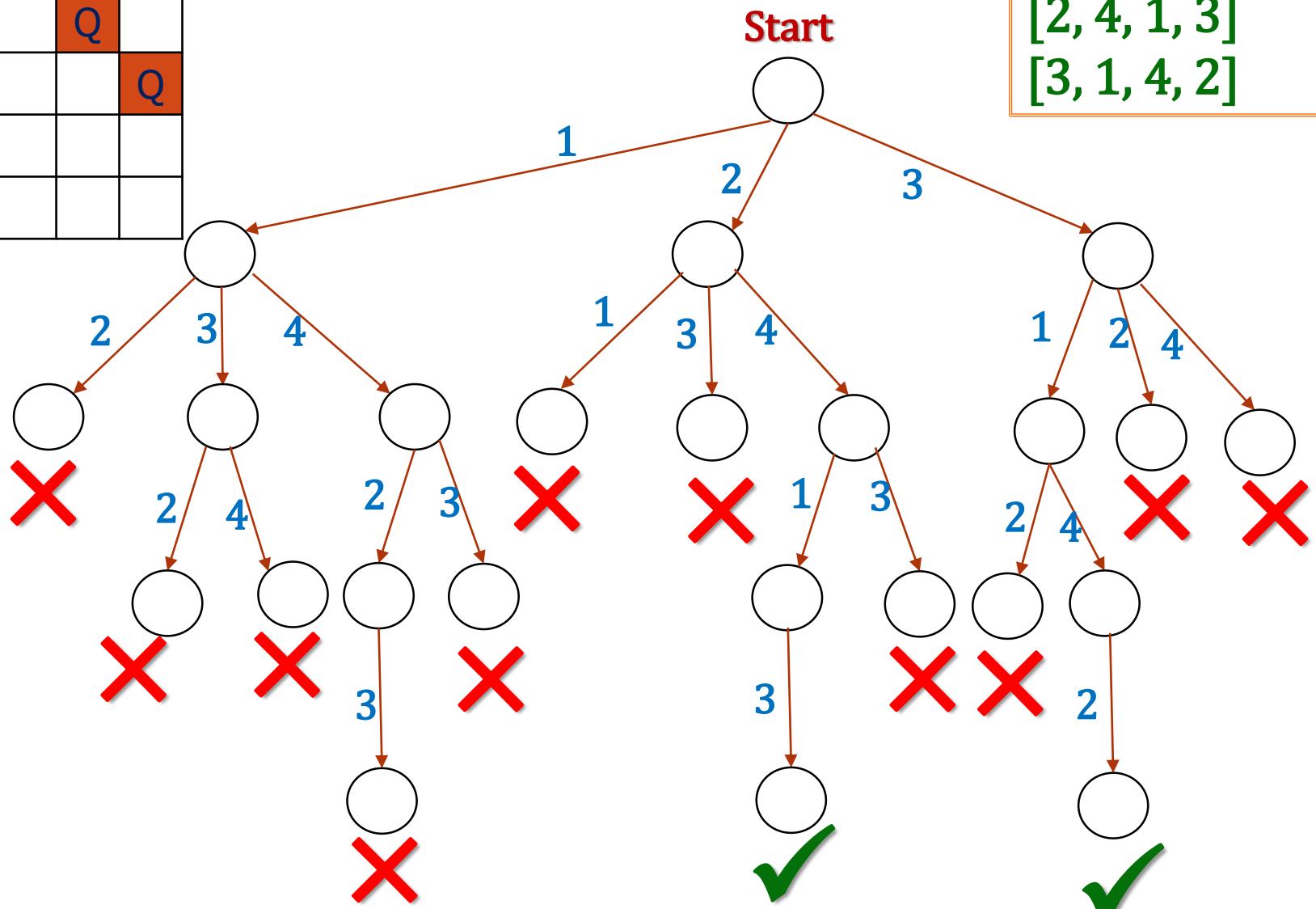
	1	2	3	4
1			Q	
2				
3				
4				



### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]

	1	2	3	4
1			Q	
2				Q
3				
4				

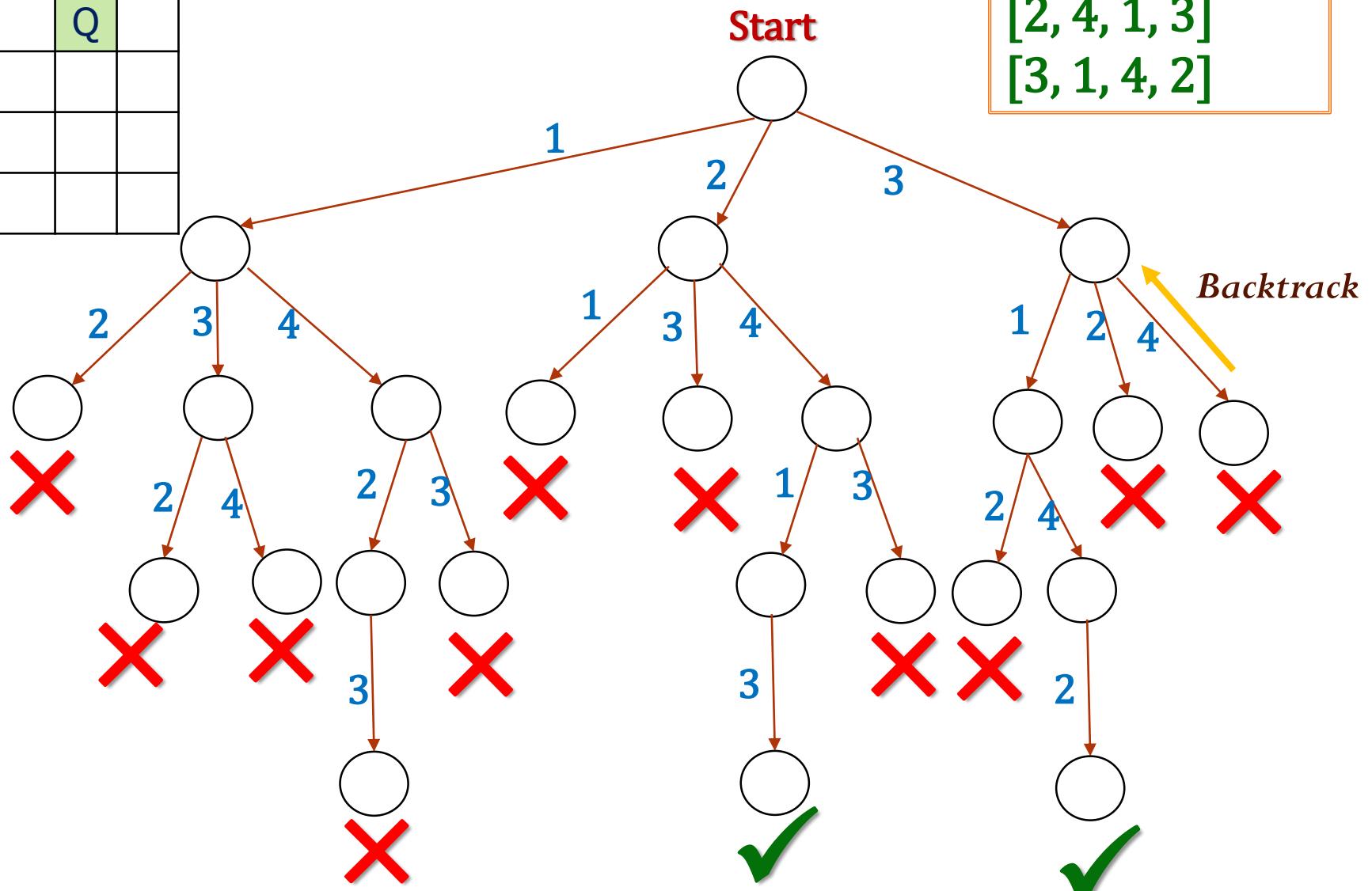


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



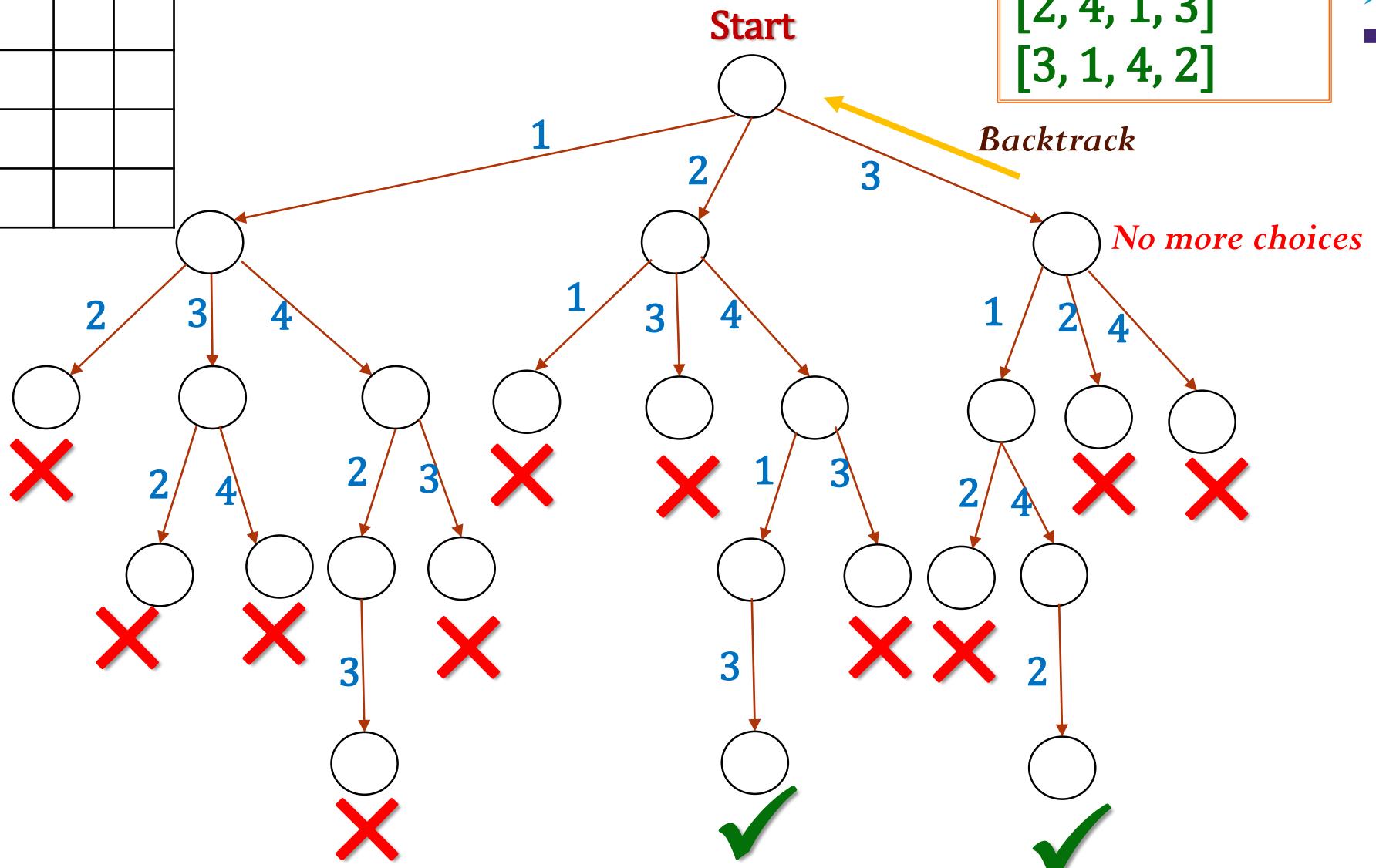
	1	2	3	4
1			Q	
2				
3				
4				



### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]

	1	2	3	4
1				
2				
3				
4				

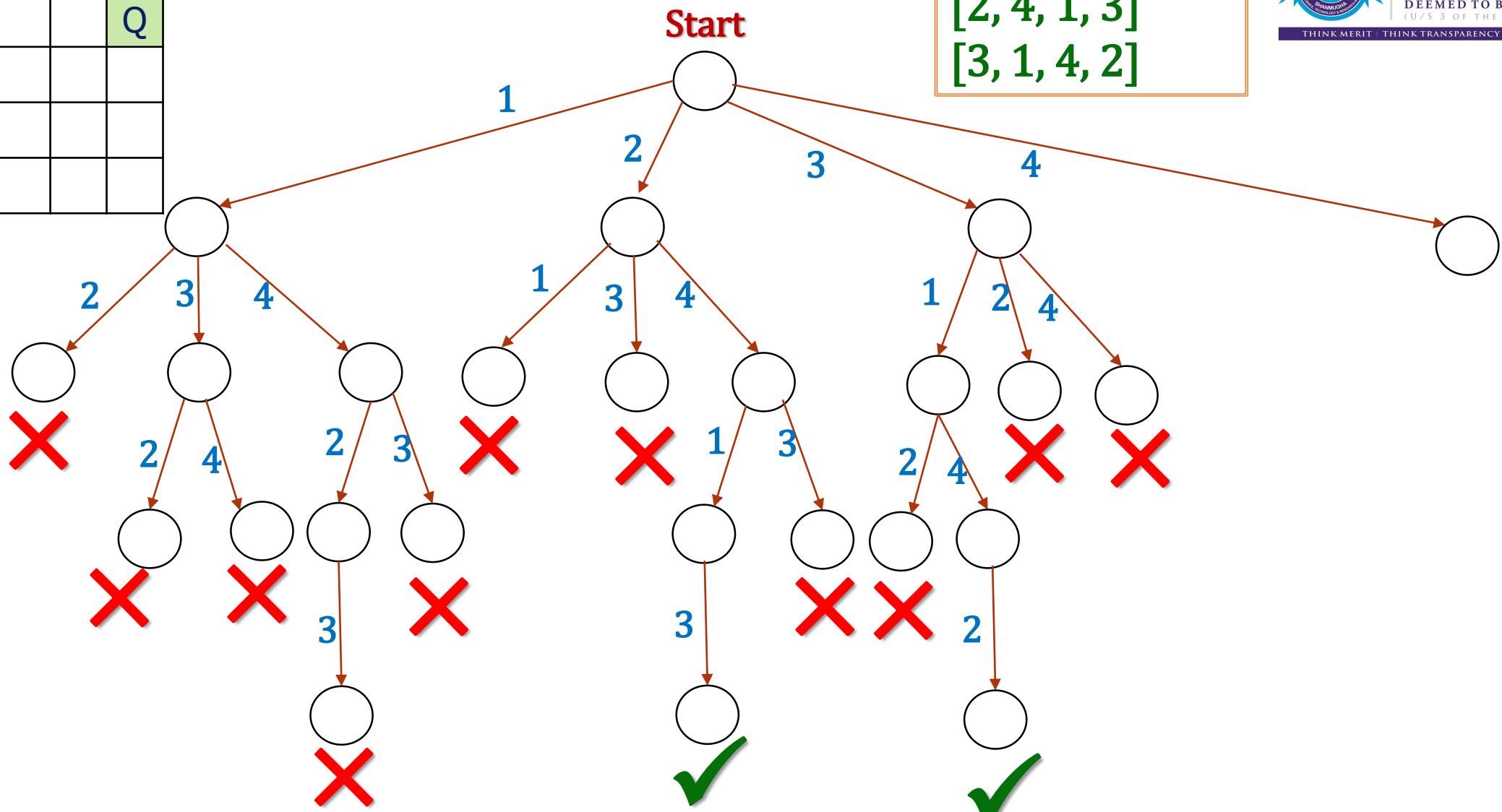


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2				
3				
4				

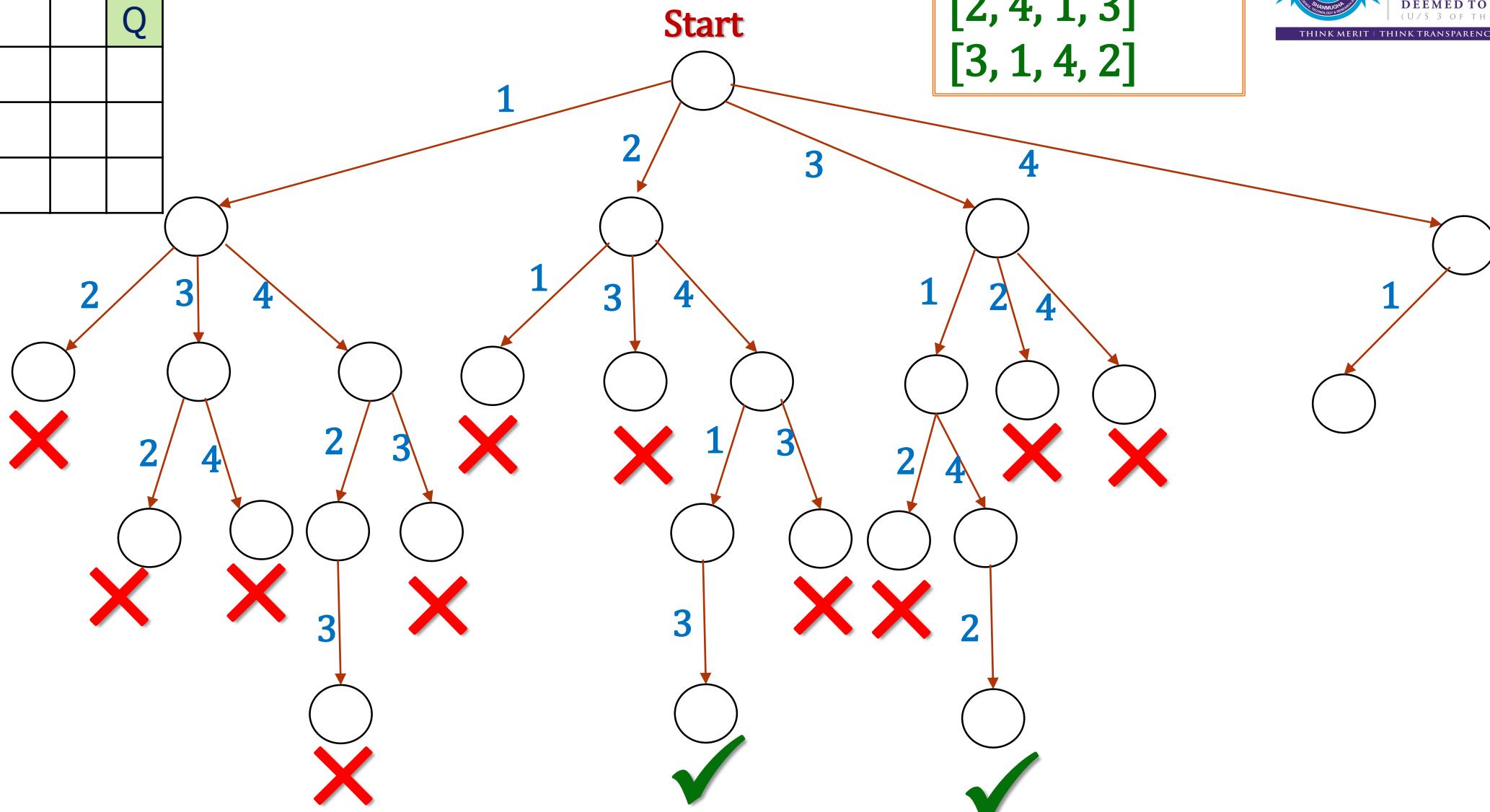


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				
2	Q			
3				
4				

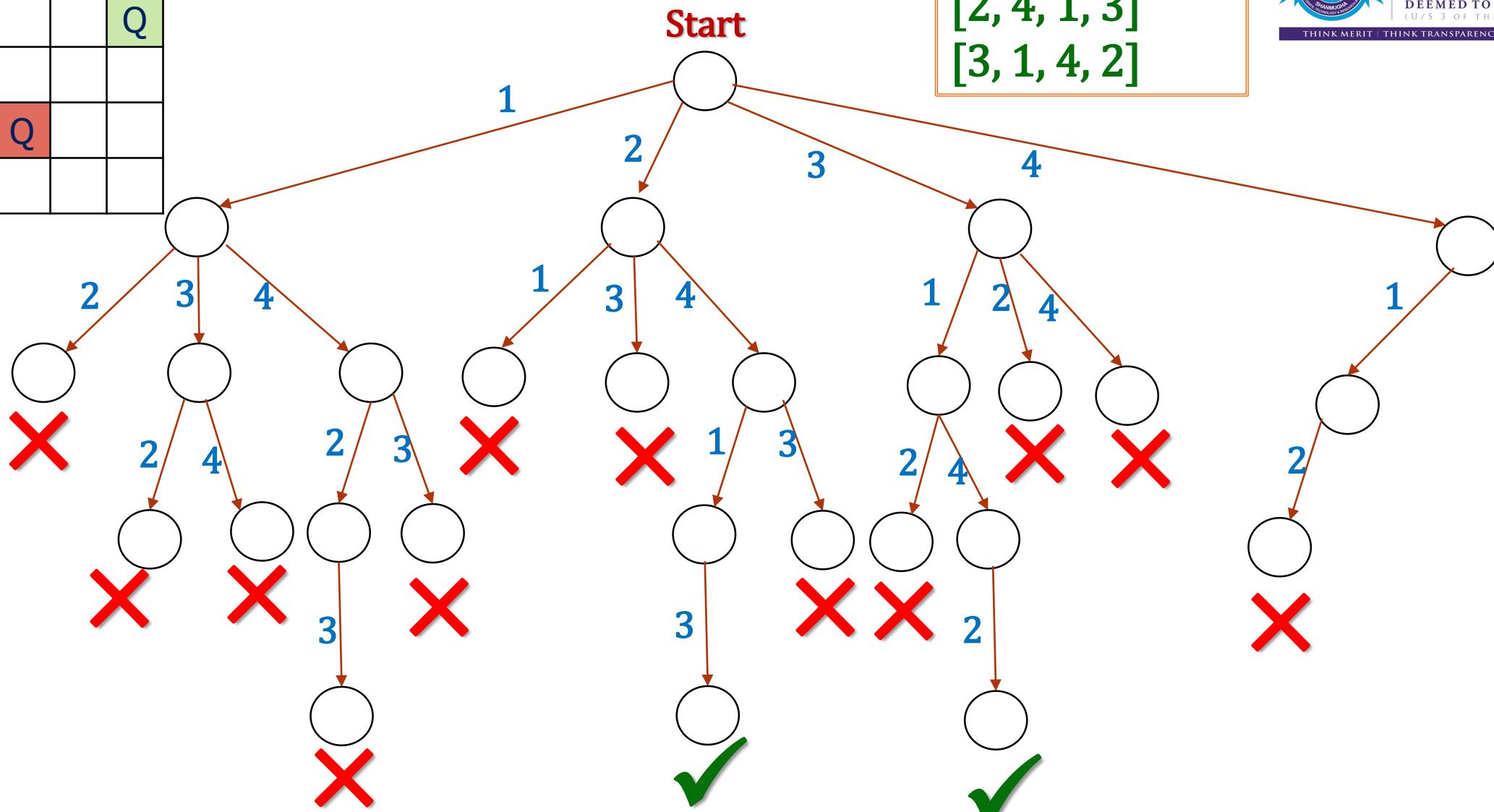


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2	Q			
3		Q		
4				

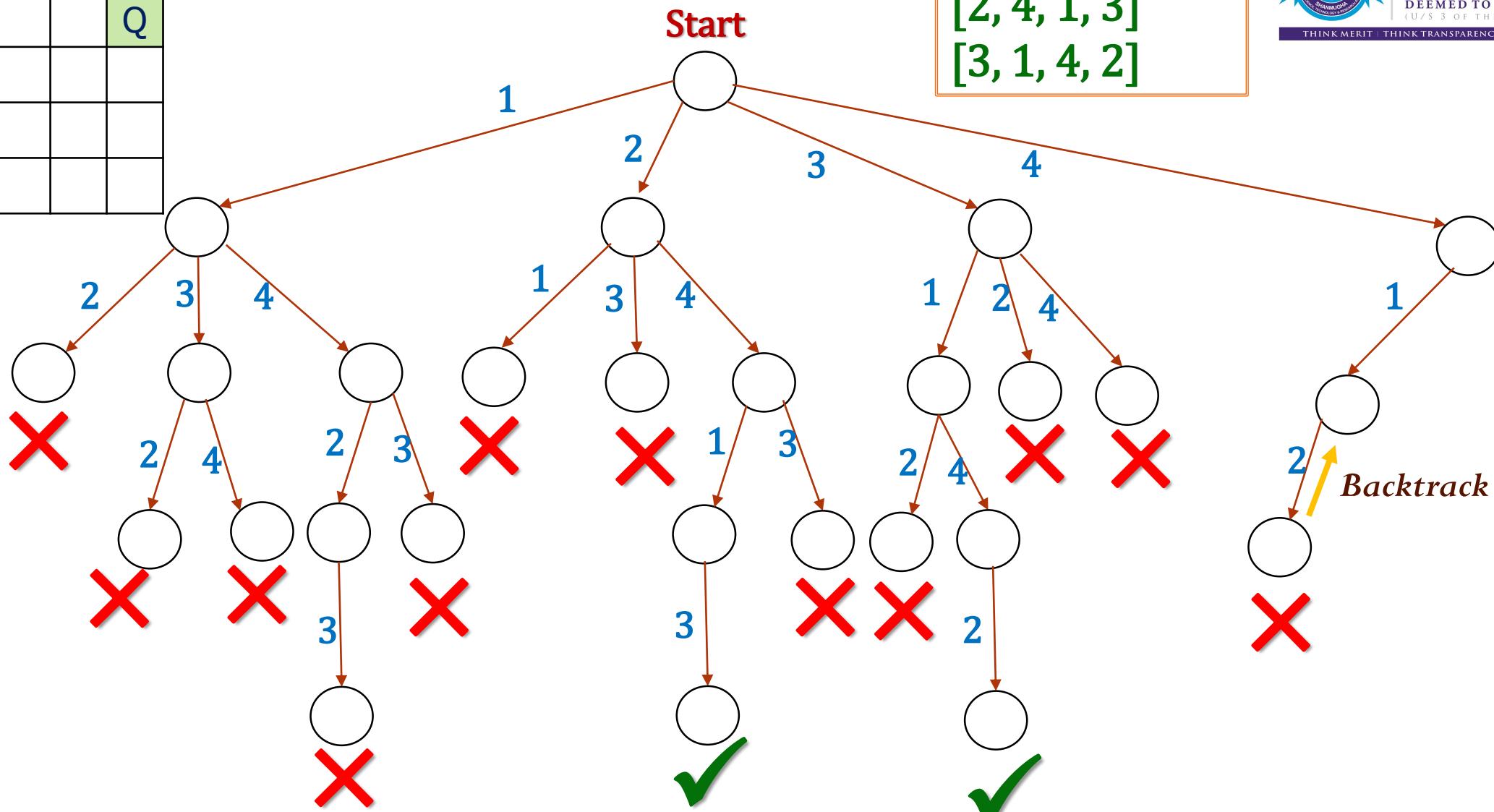


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				
2	Q			
3				
4				

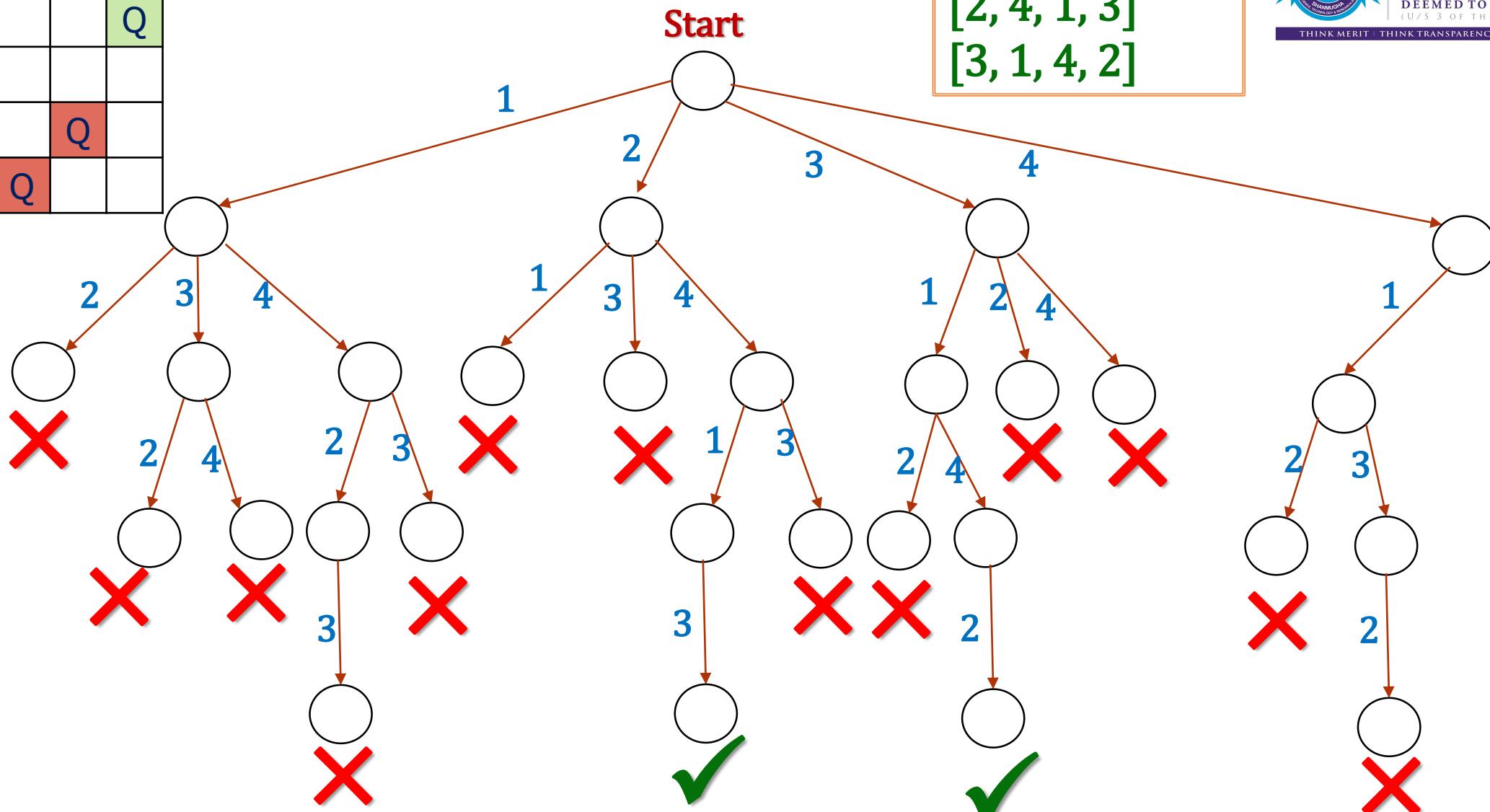


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



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

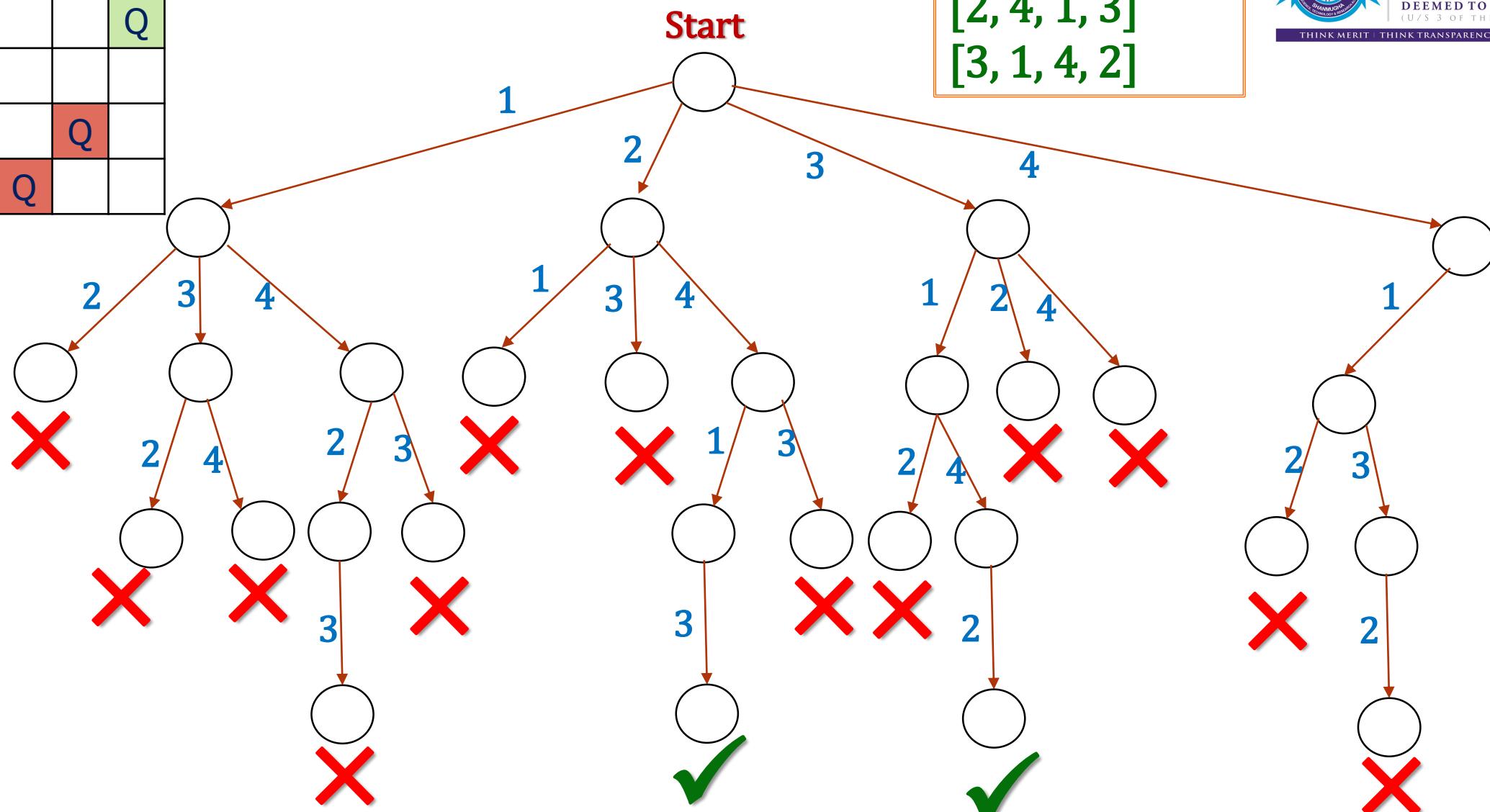


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



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



## Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]

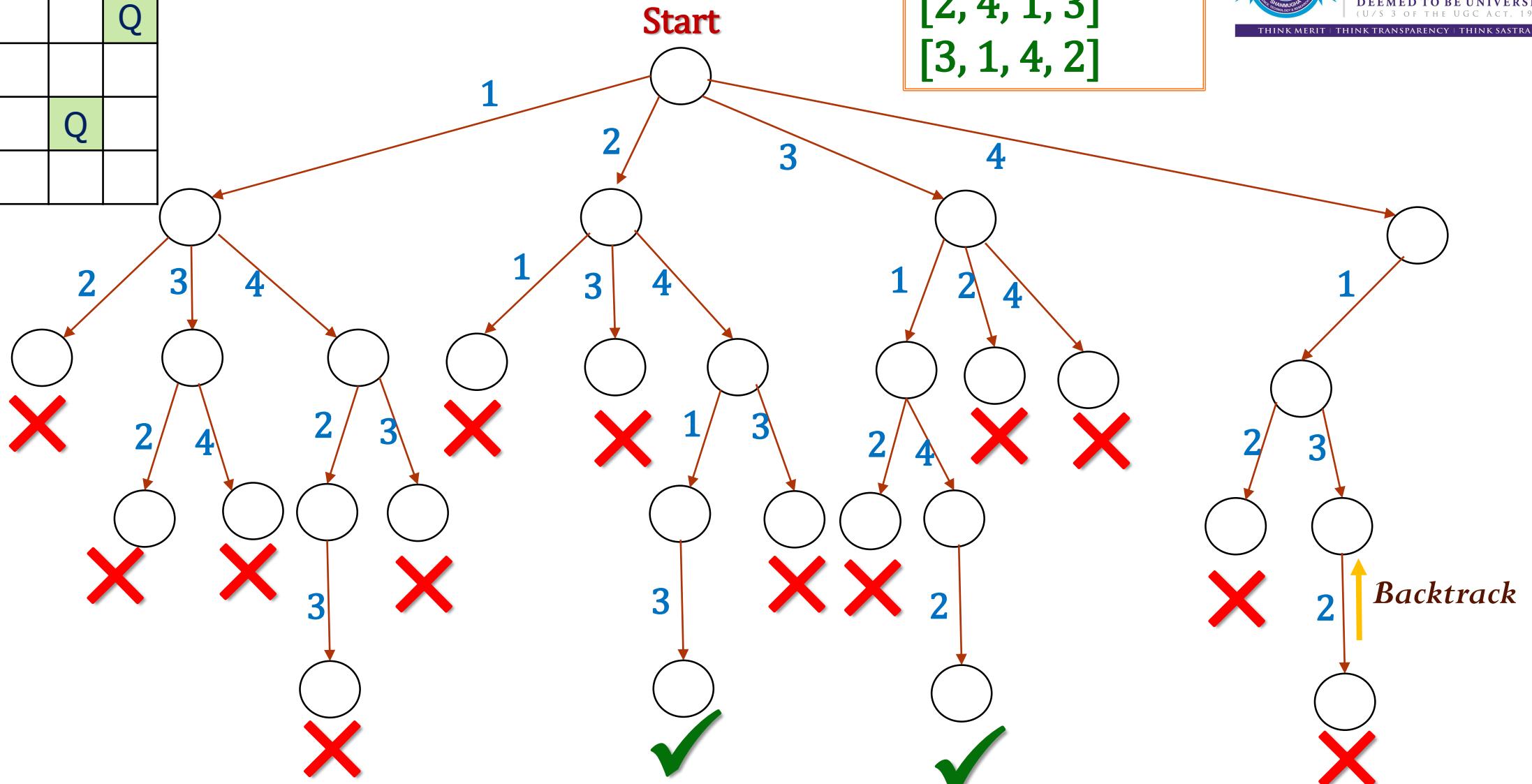


	1	2	3	4
1				Q
2	Q			
3			Q	
4				

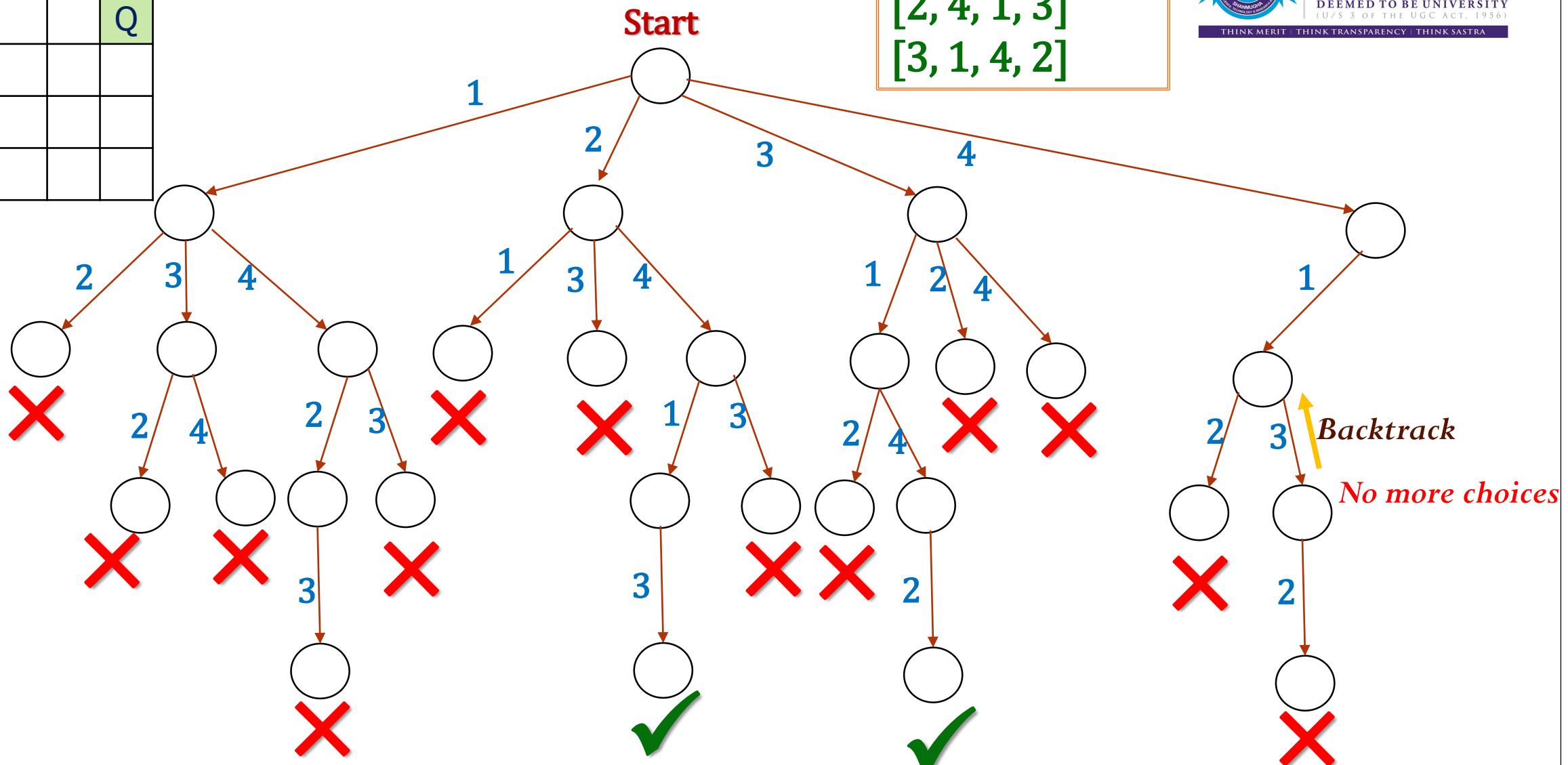
Start

### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				
2	Q			
3				
4				

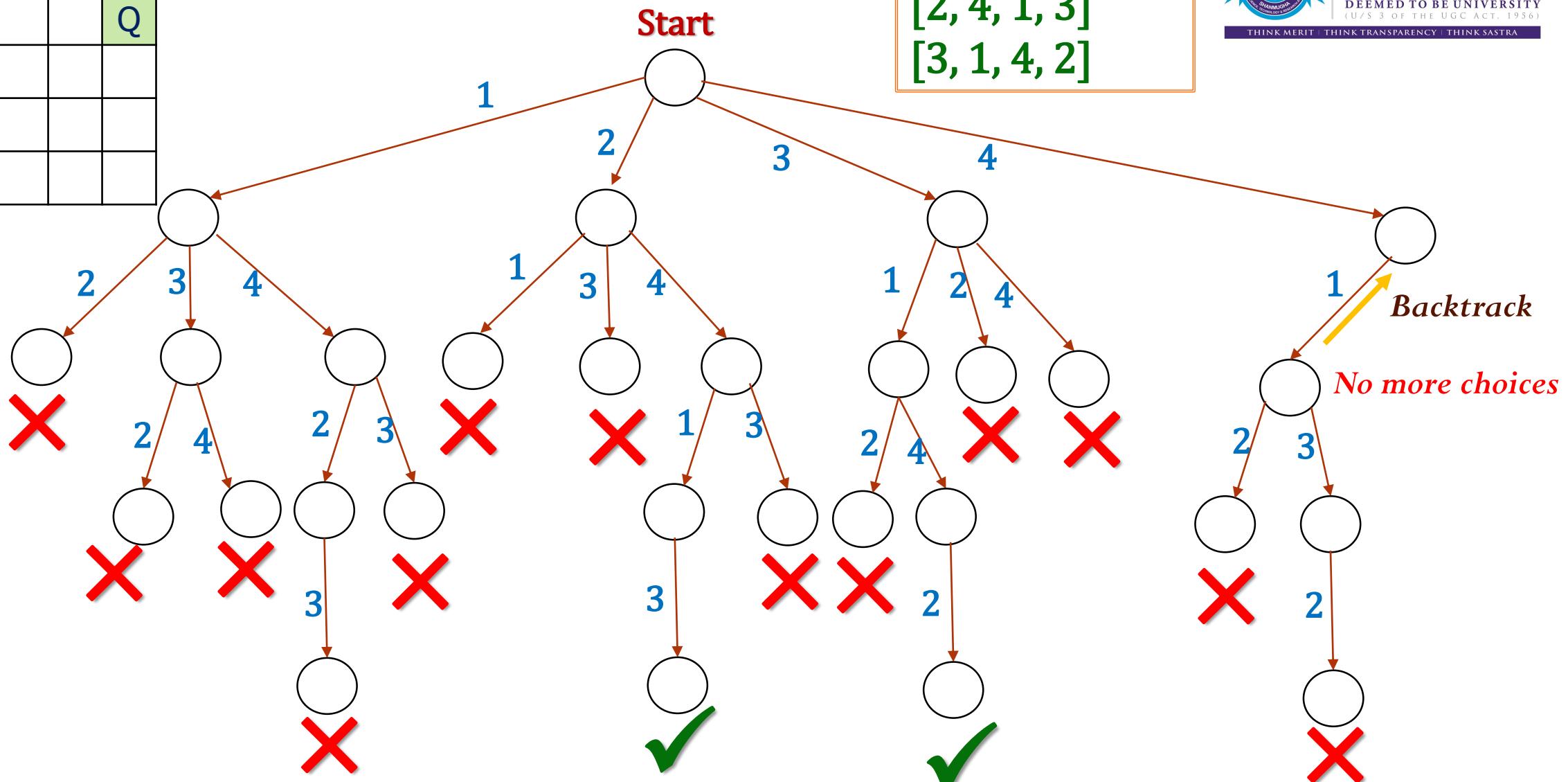


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2				
3				
4				

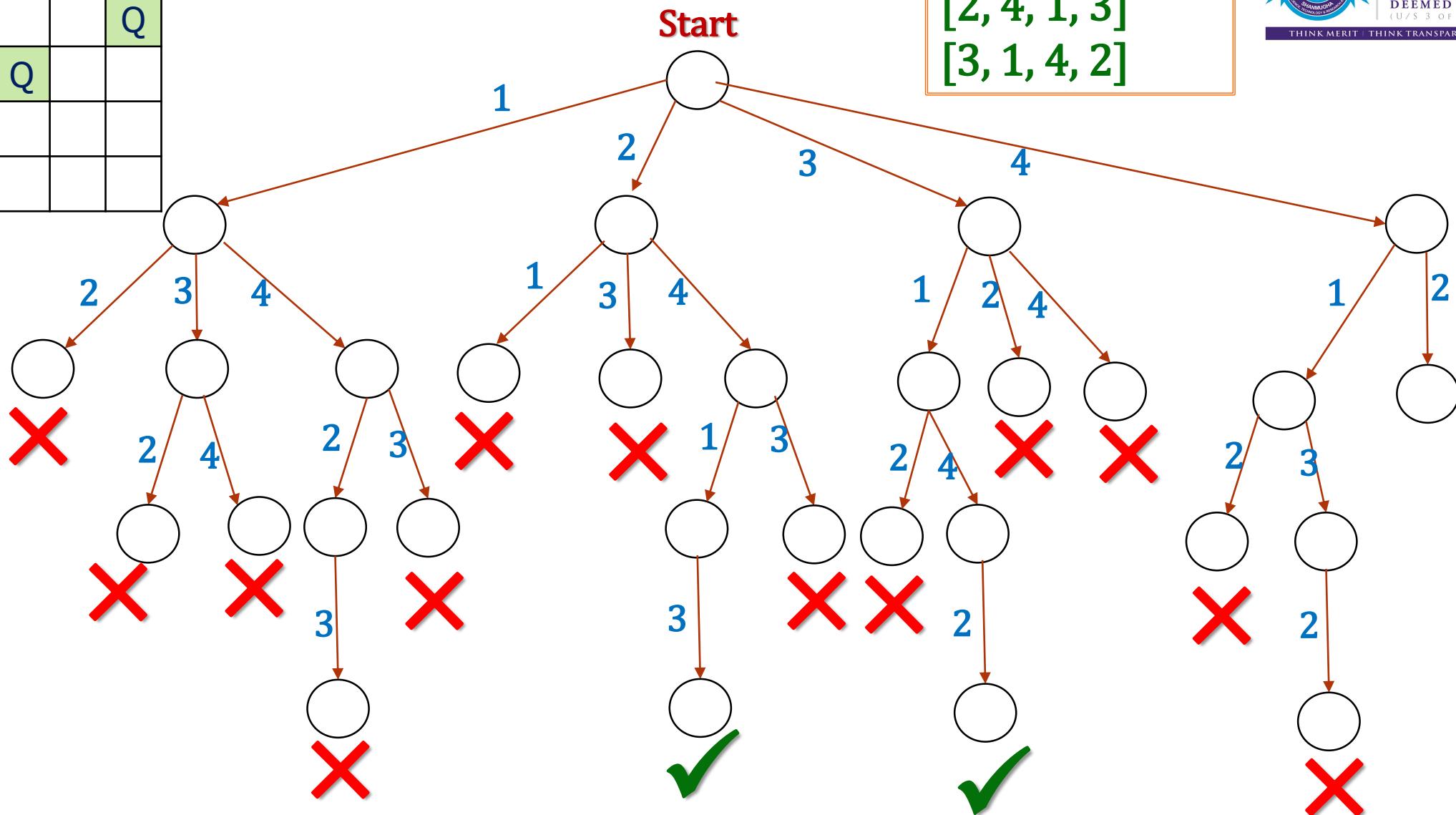


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2		Q		
3				
4				

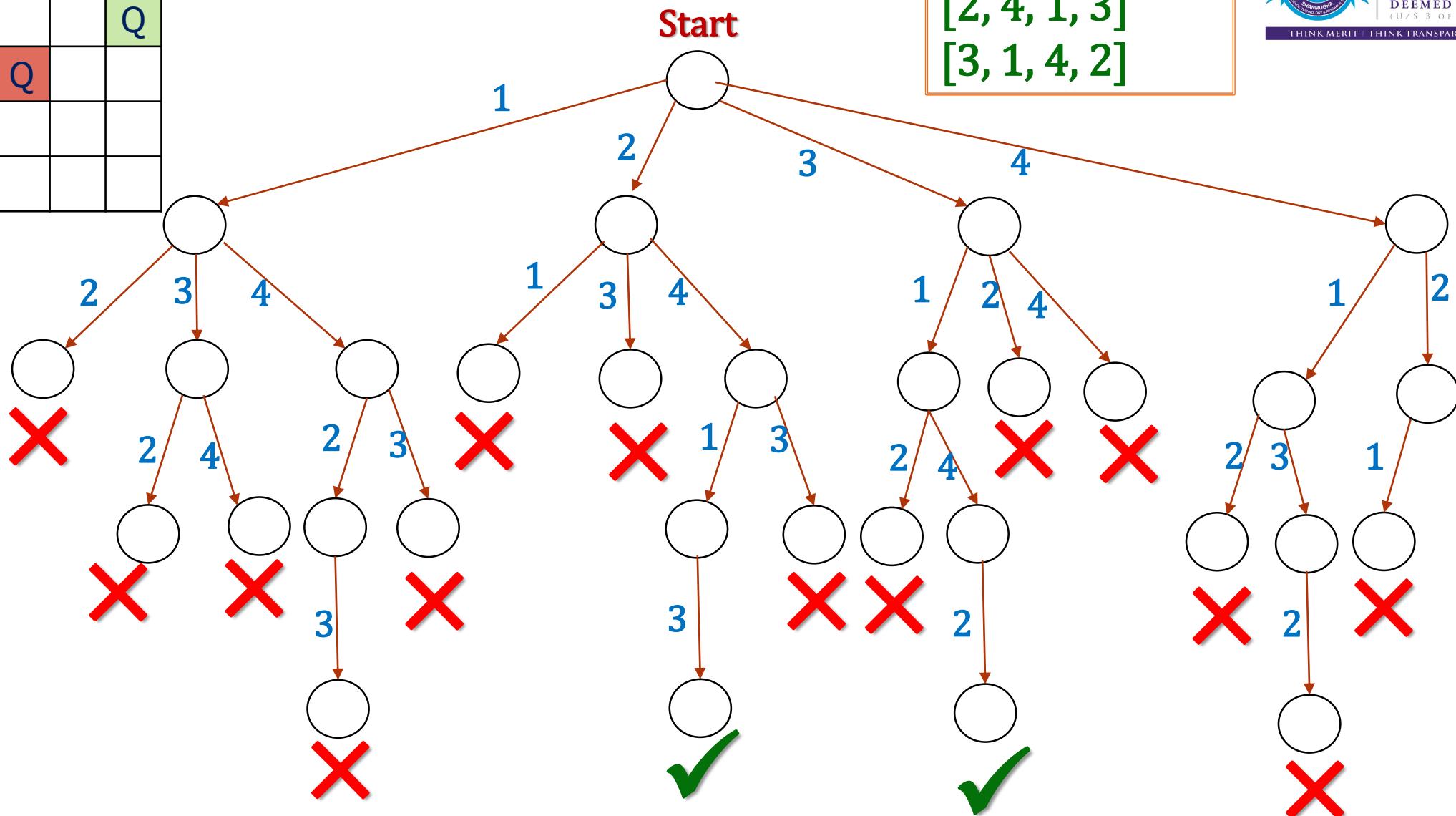


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2		Q		
3	Q			
4				

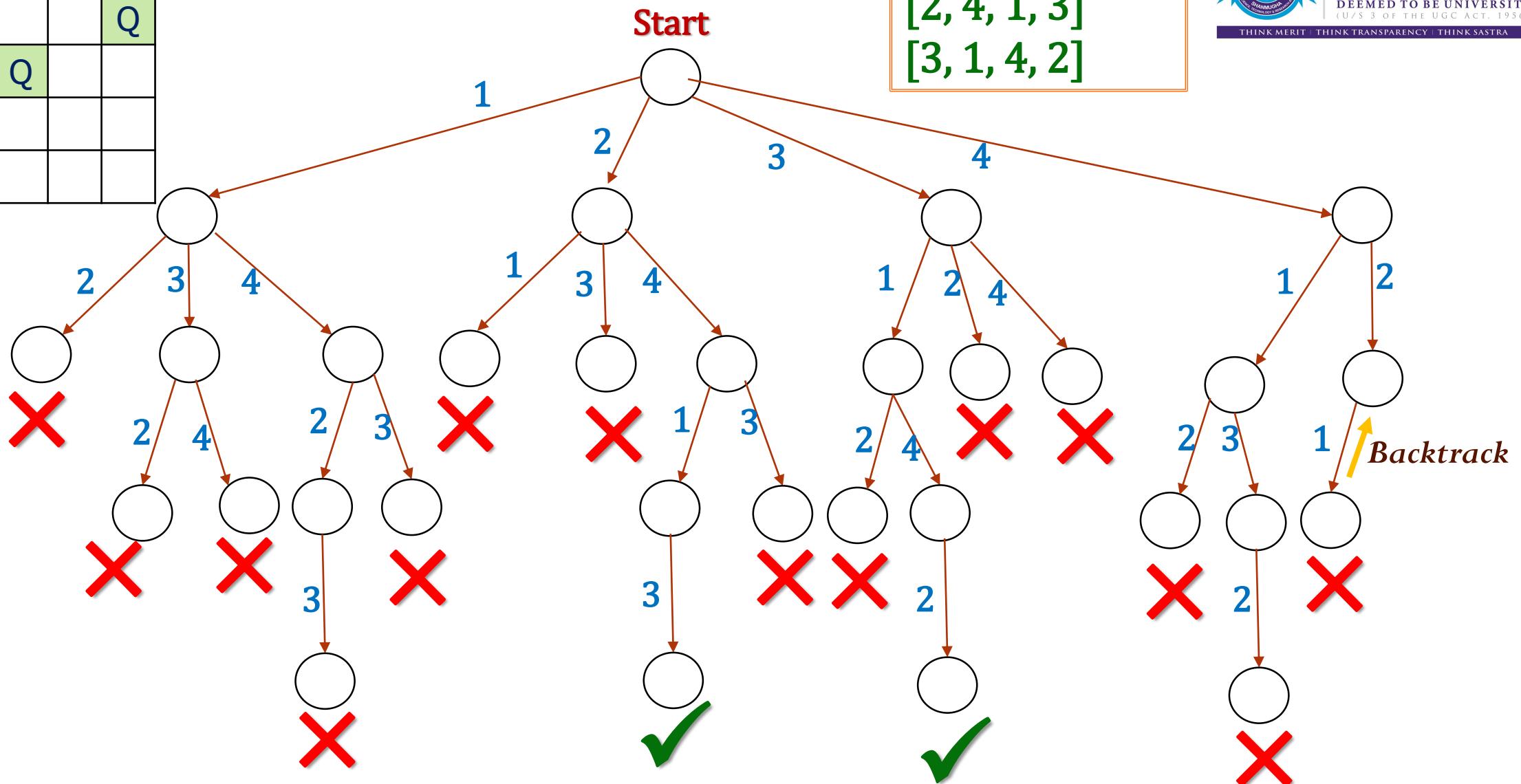


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2		Q		
3				
4				



### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2		Q		
3			Q	
4				

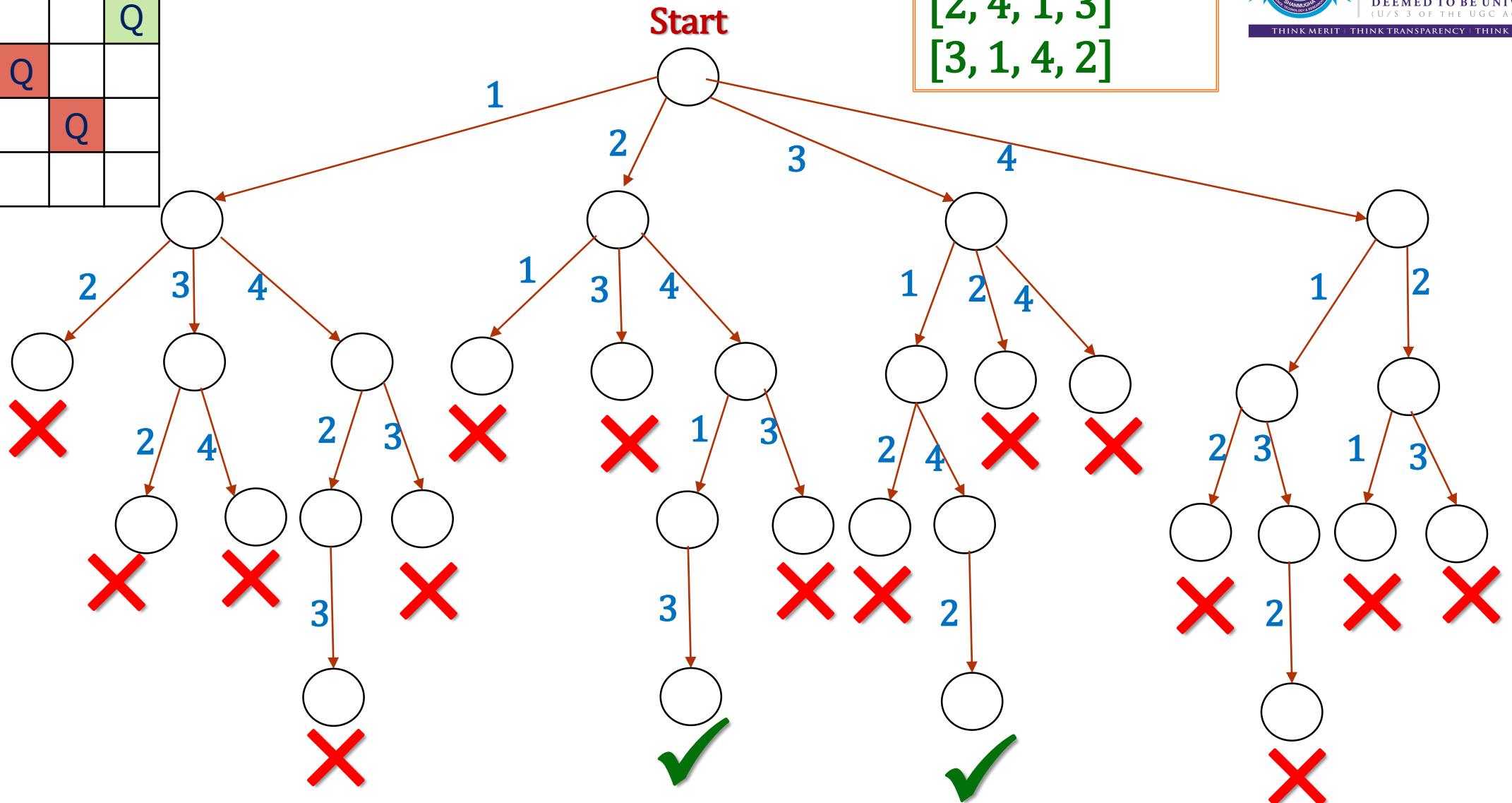
Start

### Solutions

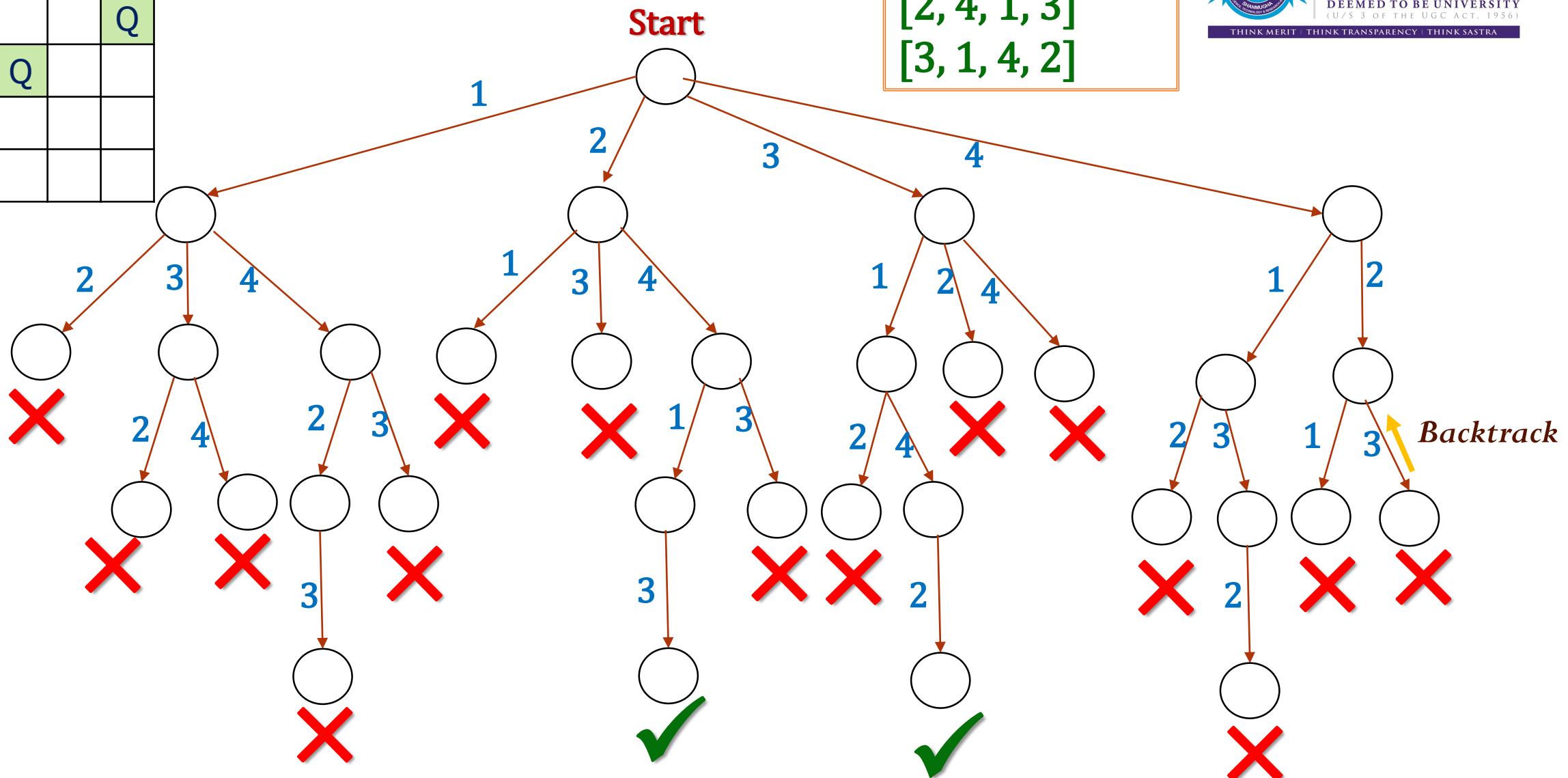
[2, 4, 1, 3]  
[3, 1, 4, 2]



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



	1	2	3	4
1				Q
2		Q		
3				
4				

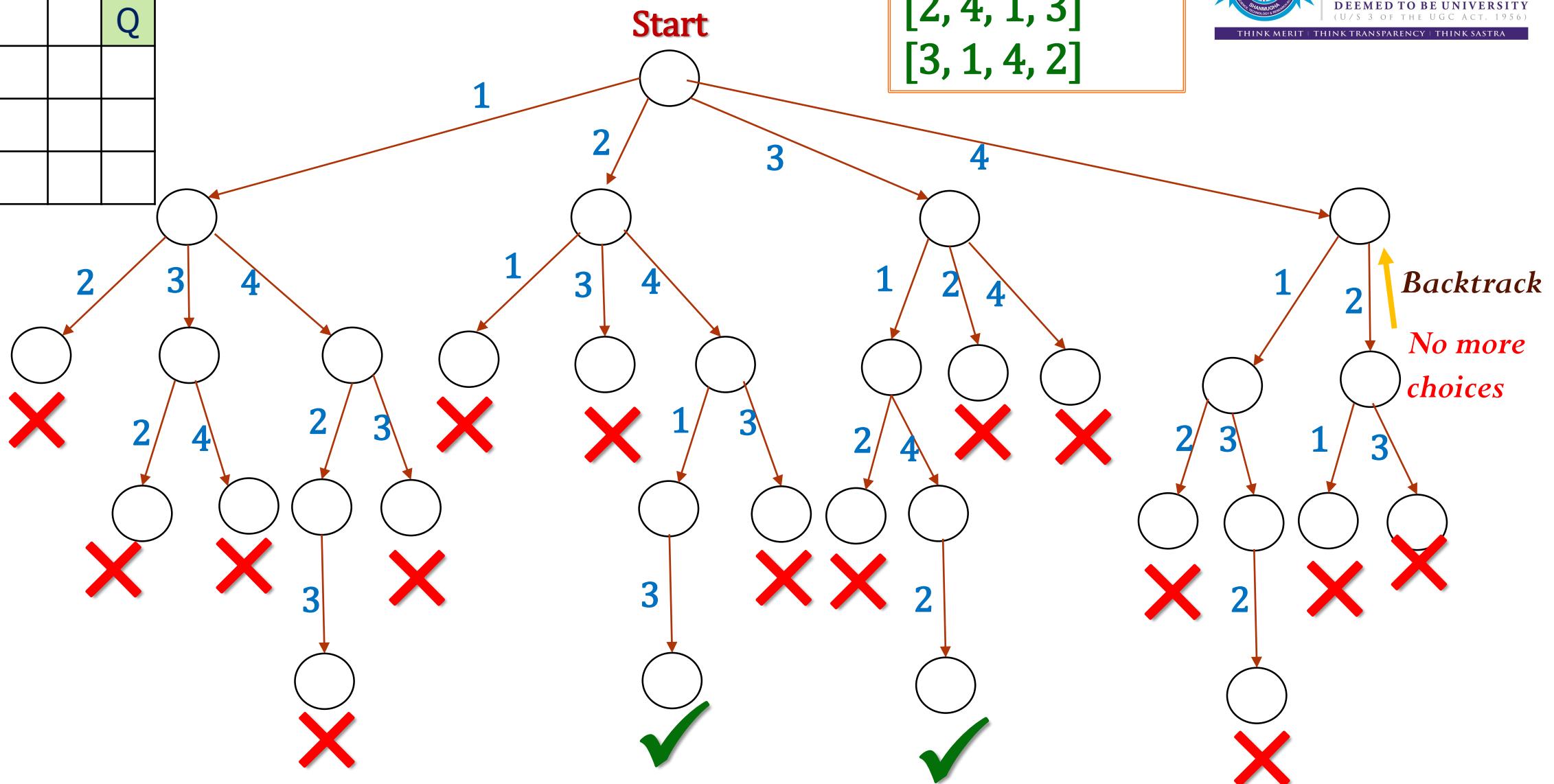


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2				
3				
4				

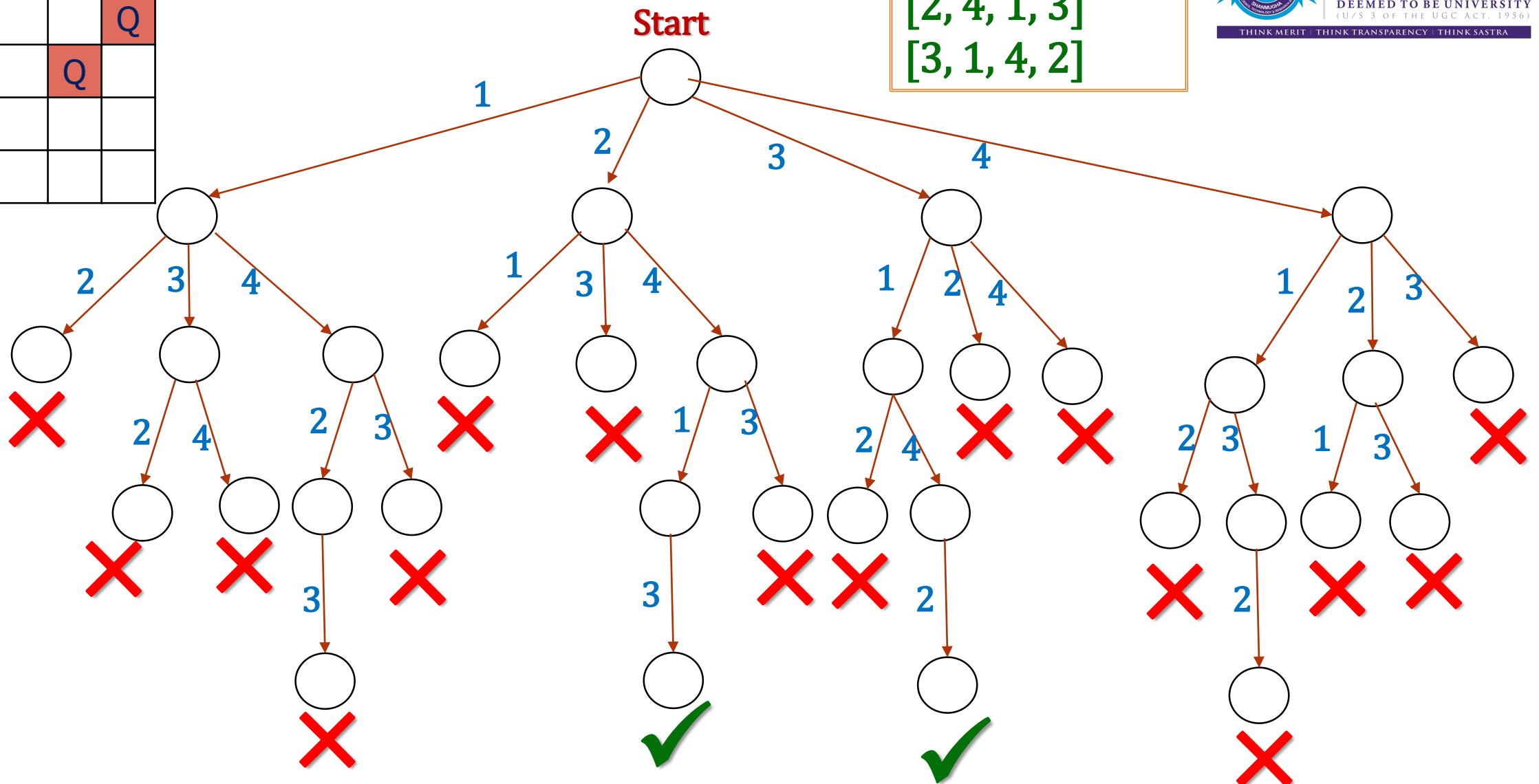


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2			Q	
3				
4				

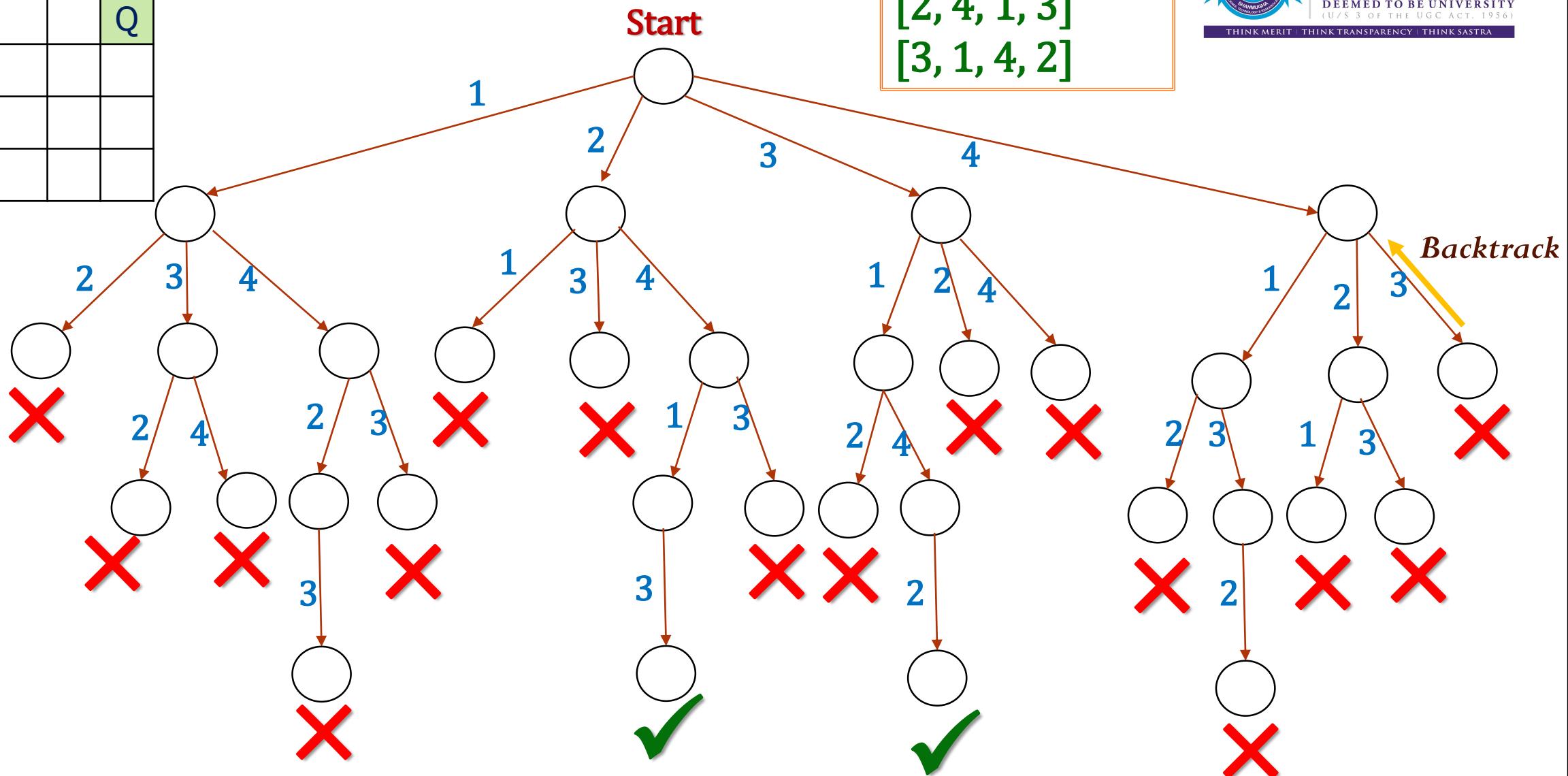


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2				
3				
4				

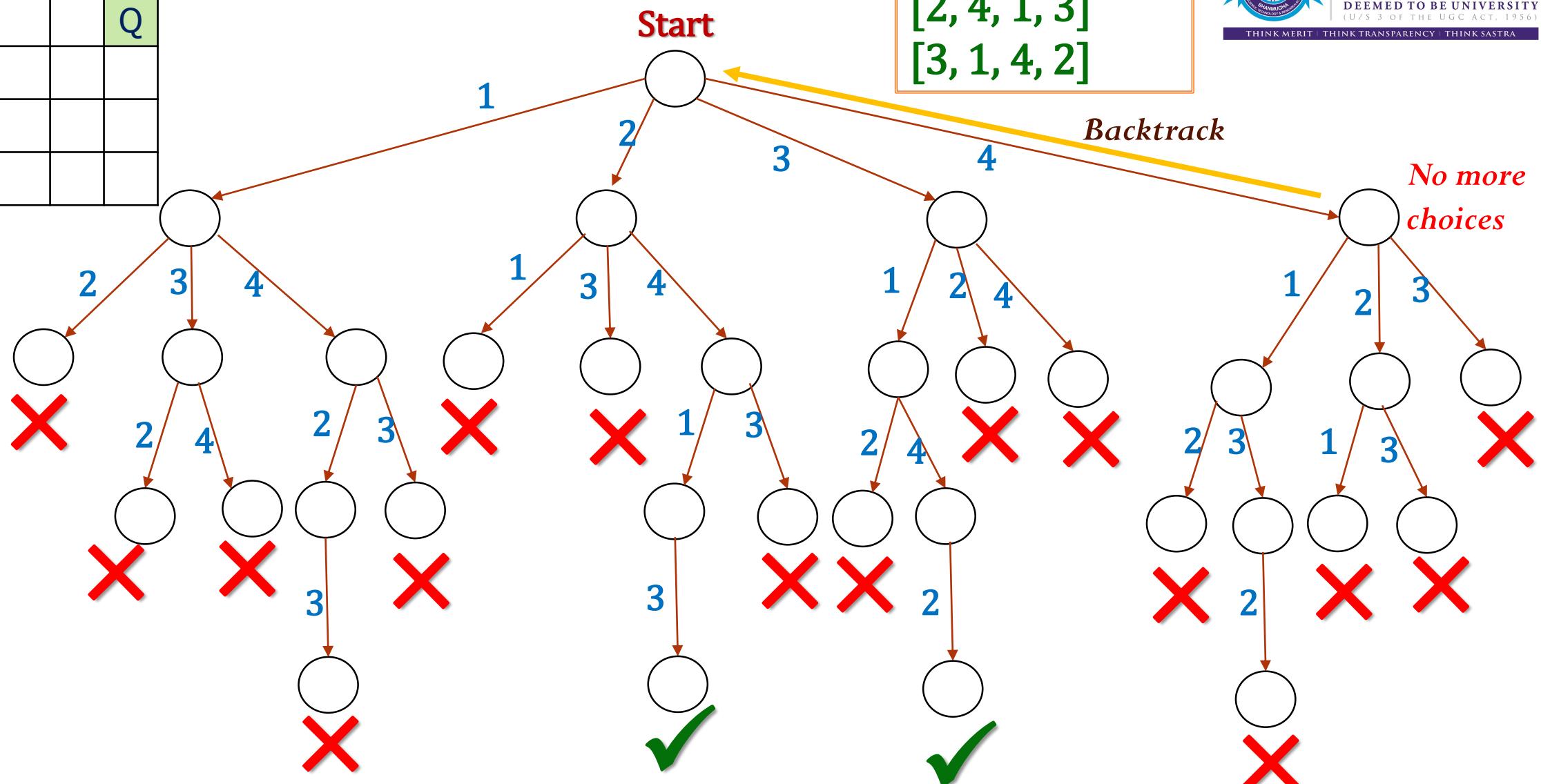


### Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



	1	2	3	4
1				Q
2				
3				
4				



## Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]

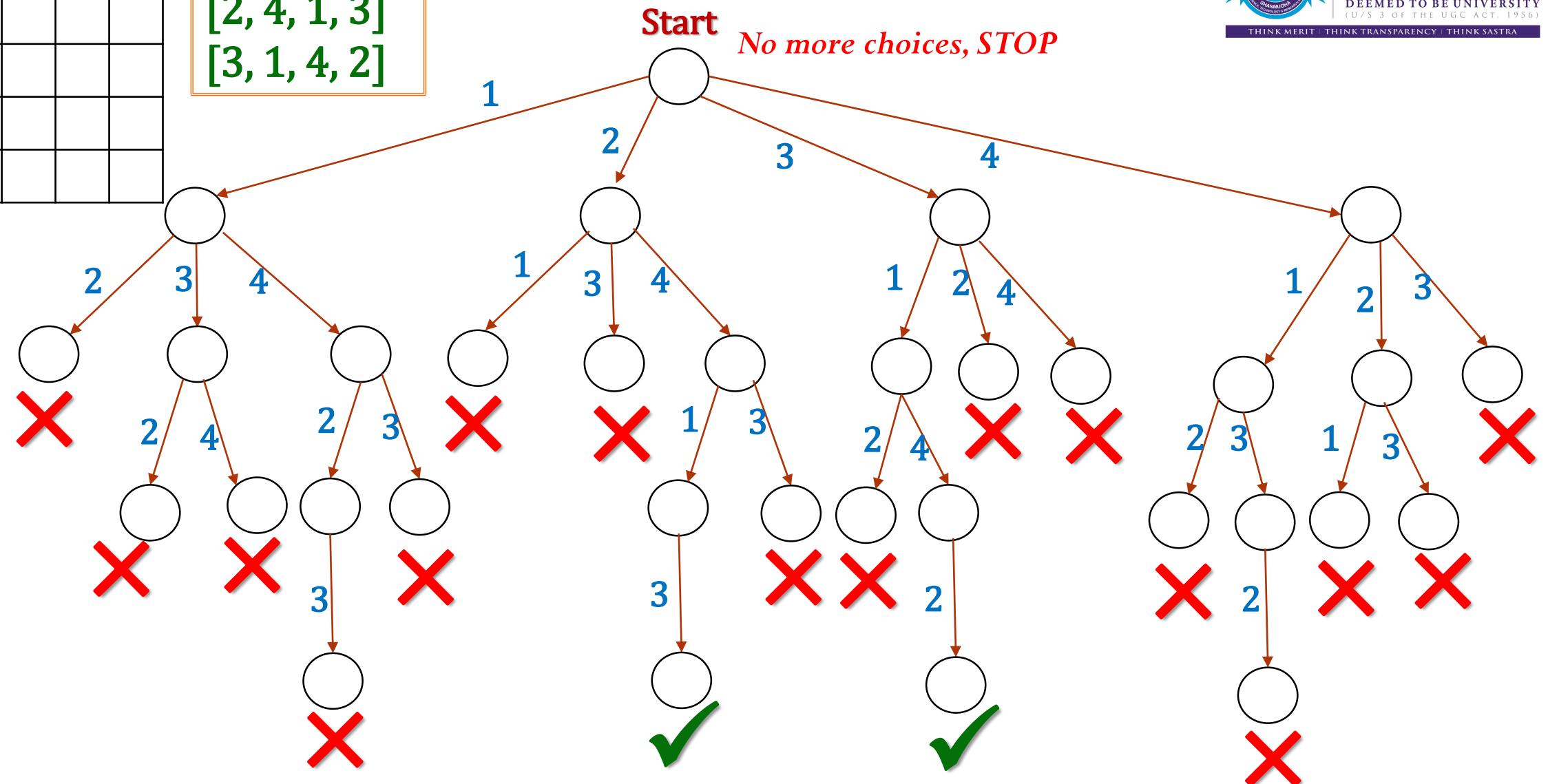


The logo for SASTRA Deemed to be University. It features the word "SASTRA" in large, bold, blue serif letters at the top. Below it, in smaller blue letters, is "ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION". Underneath that, in larger blue letters, is "DEEMED TO BE UNIVERSITY". At the bottom, in smaller blue letters, is "U/S 3 OF THE UGC ACT, 1956". A dark blue horizontal bar spans the width of the logo, containing the text "INK TRANSPARENCY | THINK SASTRA" in white.

	1	2	3	4
1				
2				
3				
4				

## Solutions

[2, 4, 1, 3]  
[3, 1, 4, 2]



Solutions

[2, 4, 1, 3]

[3, 1, 4, 2]

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

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

# N-Queen Problem - Backtracking Algorithm

**Algorithm** nQueen(*n*, Solutions[0..*m*-1][0..*n*-1], *m*)

**Input:** *n* – Number of Queens

**Output:** *m* – Number of Solutions

Solutions[0..*m*-1][0..*n*-1] – Solutions matrix – Each row represents a solution.

True or False – Can be solved or Not

Let Board[0..*n*-1][0..*n*-1] be a '*n* x *n*' Boolean matrix (values: 0 or 1)– Represents a chess board.

**For** *i*←0 to *n*-1 **do**

**For** *j*←0 to *n*-1 **do**

        Board[*i*][*j*] ← 0

**End For**

**End For**

**//Try placing a queen from 0<sup>th</sup> Row**

Row ← 0

Return PalaceQueen(Board, *n*, Row, Solutions, *m*)

**End** nQueen

# N-Queen Problem - Backtracking Algorithm

**Algorithm** PlaceQueen(Board[0..n-1][0..n-1], n, Row, Solutions[0..m-1][0..n-1], m)

**Input:** n – Number of Queens

Board[0..n-1][0..n-1] - 'n x n' Boolean matrix (values: 0 or 1)– Represents a chess board.

Row – Placing a queen at this row.

**Output:** m – Number of Solutions

Solutions[0..m-1][0..n-1] – Solutions matrix – Each row represents a solution.

True or False – Can be solved or Not

**//Base Case**

**If** Row = n **then**

**//Solution Found. Copy it into Solutions[] array**

K  $\leftarrow$  0

**For** i  $\leftarrow$  0 to n-1 **do**

**For** j  $\leftarrow$  0 to n-1 **do**

**If** Board[i][j]=1 **then**

Solutions[m][k]  $\leftarrow$  j+1

k  $\leftarrow$  k + 1

**End If**

**End For**

**End For**

**End If**

# N-Queen Problem - Backtracking Algorithm

```
Res ← False
For Col←0 to n-1 do
    If IsSafe(Board, n, Row, Col) then
        // Place this queen in board[r][c]
        Board[Row][Col] ← 1;

        If PlaceQueen(board, n, r+1, Solutions, m) = True then
            Res ← True
            End If

        // If placing queen in board[r][c] doesn't lead to a solution,
        // then remove queen from board[r][c]
        Board[Row][Col] = 0; // BACKTRACK
        End If
    End For
    Return Res
End PlaceQueen
```

# N-Queen Problem - Backtracking Algorithm

**Algorithm IsSafe**(Board[0..n-1][0..n-1], n, Row, Col)

**Input:** n – Number of Queens

Board[0..n-1][0..n-1] - 'n x n' Boolean matrix  
(values: 0 or 1)– Represents a chess board.

Row & Col – Row and Column to check placement chance.

**Output:** True or False – Can be placed at Board[Row][Col]  
or Not

**//Check this row on left side**

**For** i $\leftarrow$ 0 to row-1 **do**

**If** board[i][Col] = True **then**

**Return** False;

**End If**

**End For**

**//Check upper diagonal on left side**

i  $\leftarrow$  Row

j  $\leftarrow$  Col

**While** i  $\geq$  0 and j  $\geq$  0 **do**  
**If** Board[i][j] = True **then**  
**Return** False

**End If**

i  $\leftarrow$  i - 1

j  $\leftarrow$  j - 1

**End While**

**//Check upper diagonal on right side**

i  $\leftarrow$  Row

j  $\leftarrow$  Col

**While** i  $\geq$  0 and j < n **do**  
**If** Board[i][j] = True **then**  
**Return** False

**End If**

i  $\leftarrow$  i - 1

j  $\leftarrow$  j + 1

**End While**

**Return** True

**End IsSafe**

# N-Queen Problem - Procedure

- 1) Start in the leftmost column
- 2) If all queens are placed  
    return true
- 3) Try all rows in the current column.  
    Do following for every tried row.
  - a) If the queen can be placed safely in this row  
        then mark this [row, column] as part of the  
        solution and recursively check if placing  
        queen here leads to a solution.
  - b) If placing the queen in [row, column] leads to  
        a solution then return true.
  - c) If placing queen doesn't lead to a solution then  
        unmark this [row, column] (Backtrack) and go to  
        step (a) to try other rows.
- 3) If all rows have been tried and nothing worked,  
    return false to trigger backtracking.

# **Backtracking Approach**

## **Sum of Subset Problem**

# Problem

- ✓ Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K.
- ✓ We are considering the set contains non-negative values.
- ✓ It is assumed that the input set is unique (no duplicates are presented)

## Example

**Input:**  $w[1..7] = \{10, 7, 5, 18, 12, 20, 15\}$

$$n = 7$$

$$m = 35$$

**Output:**  $x1[1..7] = \{1, 1, 0, 1, 0, 0, 0\}$

$x2[1..7] = \{1, 0, 1, 0, 0, 1, 0\}$

$x3[1..7] = \{0, 0, 1, 1, 1, 0, 0\}$

$x4[1..7] = \{0, 0, 0, 0, 0, 1, 1\}$

## Solutions:

\*\*\*\*\*

1. {10, 7, 18}

2. {10, 5, 20}

3. {5, 18, 12}

4. {20, 15}

# Solution – Backtracking Approach

- ✓ State Space Tree - Used in representing Solution
  - ✓ Start with the Root as the given "Sum"
  - ✓ First level in the tree is explored as:
    - ✓ The 1st element is included  $\rightarrow x_1=1$
    - ✓ The 1st element is not included  $\rightarrow x_1=0$
  - ✓ Second level in the tree is explored as:
    - ✓ The 2nd element is included  $\rightarrow x_2=1$
    - ✓ The 2nd element is not included  $\rightarrow x_2=0$
  - ✓ In general, the  $i$ -th level is explored like
    - ✓ The  $i$ -th element is included  $\rightarrow x_i=1$
    - ✓ The  $i$ -th element is not included  $\rightarrow x_i=0$
- 
- ✓ If the element is include, the child value will be the balance sum. i.e., the sum will be updated as Sum-Element value.
  - ✓ If the element is not included, the child value remains same as root.

# Solution – Backtracking Approach

## Bounding Condition:

- ✓ Condition for backtracking.
- ✓ There are 3 cases, for which, backtracking is required

### Case 1:

- ✓ **Condition:** BalanceSum = 0
- ✓ **Conclusion:** Solution Found, So, Record Solution and Backtrack

### Case 2:

- ✓ **Condition:** BalanceSum < 0
- ✓ **Conclusion:** Solution Will not be Found in this path, So, Just Backtrack

### Case 3:

- ✓ **Condition:** Level Number = Size
- ✓ **Conclusion:** Solution Not Found, Just Backtrack

# Example

**Input:** Key[1..5] = {5, 2, 1, 4, 3}

n = 5

Sum = 9

1 2 3 4 5  
X 

0	0	0	0	0
---	---	---	---	---

1 2 3 4 5  
Key 

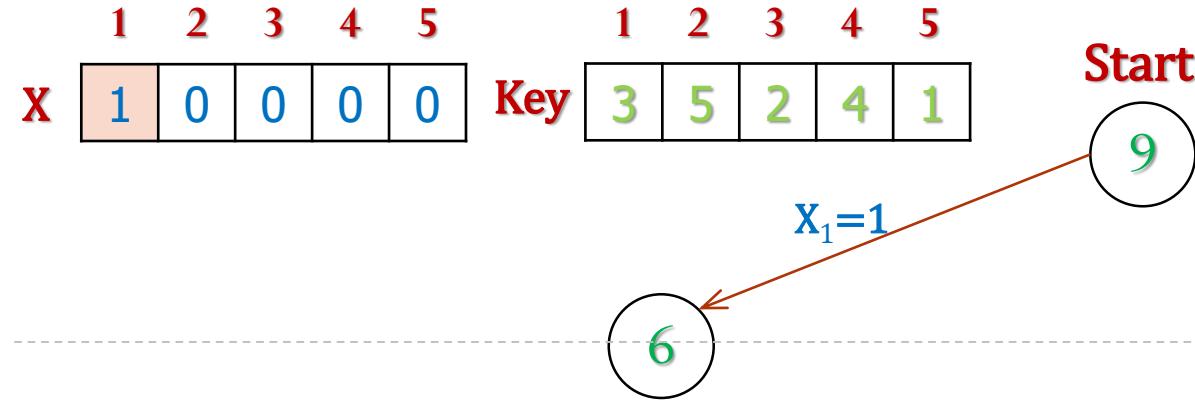
3	5	2	4	1
---	---	---	---	---

Start

9



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



Key = 3

Key = 5

Key = 2

Key = 4

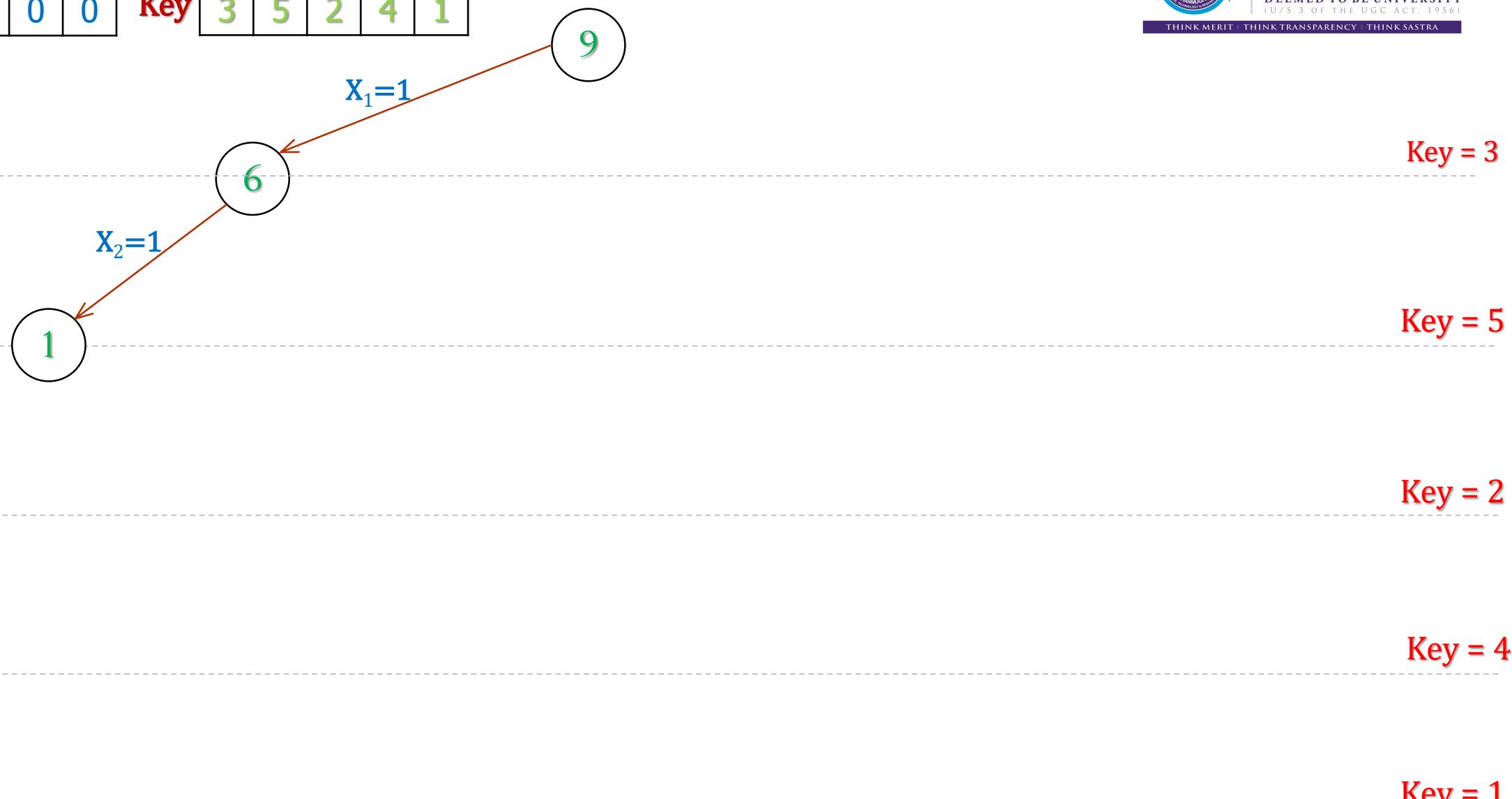
Key = 1

	1	2	3	4	5
X	1	1	0	0	0

Key

	1	2	3	4	5
Key	3	5	2	4	1

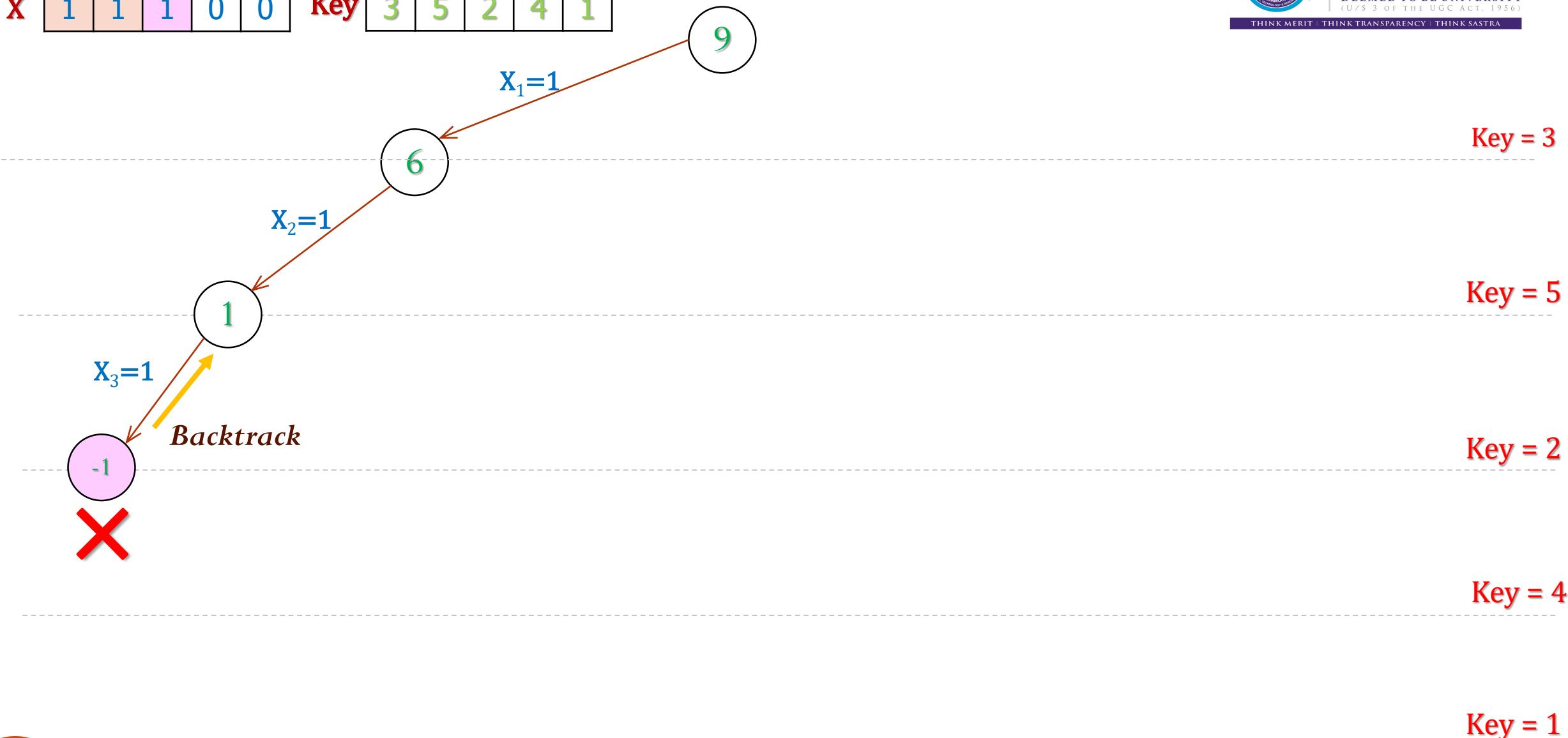
Start



	1	2	3	4	5
X	1	1	1	0	0

Key [ 3 5 2 4 1 ]

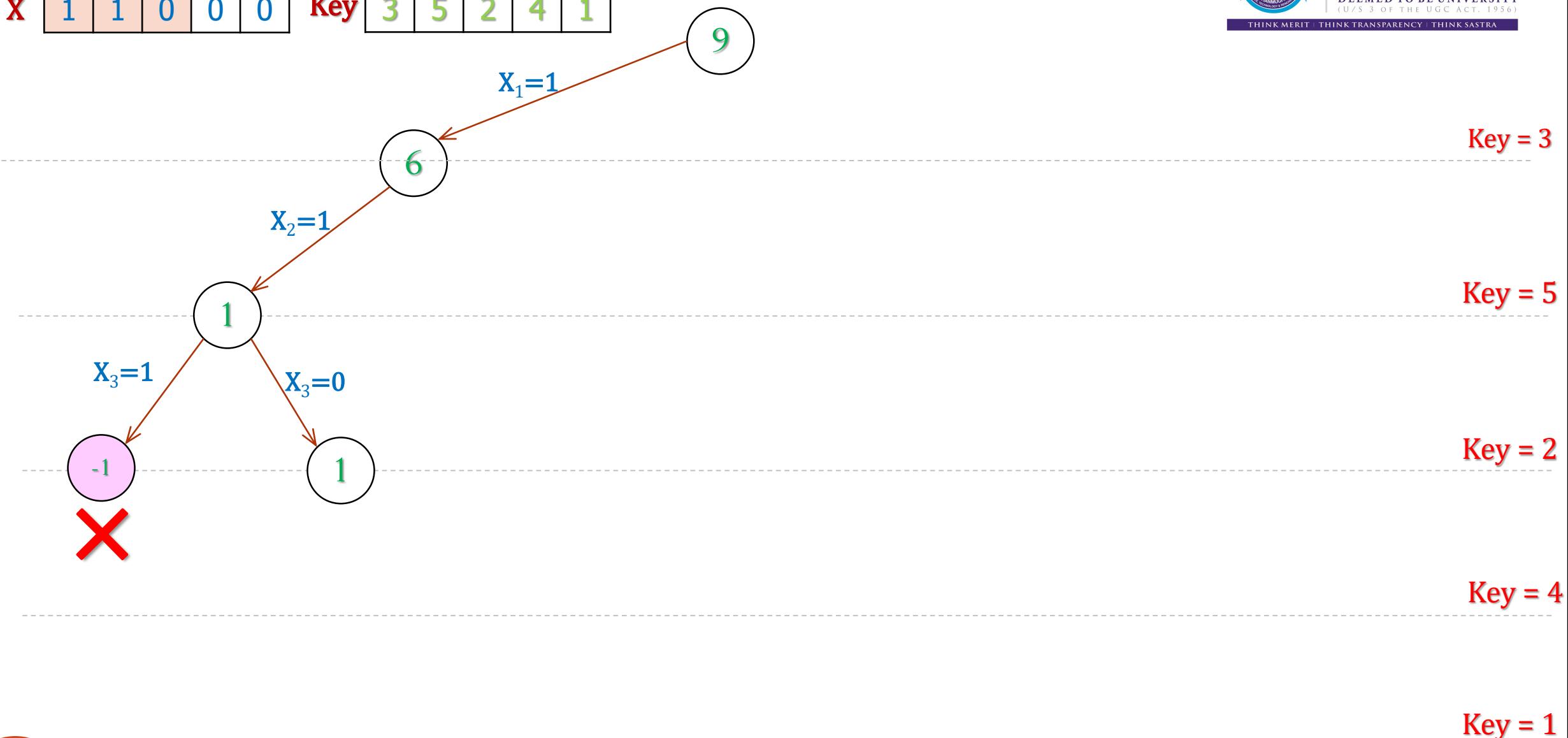
Start



	1	2	3	4	5
X	1	1	0	0	0

Key [ 3 5 2 4 1 ]

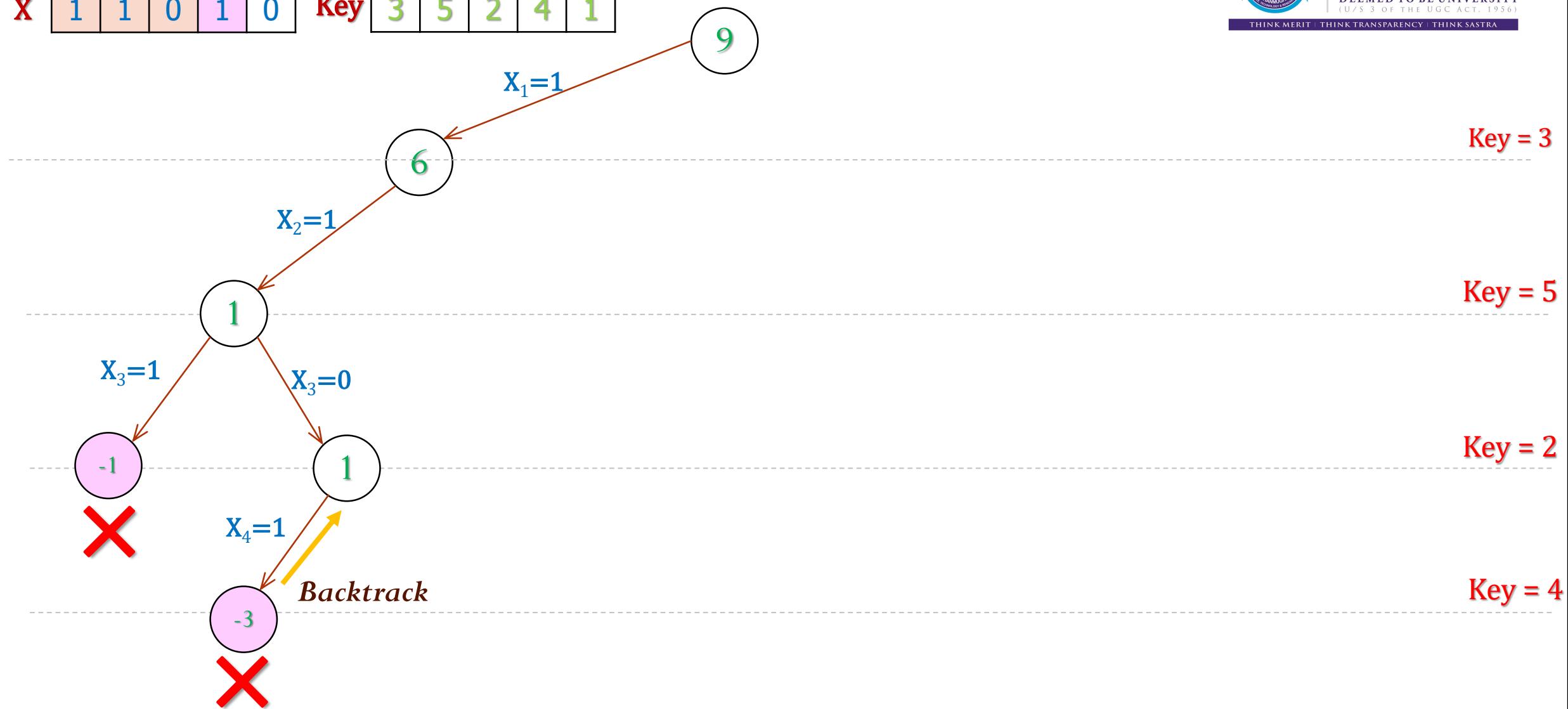
Start



	1	2	3	4	5
X	1	1	0	1	0

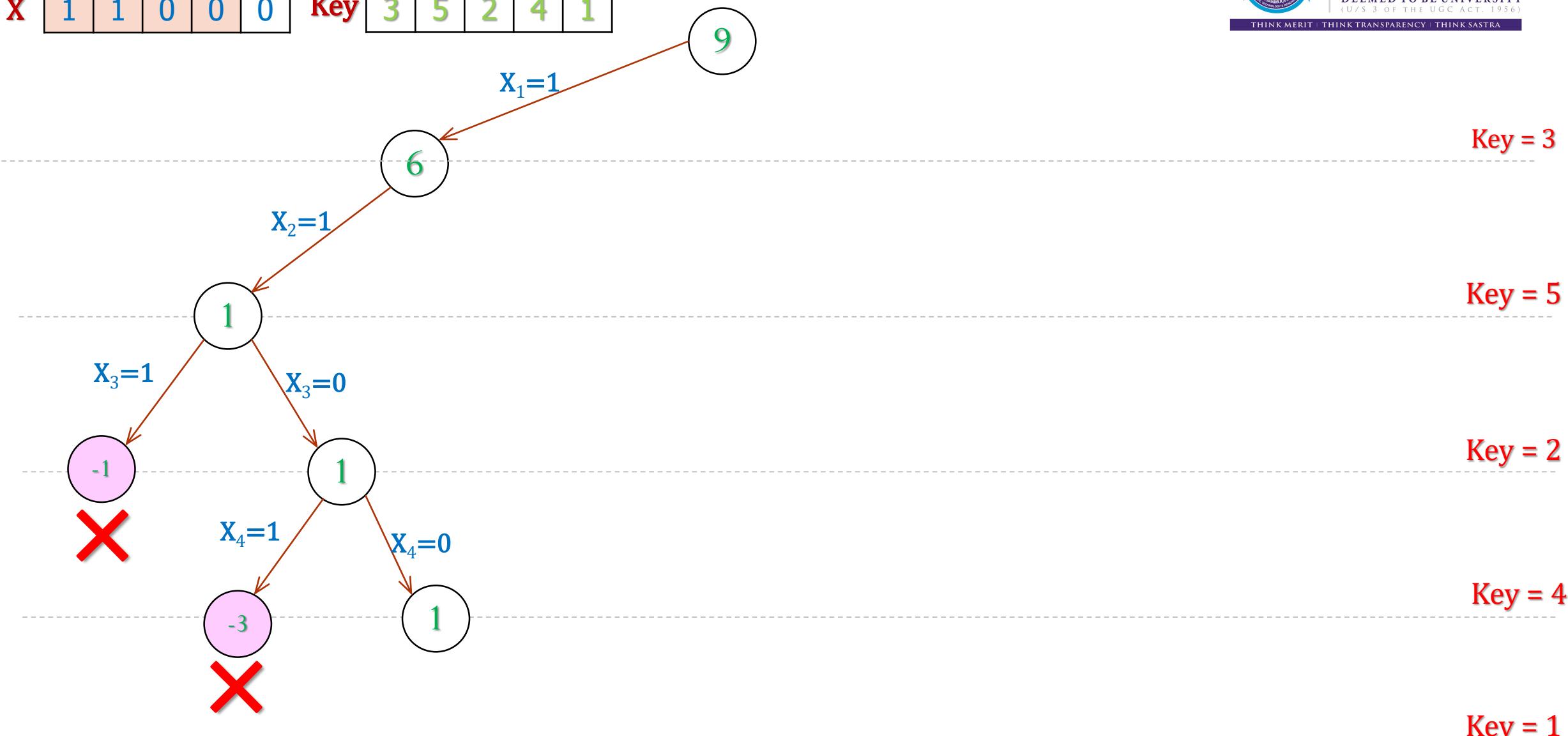
Key [ 3 5 2 4 1 ]

Start



1 2 3 4 5  
**X** [ 1 1 0 0 0 ]      **Key** [ 3 5 2 4 1 ]

Start



**X**    1    2    3    4    5

1	1	0	0	1
---	---	---	---	---

**Key**    1    2    3    4    5

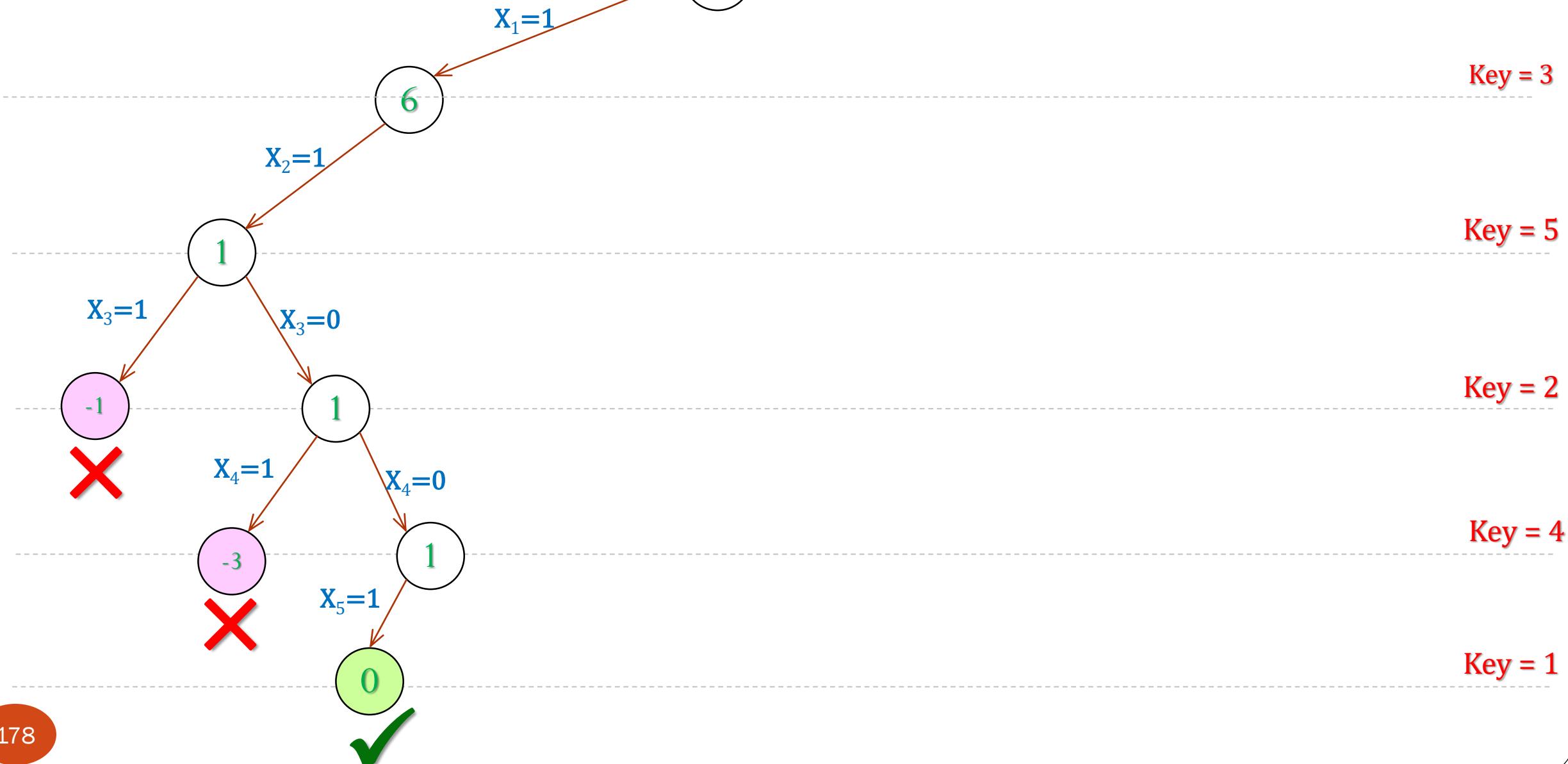
3	5	2	4	1
---	---	---	---	---

**Start**

**Solutions**    1    1    0    0    1



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



**X**    1    2    3    4    5

1	1	0	0	0
---	---	---	---	---

**Key**    1    2    3    4    5

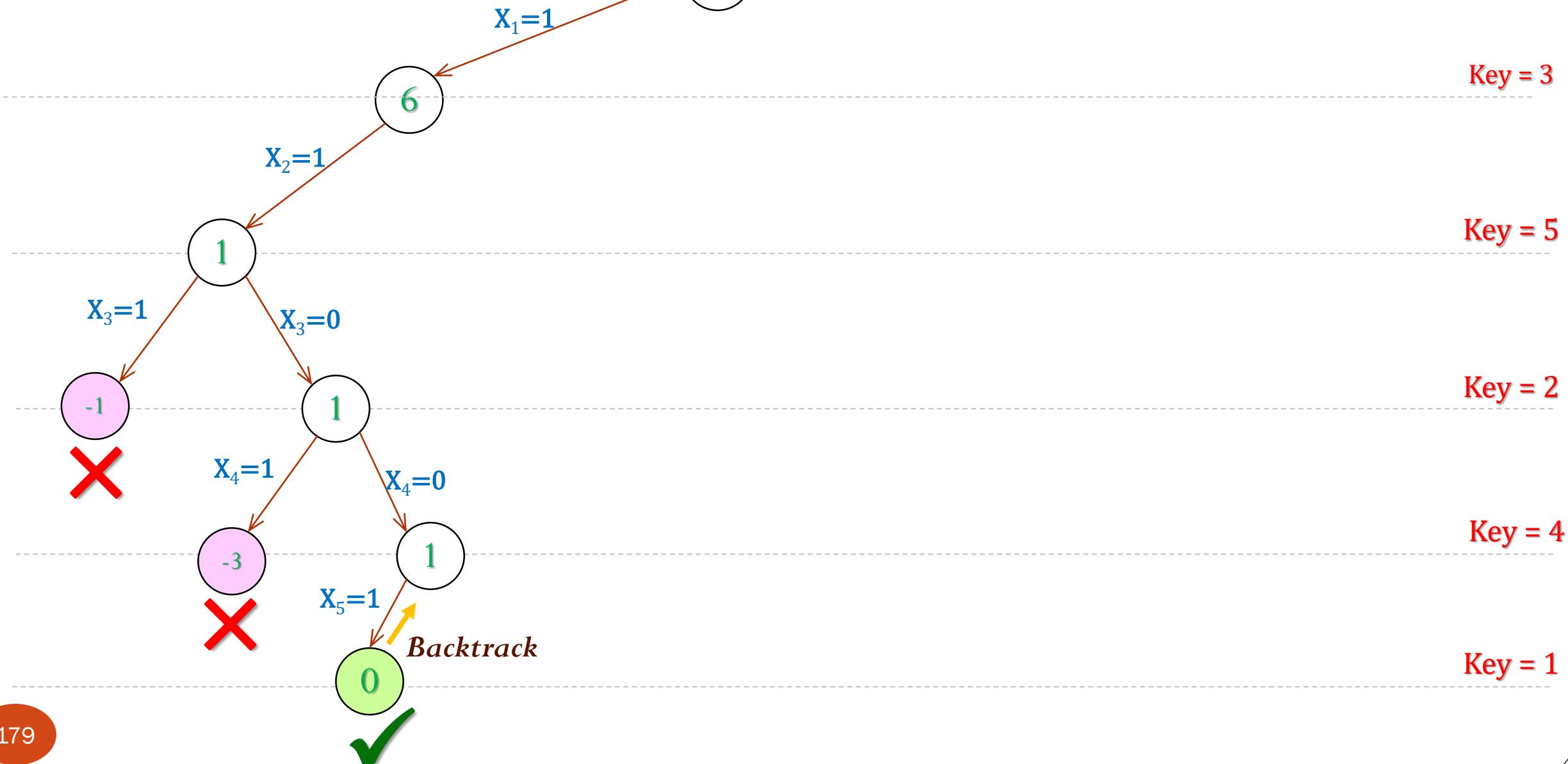
3	5	2	4	1
---	---	---	---	---

**Start**

**Solutions**    1    1    0    0    1



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



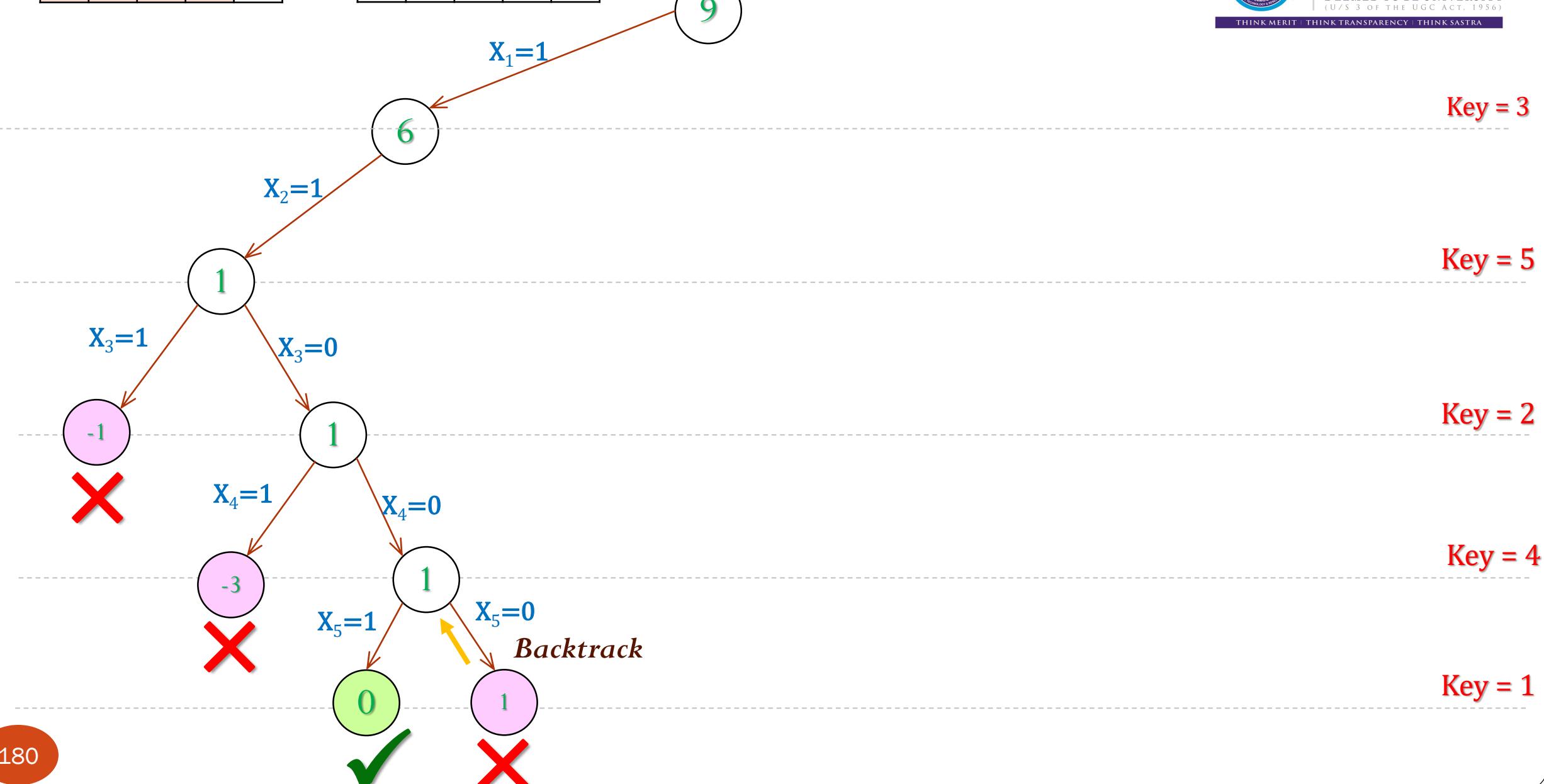
	1	2	3	4	5
X	1	1	0	0	0

1	2	3	4	5
Key	3	5	2	4

# Start

## Solution

S	1	1	0	0	1
---	---	---	---	---	---



	1	2	3	4	5
X	1	1	0	0	0

	1	2	3	4	5
Key	3	5	2	4	1

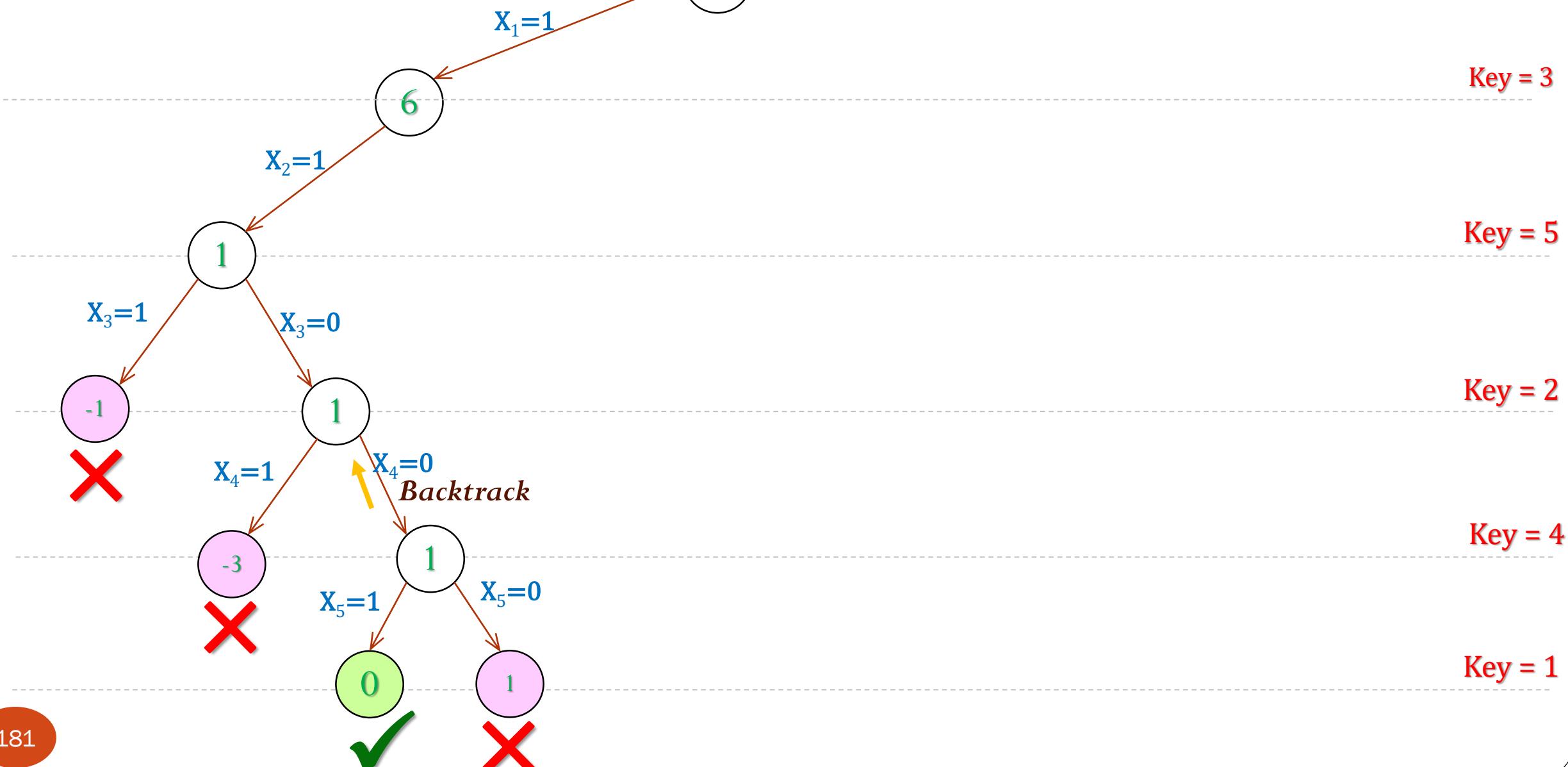
Start

Solutions

1	1	0	0	1
---	---	---	---	---



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



	1	2	3	4	5
X	1	1	0	0	0

	1	2	3	4	5
Key	3	5	2	4	1

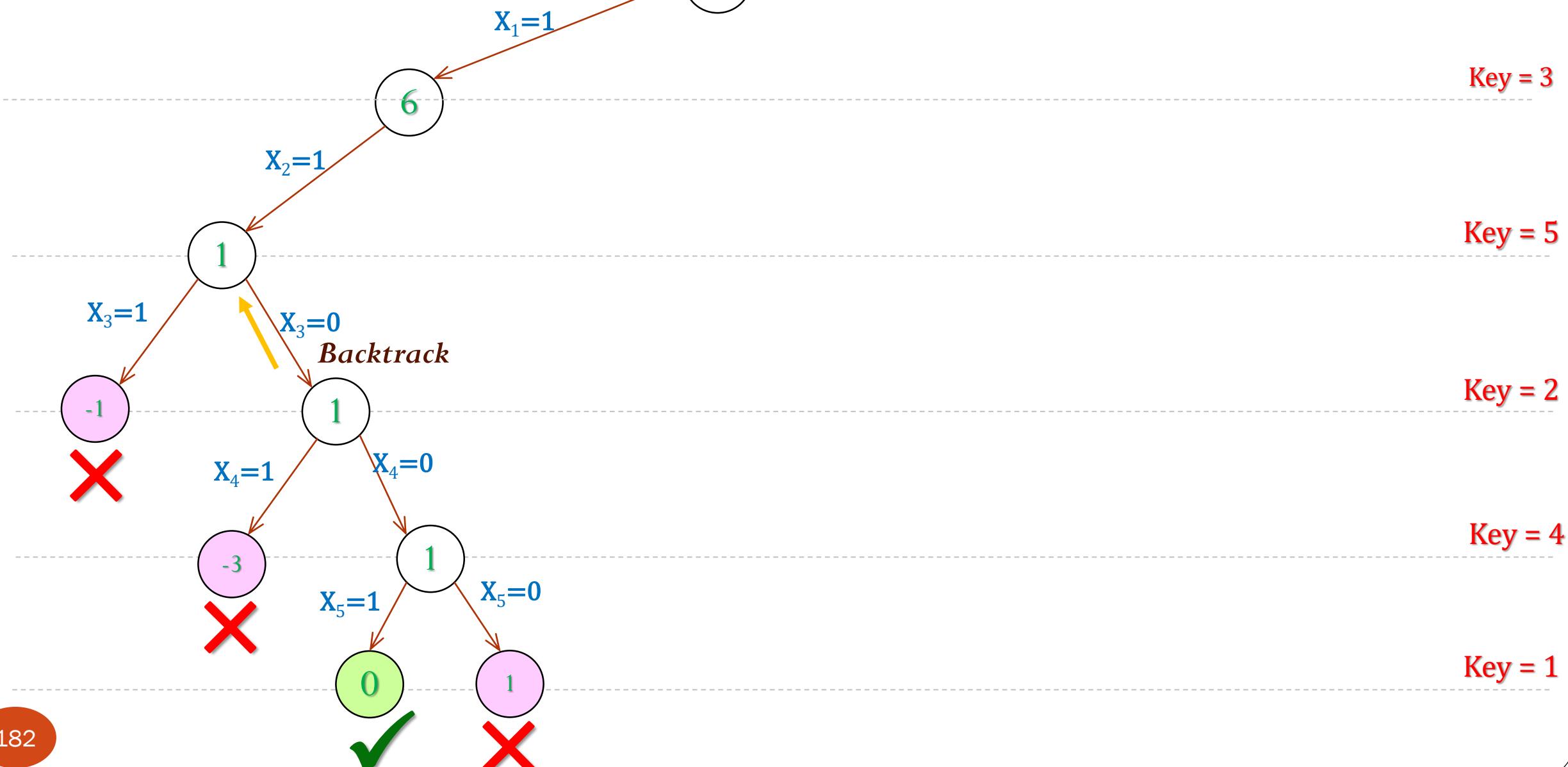
Start

Solutions

1	1	0	0	1
---	---	---	---	---



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



X	1	0	0	0	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

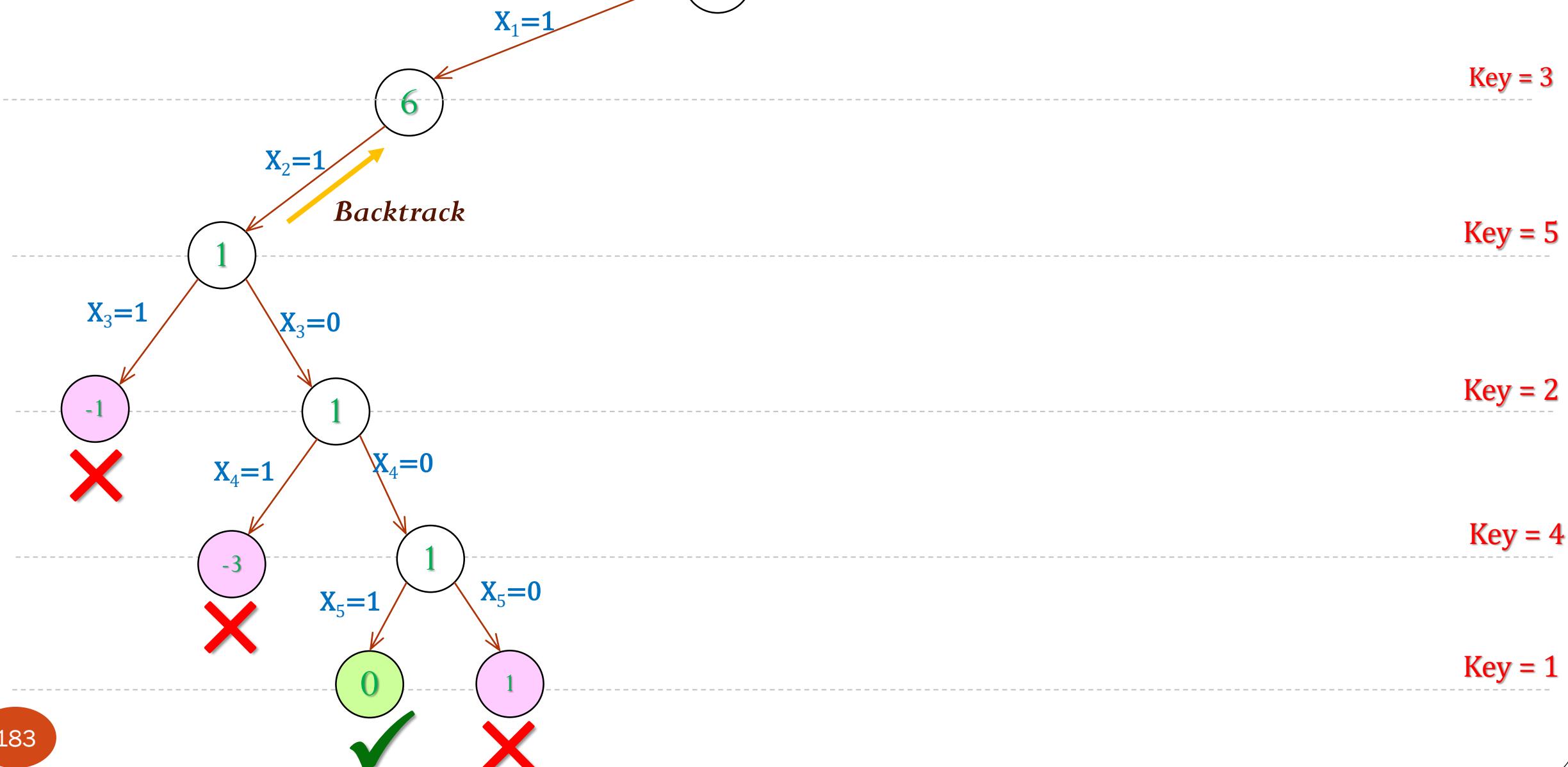
Start

Solutions

1	1	0	0	1
---	---	---	---	---



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



**X**

1	0	0	0	0
---	---	---	---	---

**Key**

3	5	2	4	1
---	---	---	---	---

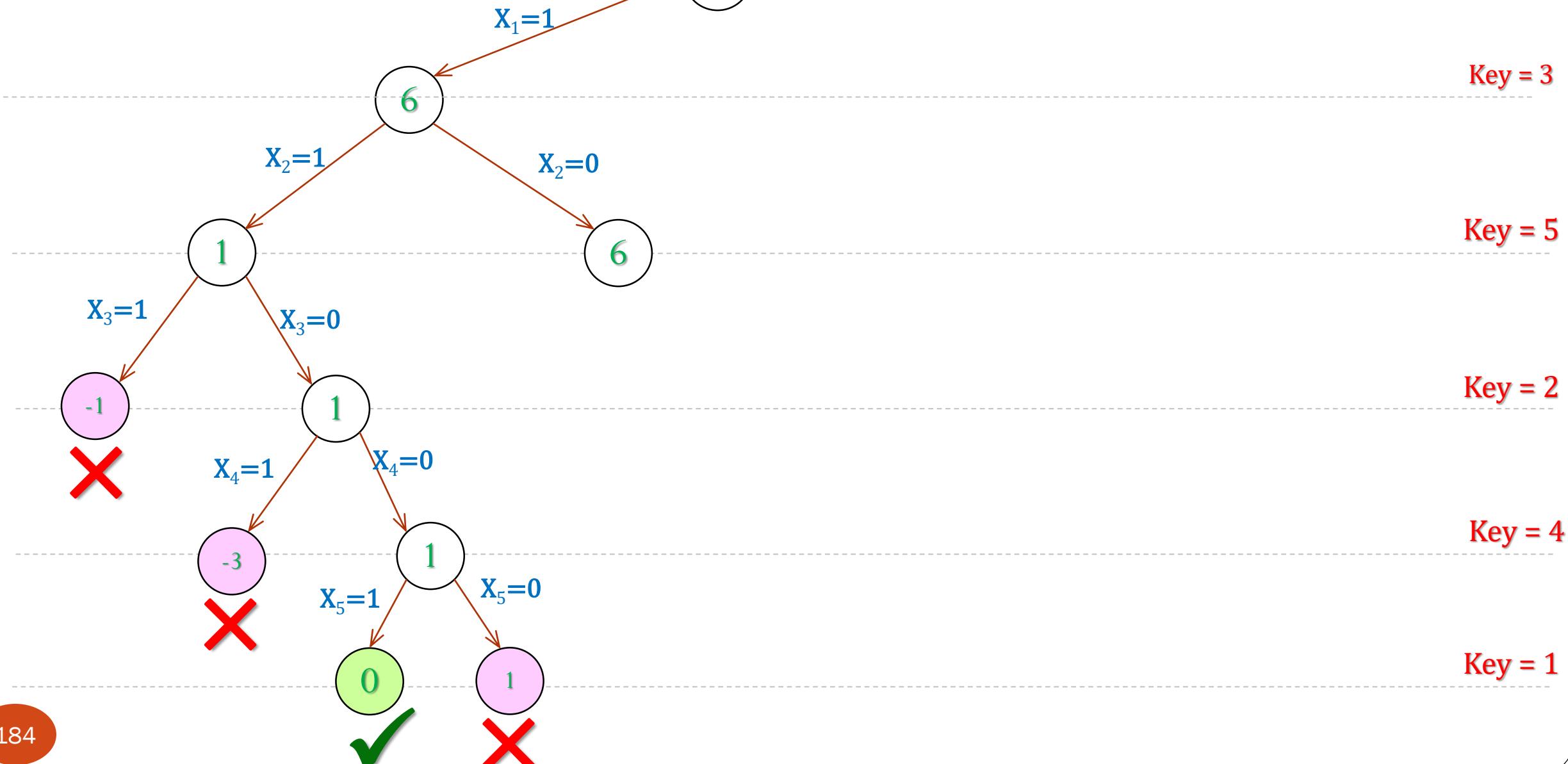
**Start**

**Solutions**

1	1	0	0	1
---	---	---	---	---



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



**X**

1	0	1	0	0
---	---	---	---	---

**Key**

3	5	2	4	1
---	---	---	---	---

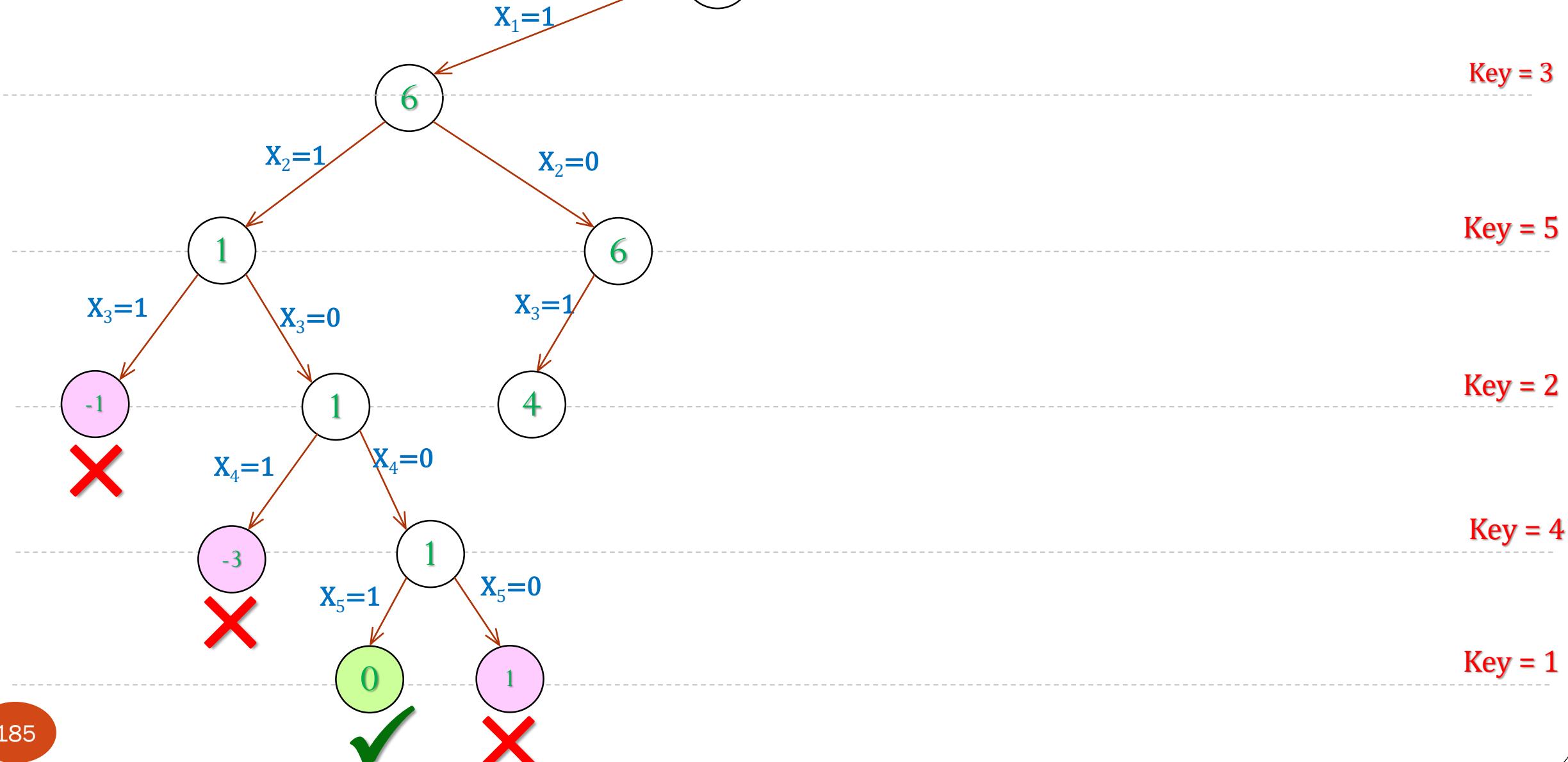
**Start**

**Solutions**

1	1	0	0	1
---	---	---	---	---



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



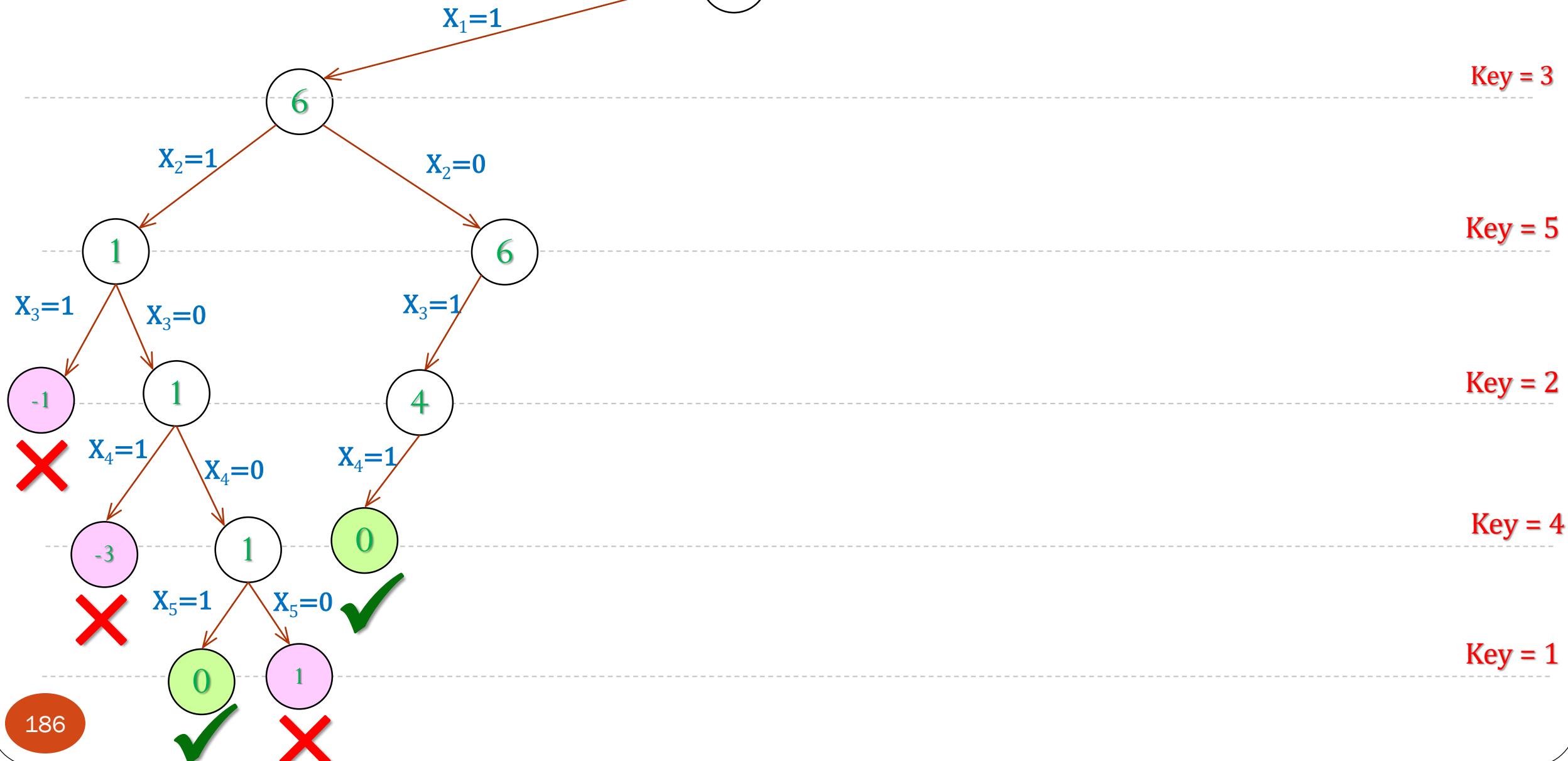
	1	2	3	4	5
X	1	0	1	1	0

1	2	3	4	5
Key	3	5	2	4

# Start

# Solutions

1	1	0	0	1
1	0	1	1	0



X	1	0	1	0	0
1 2 3 4 5	1	2	3	4	5

Key	3	5	2	4	1
1 2 3 4 5	1	2	3	4	5

Start

9

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

$X_2 = 1$

$X_2 = 0$

$X_3 = 1$

$X_3 = 0$

$X_3 = 1$

$X_4 = 1$

$X_4 = 0$

$X_4 = 1$

$X_5 = 1$

$X_5 = 0$

Key = 3

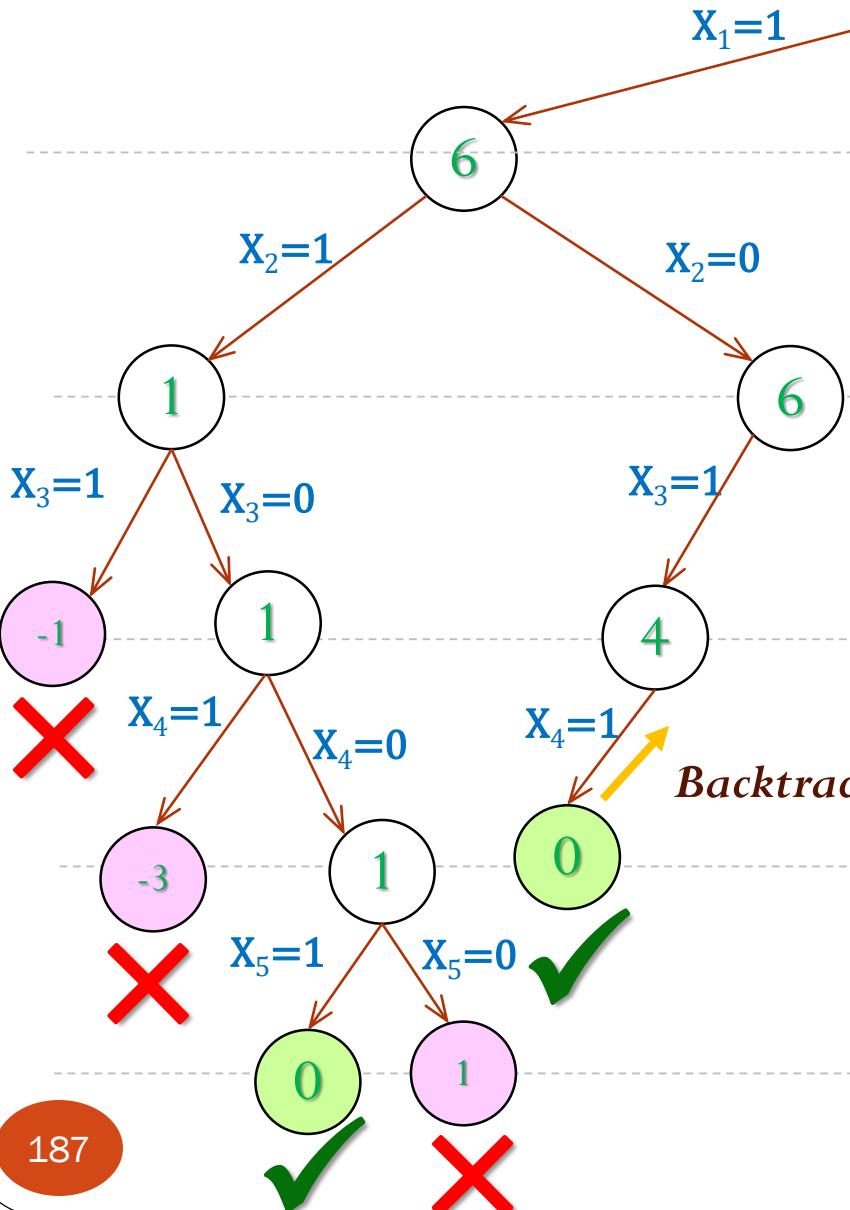
Key = 5

Key = 2

Key = 4

Key = 1

Backtrack



	1	2	3	4	5
X	1	0	1	0	0

	1	2	3	4	5
Key	3	5	2	4	1

Start

9

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

6

$X_2 = 1$

1

$X_2 = 0$

6

$X_3 = 1$

-1

$X_3 = 0$

1

$X_3 = 1$

4

$X_3 = 0$

0

$X_4 = 1$

-3

X

$X_4 = 0$

1

$X_4 = 1$

4

X

$X_5 = 1$

0

✓

$X_5 = 0$

1

$X_5 = 1$

4

✓

Key = 3

Key = 5

Key = 2

Key = 4

Key = 1

X	1	0	1	0	1
1 2 3 4 5	1	2	3	4	5

Key	3	5	2	4	1
1 2 3 4 5	1	2	3	4	5

Start

9

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

6

$X_2 = 1$

$X_2 = 0$

1

$X_3 = 1$

$X_3 = 0$

-1

+

X

$X_3 = 1$

$X_3 = 0$

1

+

X

4

+

X

0

+

X

4

+

X

1

+

X

0

+

X

1

+

X

3

+

X

0

+

X

1

+

X

2

+

X

4

+

X

5

+

X

Key = 3

Key = 5

Key = 2

Key = 4

Key = 1

Backtrack

X	1	0	1	0	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3

$X_2 = 1$

$X_2 = 0$

Key = 5

$X_3 = 1$

$X_3 = 0$

Key = 2

$X_3 = 1$

$X_3 = 0$

Key = 4

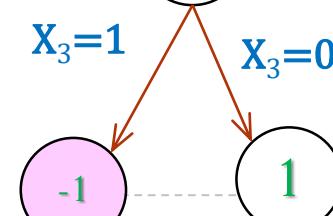
$X_4 = 1$

$X_4 = 0$

Key = 1

$X_4 = 1$

$X_4 = 0$



Backtrack

X	1	0	1	0	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

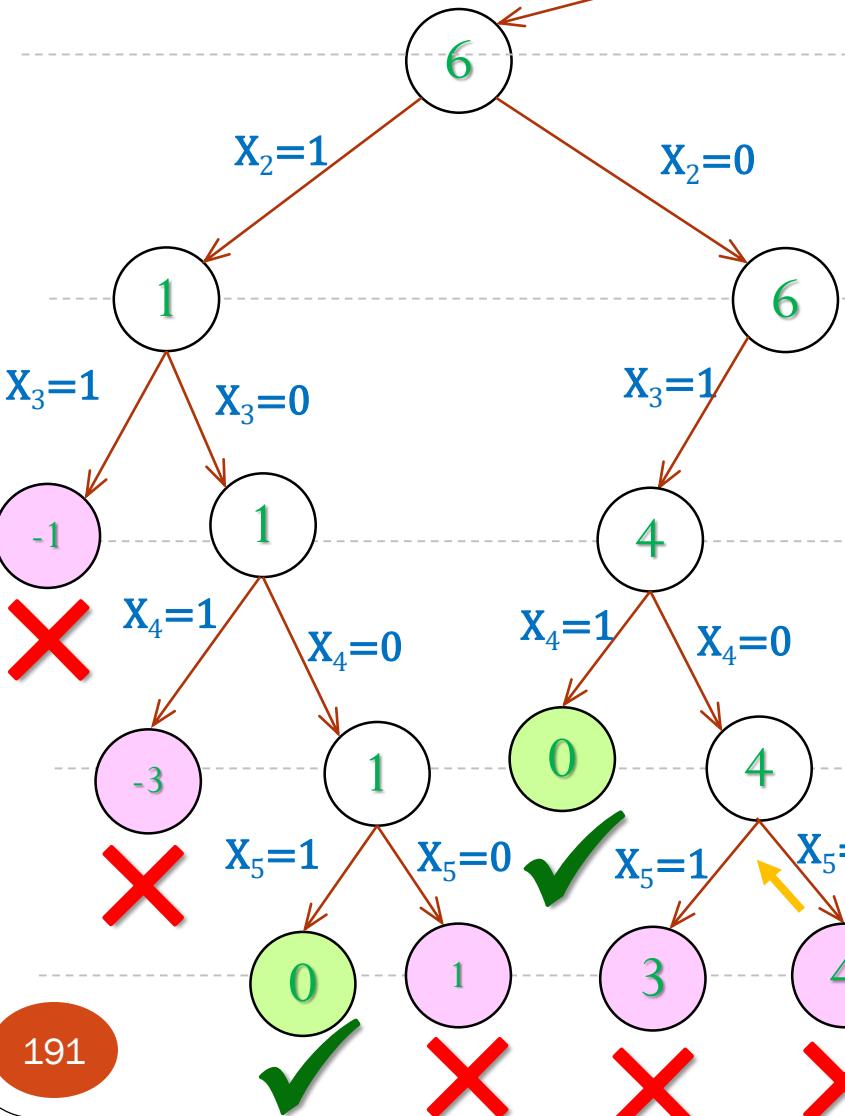
1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3



Key = 5

Key = 2

Key = 4

Key = 1

X	1	0	1	0	0
1 2 3 4 5	1	2	3	4	5

Key	3	5	2	4	1
1 2 3 4 5	1	2	3	4	5

Start

9

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

6

$X_2 = 1$

$X_2 = 0$

1

$X_3 = 1$

$X_3 = 0$

-1

+

1

$X_3 = 1$

$X_3 = 0$

1

+

1

+

1

-3

+

1

4

+

4

+

0

+

1

+

0

+

1

+

3

+

4

0

+

3

+

4

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+

0

+

1

+</

	1	2	3	4	5
X	1	0	0	0	0

1	2	3	4	5
Key	3	5	2	4

# Start

# Solutions

1	1	0	0	1
1	0	1	1	0



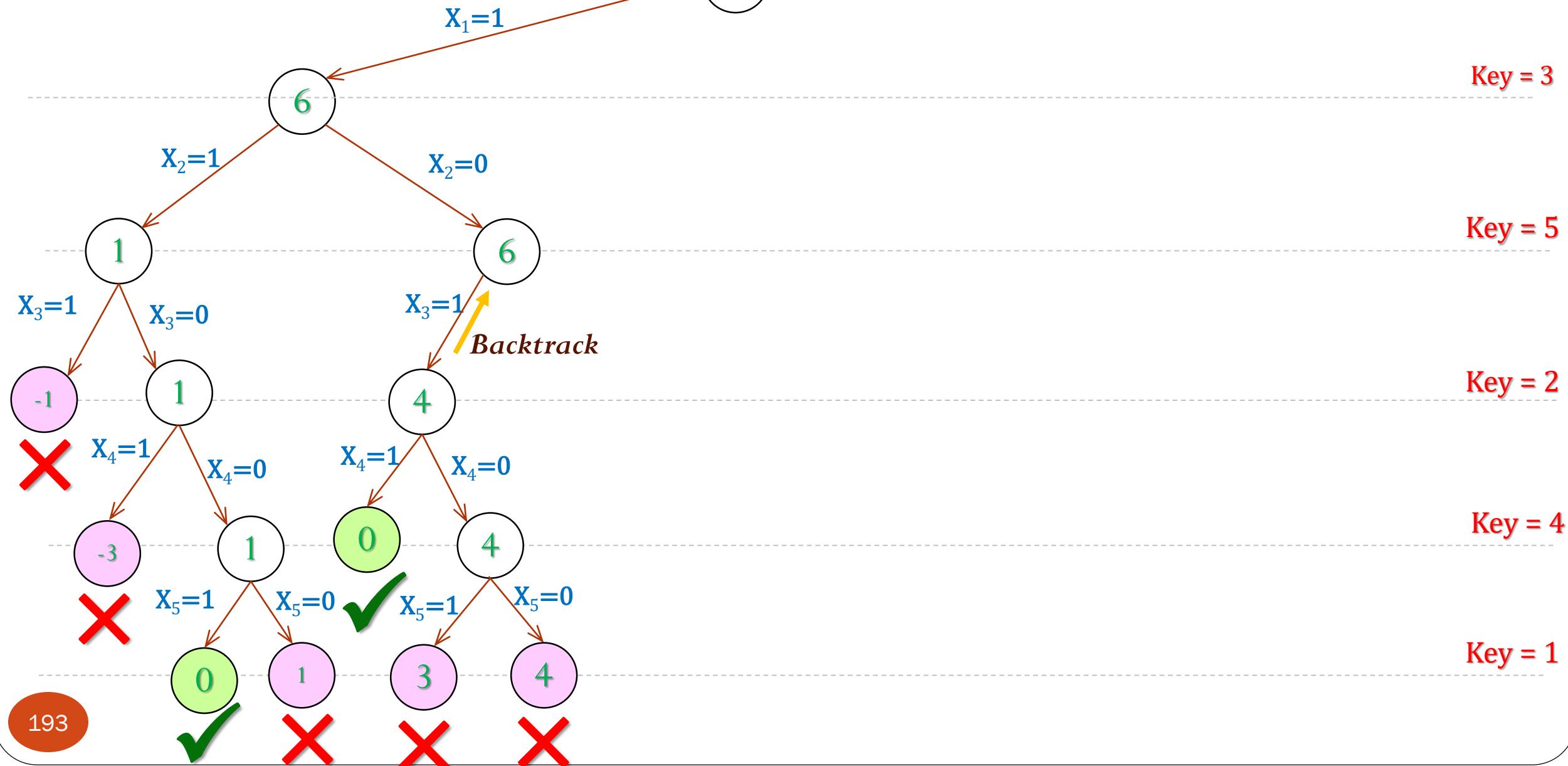
# SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

## DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

LINK TRANSPARENCY | THINK SASTRA



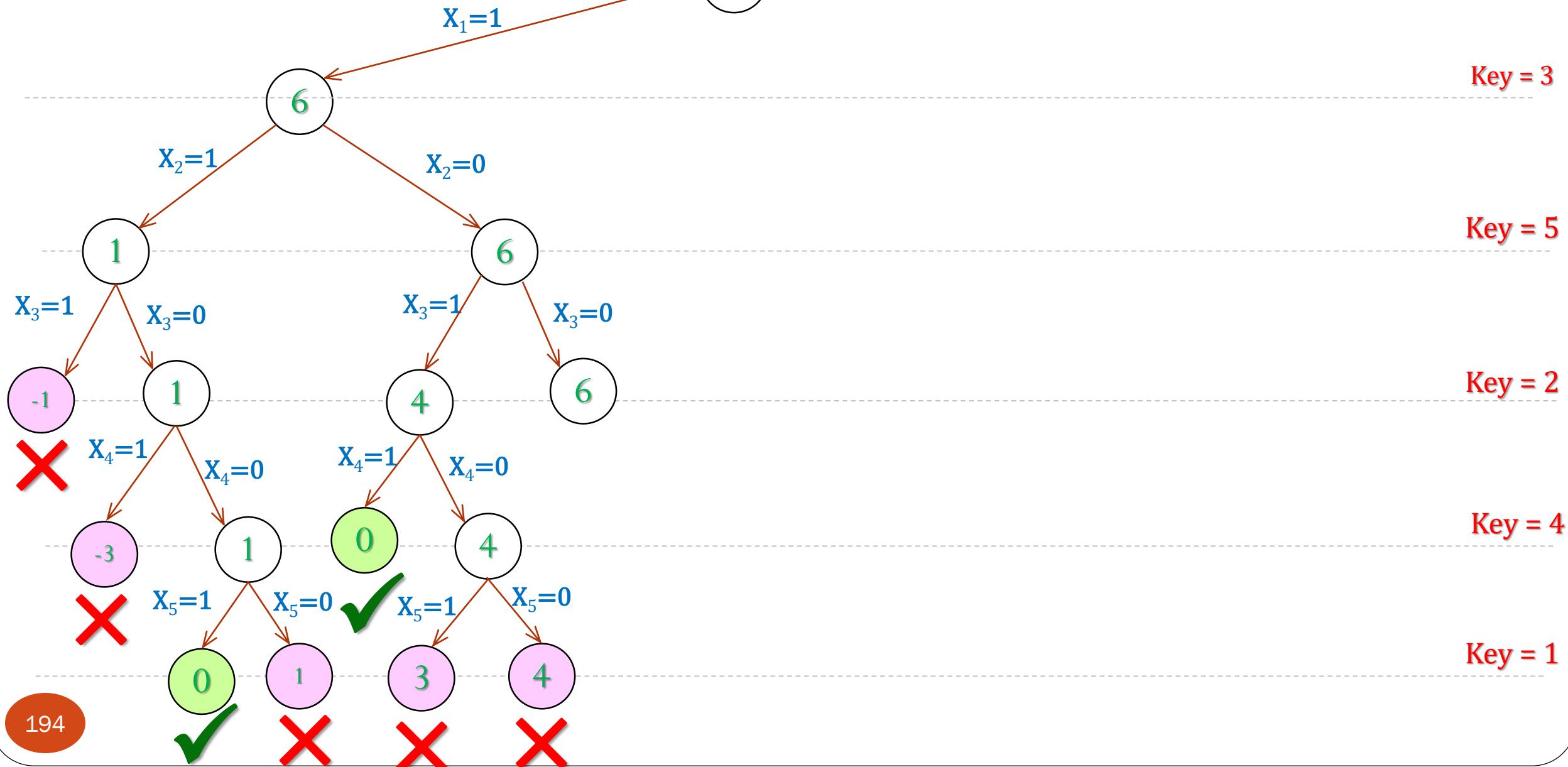
	1	2	3	4	5
X	1	0	0	0	0

1	2	3	4	5
Key	3	5	2	4

# Start

# Solutions

1	1	0	0	1
1	0	1	1	0



X	1	0	0	1	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3

$X_2 = 1$

6

$X_2 = 0$

6

Key = 5

$X_3 = 1$

1

$X_3 = 0$

1

$X_3 = 1$

6

$X_3 = 0$

6

Key = 2

$X_4 = 1$

-1

$X_4 = 0$

1

$X_4 = 1$

1

$X_4 = 0$

4

$X_4 = 1$

0

$X_4 = 0$

4

$X_4 = 1$

2

Key = 4

$X_5 = 1$

-3

$X_5 = 0$

1

$X_5 = 1$

0

$X_5 = 0$

3

$X_5 = 1$

4

Key = 1

X	1	0	0	1	1
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3

$X_2 = 1$

$X_2 = 0$

Key = 5

$X_3 = 1$   
 $X_3 = 0$

$X_3 = 1$   
 $X_3 = 0$

Key = 2

$X_4 = 1$   
 $X_4 = 0$

$X_4 = 1$   
 $X_4 = 0$

Key = 4

$X_5 = 1$   
 $X_5 = 0$

$X_5 = 1$   
 $X_5 = 0$

Key = 1

Backtrack

X	1	0	0	1	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

1	1	0	0	1
1	0	1	1	0



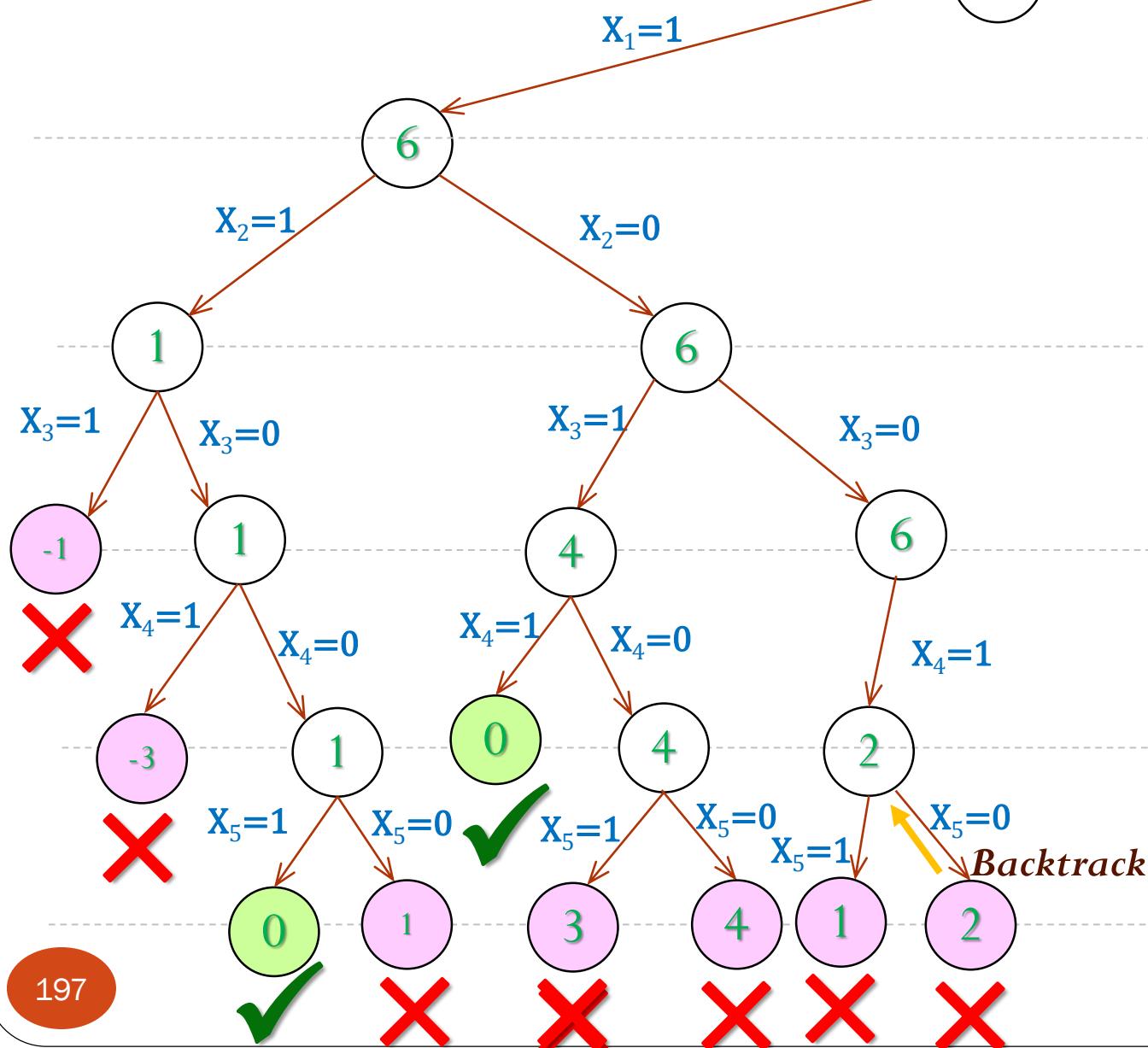
Key = 3

Key = 5

Key = 2

Key = 4

Key = 1



X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3

$X_2 = 1$

$X_2 = 0$

Key = 5

$X_3 = 1$   
 $X_3 = 0$

$X_3 = 1$   
 $X_3 = 0$

Key = 2

$X_4 = 1$   
 $X_4 = 0$

$X_4 = 1$   
 $X_4 = 0$

Key = 4

$X_5 = 1$   
 $X_5 = 0$

$X_5 = 1$   
 $X_5 = 0$

Key = 1

Backtrack

X	1	0	0	0	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3

$X_2 = 1$

$X_2 = 0$

Key = 5

$X_3 = 1$   
 $X_3 = 0$

$X_3 = 1$   
 $X_3 = 0$

Key = 2

$X_4 = 1$   
 $X_4 = 0$

$X_4 = 1$   
 $X_4 = 0$

Key = 4

$X_5 = 1$   
 $X_5 = 0$

$X_5 = 1$   
 $X_5 = 0$

Key = 1

$X_5 = 1$   
 $X_5 = 0$

$X_5 = 1$   
 $X_5 = 0$

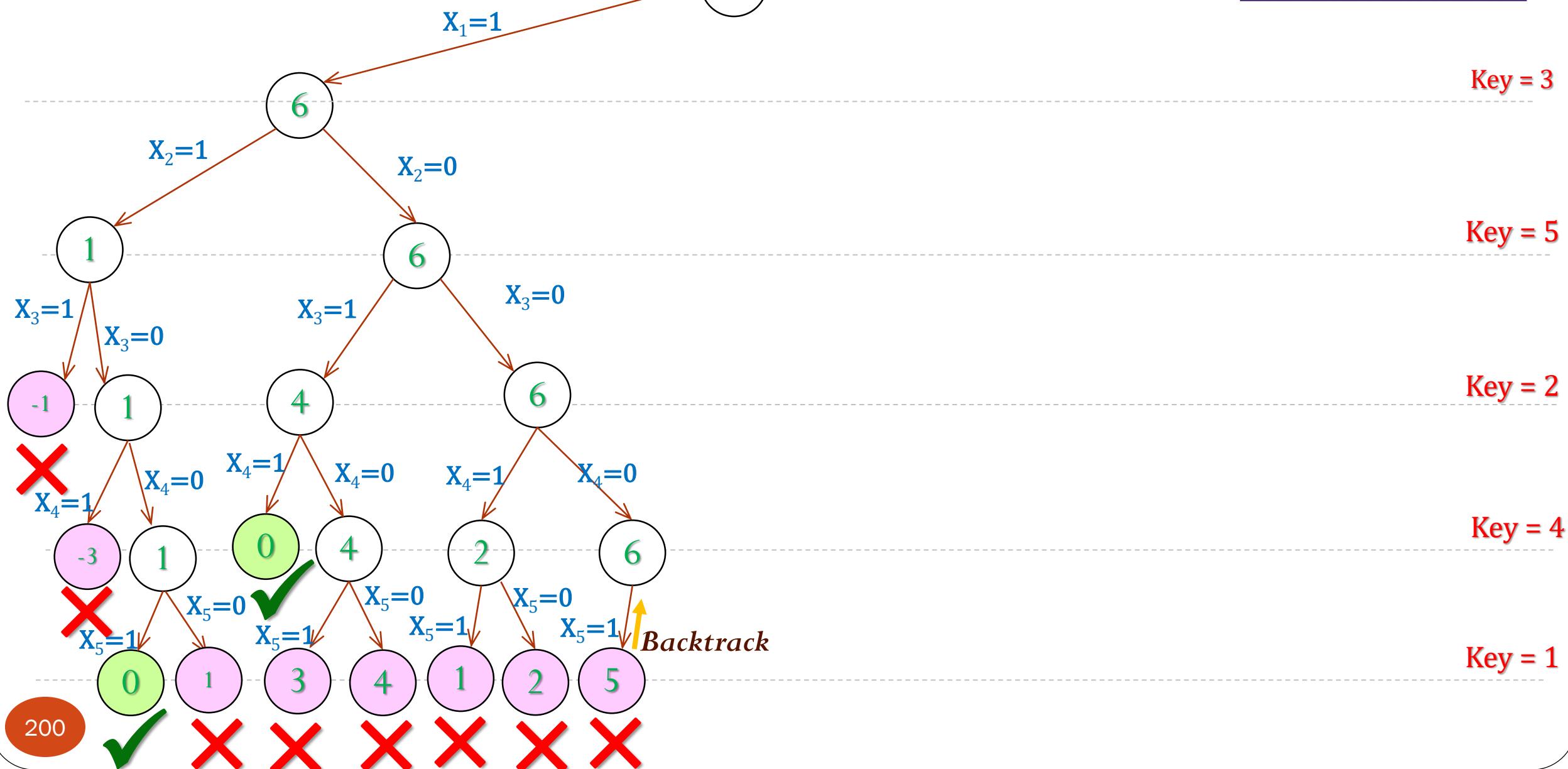
X	1	2	3	4	5
	1	0	0	0	1

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



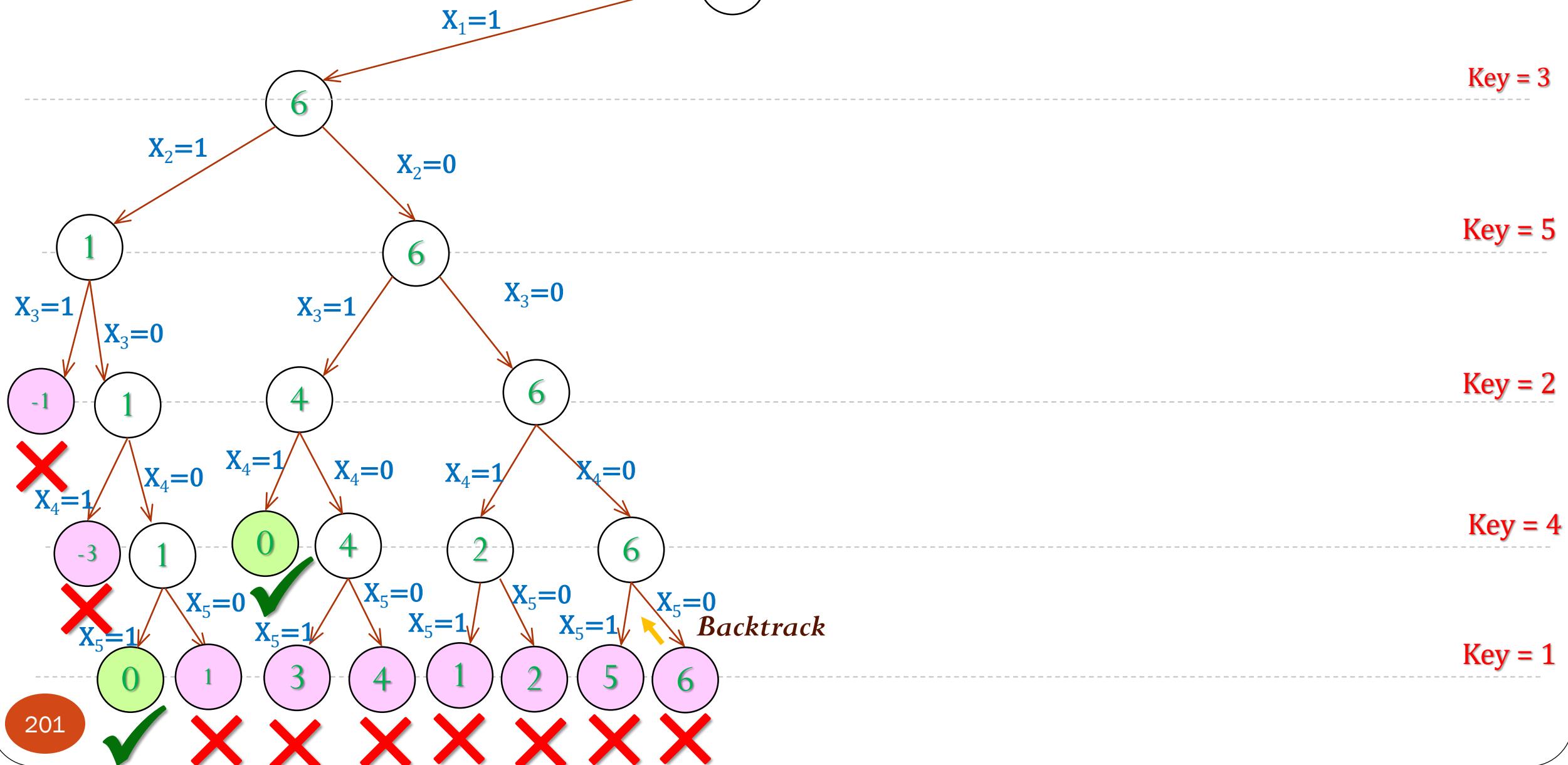
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



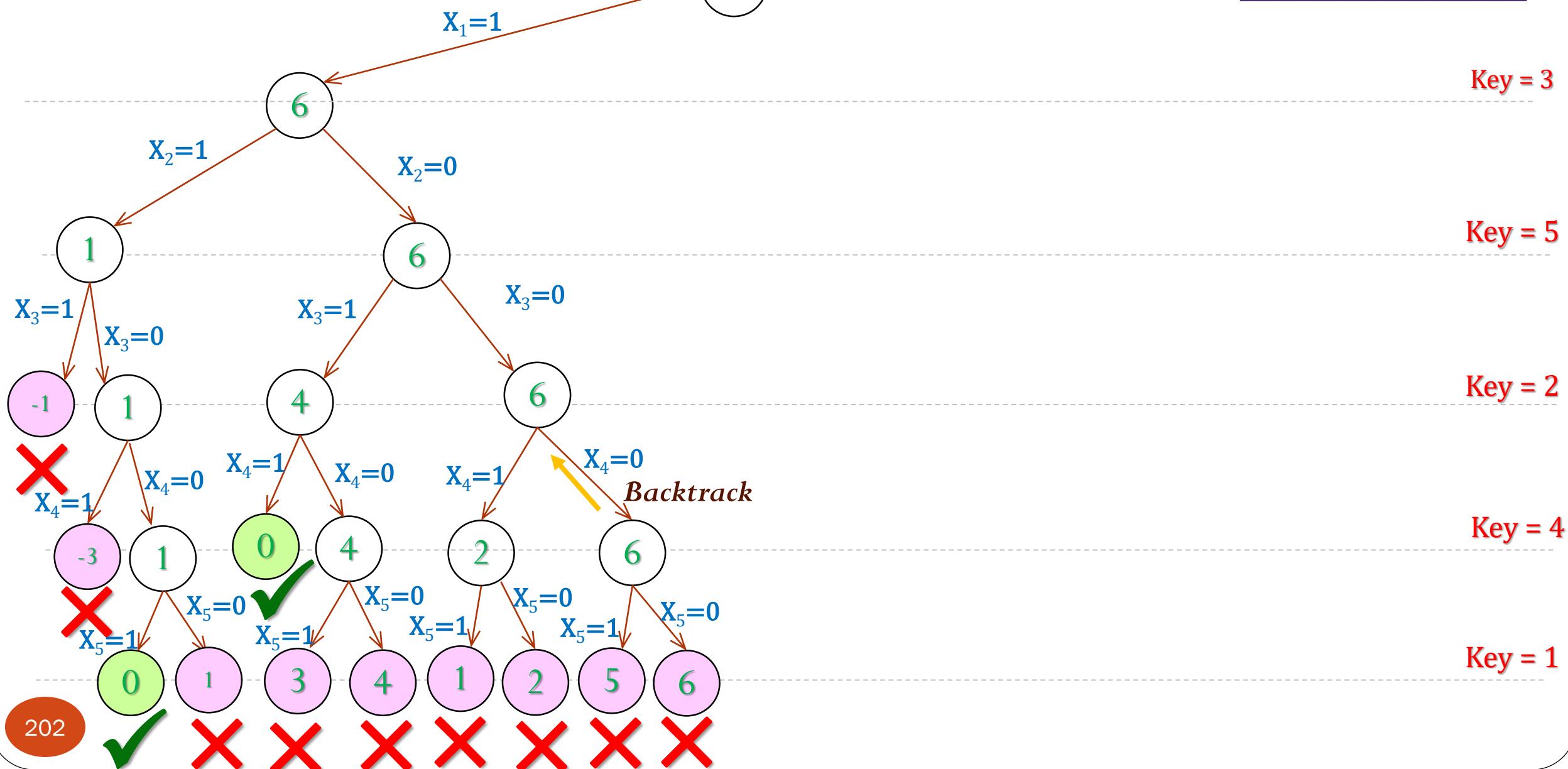
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



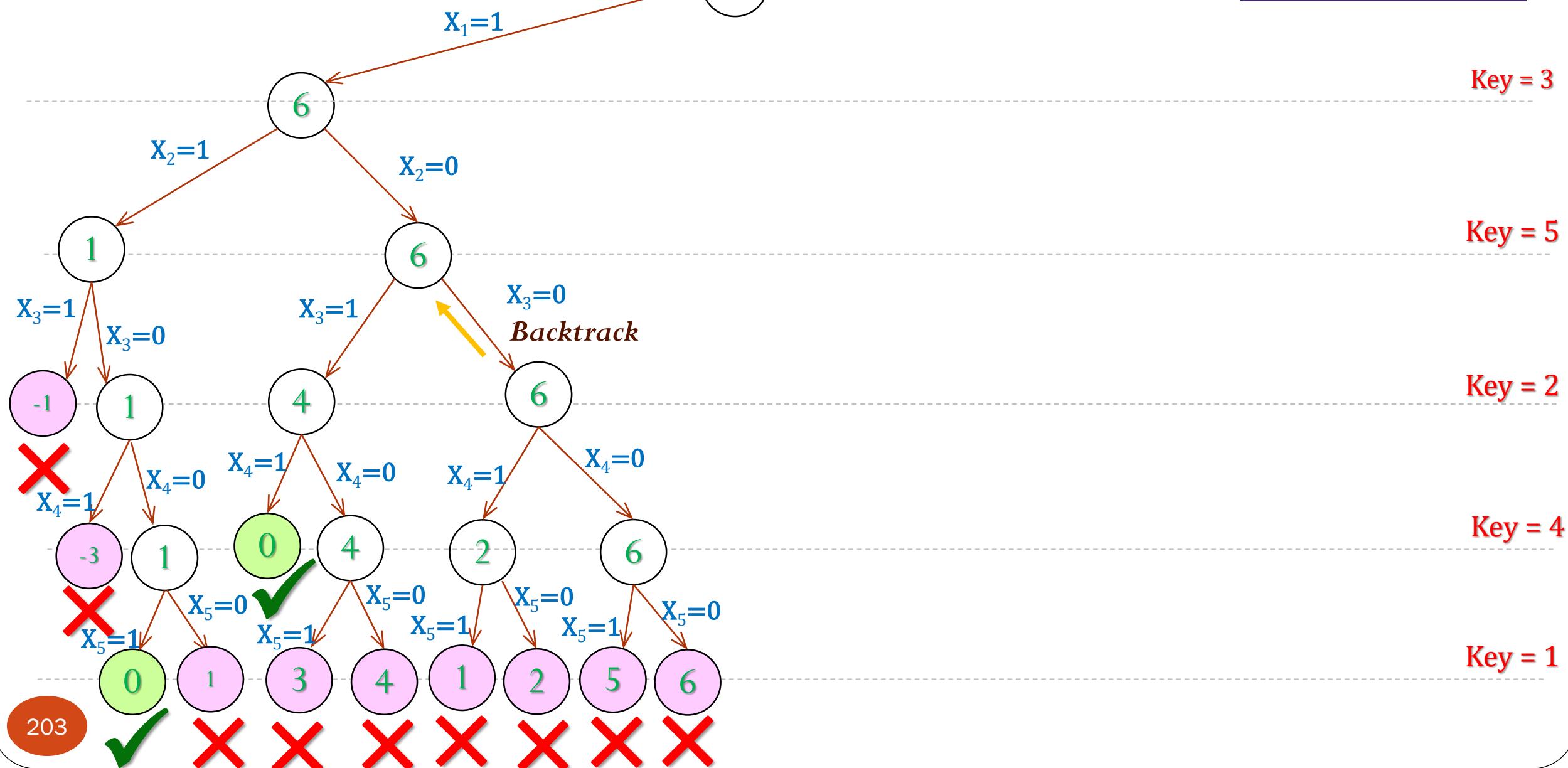
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



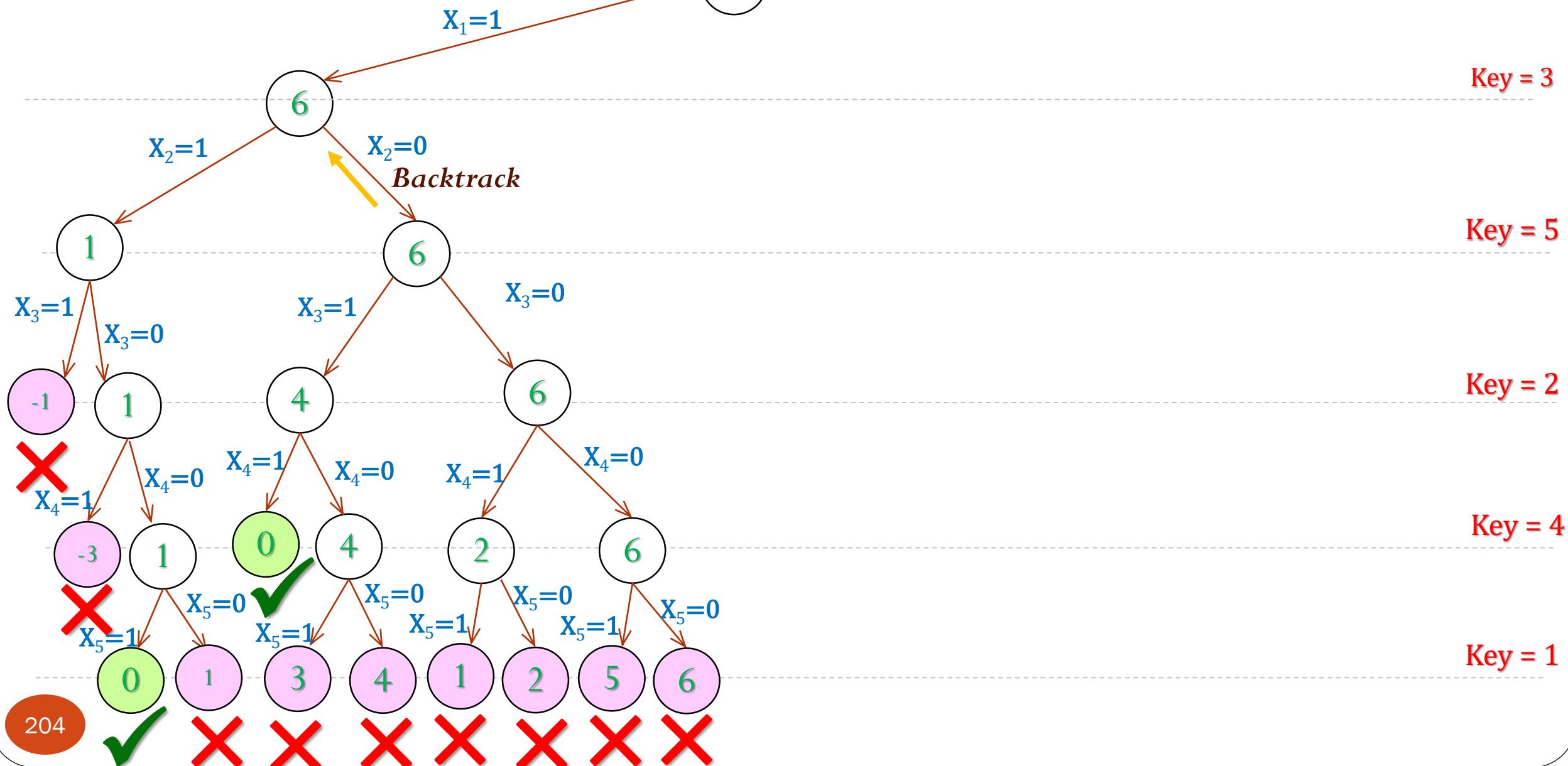
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



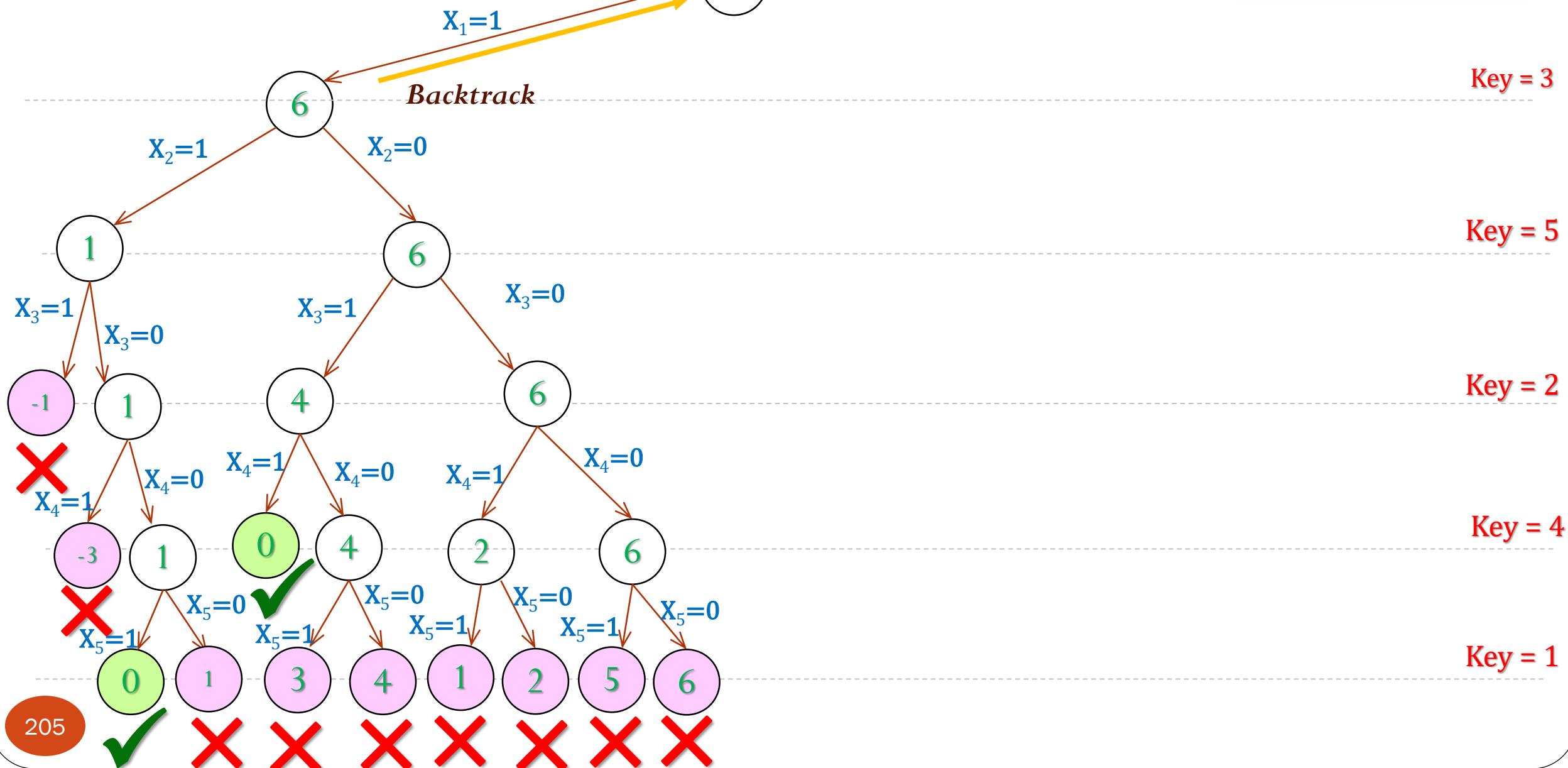
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



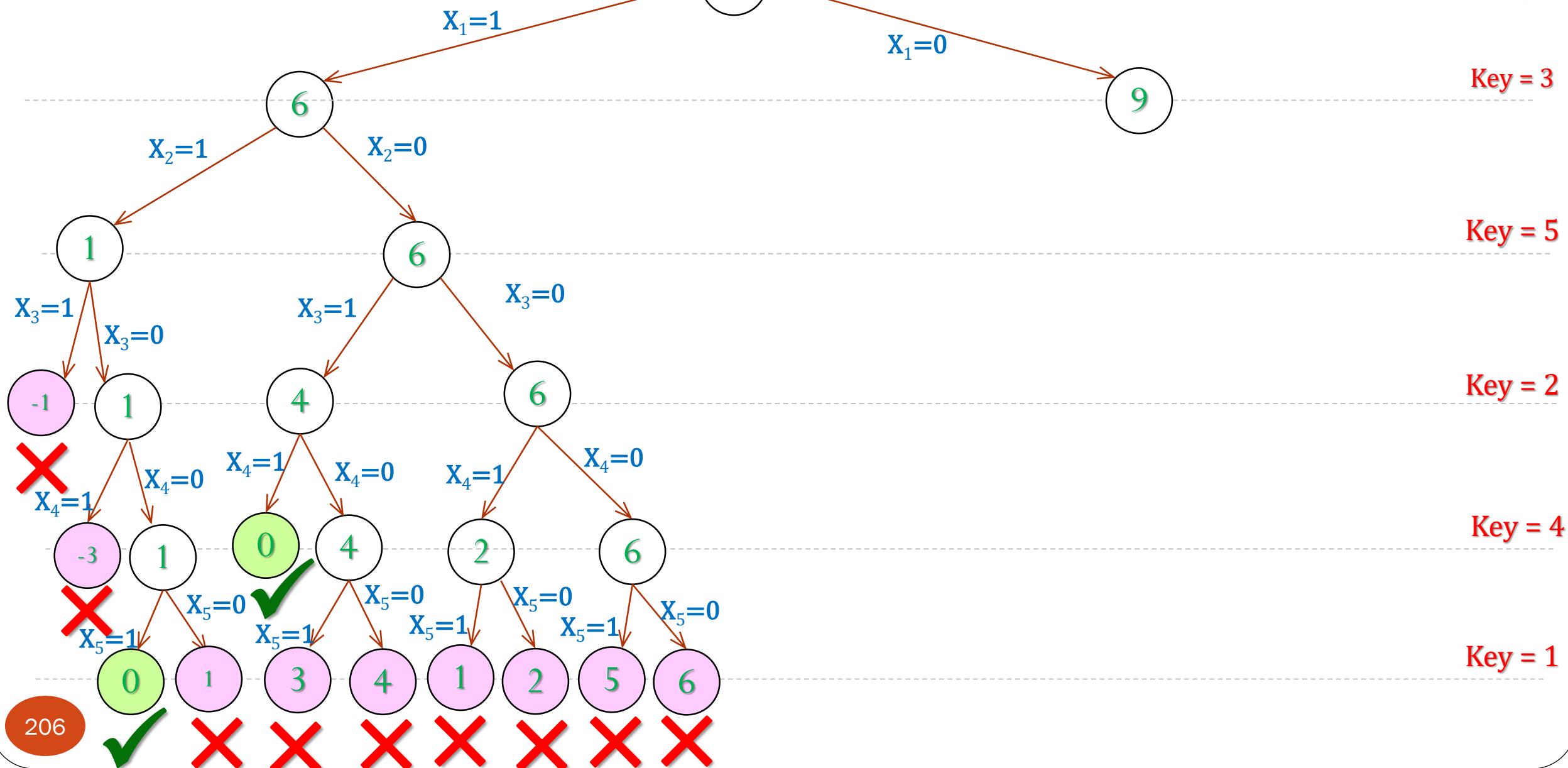
1	2	3	4	5
X	0	0	0	0

1	2	3	4	5
Key	3	5	2	4

Start

Solutions

1	1	0	0	1
1	0	1	1	0



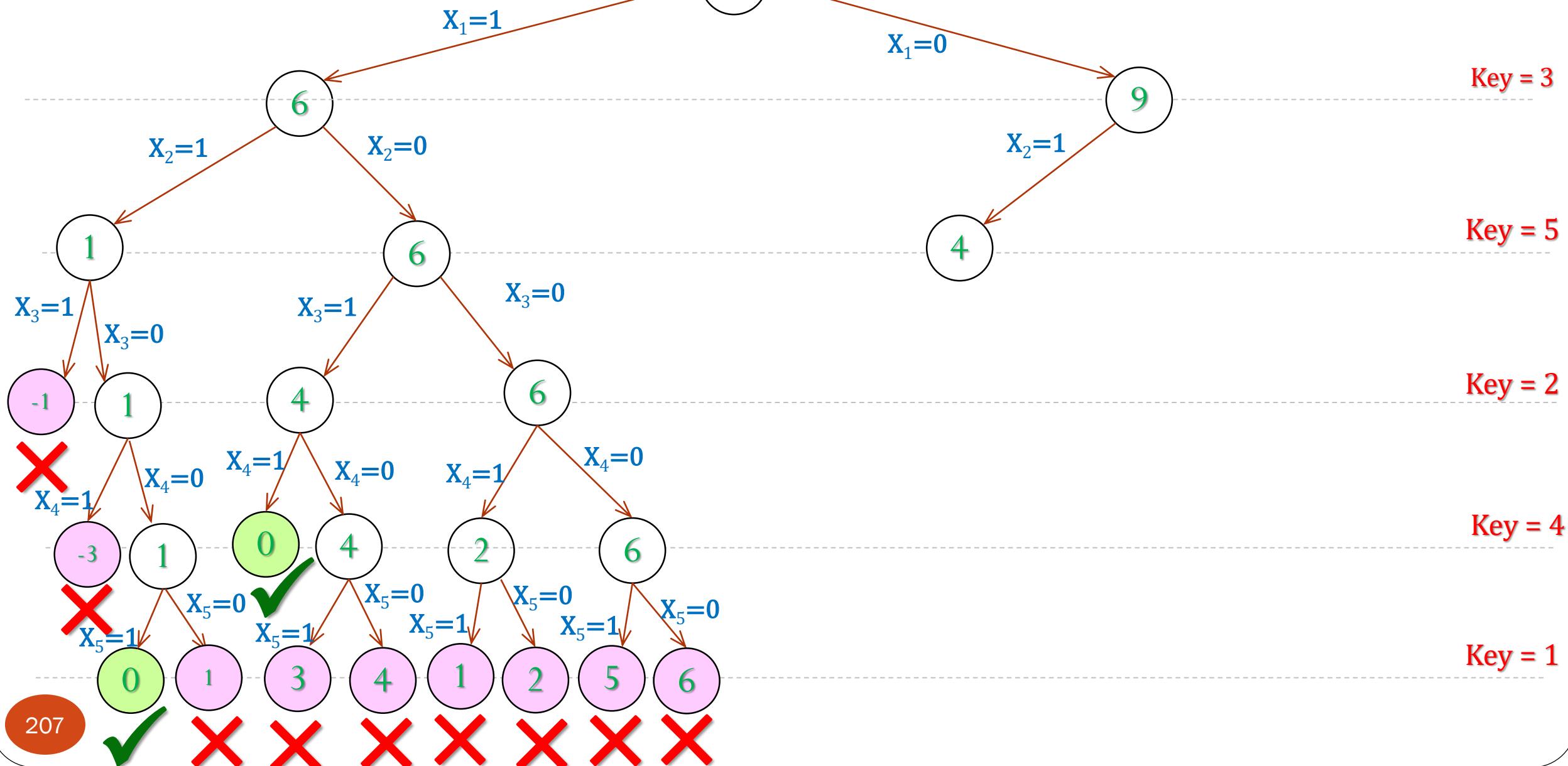
1	2	3	4	5
0	1	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



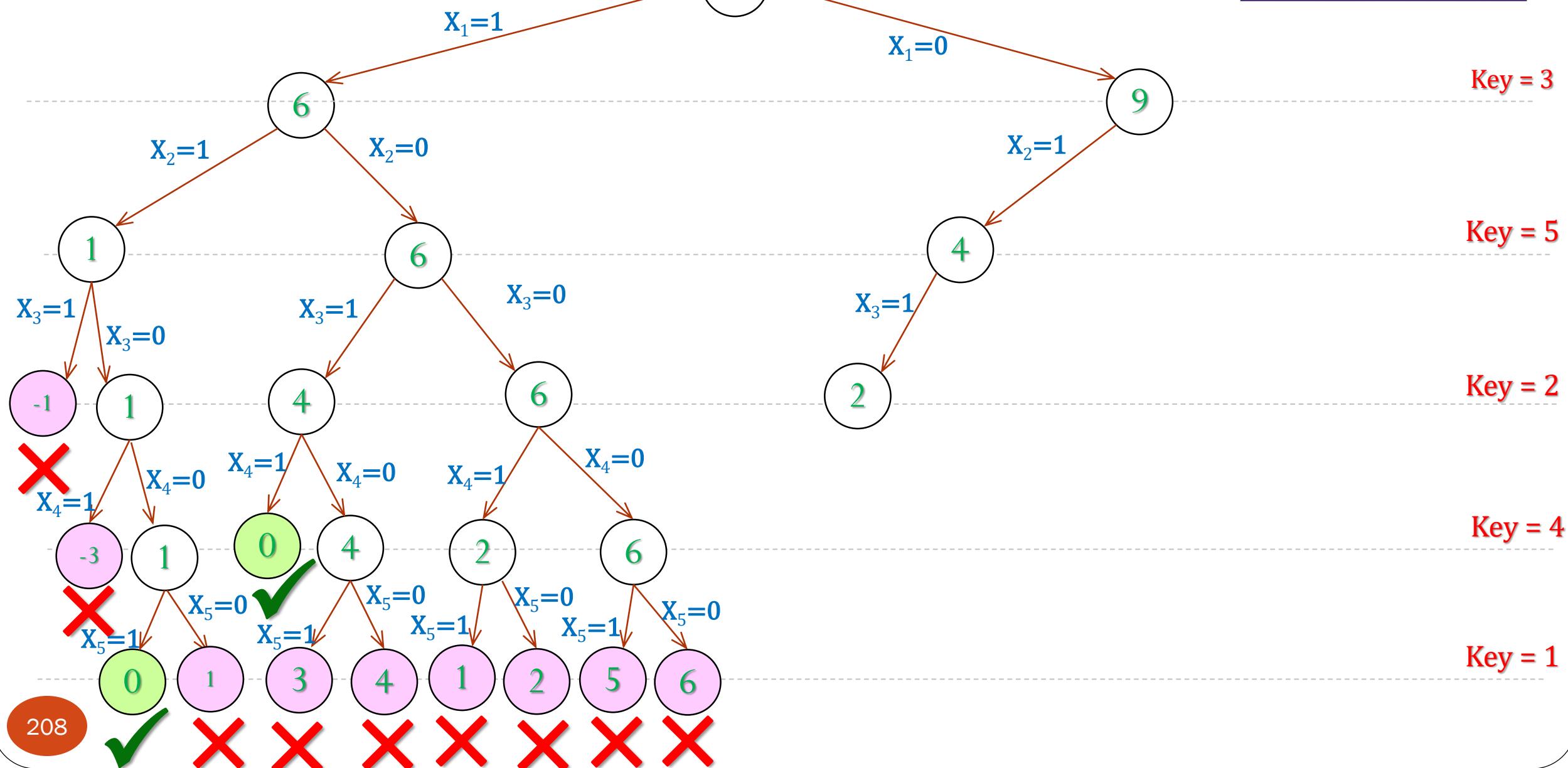
X	1	2	3	4	5
0	0	1	1	0	0

Key	1	2	3	4	5
3	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



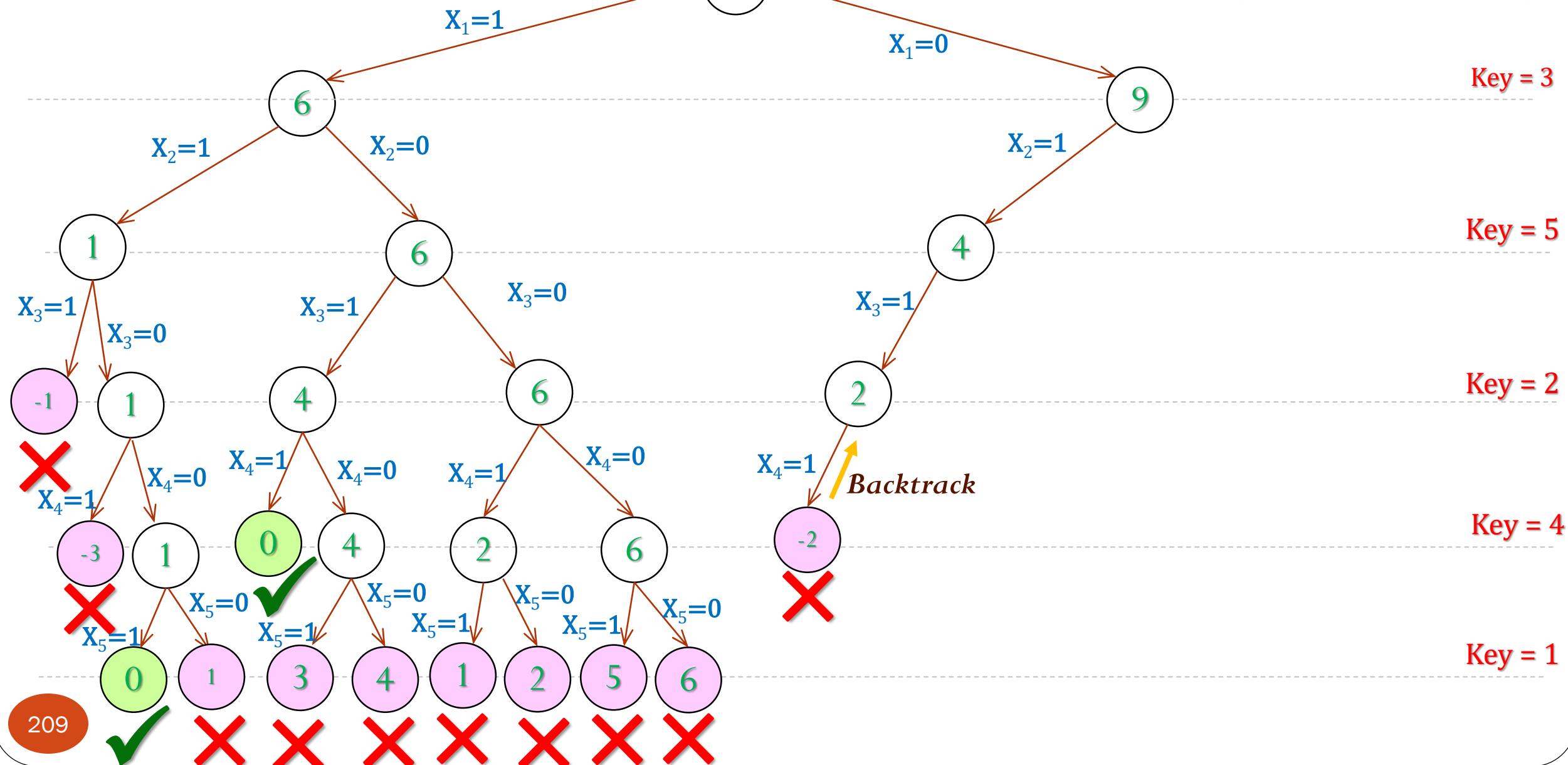
	1	2	3	4	5
X	1	1	1	1	0

	1	2	3	4	5
Key	3	5	2	4	1

# Start

# Solutions

S	1	1	0	0	1
	1	0	1	1	0



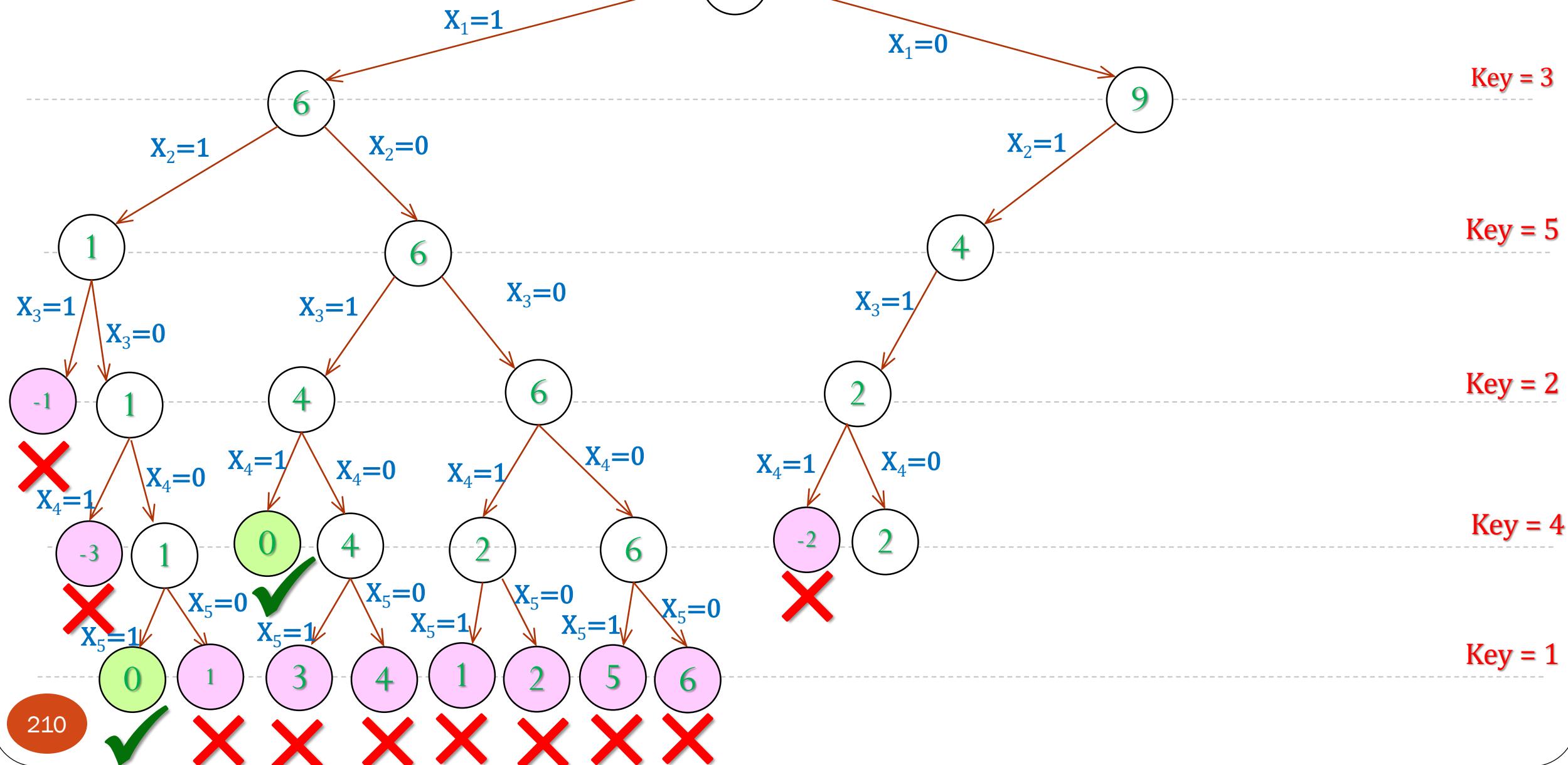
1	2	3	4	5
0	1	1	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



	1	2	3	4	5
X	0	1	1	0	1

1	2	3	4	5
Key	3	5	2	4

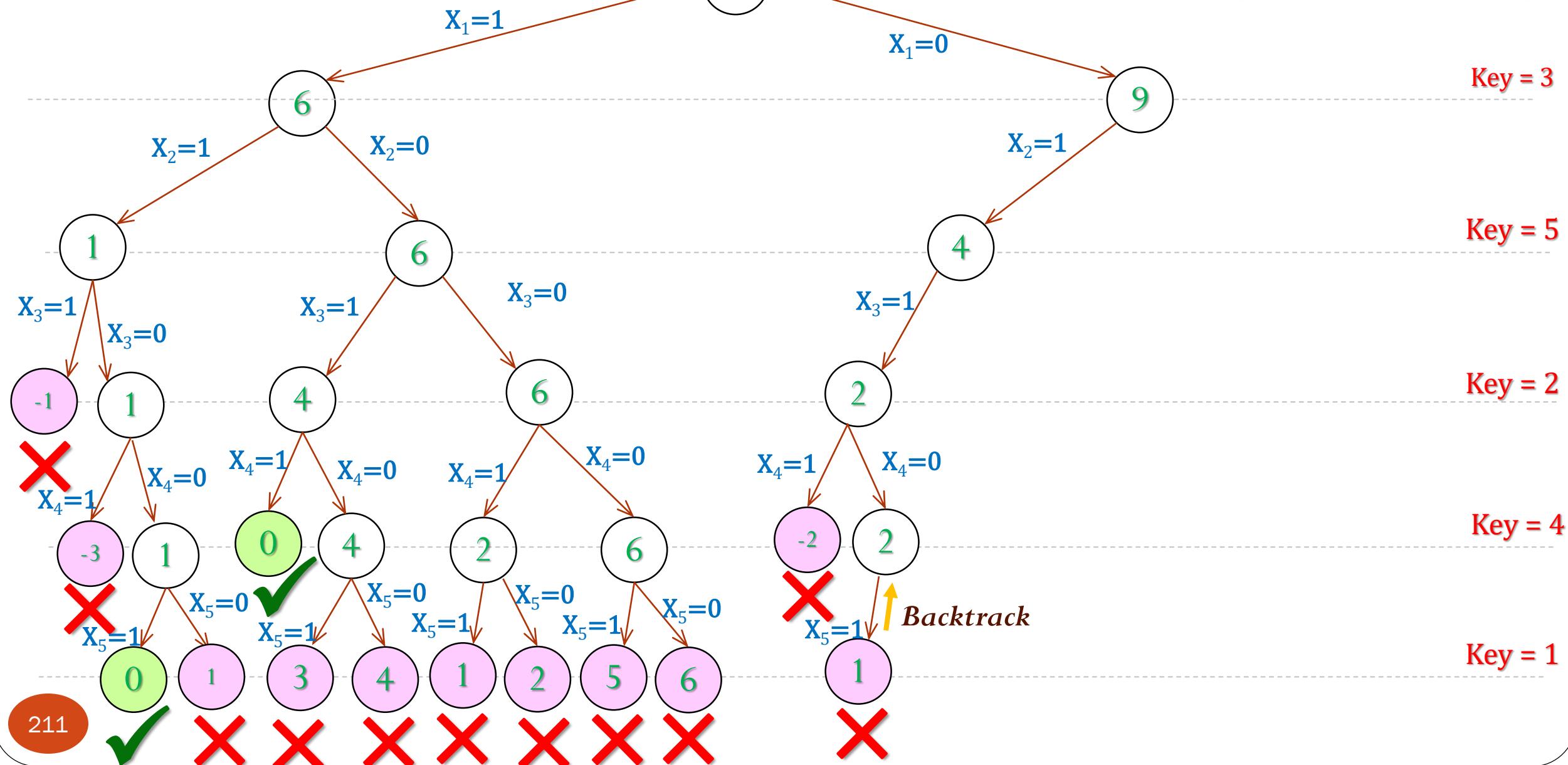
# Start

# Solutions

S	1	1	0	0	1
	1	0	1	1	0



The logo for SASTRA Deemed to be University. It features the word "SASTRA" in large, bold, blue serif capital letters at the top. Below it, in smaller blue serif capital letters, is the text "ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION". Underneath that, in larger blue serif capital letters, is "DEEMED TO BE UNIVERSITY". At the bottom, in smaller blue serif capital letters, is "(U/S 3 OF THE UGC ACT, 1956)".



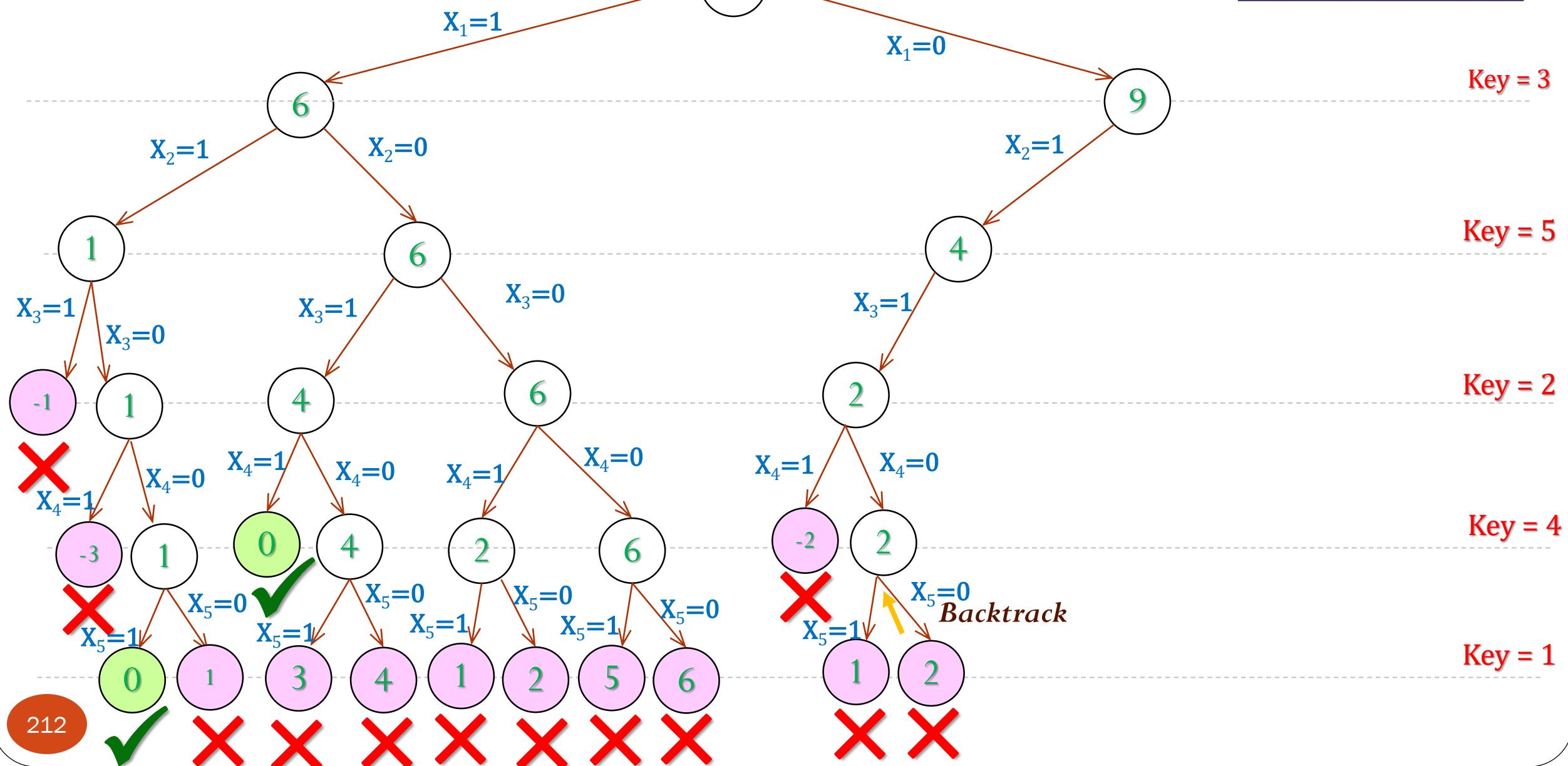
	1	2	3	4	5
X	0	1	1	0	0

	1	2	3	4	5
Key	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



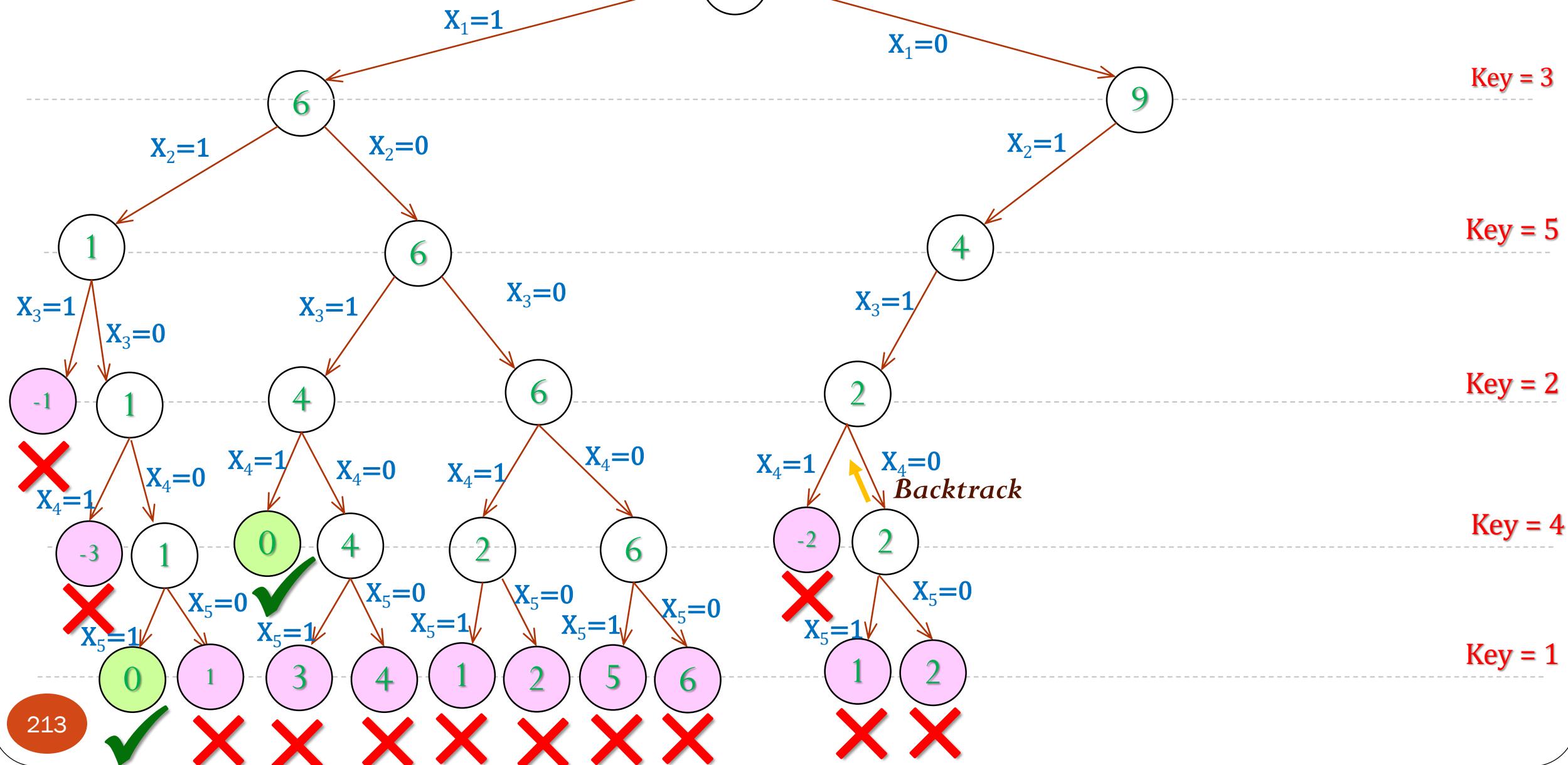
1	2	3	4	5	
X	1	0	1	0	0

1	2	3	4	5	
Key	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



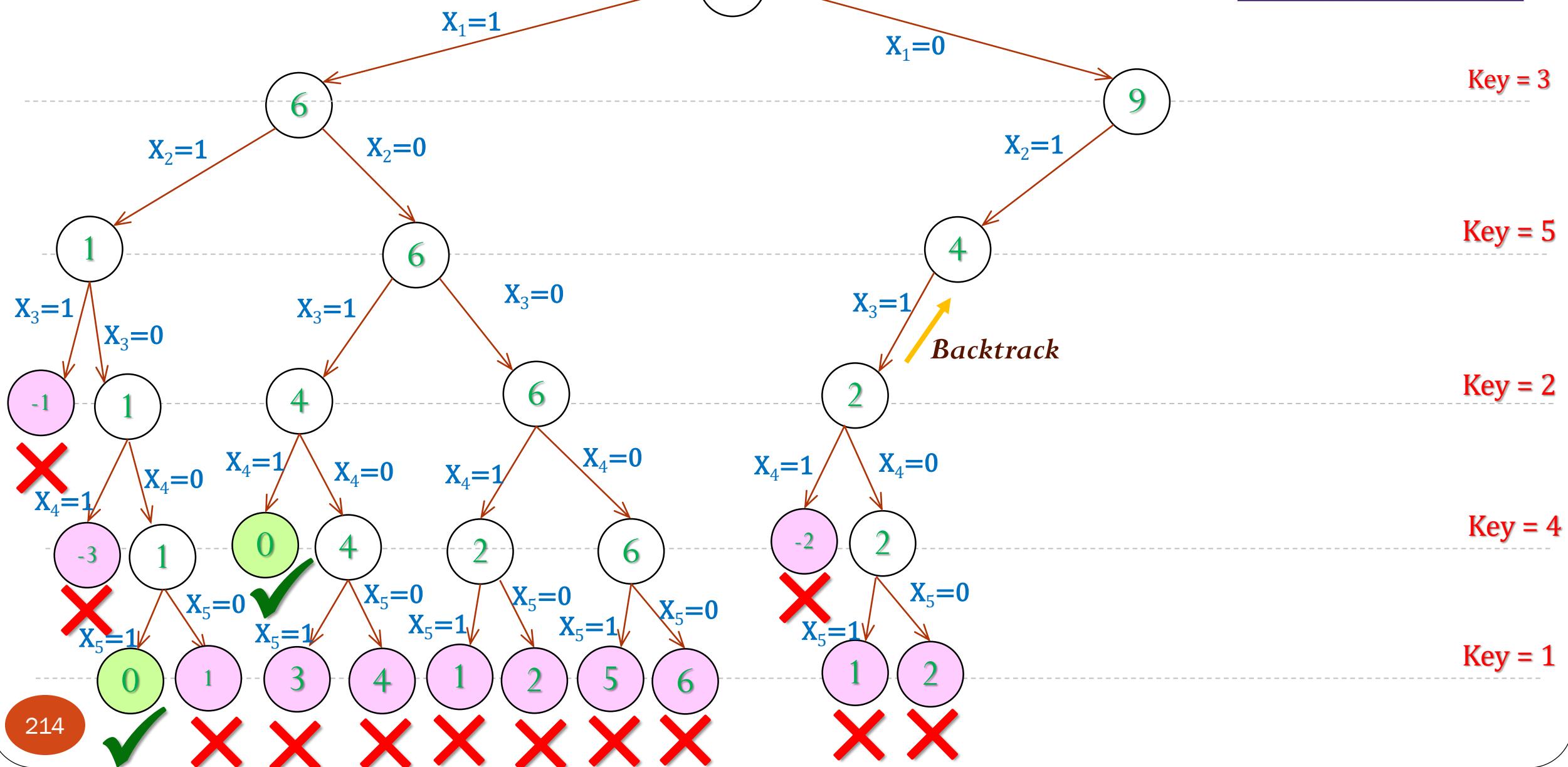
1	2	3	4	5
0	1	1	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



**X**

1	2	3	4	5
0	1	0	1	0

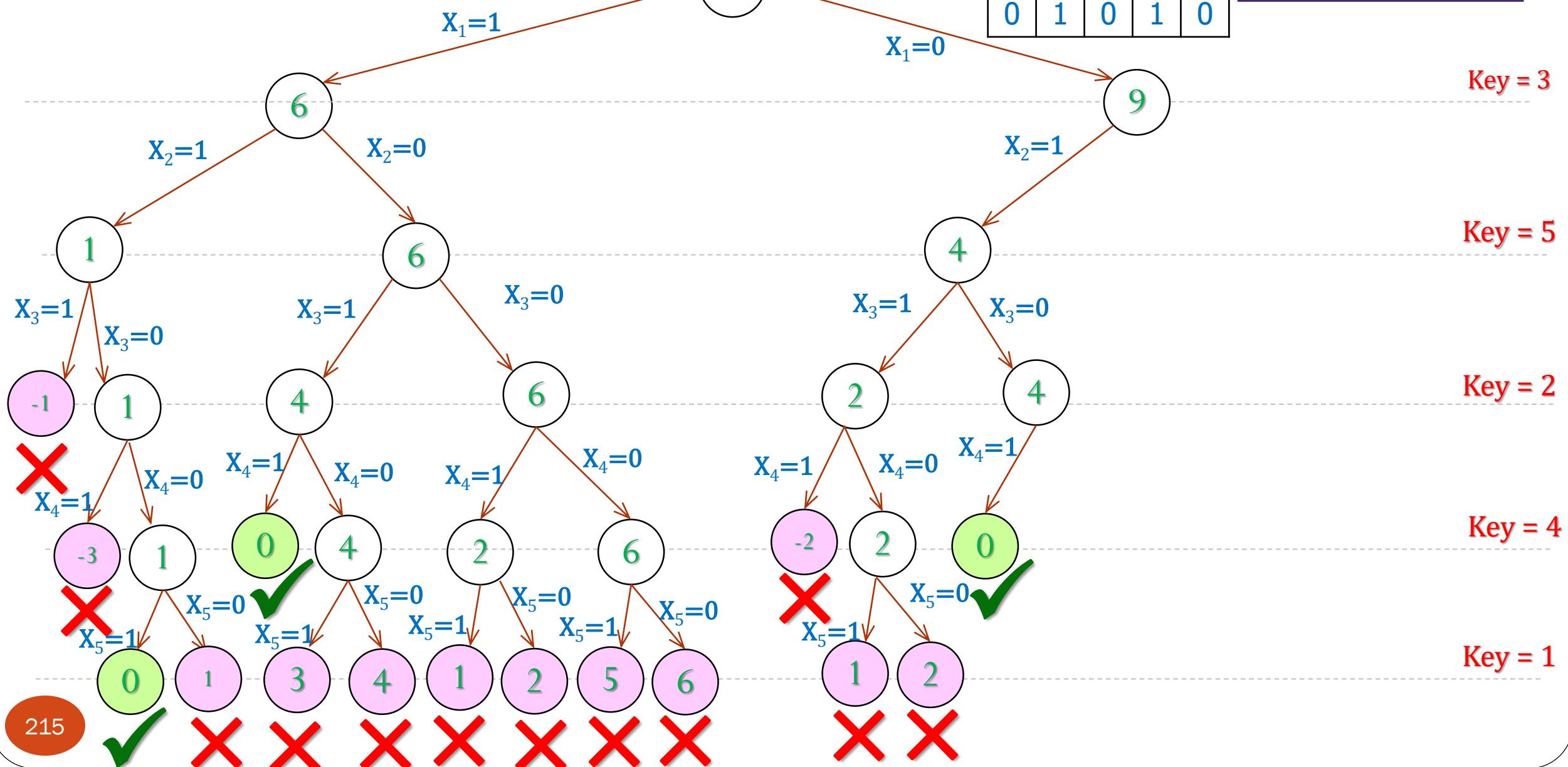
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
0	1	0	1	0



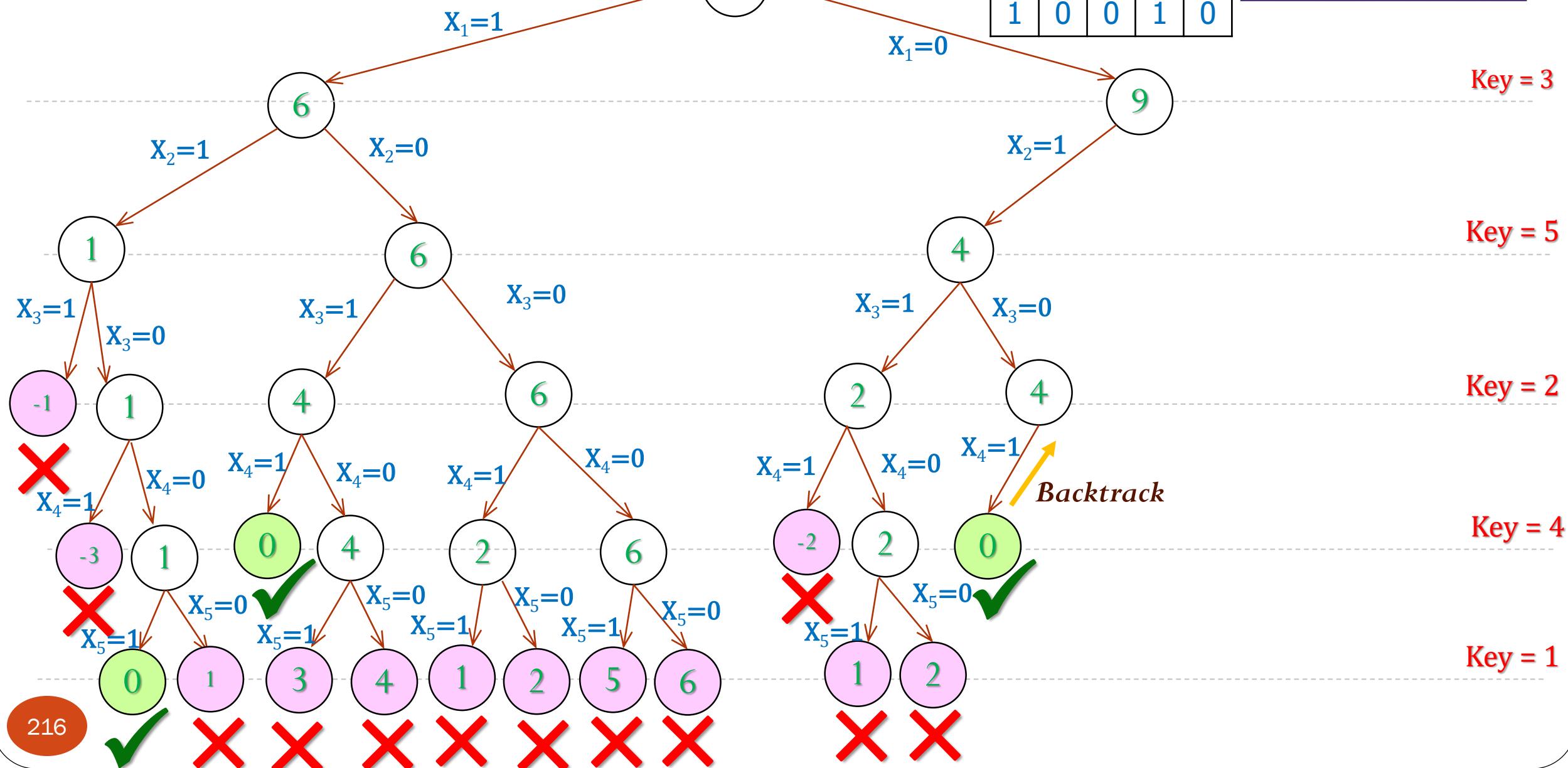
1	2	3	4	5
0	1	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



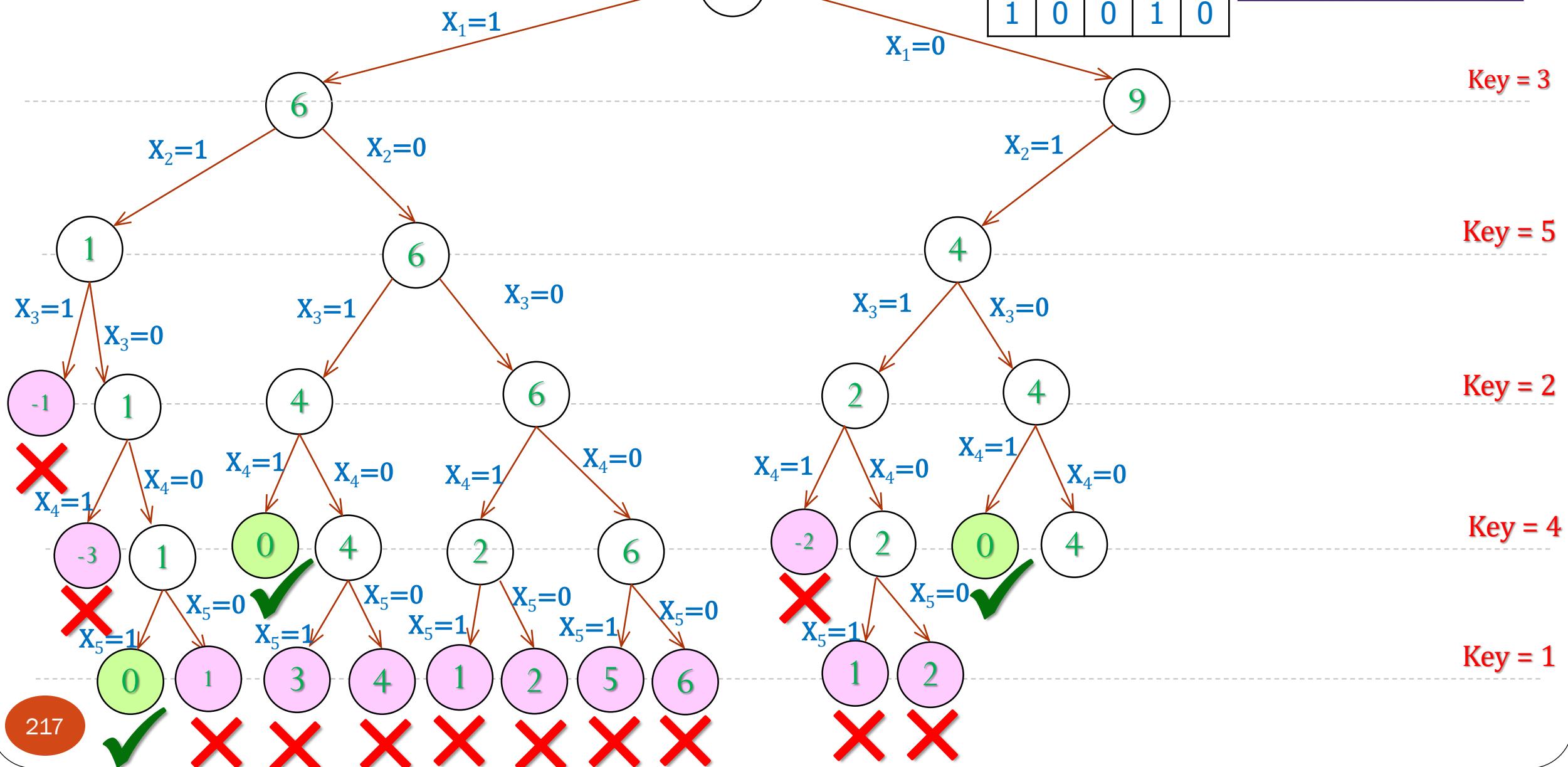
1	2	3	4	5
0	1	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**

1	2	3	4	5
0	1	0	0	1

**Key**

1	2	3	4	5
3	5	2	4	1

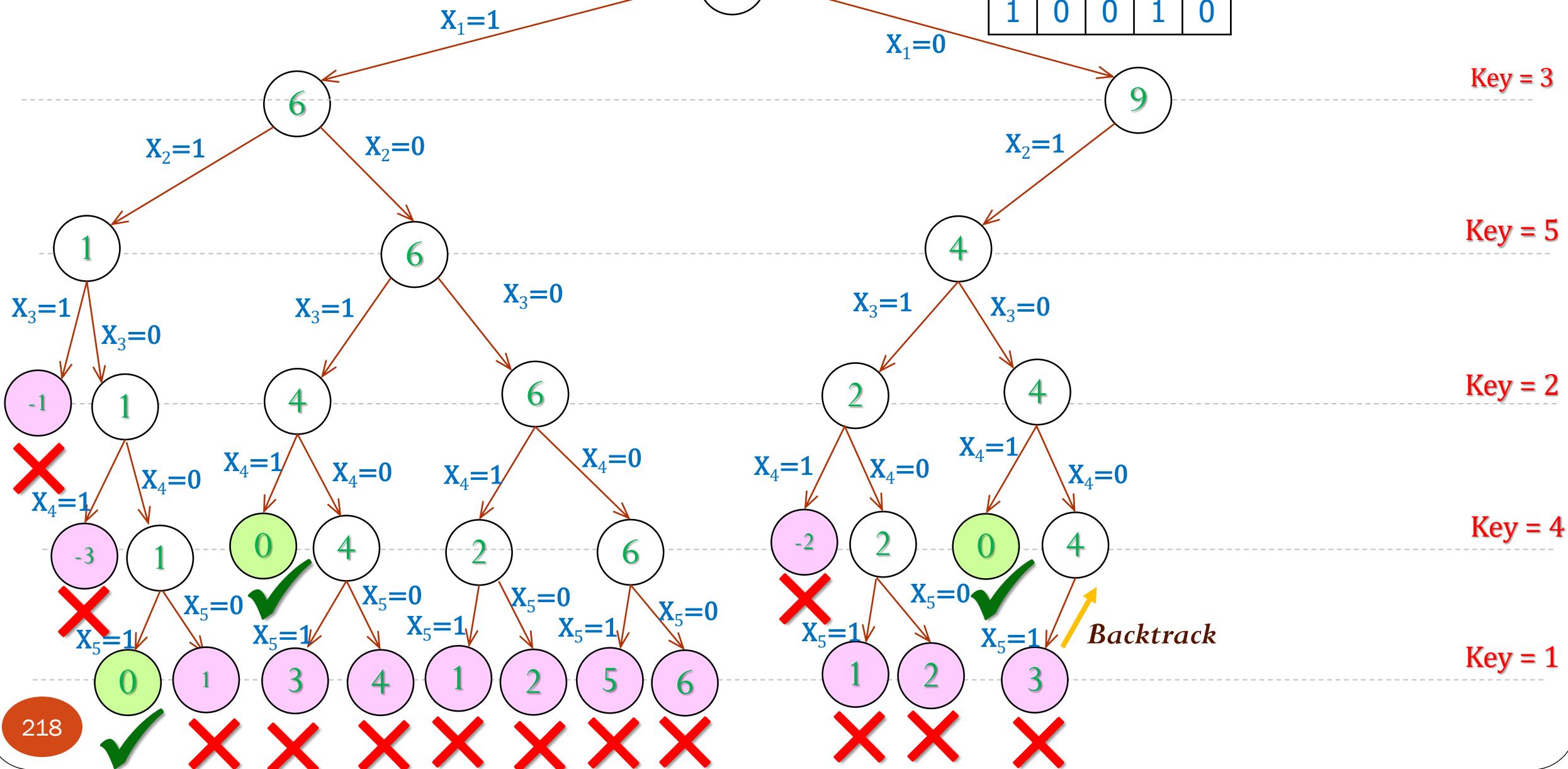
**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



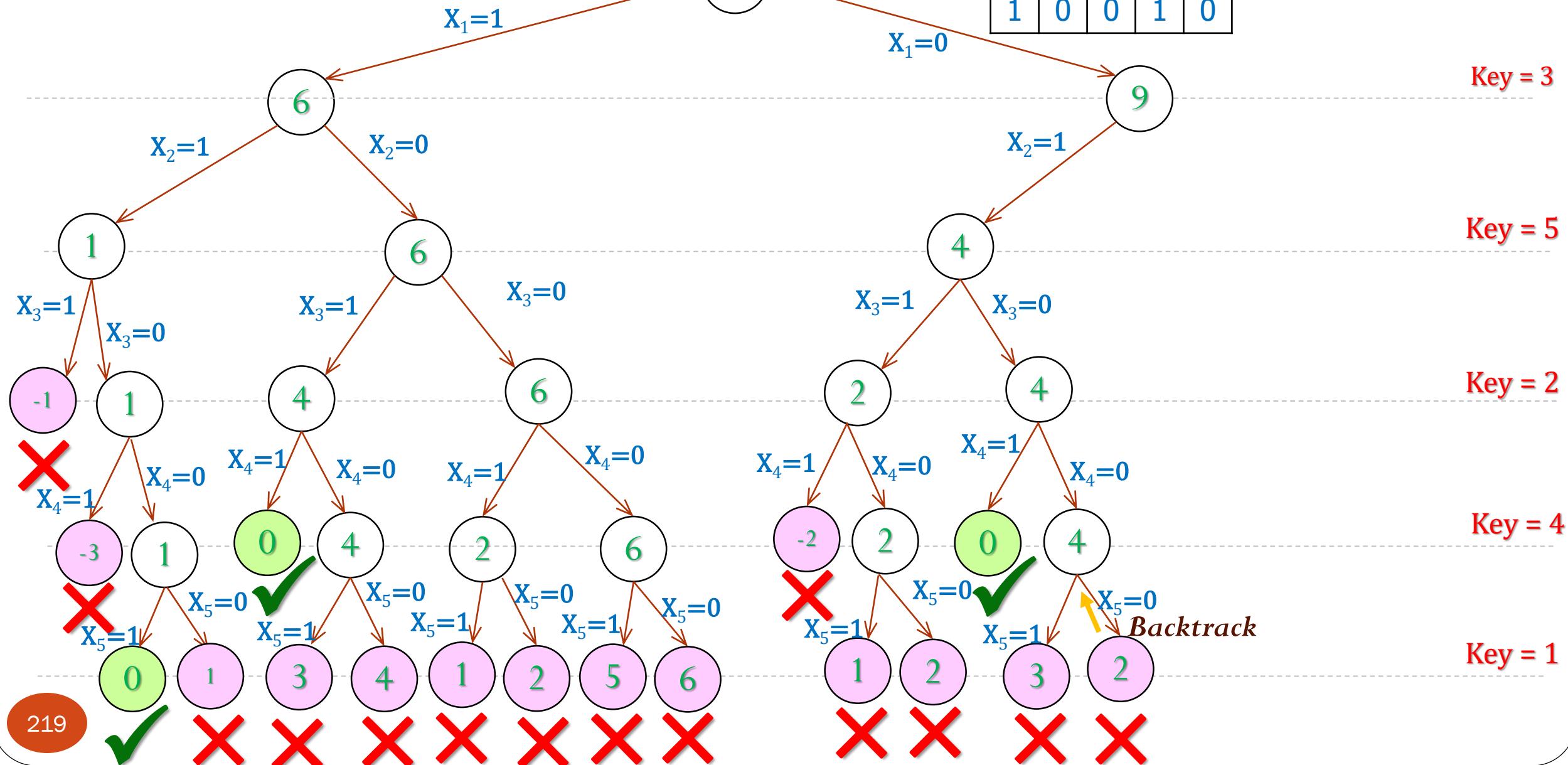
**Key = 3**



	1	2	3	4	5
X	0	1	0	0	0

1	2	3	4	5
Key	3	5	2	4

## Start



1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
**DEEMED TO BE UNIVERSITY**  
(U/S 3 OF THE UGC ACT, 1956)  
INK TRANSPARENCY   THINK SASTRA

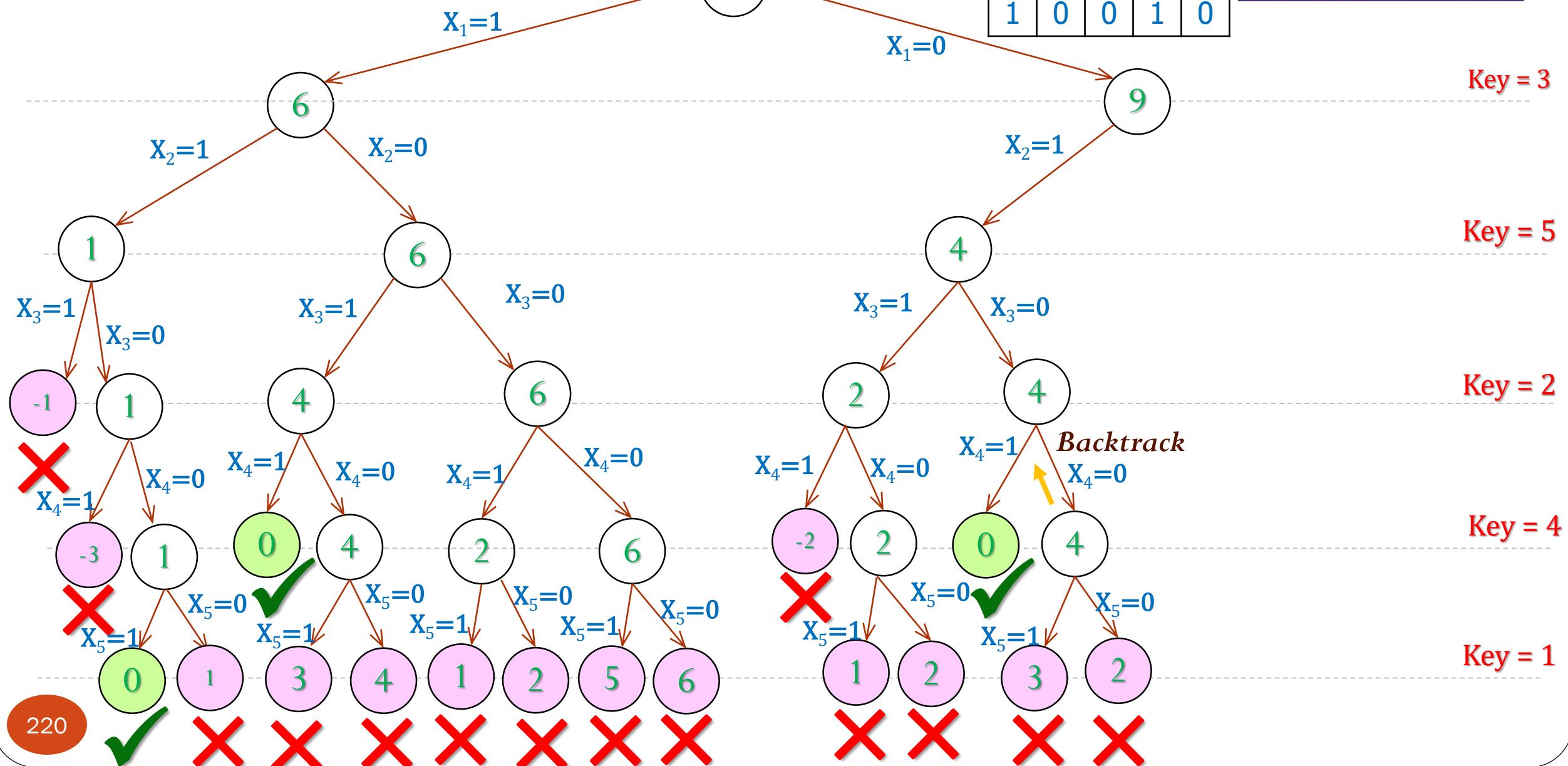
X	1	2	3	4	5
0	0	1	0	0	0

Key	1	2	3	4	5
3	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**

1	2	3	4	5
0	1	0	0	0

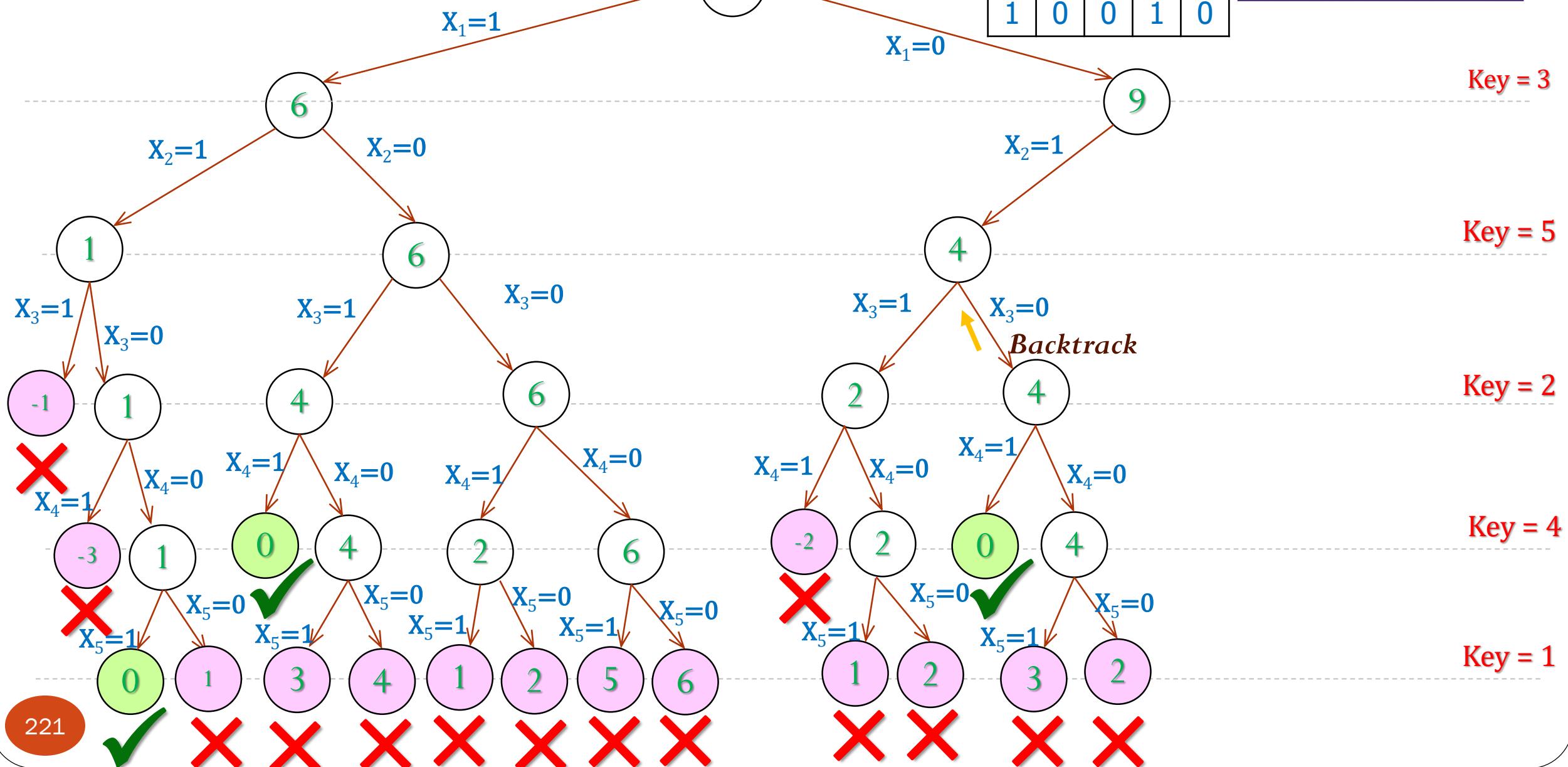
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



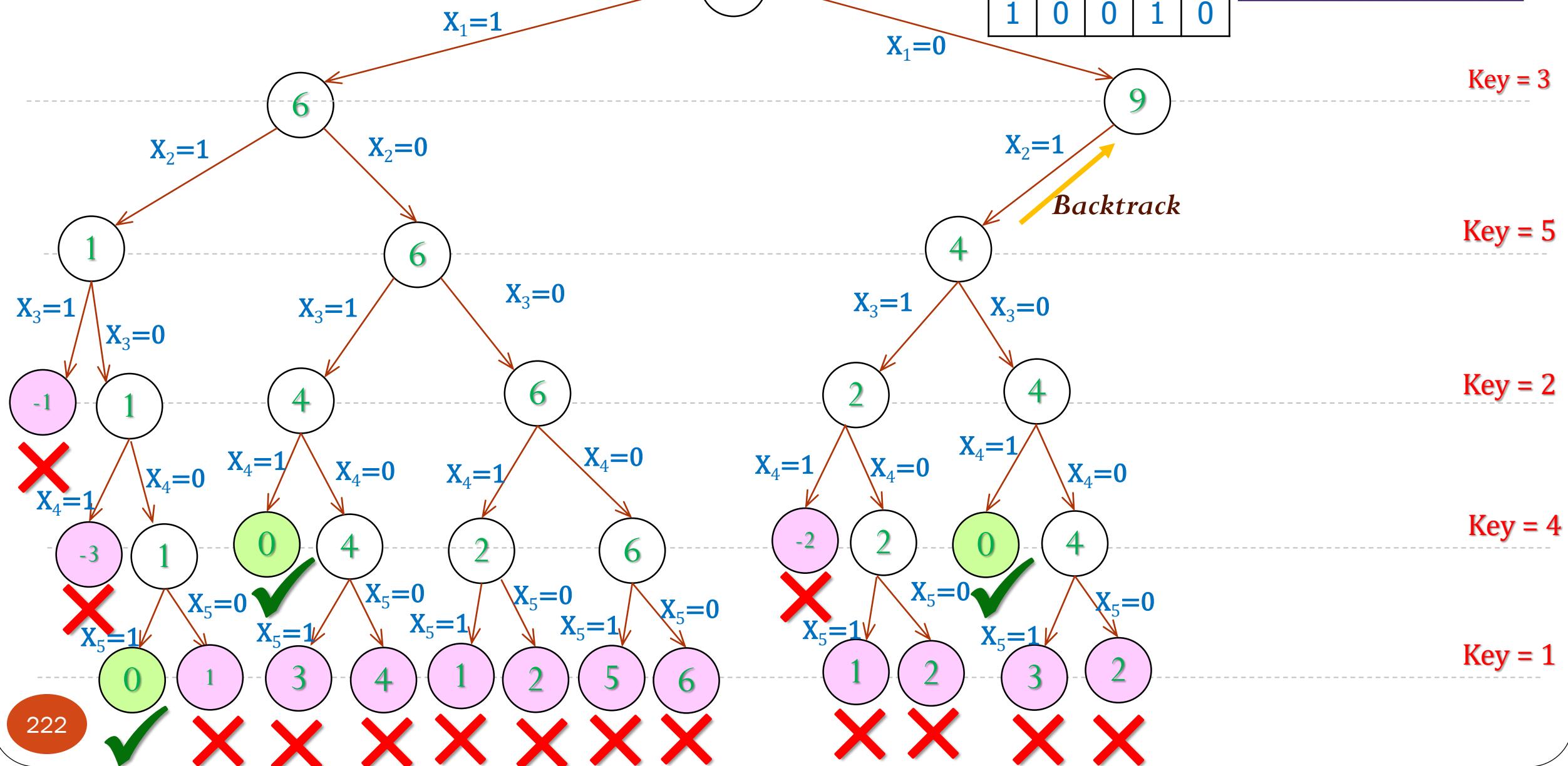
1	2	3	4	5
0	1	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

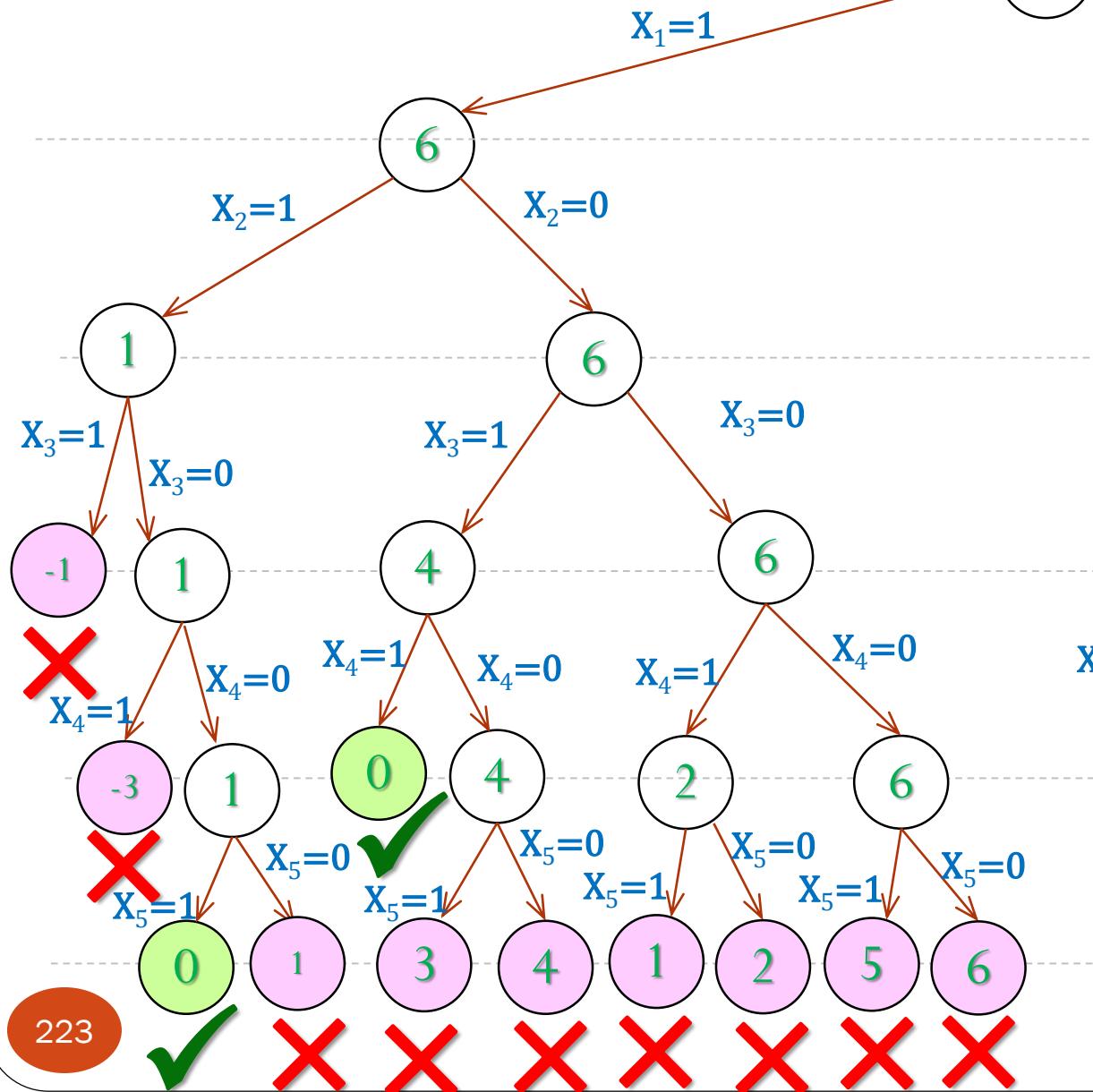
1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



	1	2	3	4	5
X	0	0	0	0	0

1	2	3	4	5
Key	3	5	2	4

## Start



# Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**Key = 3**

Key = 5

**Key = 2**

Key = 4

Key = 1

223

**X**

1	2	3	4	5
0	0	1	0	0

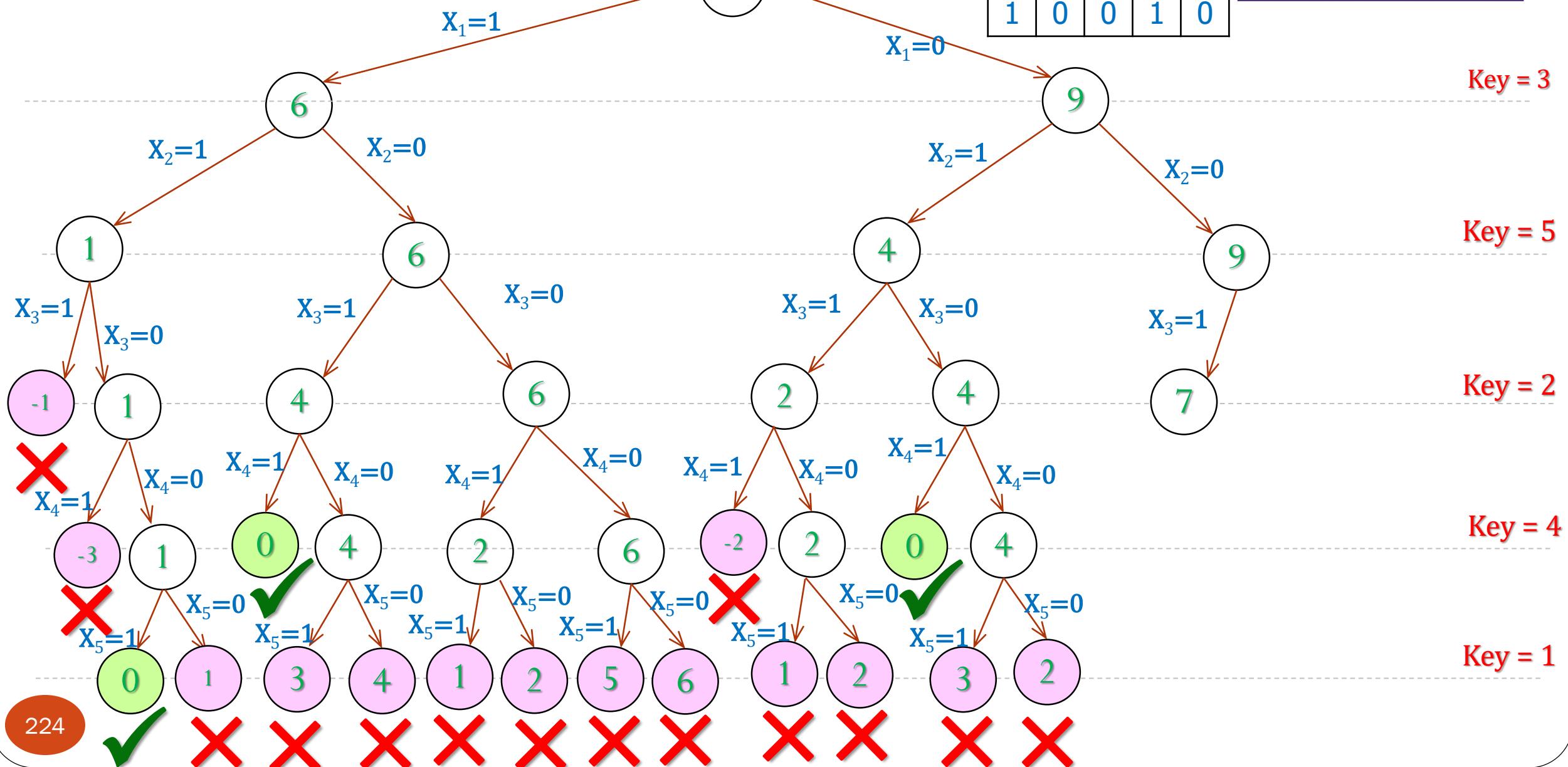
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**    1 2 3 4 5  

0	0	1	1	0
---	---	---	---	---

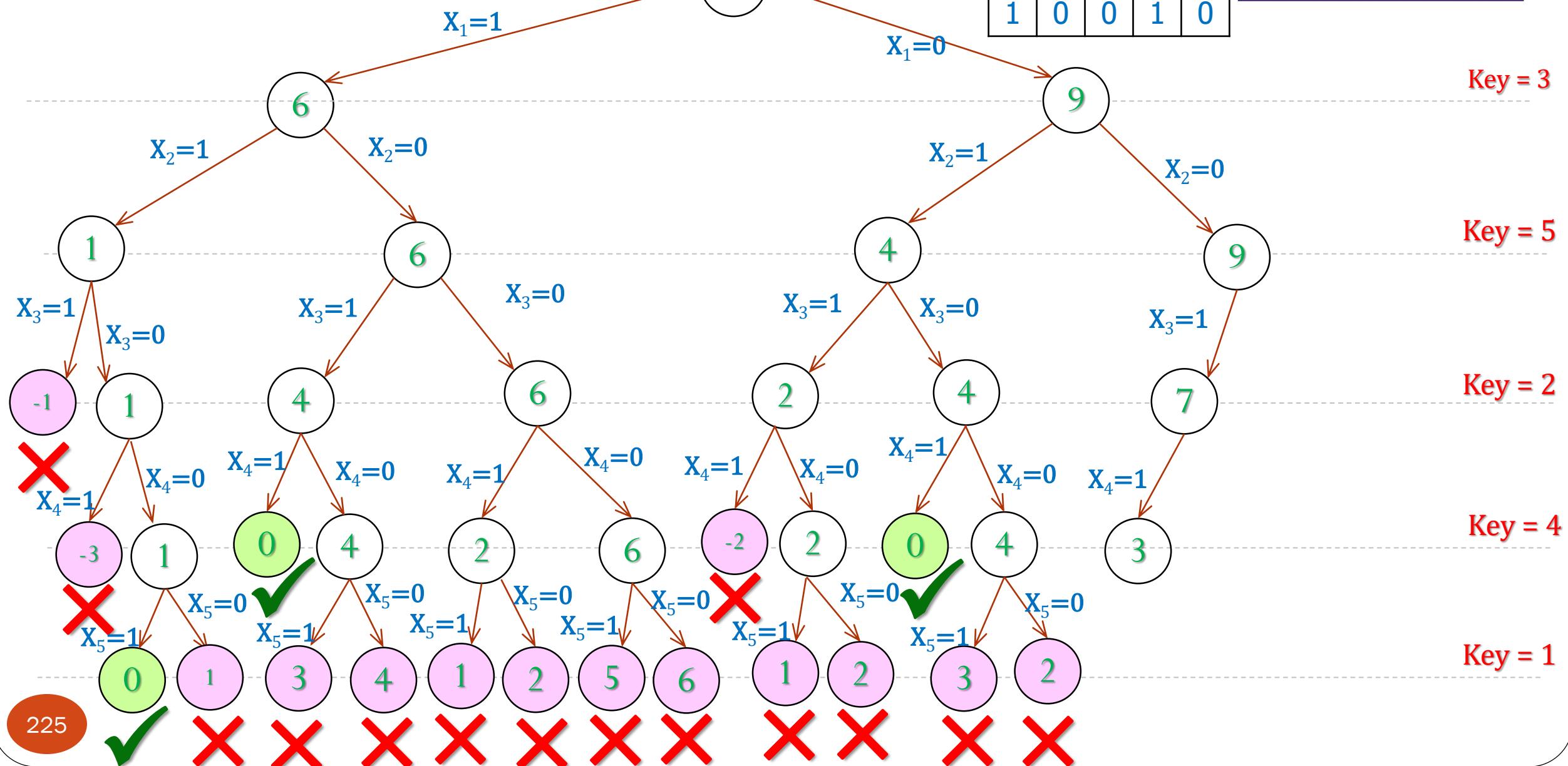
**Key**    1 2 3 4 5  

3	5	2	4	1
---	---	---	---	---

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**

1	2	3	4	5
0	0	1	1	1

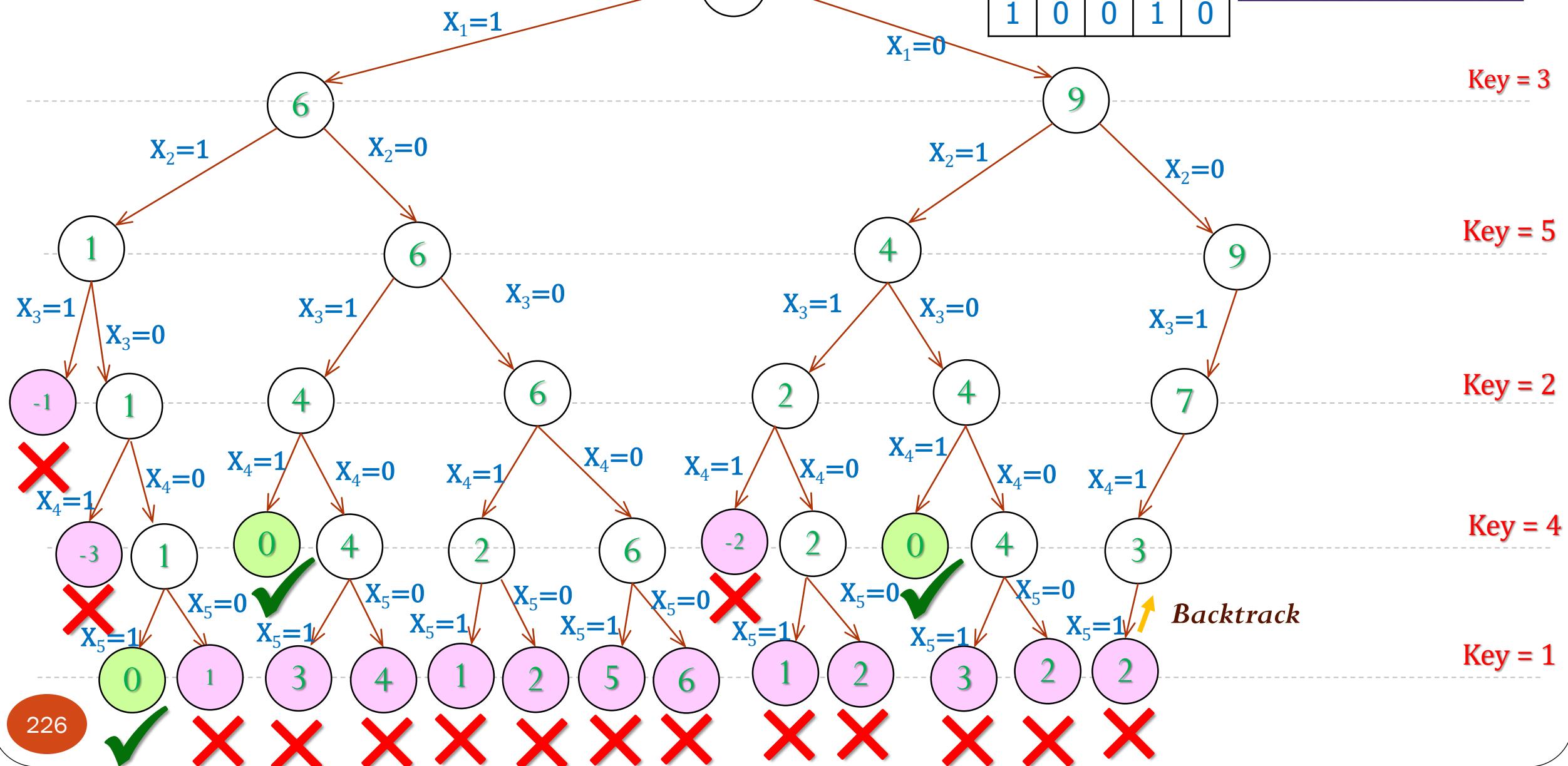
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**

1	2	3	4	5
0	0	1	1	0

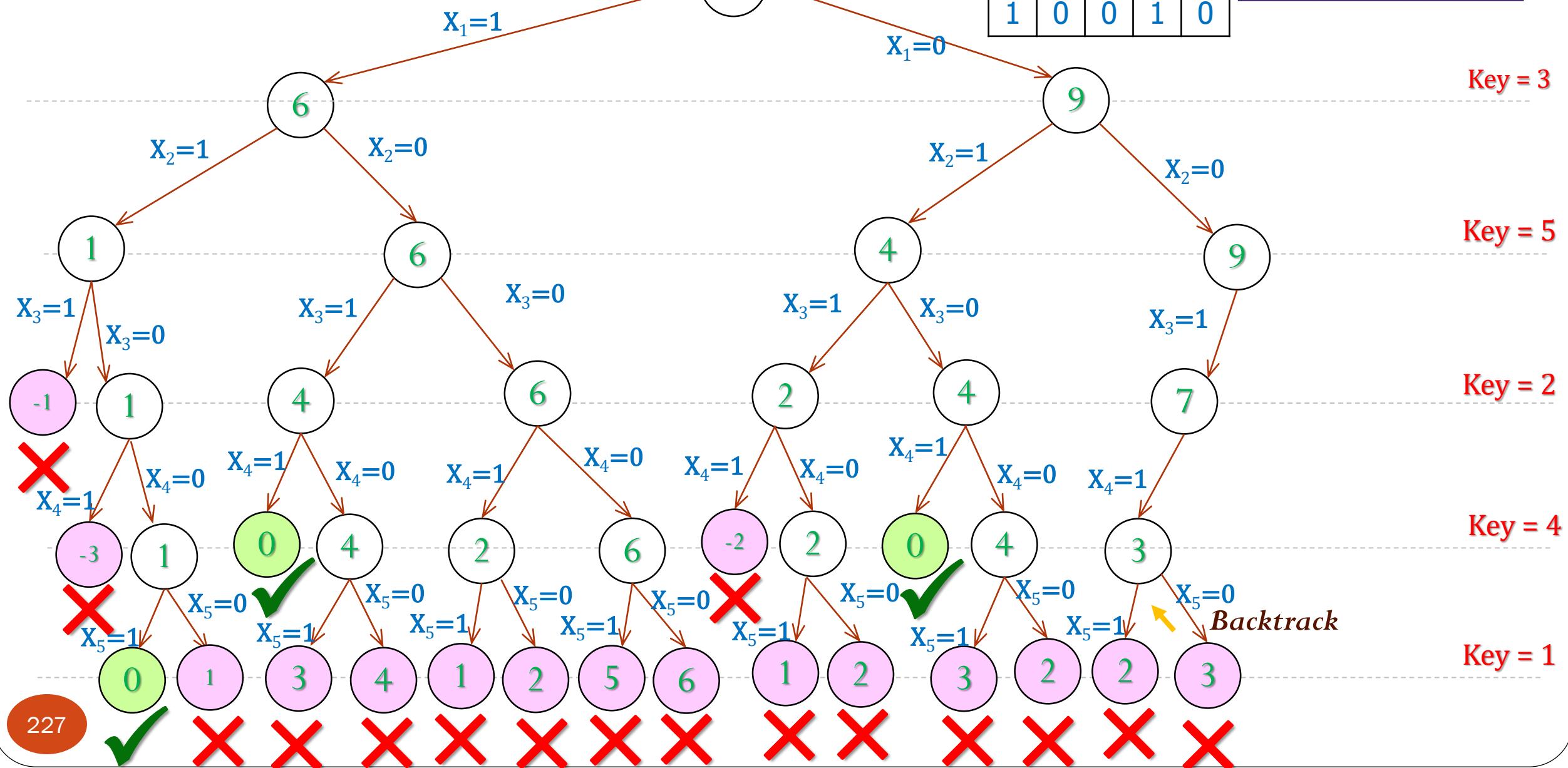
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



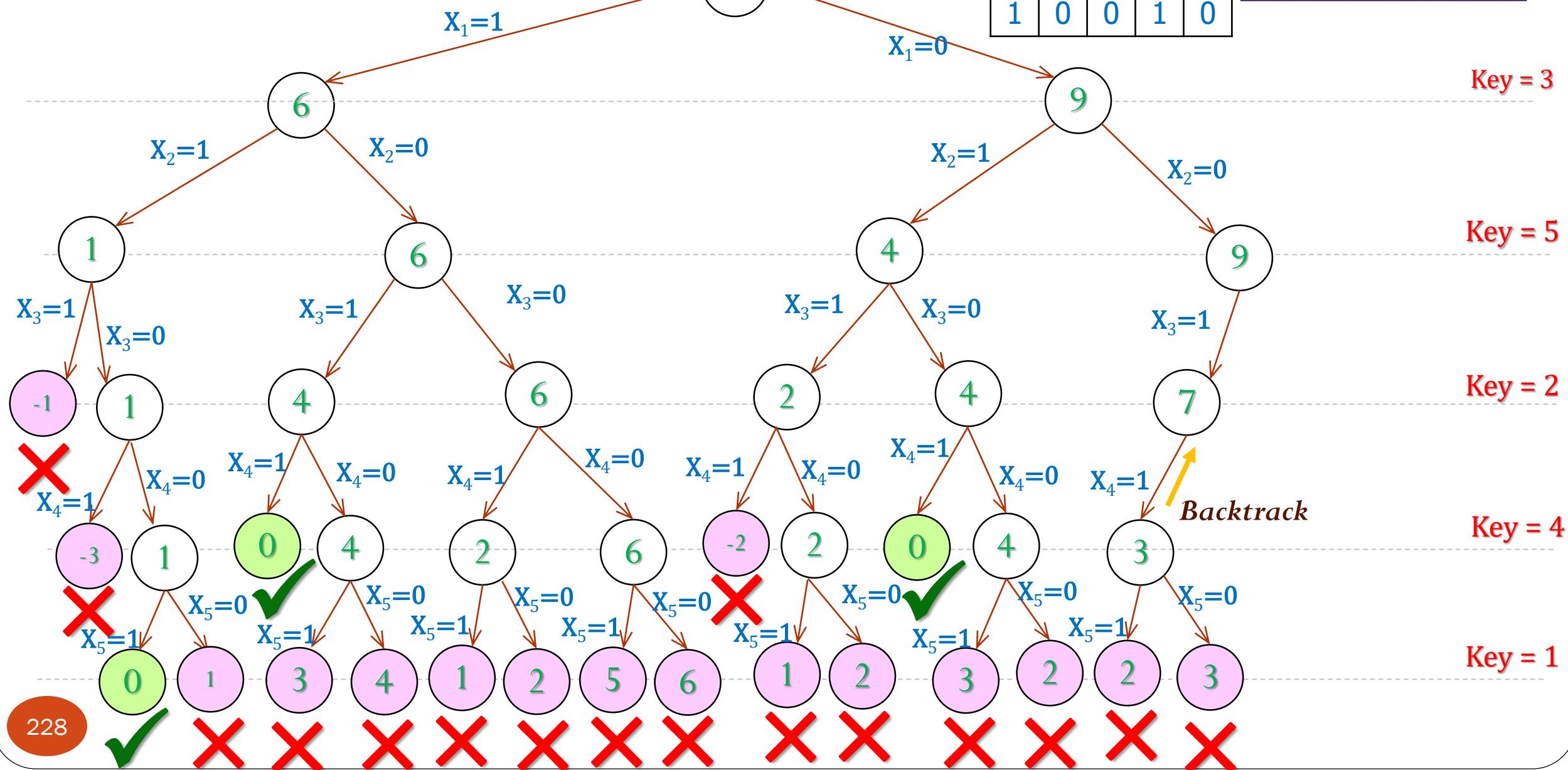
1	2	3	4	5
0	0	1	1	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



	1	2	3	4	5
X	0	0	1	0	0

1	2	3	4	5
Key	3	5	2	4

## Start

Solutions	1	1	0	0	1
	1	0	1	1	0
	1	0	0	1	0



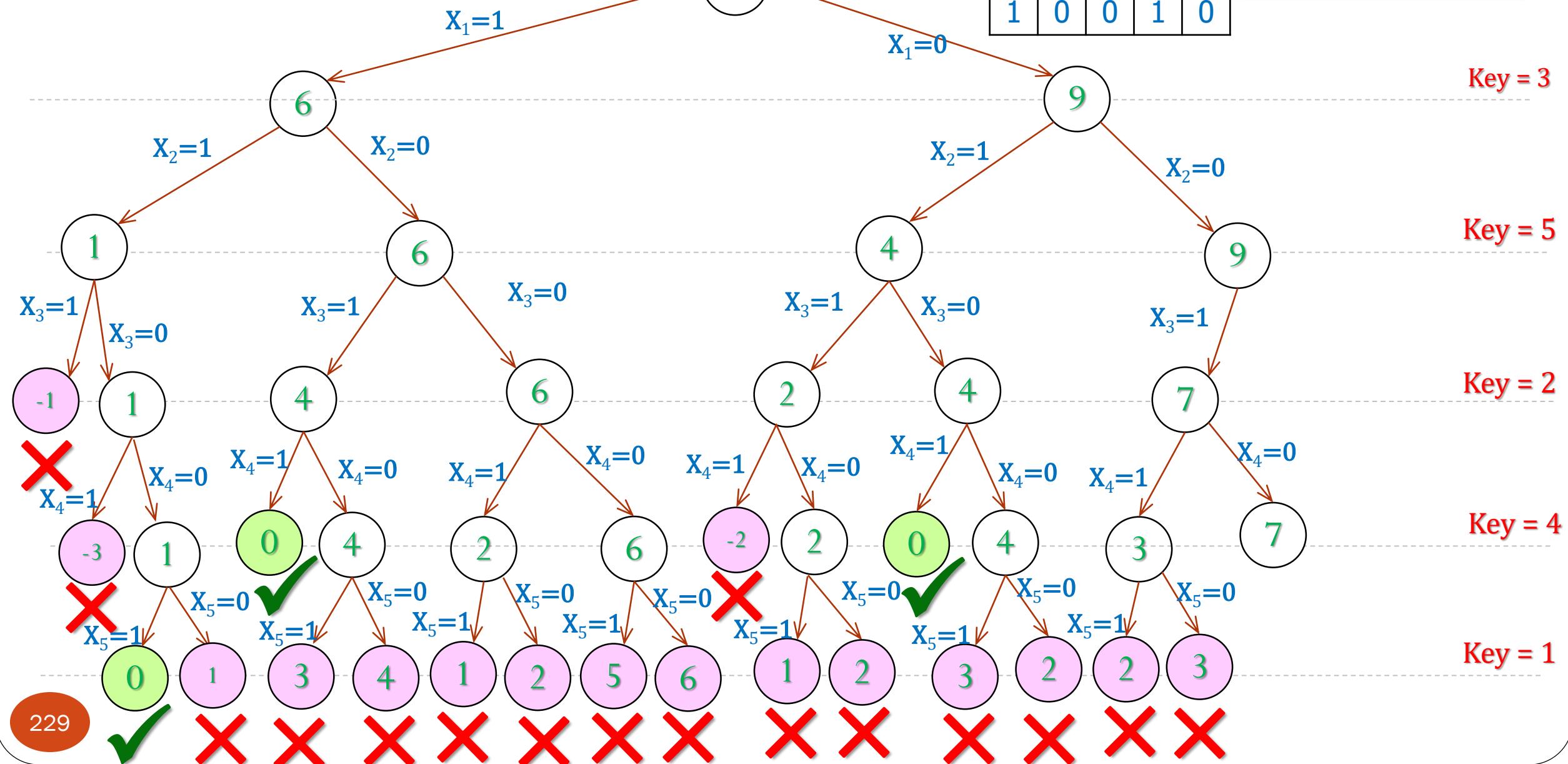
# **SASTRA**

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

## **DEEMED TO BE UNIVERSITY**

(U/S 3 OF THE UGC ACT, 1956)

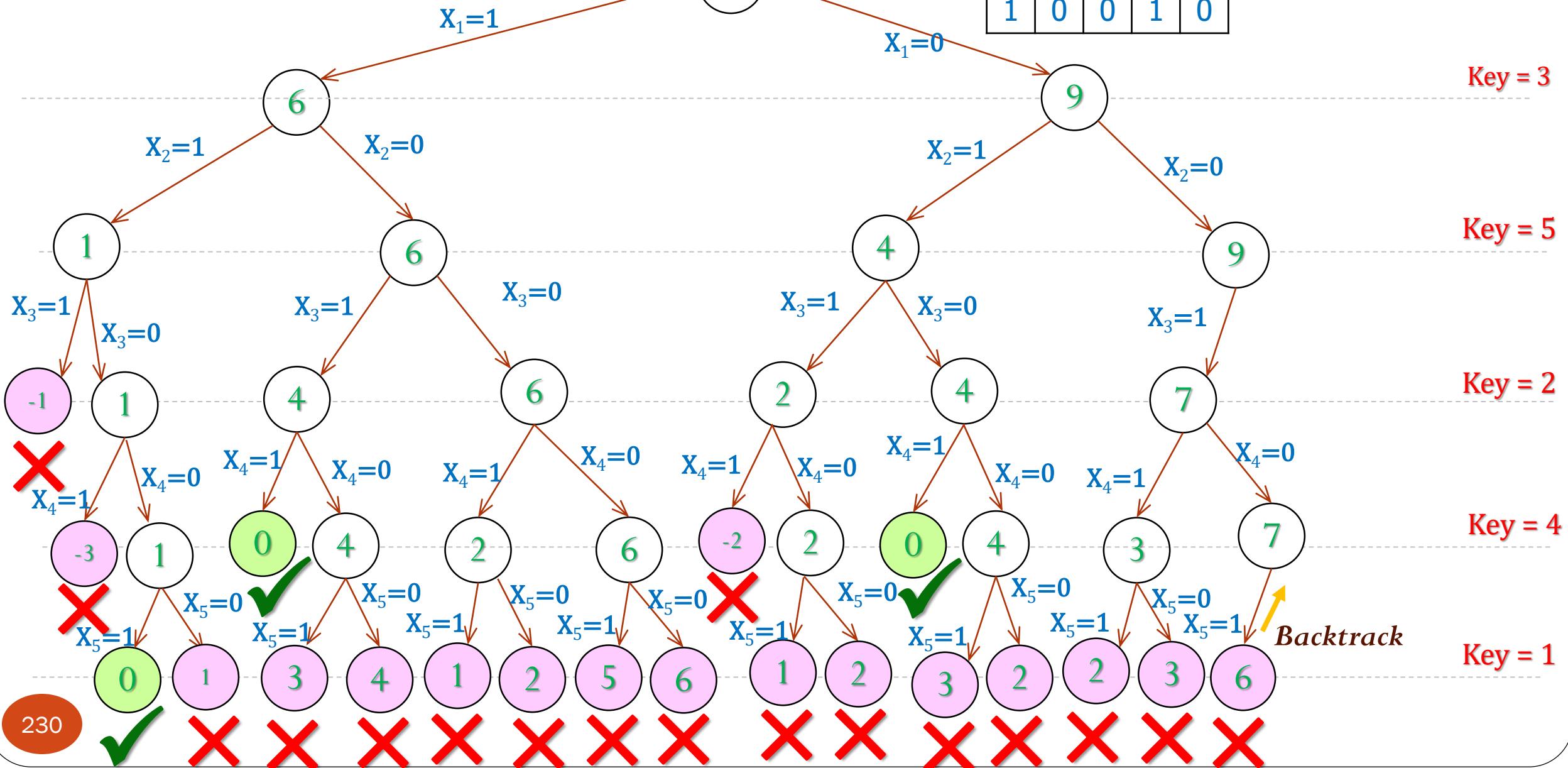
INK TRANSPARENCY · THINK SASTRA



	1	2	3	4	5
X	0	0	1	0	1

1	2	3	4	5
ey	3	5	2	4

# Start



	1	1	0	0	1
	1	0	1	1	0
	1	0	0	1	0



**X**

1	2	3	4	5
0	0	1	0	0

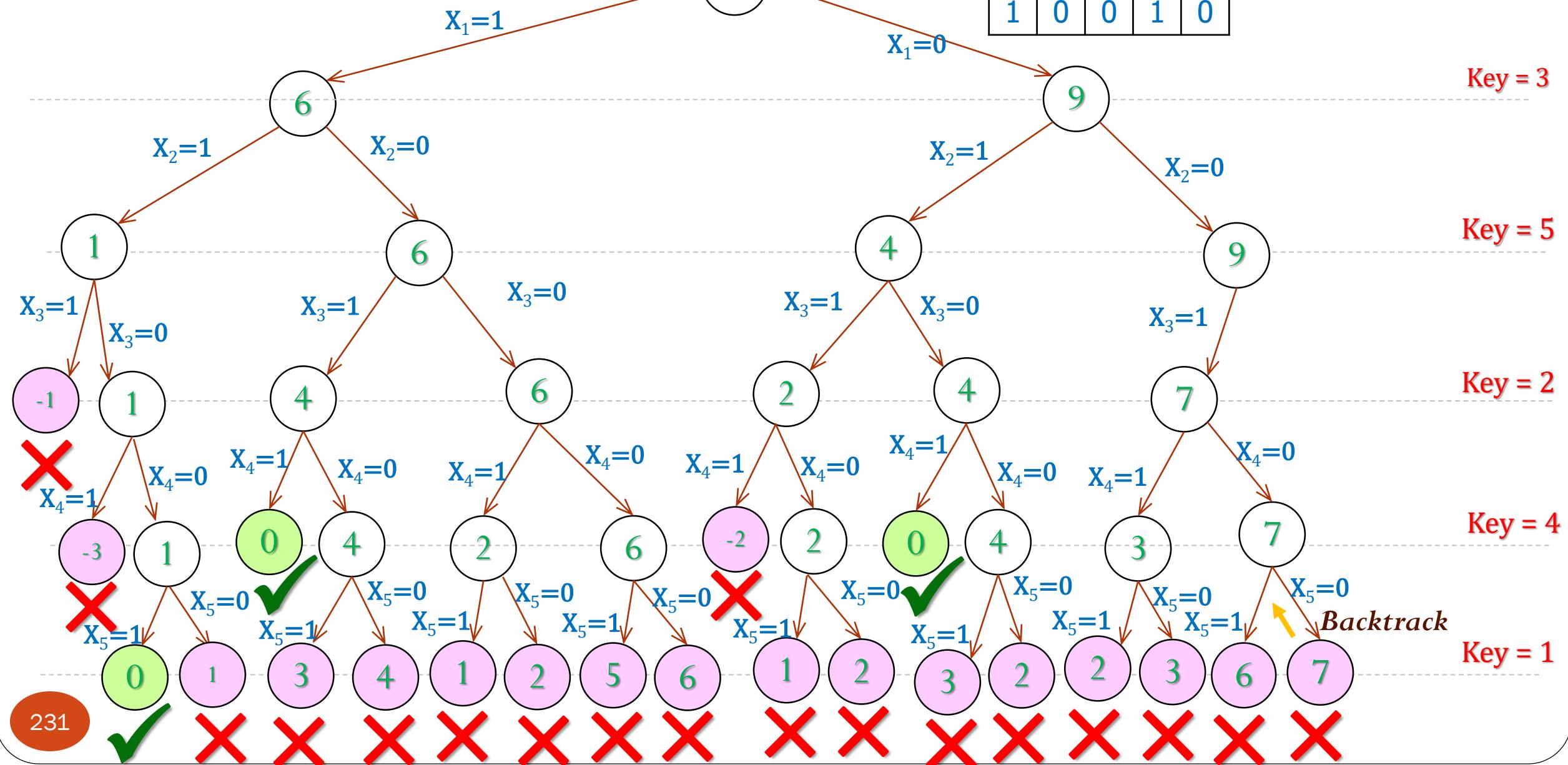
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



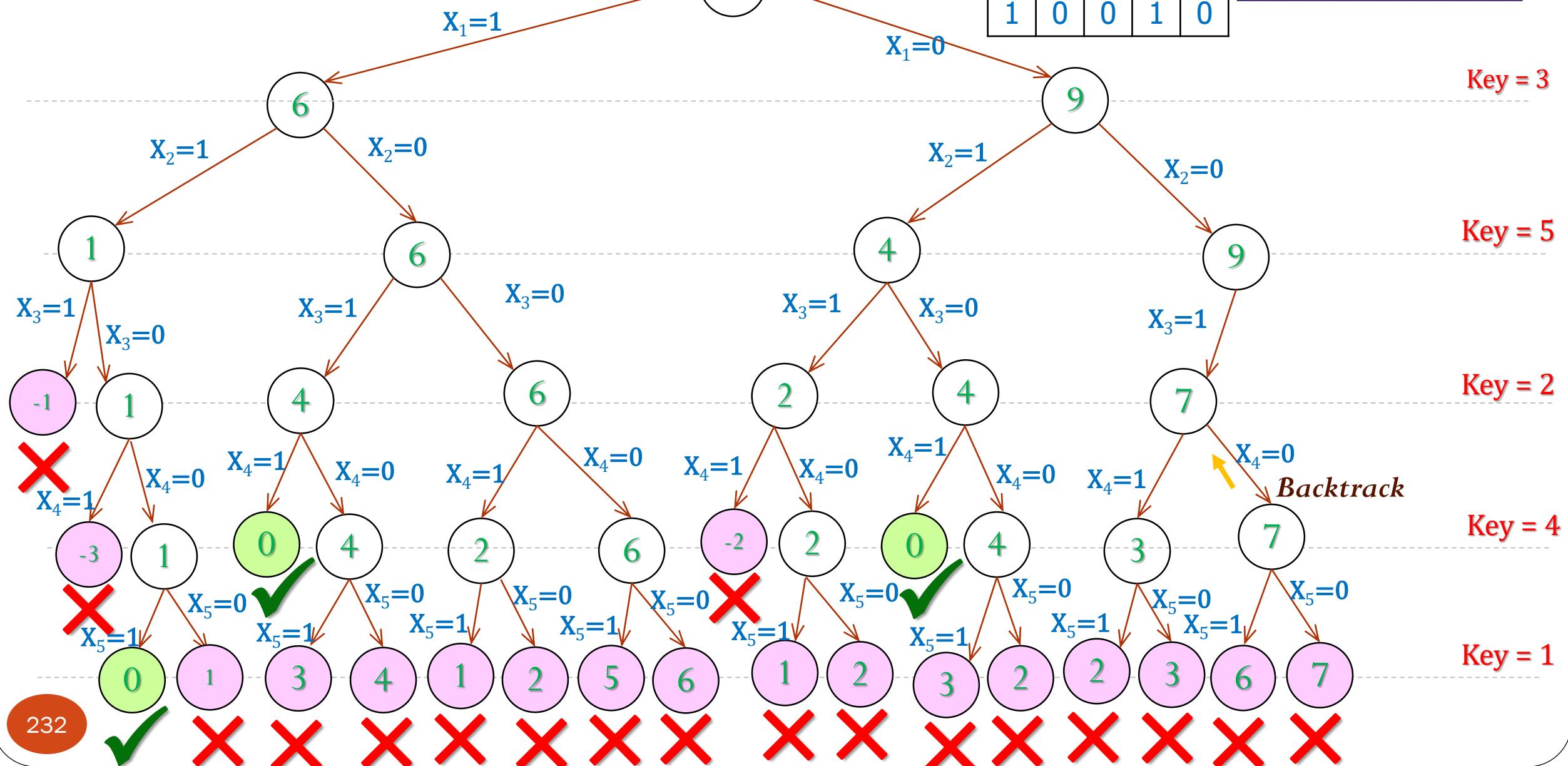
1	2	3	4	5
0	0	1	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



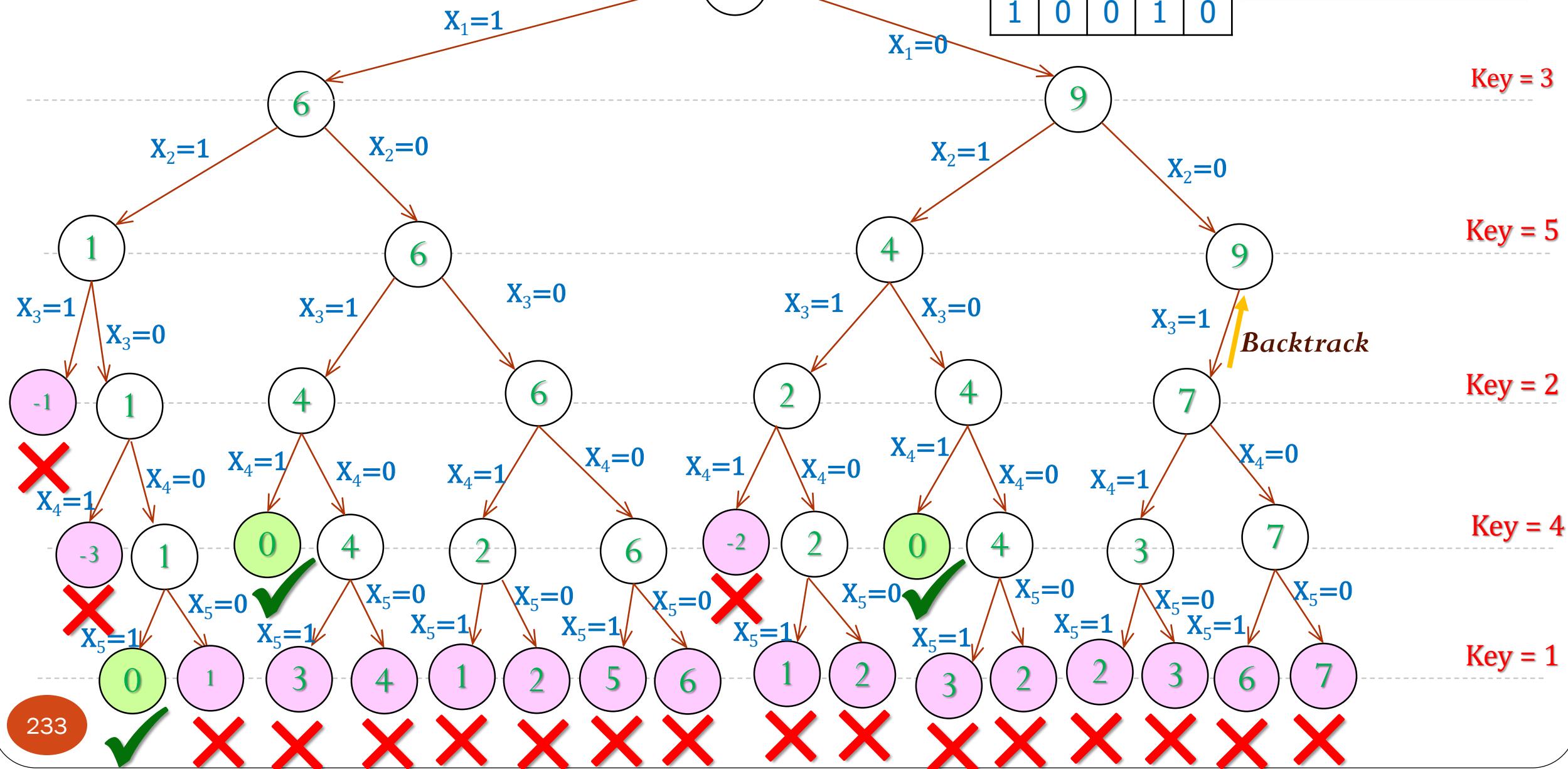
1	2	3	4	5	
X	0	0	1	0	0

1	2	3	4	5	
Key	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



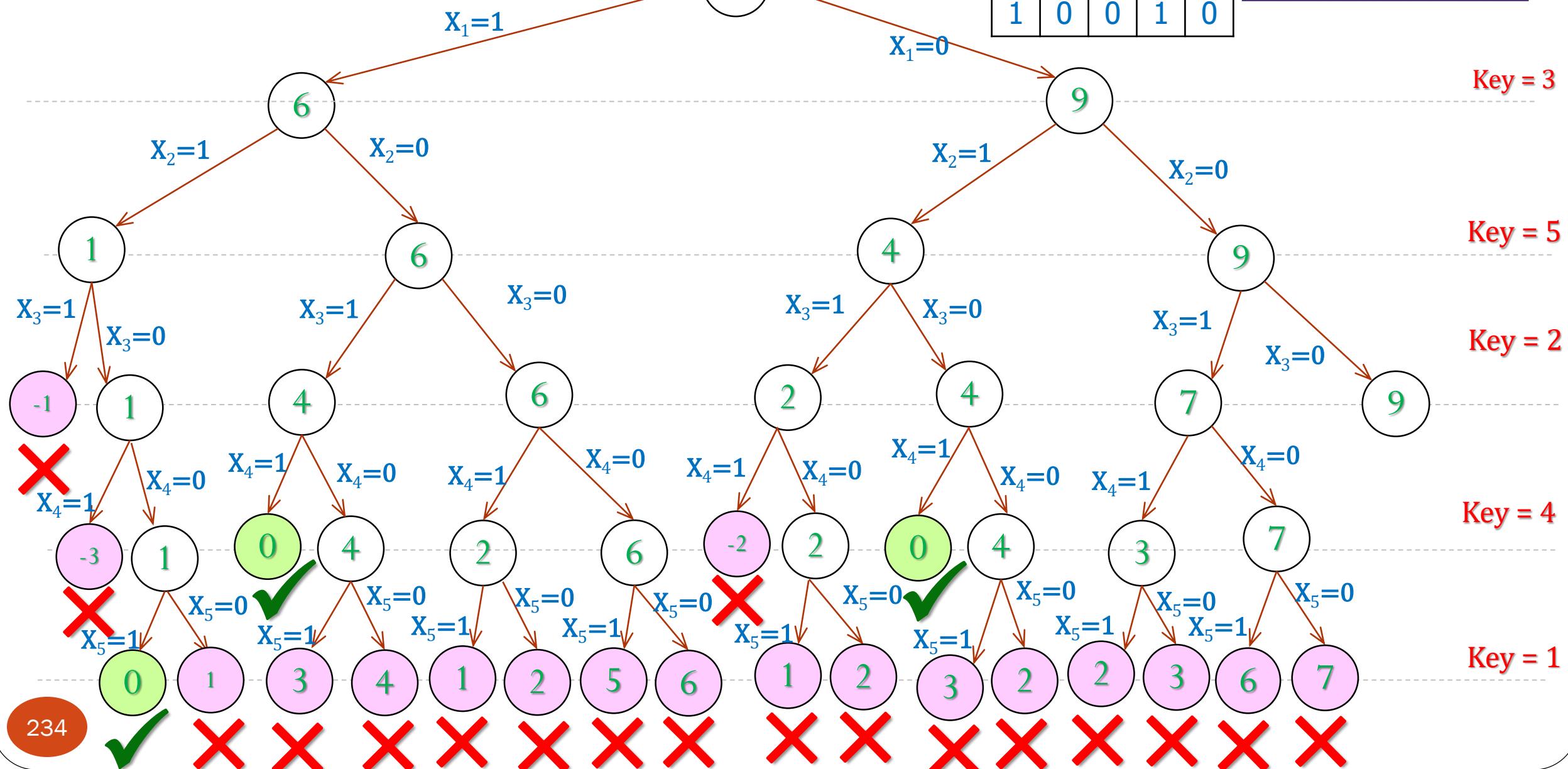
1	2	3	4	5
0	0	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

## Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



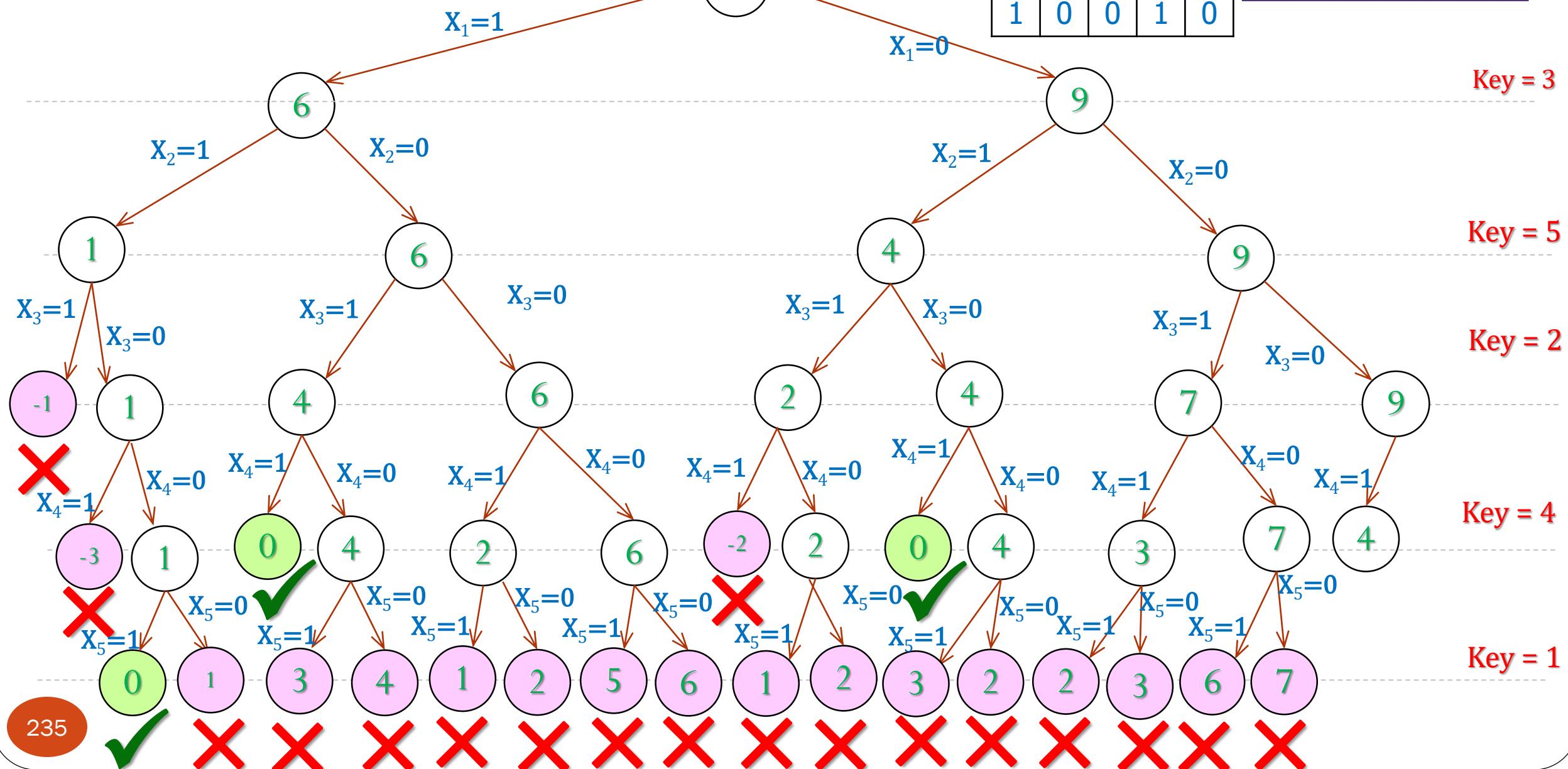
1	2	3	4	5
0	0	0	1	0

1	2	3	4	5
3	5	2	4	1

Start

## Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**

1	2	3	4	5
0	0	0	1	1

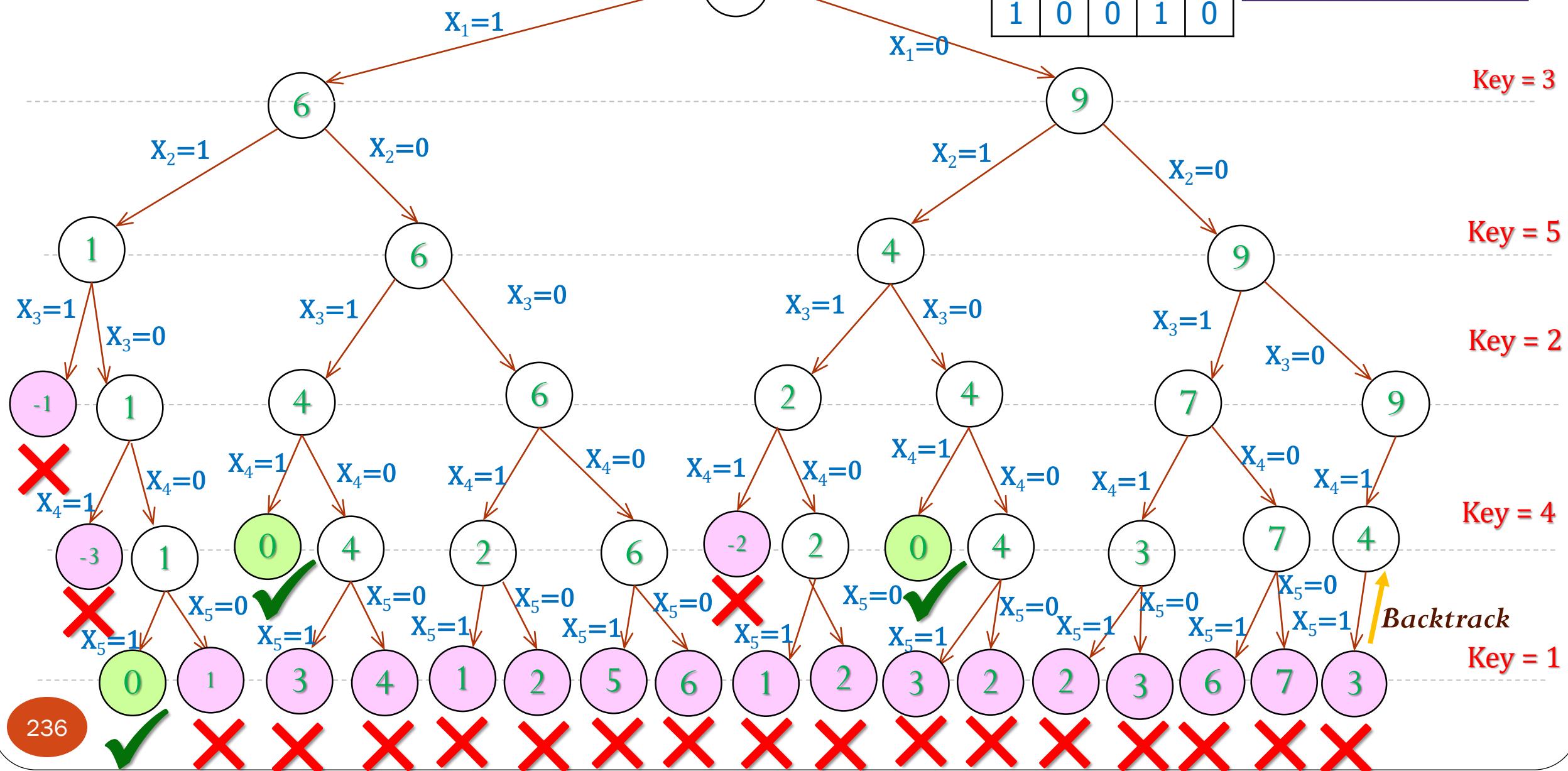
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**

1	2	3	4	5
0	0	0	1	0

**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



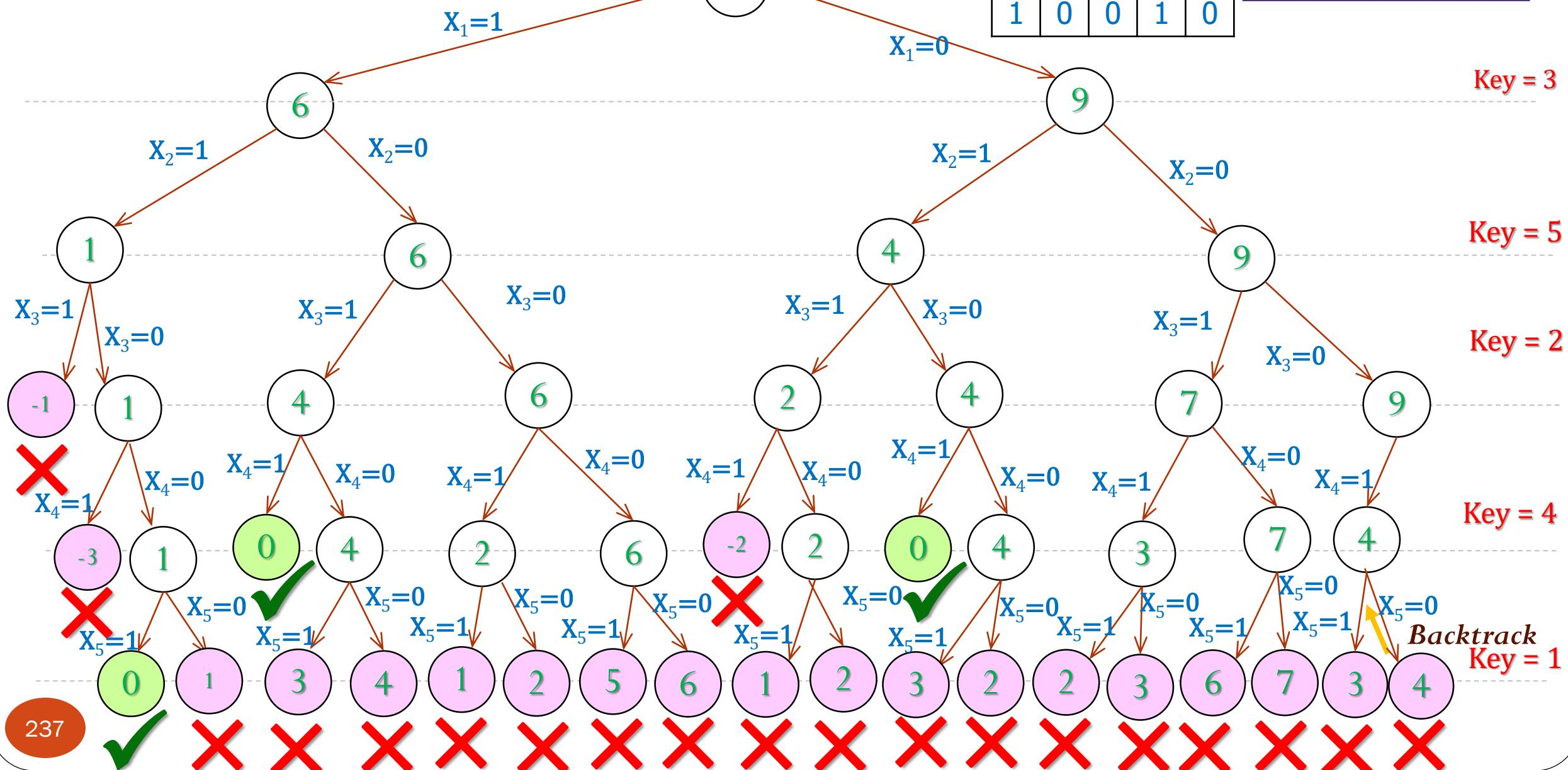
**Key = 3**

**Key = 5**

**Key = 2**

**Key = 4**

**Backtrack  
Key = 1**



**X** [ 1 2 3 4 5 ]  

0	0	0	1	0
---	---	---	---	---

**Key** [ 1 2 3 4 5 ]  

3	5	2	4	1
---	---	---	---	---

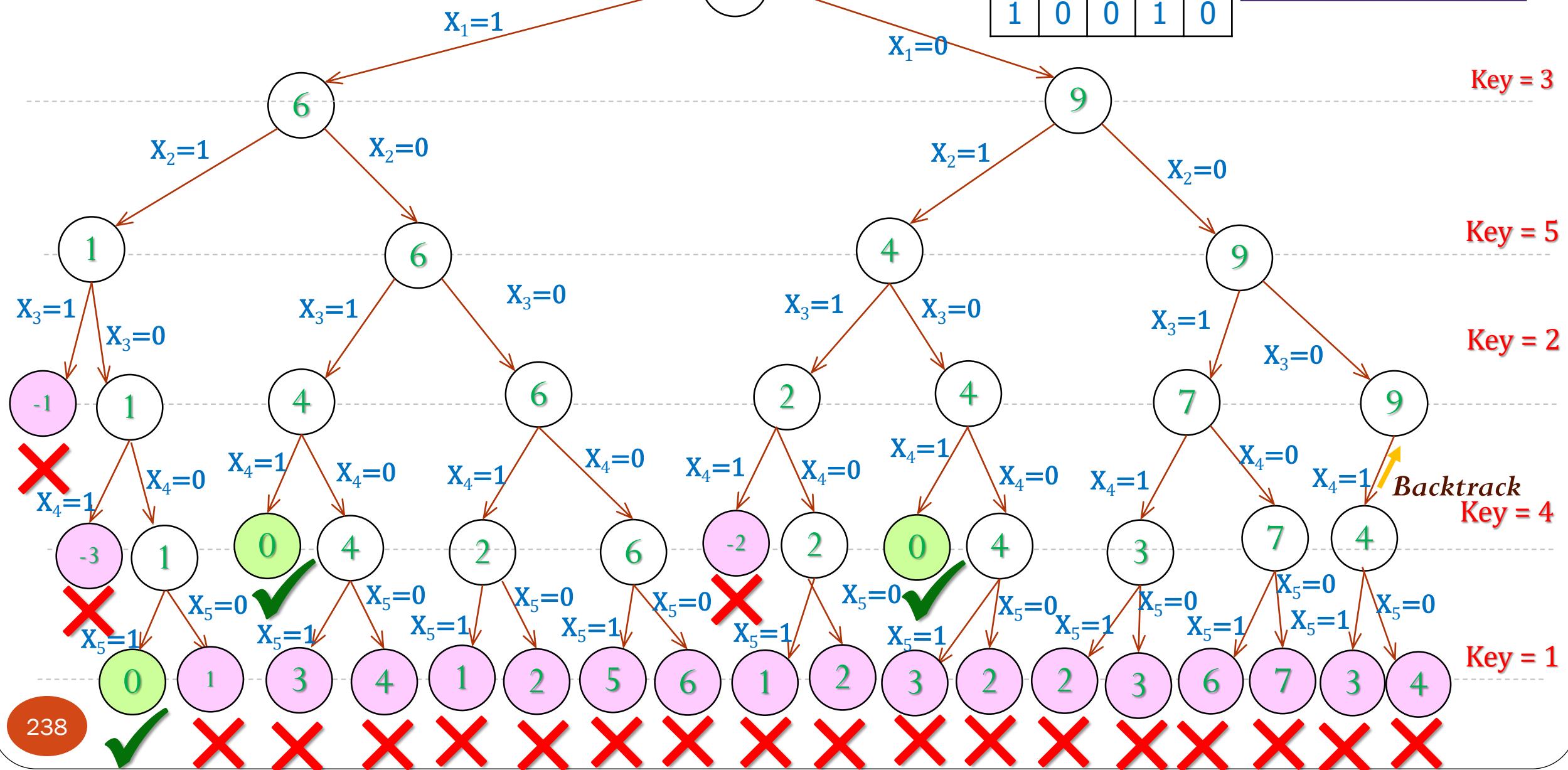
**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



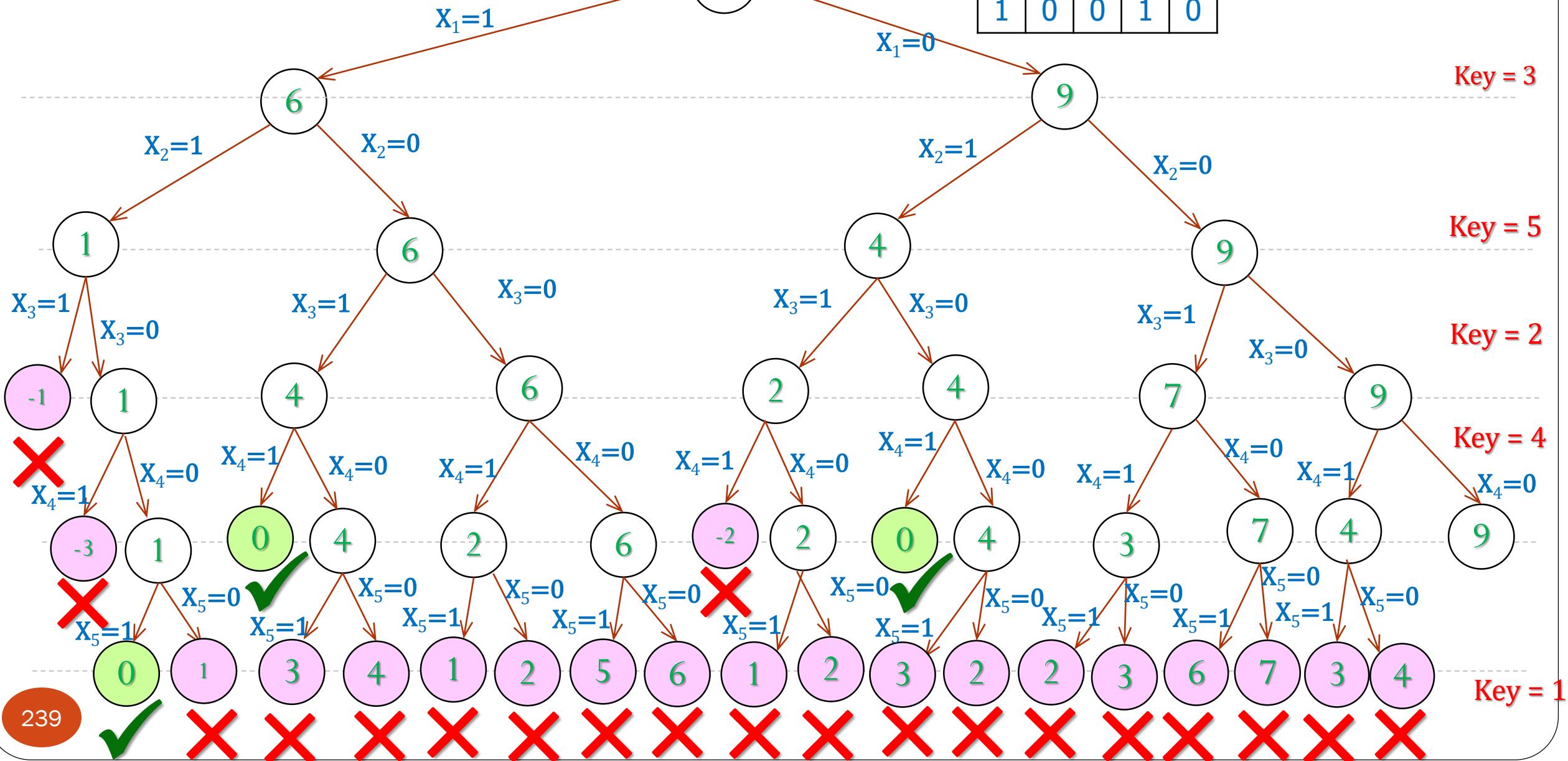
**SASTRA**  
 ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
 DEEMED TO BE UNIVERSITY  
 (U/A 3 OF THE UGC ACT, 1956)  
 THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



	1	2	3	4	5
X	0	0	0	0	0

1	2	3	4	5
Key	3	5	2	4

# Start



	1	1	0	0	1
	1	0	1	1	0
	1	0	0	1	0



**X**

1	2	3	4	5
0	0	0	0	1

**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0

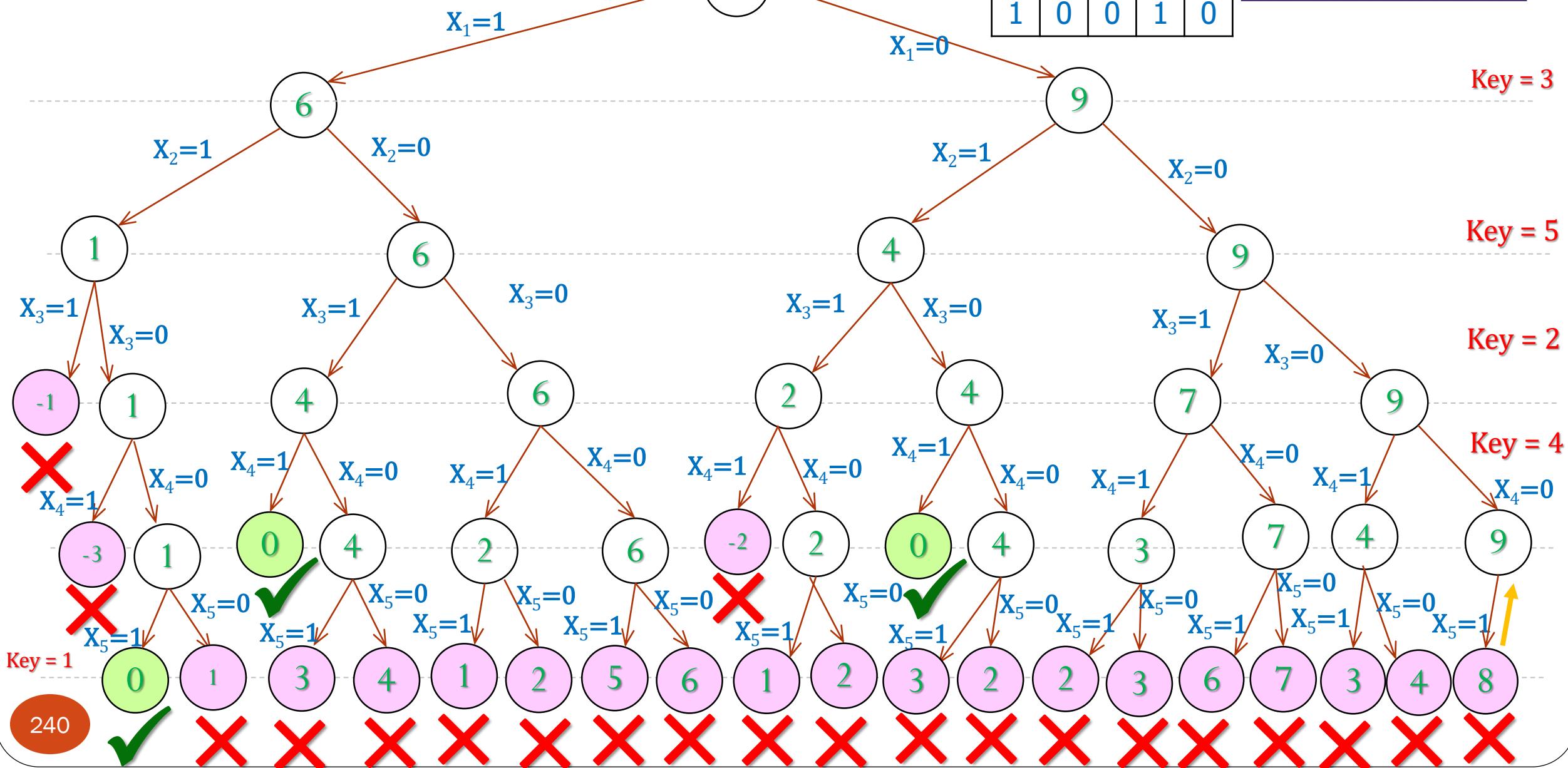


**Key = 3**

**Key = 5**

**Key = 2**

**Key = 4**



**X**

1	2	3	4	5
0	0	0	0	0

**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0

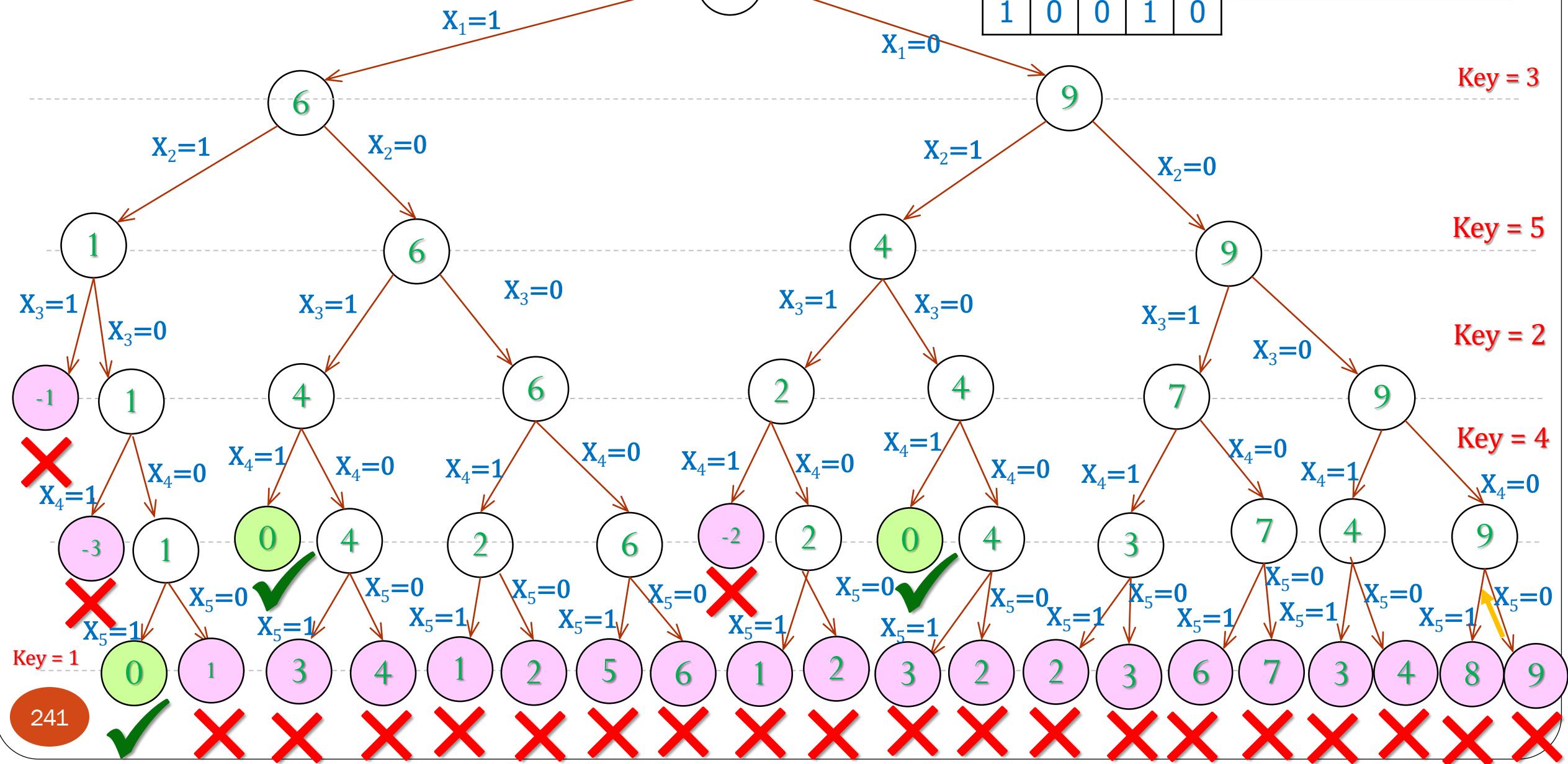


**Key = 3**

**Key = 5**

**Key = 2**

**Key = 4**



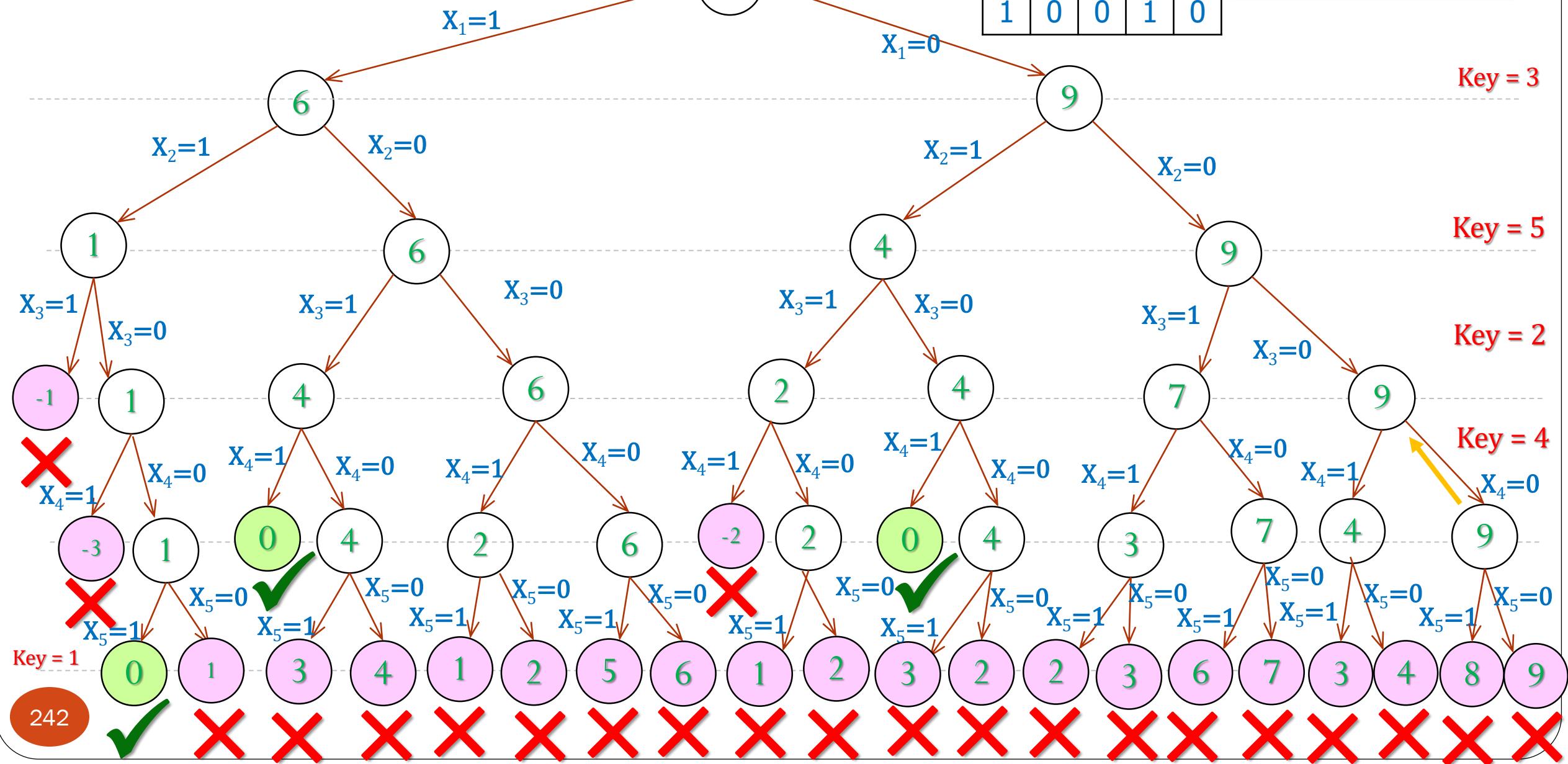
**X** [ 1 2 3 4 5 ]  
 0 0 0 0 0

**Key** [ 1 2 3 4 5 ]  
 3 5 2 4 1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X**

1	2	3	4	5
0	0	0	0	0

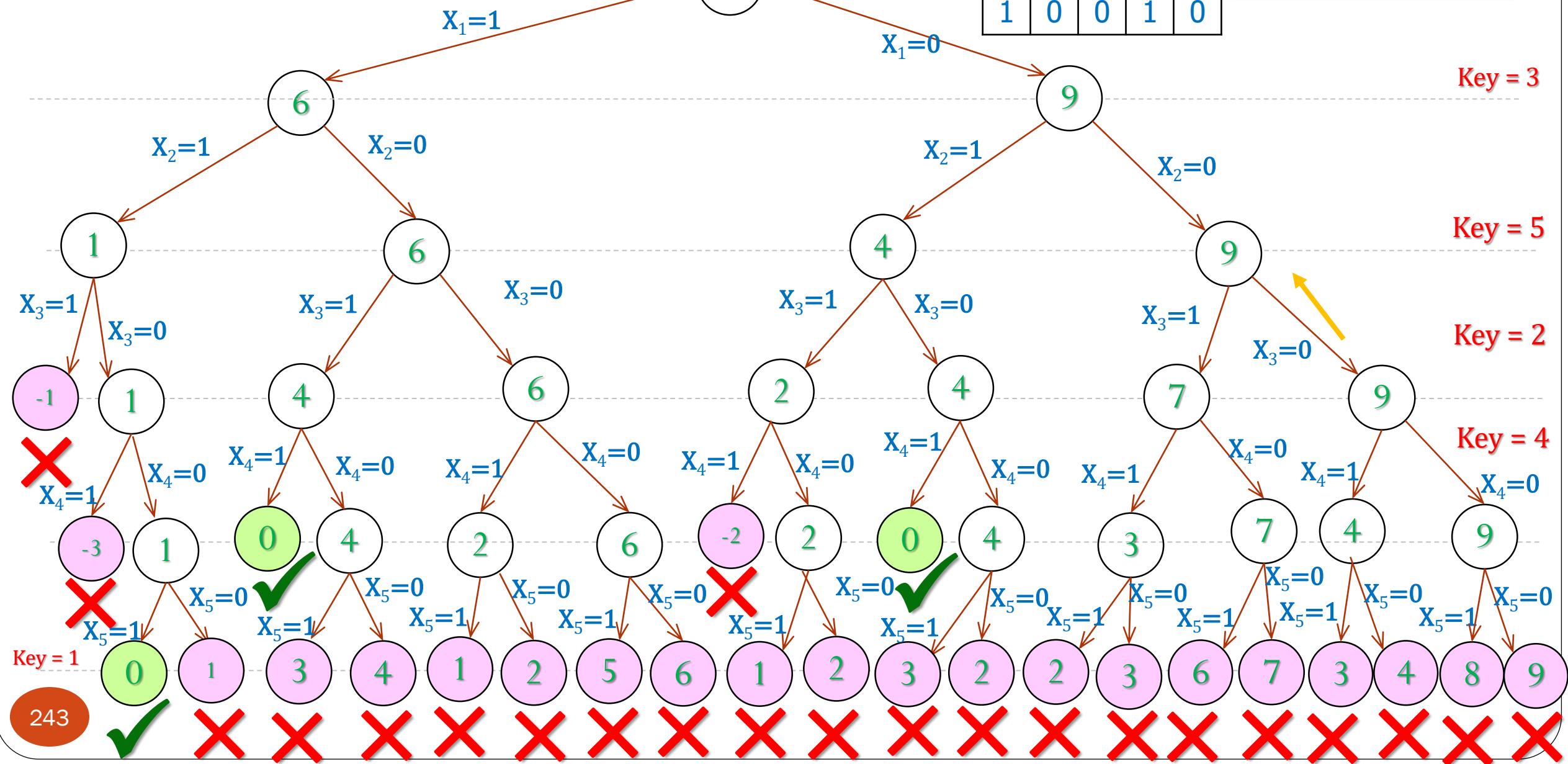
**Key**

1	2	3	4	5
3	5	2	4	1

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



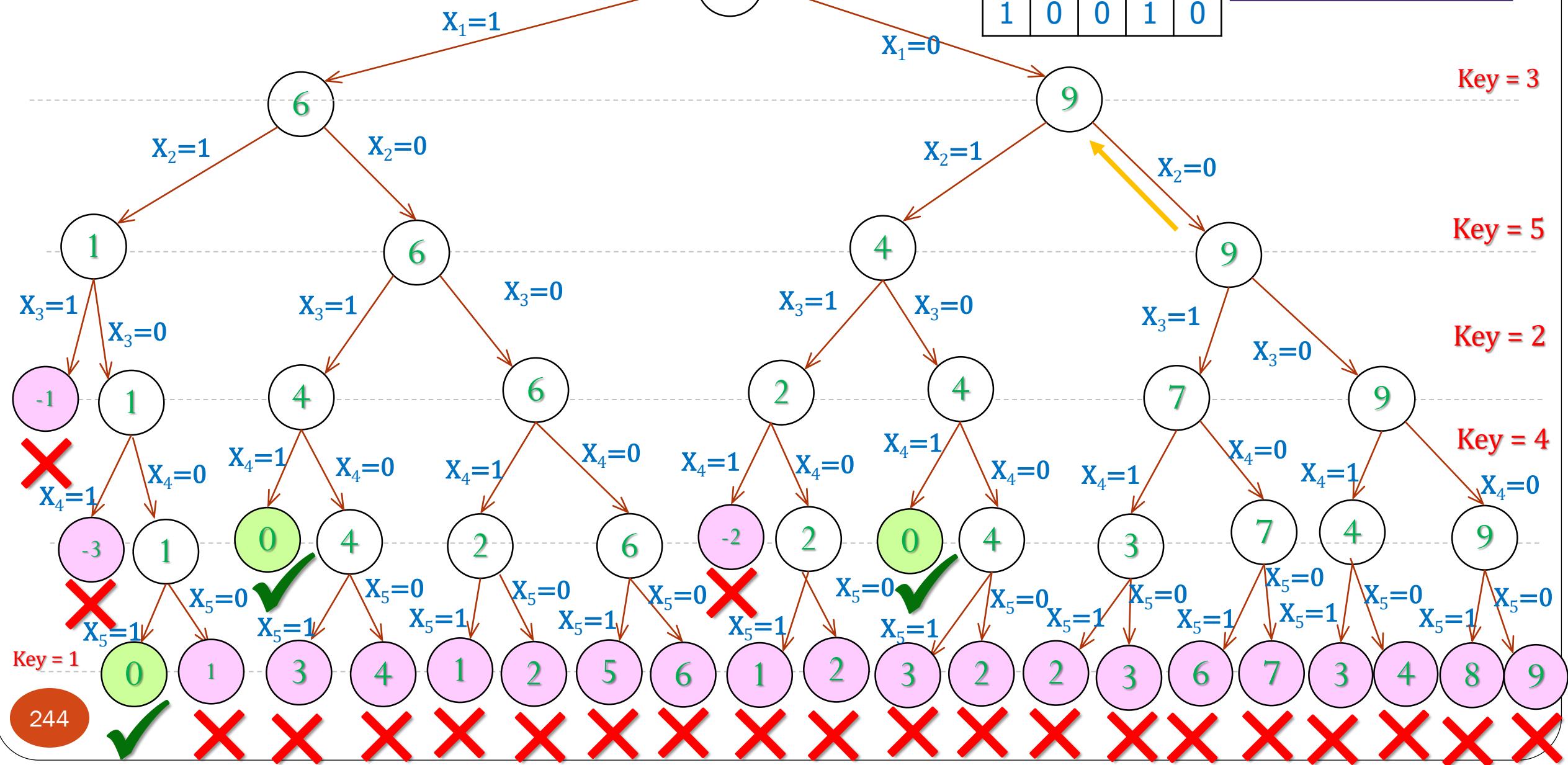
1	2	3	4	5
0	0	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



**X** [ 1 2 3 4 5 ]  

0	0	0	0	0
---	---	---	---	---

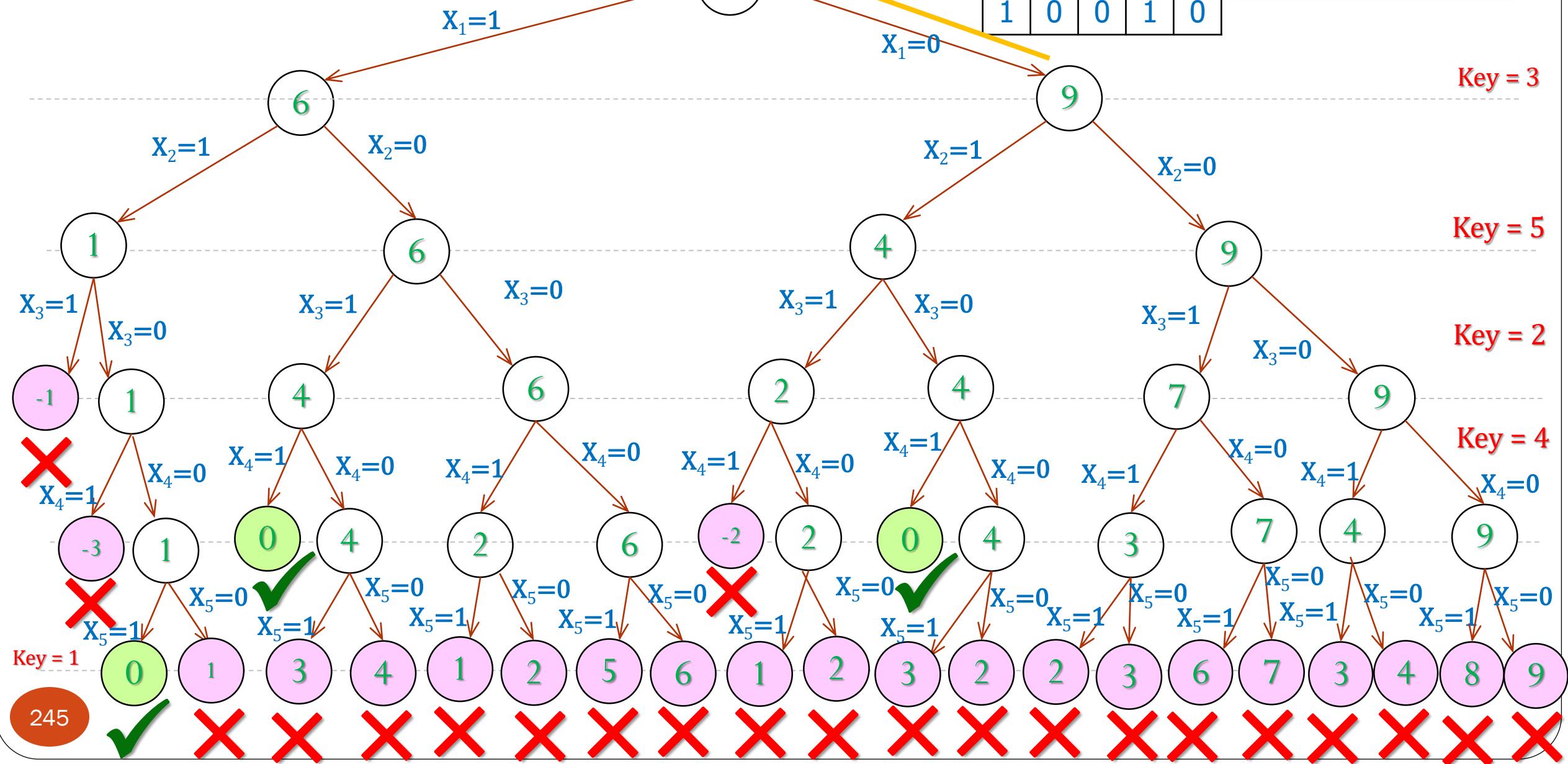
**Key** [ 1 2 3 4 5 ]  

3	5	2	4	1
---	---	---	---	---

**Start**

**Solutions**

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



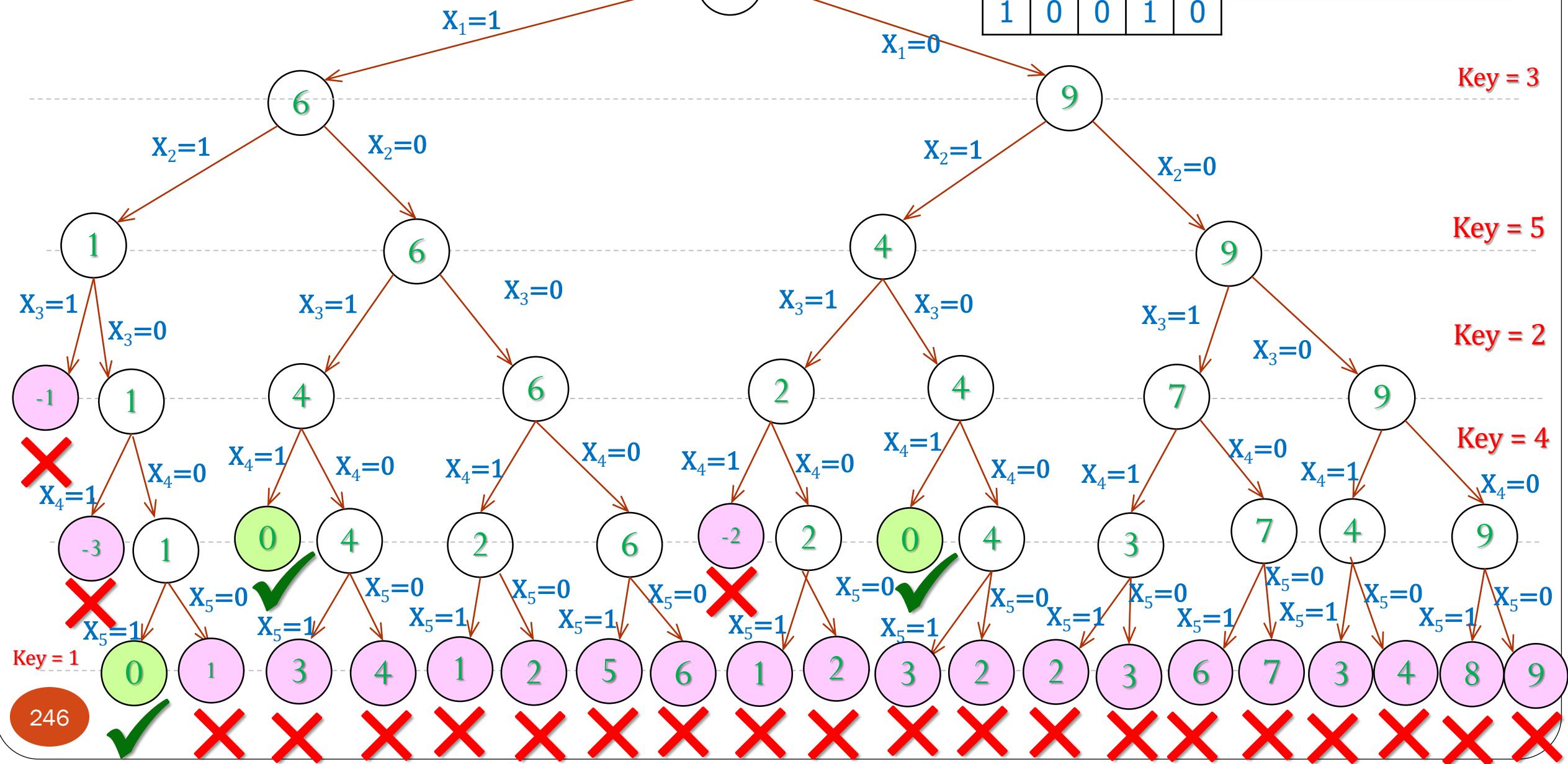
1	2	3	4	5
0	0	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

## Solutions

1	1	0	0	1
1	0	1	1	0
1	0	0	1	0



# Subset Sum – Algorithm - Backtracking



**Algorithm** SubSetSum(*next*, *n*, set[0..*n*-1], balanceSum, subset, solutionSet[], *soutionSize*)

//No Solution found... Backtrack..

**If** balanceSum<0 **Then**

**Return**

**End If**

//Solution Found... Record the Solution... Backtrack...

**If** balanceSum=0 **Then**

    solutionSet [*solutionSize*++] ← subset

**Return**

**End If**

//If No more choices... Backtrack...

**If** *next* = *n* **Then**

**Return**

**End If**

//If next is included

subset.X[*next*] ← 1

SubSetSum(*next* +1, *n*, set, balanceSum - set[i], subset, solutionSet, *solutionSize*)

//If next is not included

subset.X[*next*] ← 0

SubSetSum(*next* +1, *n*, set, balanceSum, subset, solutionSet, *solutionSize*)

**End** SubSetSum

# Problem

**Input:**  $w[1..6] = \{ 5, 10, 12, 13, 15, 18 \}$

$n = 6$

$m = 30$

**Output:**  $x1[1..6] = \{1, 1, 0, 0, 1, 0\}$  ( $w1+w2+w5=5+10+15=30$ )

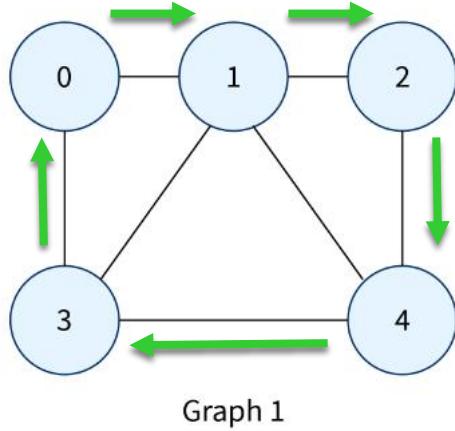
$x2[1..6] = \{0, 0, 1, 0, 0, 1\}$  ( $w2+w6=12+18=30$ )

# **Backtracking Approach**

# **Hamiltonian Cycles Problem**

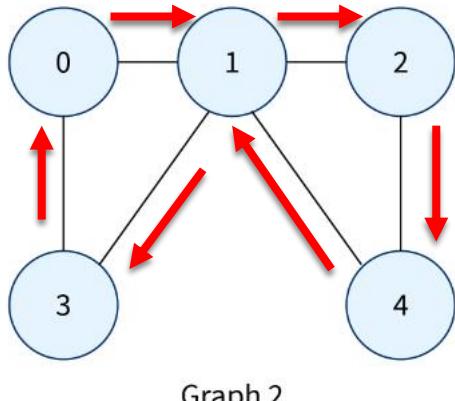
# Hamiltonian Cycles

- ✓ It is a cycle in a graph, which visit each vertex exactly once.



In Graph 1, Hamiltonian Cycle is present: 0-1-2-4-3-0

The cycles 0-1-2-4-3-0 and 1-2-4-3-0-1 both refers to same cycles.

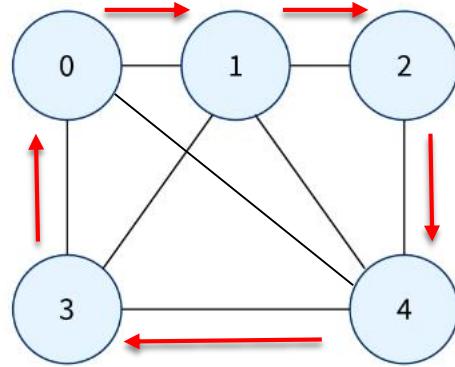


In Graph 2, No Hamiltonian Cycle Present

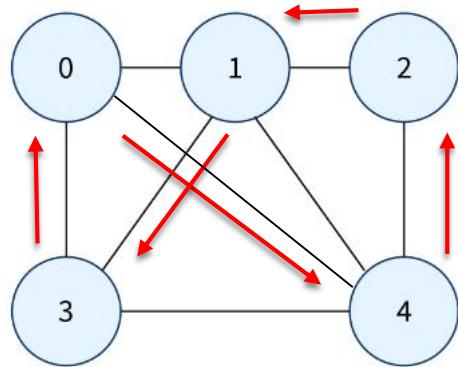
The cycle 0-1-2-4-1-3-0 is not a Hamiltonian cycle. The vertex 1 is visited twice.

# Hamiltonian Cycles Problem

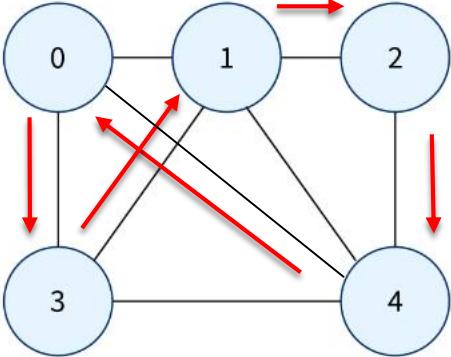
- ✓ Problem of finding all the Hamiltonian cycles present in the graph.



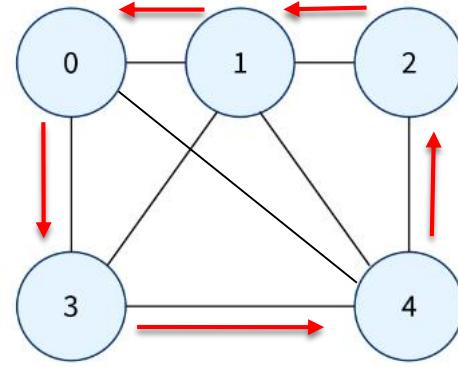
0-1-2-4-3-0



0-4-2-1-3-0



0-3-1-2-4-0



0-3-4-2-1-0

## Simple Representation of Solutions – Sequence of Vertices

0-1-2-4-3

0-4-2-1-3

0-3-1-2-4

0-3-4-2-1

# Hamiltonian Cycles – Brute-force

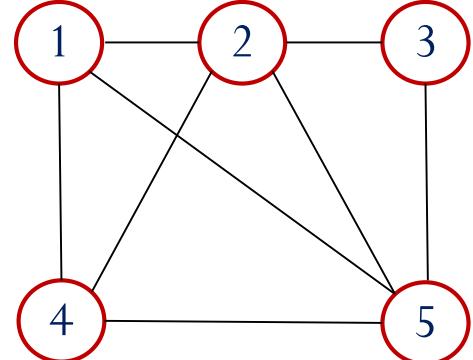


- ✓ Leave the starting vertex
- ✓ Generate the permutations for the remaining vertex
- ✓ Check the generated paths are Hamiltonian cycles or not.
- ✓ Save it, if it is Hamiltonian cycles.

Solution Space: $(n-1)! = 4! = 24$								
1	0	1	2	3	4	0	No	
2	0	1	2	4	3	0	Yes	
3	0	1	3	2	4	0	No	
4	0	1	3	4	2	0	No	
5	0	1	4	2	3	0	No	
6	0	1	4	3	2	0	No	
7	0	2	1	3	4	0	No	
8	0	2	1	4	3	0	No	
9	0	2	3	1	4	0	No	
10	0	2	3	4	1	0	No	
11	0	2	4	1	3	0	No	
12	0	2	4	3	1	0	No	
13	0	3	2	1	4	0	No	
14	0	3	2	4	1	0	No	
15	0	3	1	2	4	0	Yes	
16	0	3	1	4	2	0	No	
17	0	3	4	2	1	0	Yes	
18	0	3	4	1	2	0	No	
19	0	4	2	3	1	0	No	
20	0	4	2	1	3	0	Yes	
21	0	4	3	2	1	0	No	
22	0	4	3	1	2	0	No	
23	0	4	1	2	3	0	No	
24	0	4	1	3	2	0	No	
Total Solutions: 4								

# Hamiltonian Cycles - Backtracking

- ✓ Let an array (C) with size equivalent to the number of vertices (n) in the graph.
- ✓ In this example, n=5
- ✓ The values in array: vertices numbers like 1,2,3....
- ✓ If value=0, the no vertex
- ✓ Initialize the array values as 0 – Indicates no vertex is added in cycle.
- ✓ Cycle may start with any vertex.
- ✓ Let us consider the starting vertex is 1 in this example.
- ✓ So,  $C[1] = 1$ .
- ✓ The next vertex values are increased like 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, etc.,
- ✓ The next vertex for the cycles will be selected based on the bounding conditions.
- ✓ Each level in state space tree represents the selection of next vertex
- ✓ So, the tree's maximum level is  $n-1$



C	0	0	0	0	0
	1	2	3	4	5

C	1	0	0	0	0
	1	2	3	4	5

# Hamiltonian Cycles - Backtracking

## Bounding Conditions:

- ✓ Let the starting node,  $s=1$
- ✓ Let the current vertex is 'i' and the selected next vertex is 'j'
- ✓ For every vertex  $i = \{2,3,4,\dots,n\}$ , the next vertex 'j' is selected as follows.

### Case 1:

- ✓ If  $i=n$  and there is an edge  $(i, s)$ , then record the solution and backtrack

### Case 2:

- ✓ If  $i=n$  and there is no edge  $(i, s)$ , no solution found, just backtrack

### Case 3:

- ✓ If  $j$  is already presented in the array, then backtrack

### Case 4:

- ✓ If there is no edge  $(i, j)$ , then backtrack

- ✓ For the next level, need to select possible values for C[2].

- ✓ The current value of C[2] is 0.

- ✓ It will be increased as 1,2,3...

- ✓ So next possible values is 1.

But this 1 already presented in array.

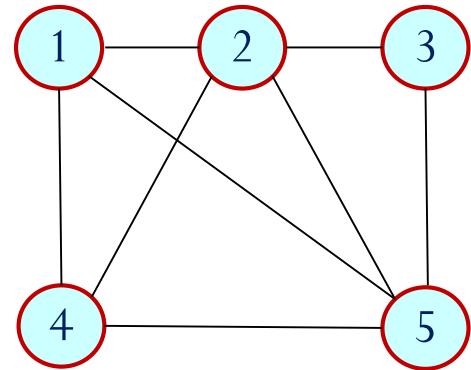
- ✓ So go for next value: 2

- ✓ Check for 1 to 2, edge is present or not.

- ✓ Yes. Edge (1, 2) is presented

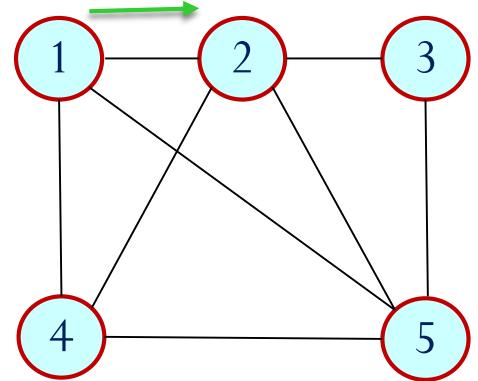
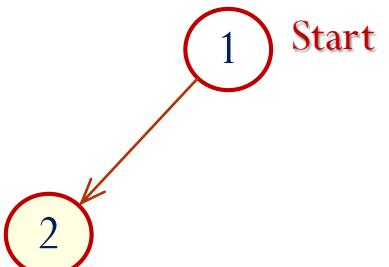
- ✓ So proceed for next level with selected vertex as 2

1 Start



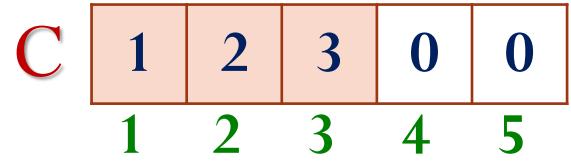
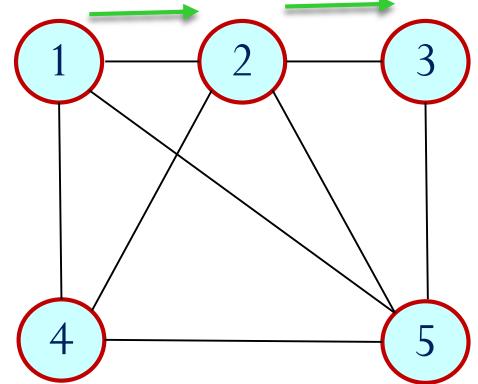
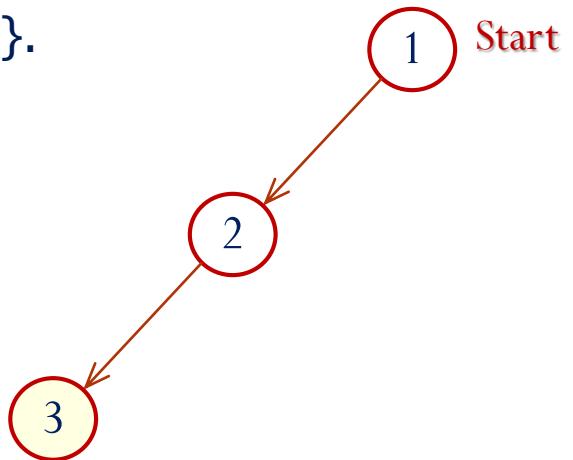
C	1	0	0	0	0
	1	2	3	4	5

- ✓ The next possible vertices for 2 is {3, 4, 5}.
- ✓ So proceed with 3 as next vertex

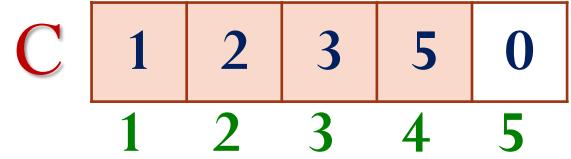
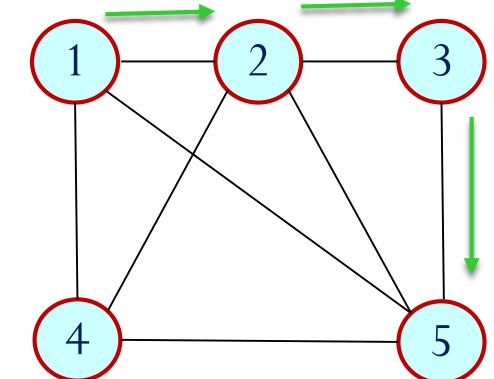
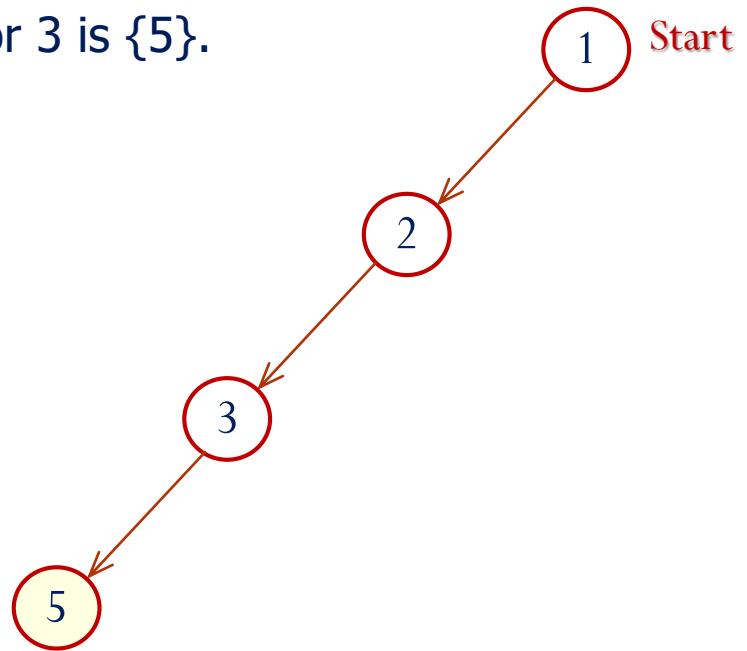


C	1	2	0	0	0
	1	2	3	4	5

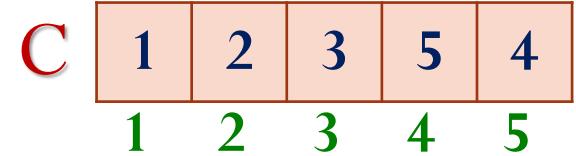
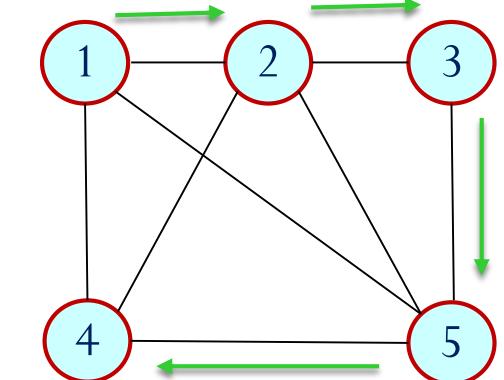
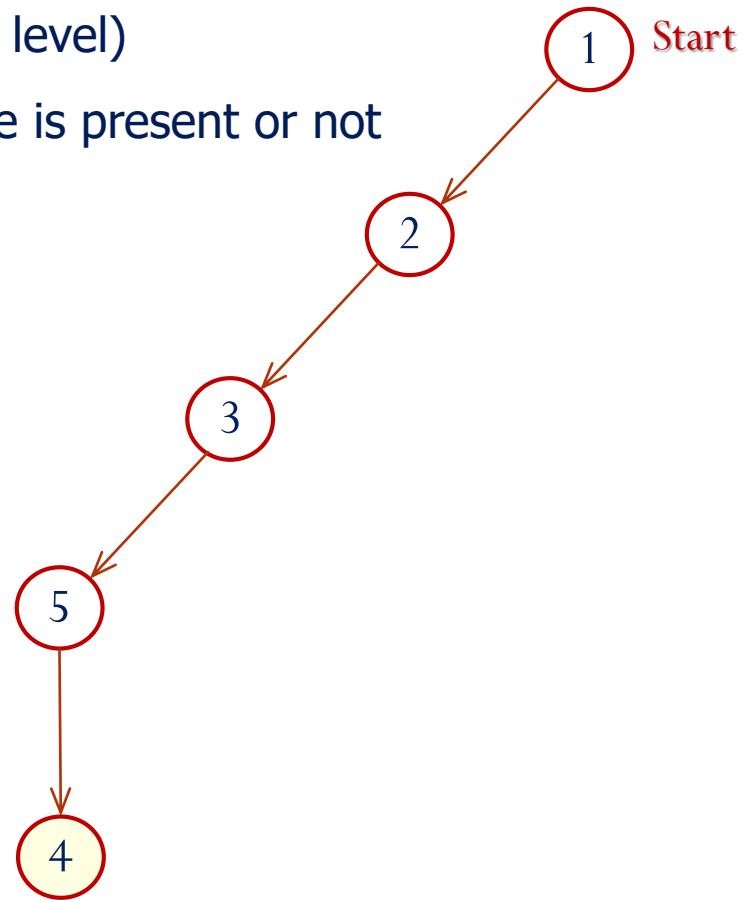
- ✓ The next possible vertices for 3 is {5}.



- ✓ The next possible vertices for 3 is {5}.



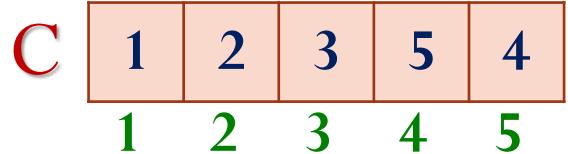
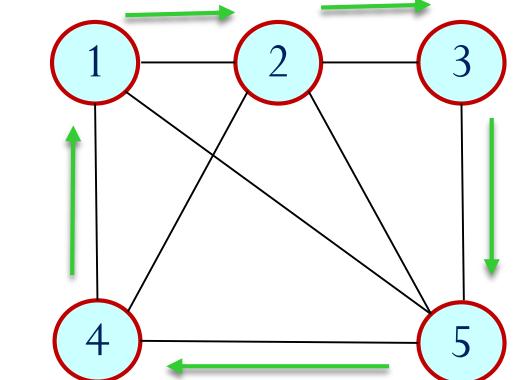
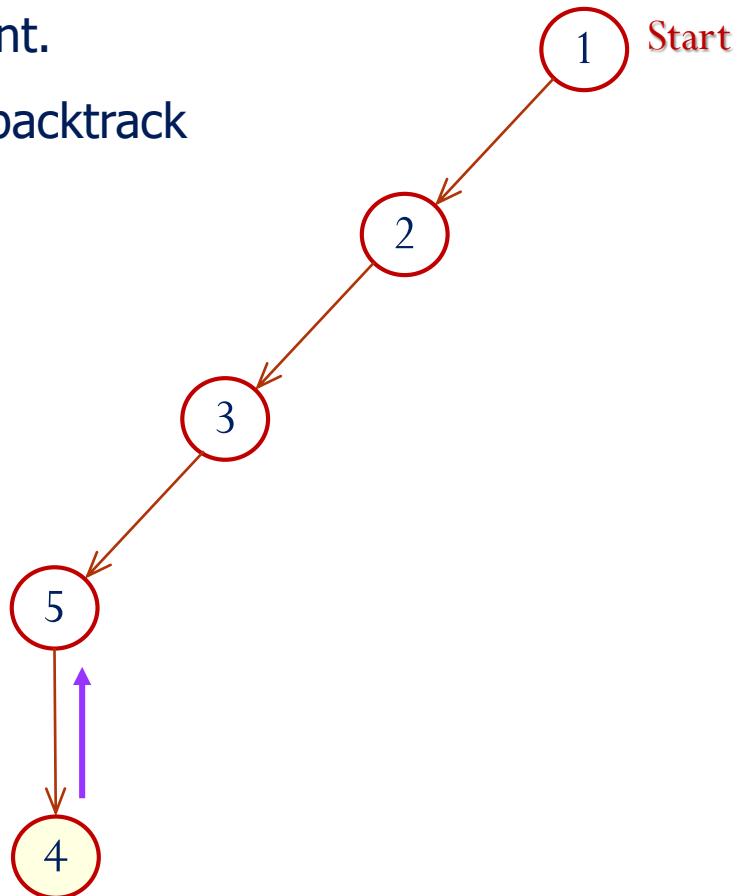
- ✓ Now reached last level. (nth level)
- ✓ So, need to check (4,1) edge is present or not



- ✓ Yes. The edge (4,1) is Present.
- ✓ So, record the solution and backtrack

Solutions

1	2	3	5	4
---	---	---	---	---

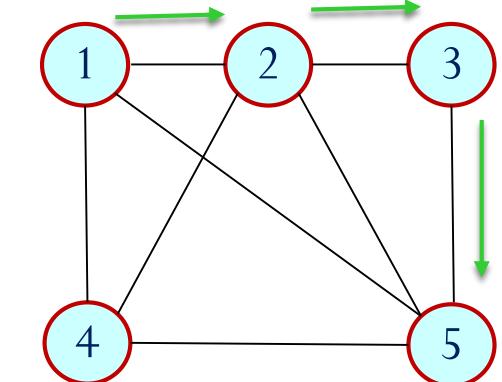
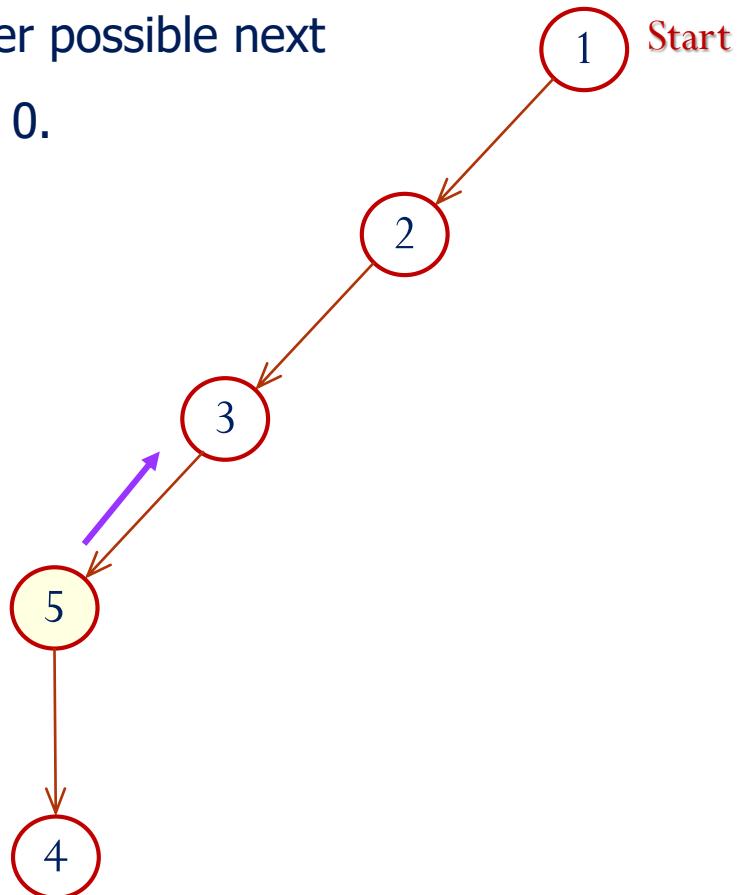


- ✓ For vertex 5, there is no other possible next vertex. So,  $C[5]$  will become 0.

- ✓ Backtrack to previous state

Solutions

1	2	3	5	4
---	---	---	---	---



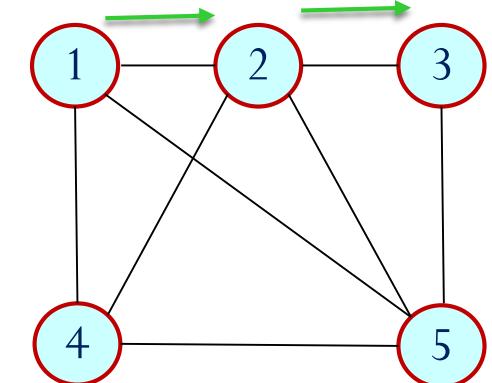
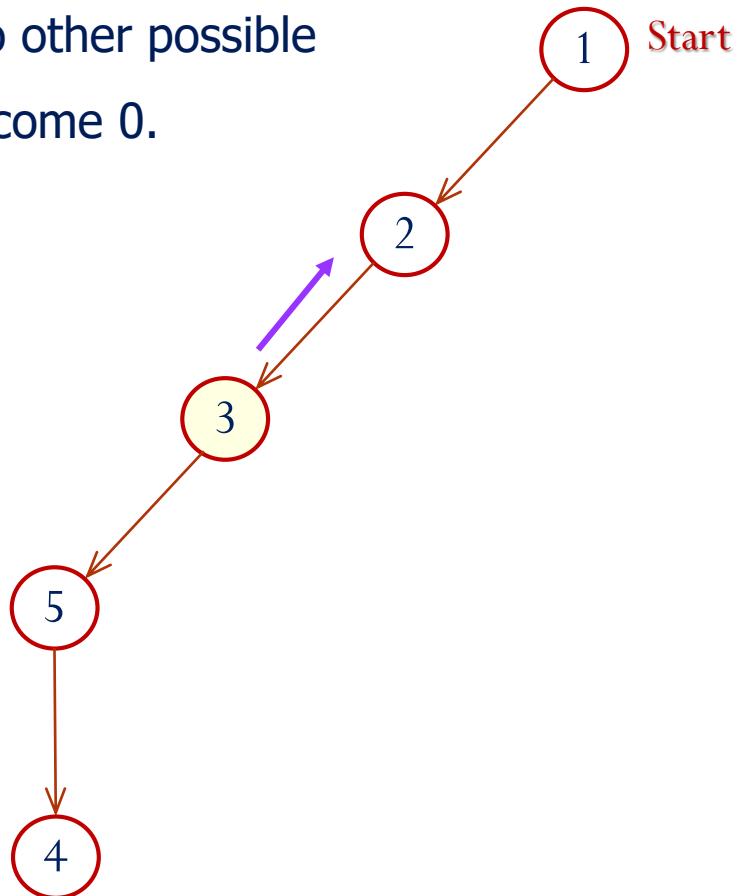
C	1	2	3	5	0
	1	2	3	4	5

- ✓ For vertex 3 also, there is no other possible next vertex. So,  $C[4]$  will become 0.

- ✓ Backtrack to previous state

Solutions

1	2	3	5	4
---	---	---	---	---

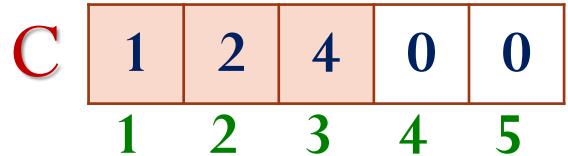
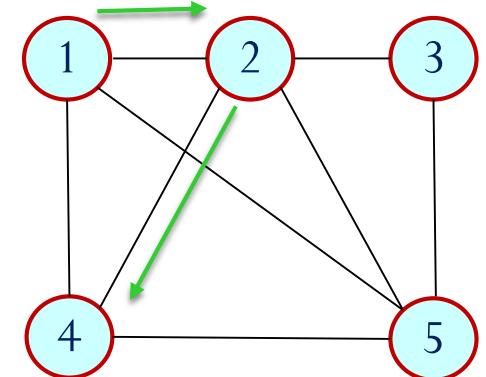
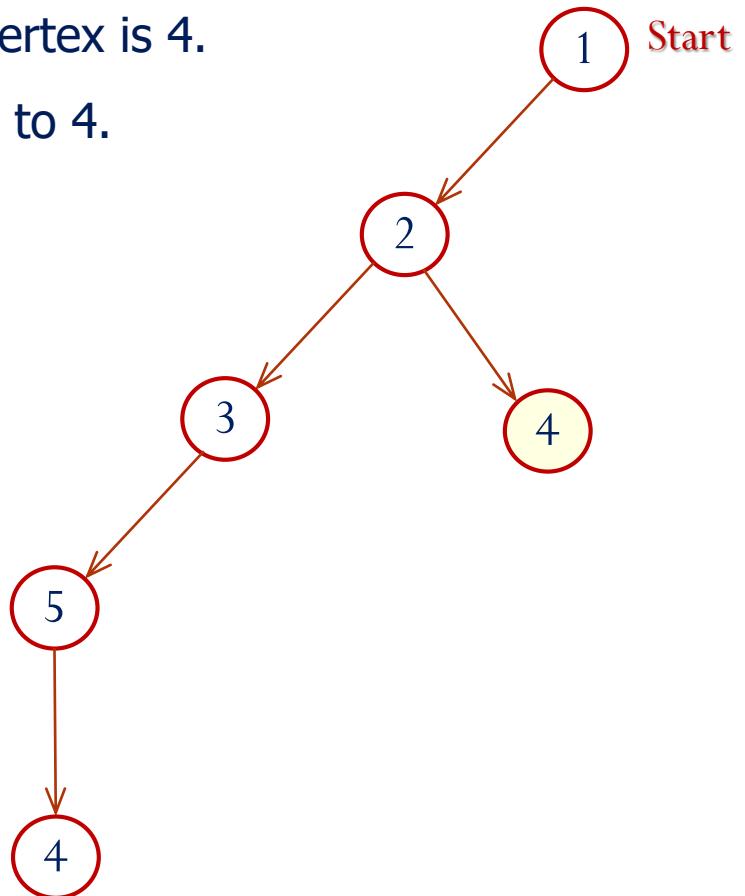


C	1	2	3	0	0
	1	2	3	4	5

- ✓ For vertex 2, next possible vertex is 4.
- ✓ So, C[3] will be incremented to 4.

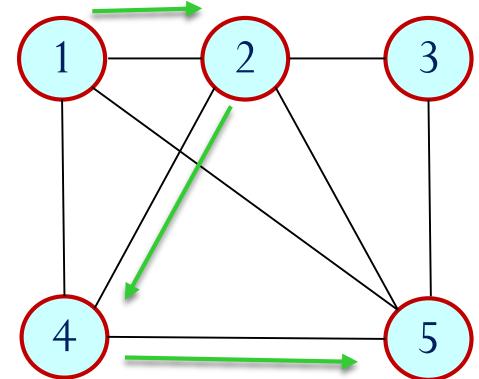
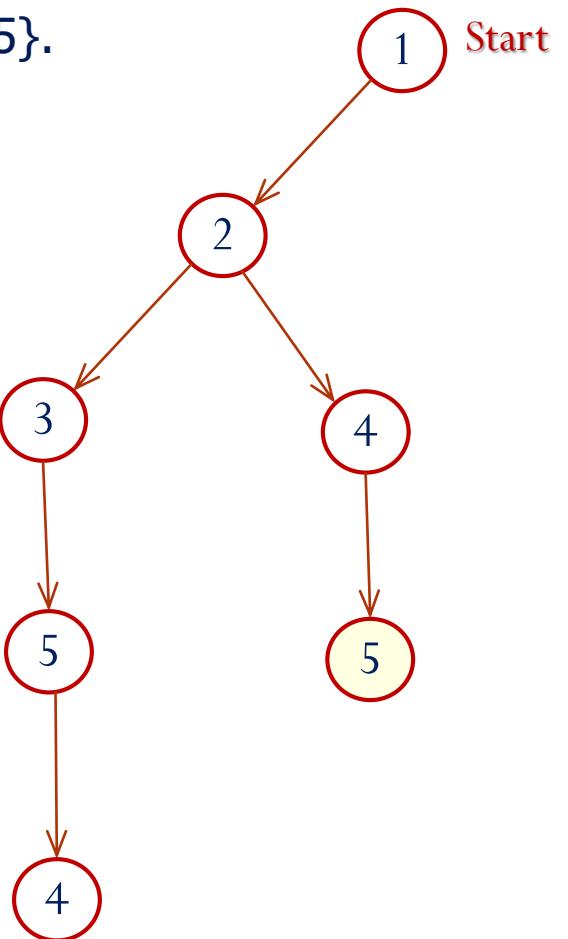
Solutions

1	2	3	5	4
---	---	---	---	---



✓ For vertex 4, next possible vertices {5}.

✓ So, C[4] will be incremented to 5.



C

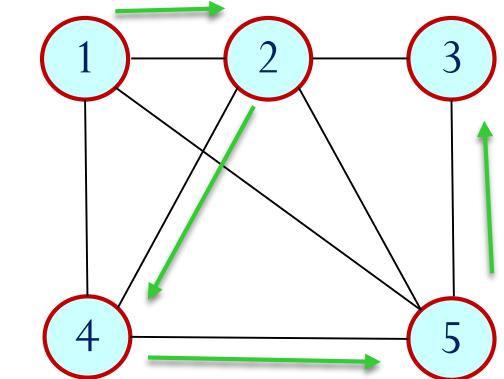
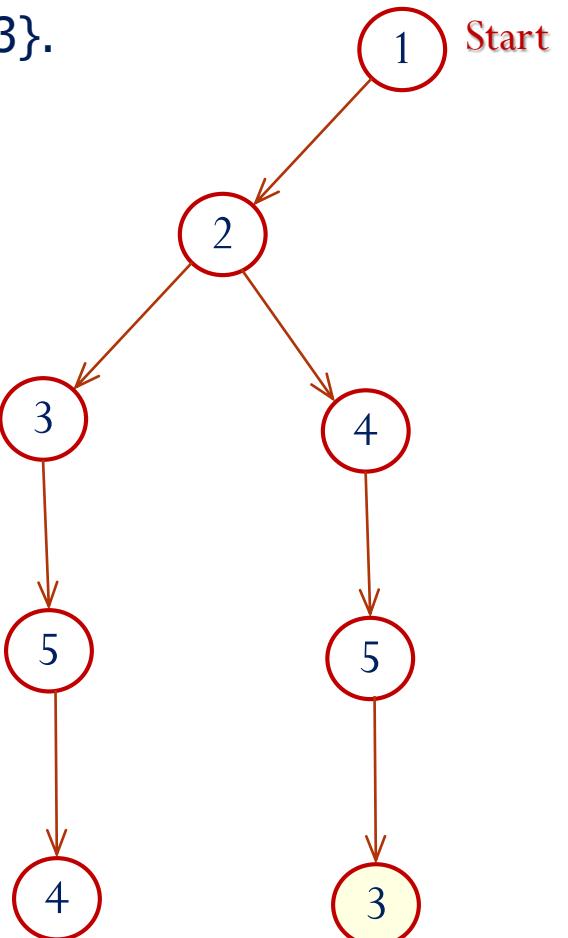
1	2	4	5	0
1	2	3	4	5

✓ For vertex 5, next possible vertices {3}.

✓ So, C[5] will be incremented to 3.

Solutions

1	2	3	5	4
---	---	---	---	---



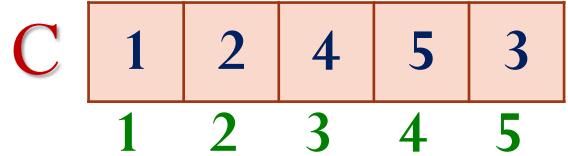
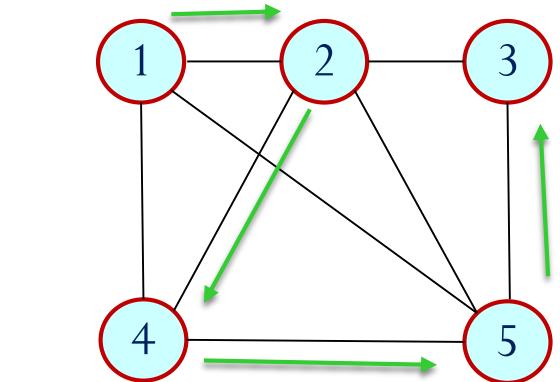
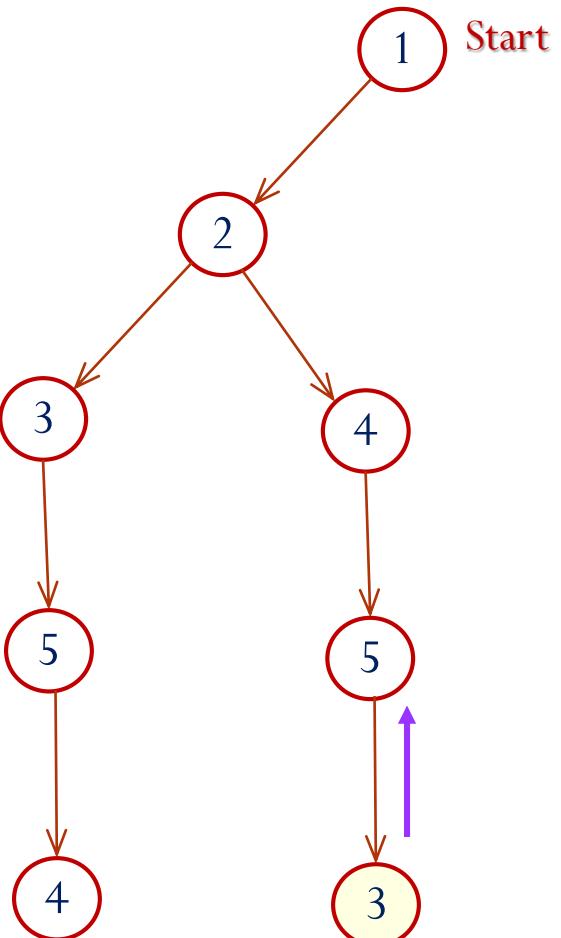
C

1	2	4	5	3
1	2	3	4	5

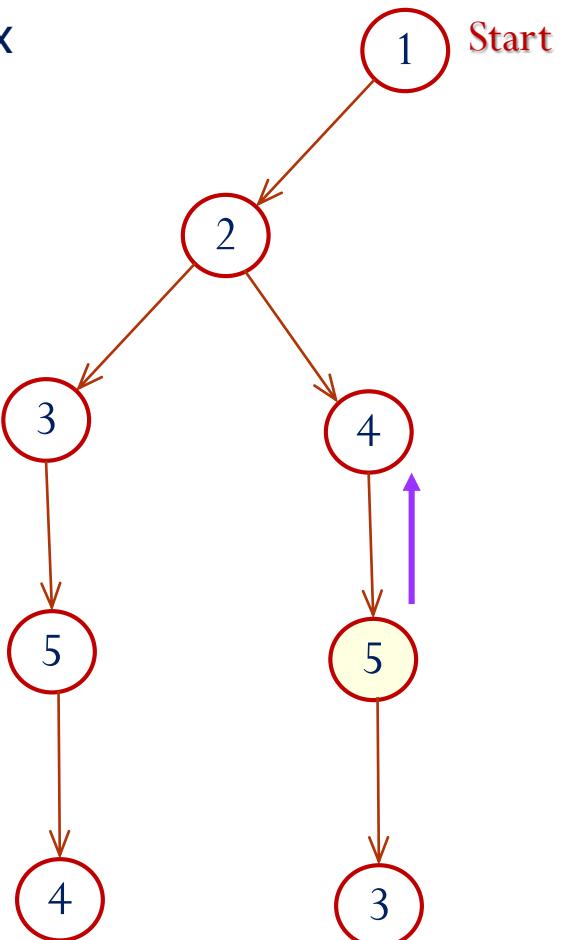
- ✓ Now, last level is reached.
- ✓ (3,1) edge is not present
- ✓ So no solution found
- ✓ Just Backtrack

Solutions

1	2	3	5	4
---	---	---	---	---

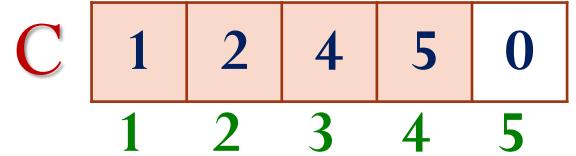
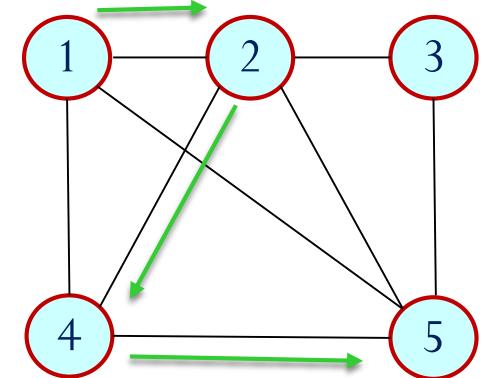


- ✓ For vertex 5, no other possible vertex
- ✓ So backtrack



Solutions

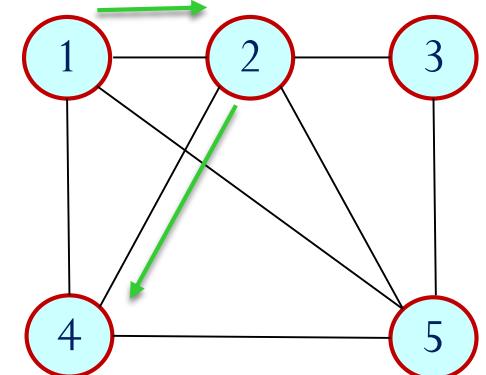
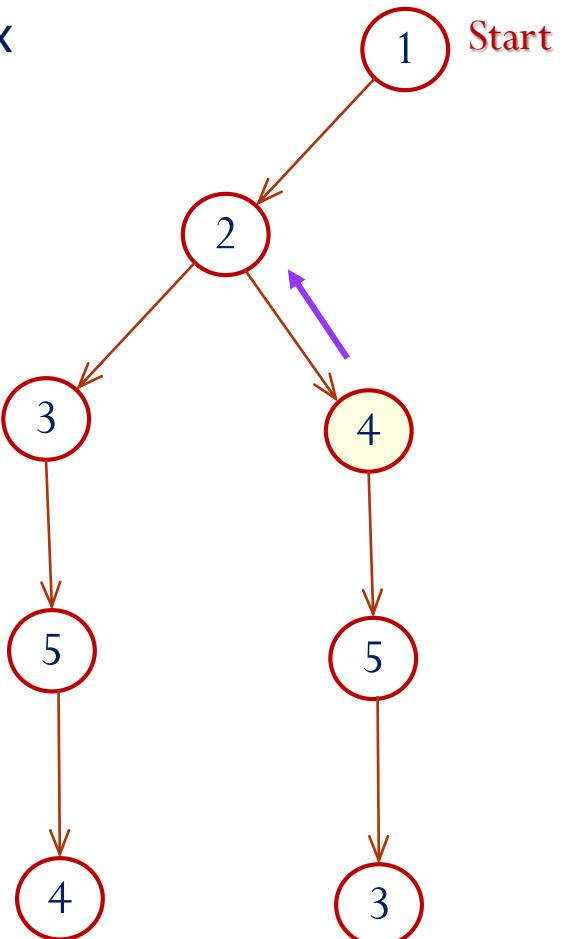
1	2	3	5	4
---	---	---	---	---



- ✓ For vertex 4, no other possible vertex
- ✓ So backtrack

Solutions

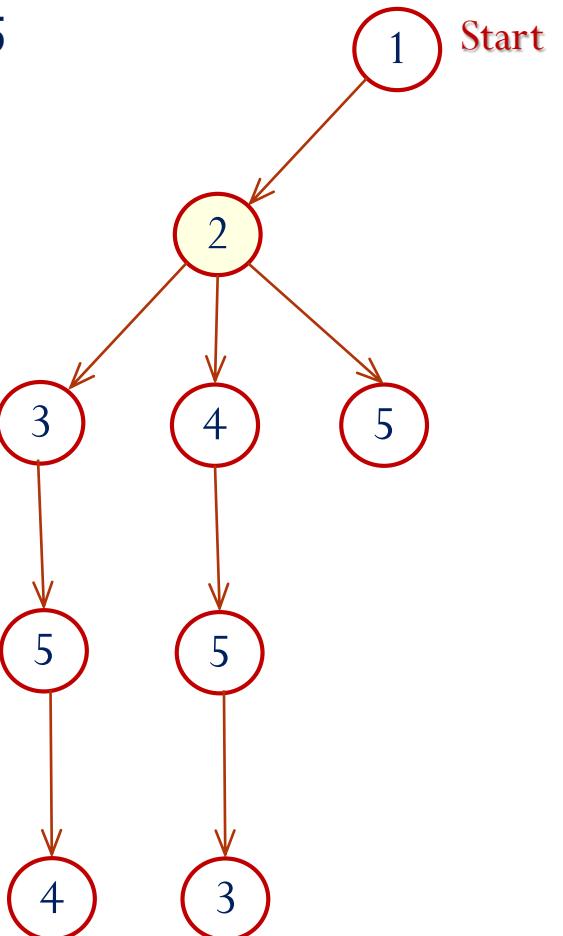
1	2	3	5	4
---	---	---	---	---



C

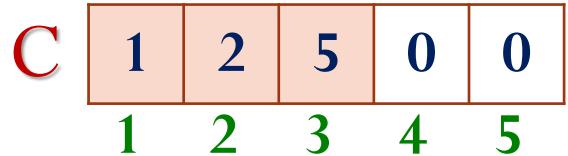
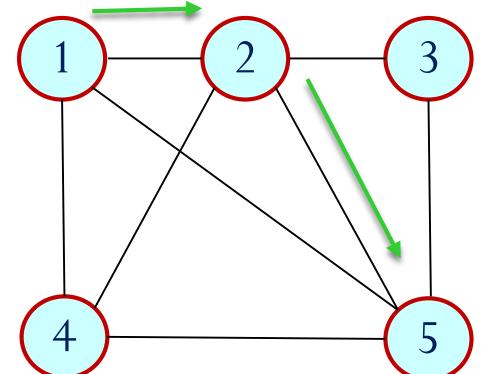
1	2	4	0	0
1	2	3	4	5

- ✓ For vertex 2, next possible vertex is 5



Solutions

1	2	3	5	4
---	---	---	---	---

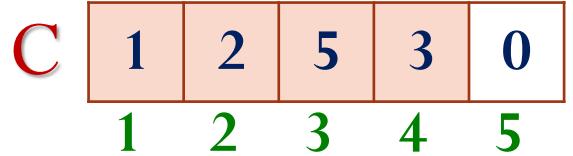
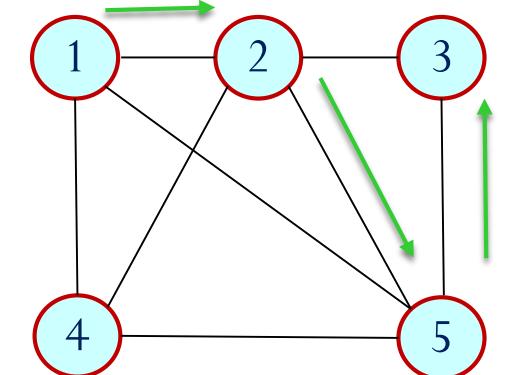
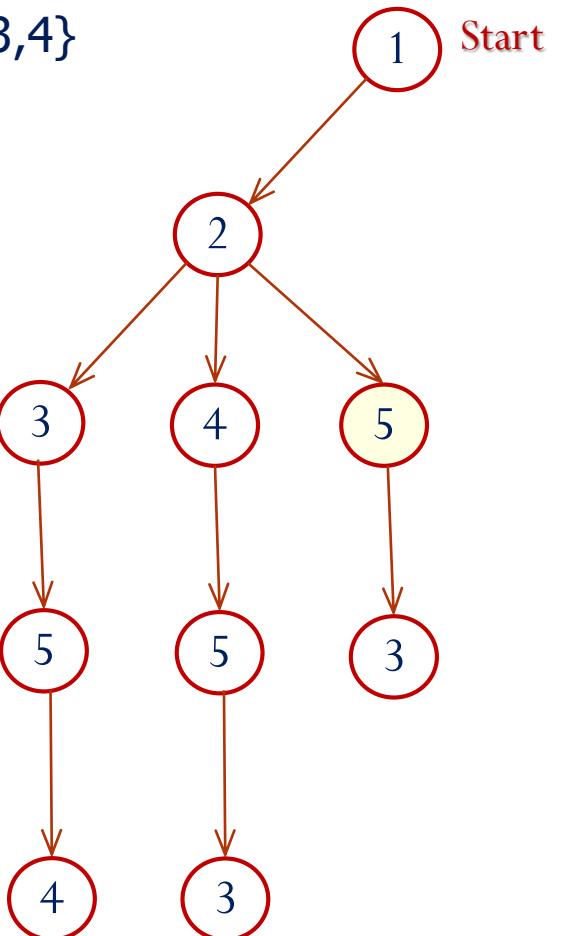


✓ For vertex 5, next possible vertices {3,4}

✓ Proceed with 3

Solutions

1	2	3	5	4
---	---	---	---	---

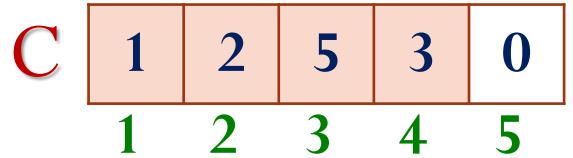
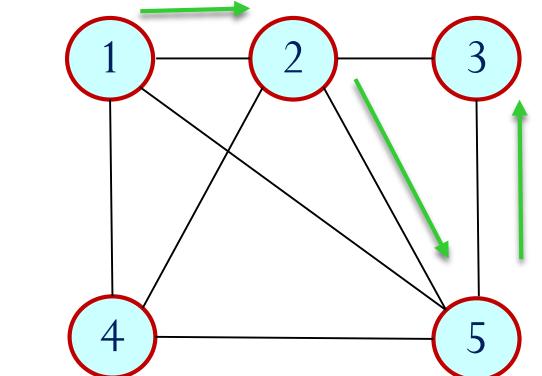
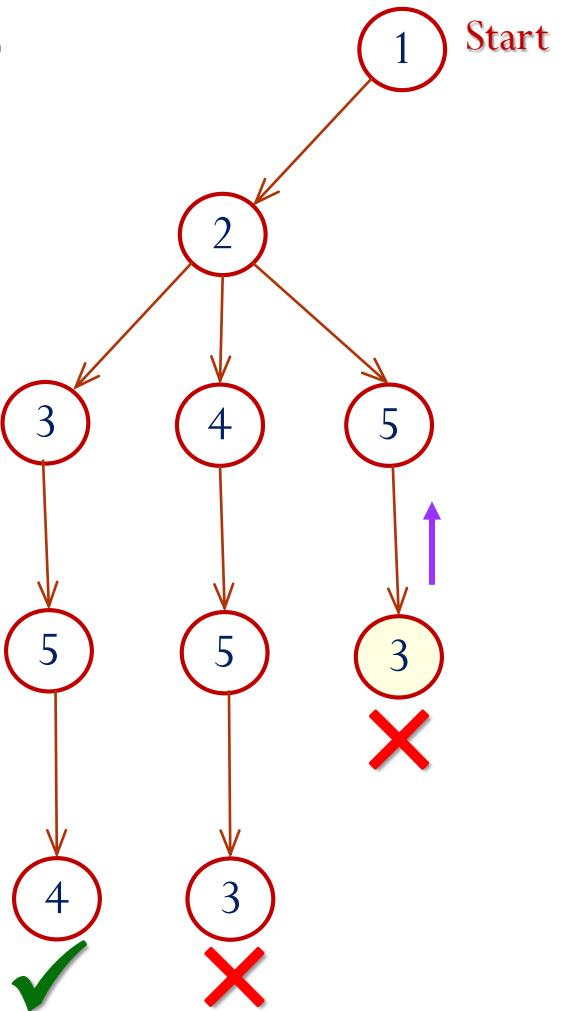


✓ For vertex 3, no possible next vertex.

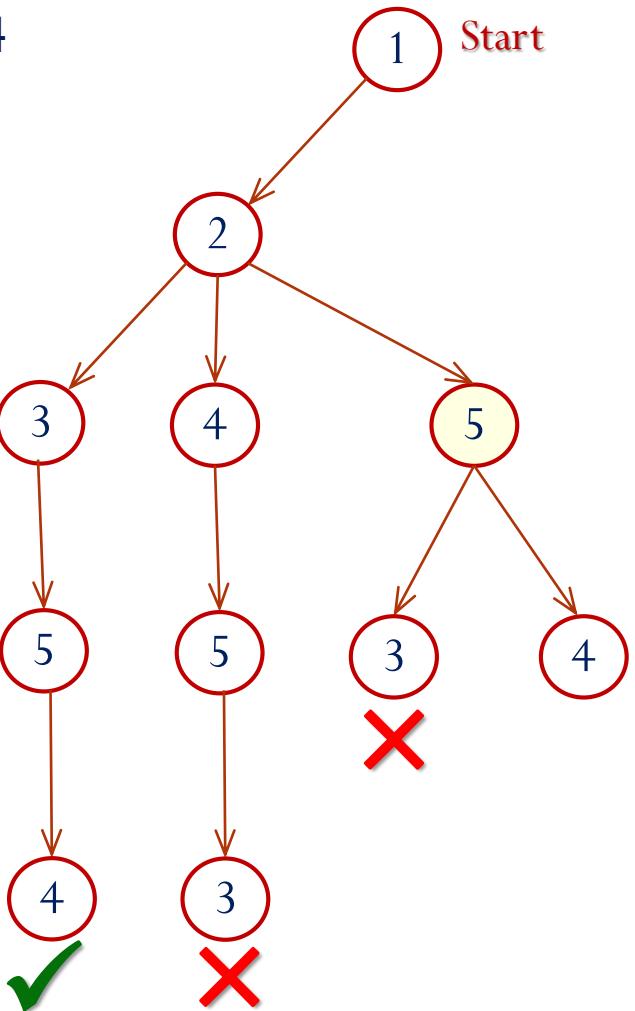
✓ So, backtrack

Solutions

1	2	3	5	4
---	---	---	---	---

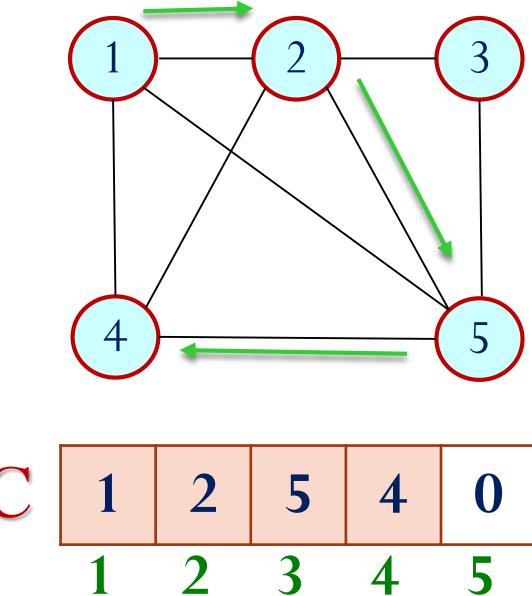


- ✓ For vertex 5, possible next vertex is 4

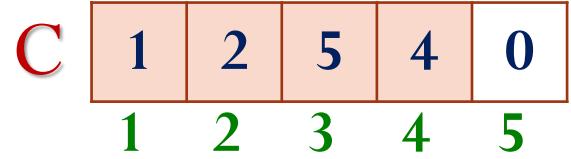
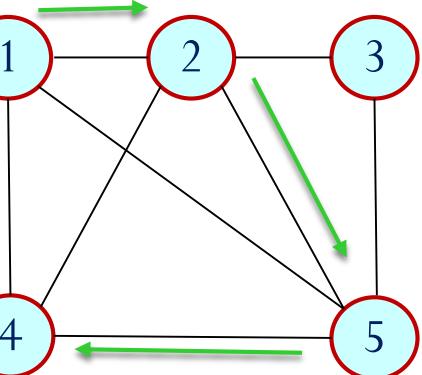


Solutions

1	2	3	5	4
---	---	---	---	---

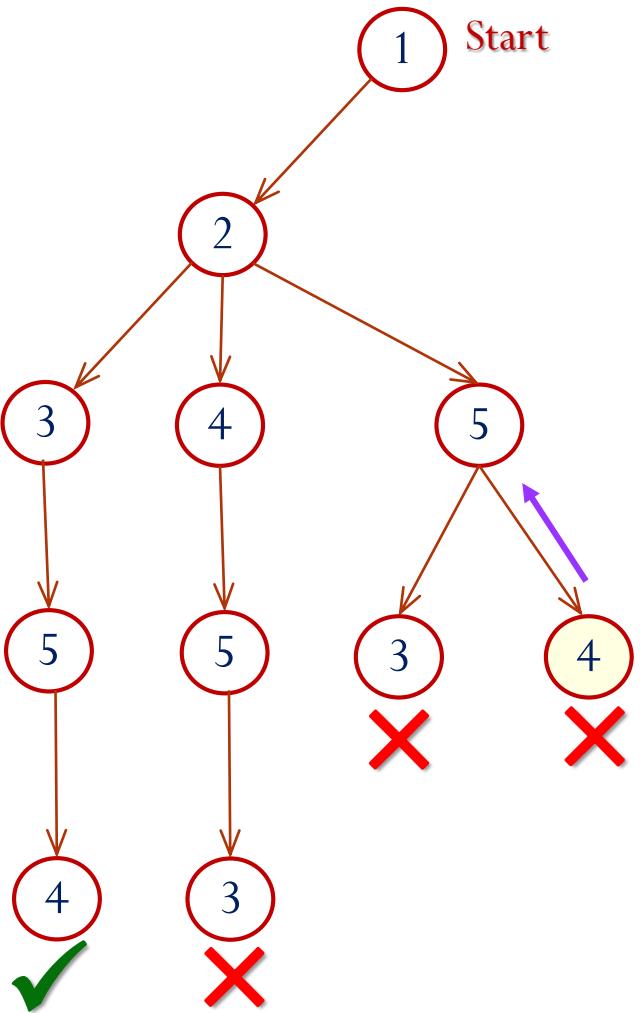


- ✓ For vertex 4, no possible vertex
- ✓ So, backtrack



Solutions

1	2	3	5	4
---	---	---	---	---

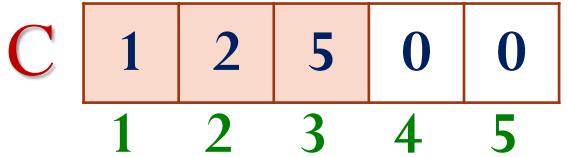
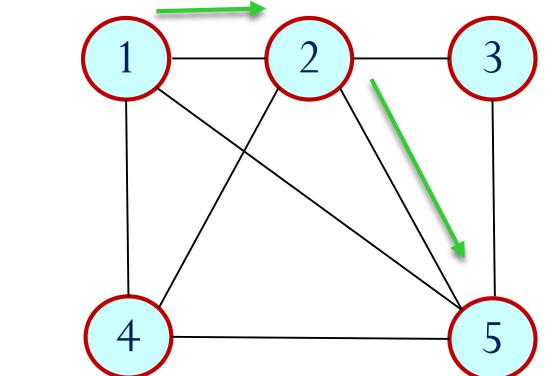
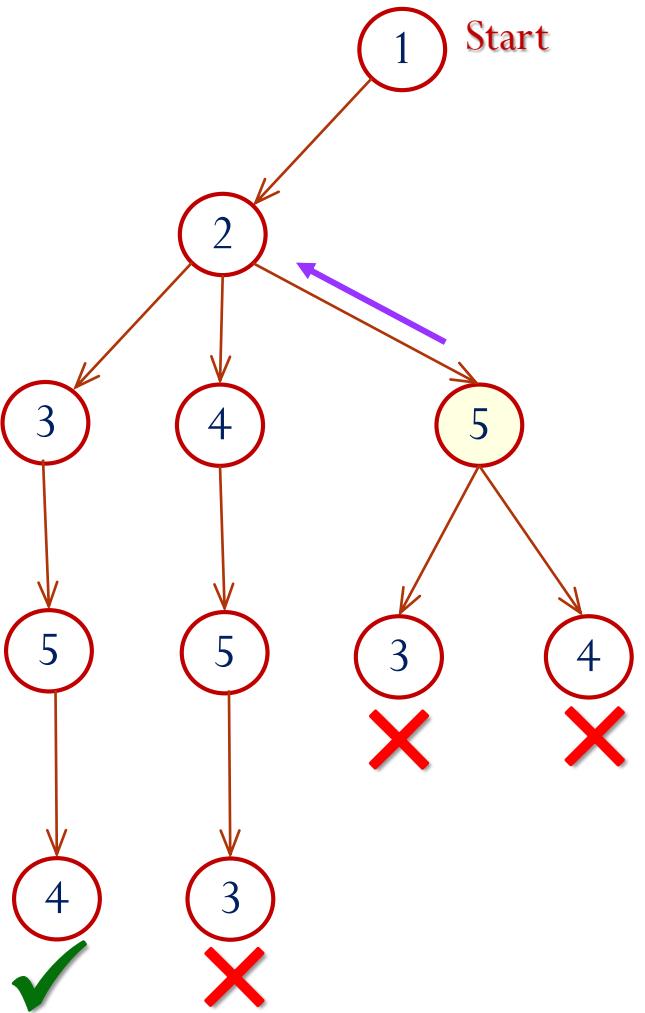


- ✓ For vertex 5, no possible vertex
- ✓ So, backtrack



Solutions

1	2	3	5	4
---	---	---	---	---

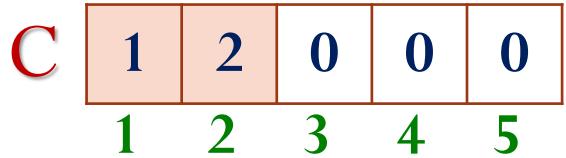
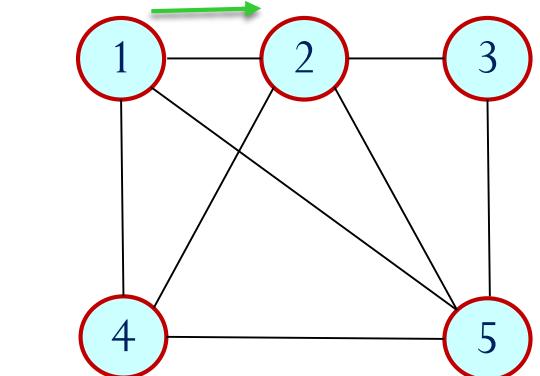
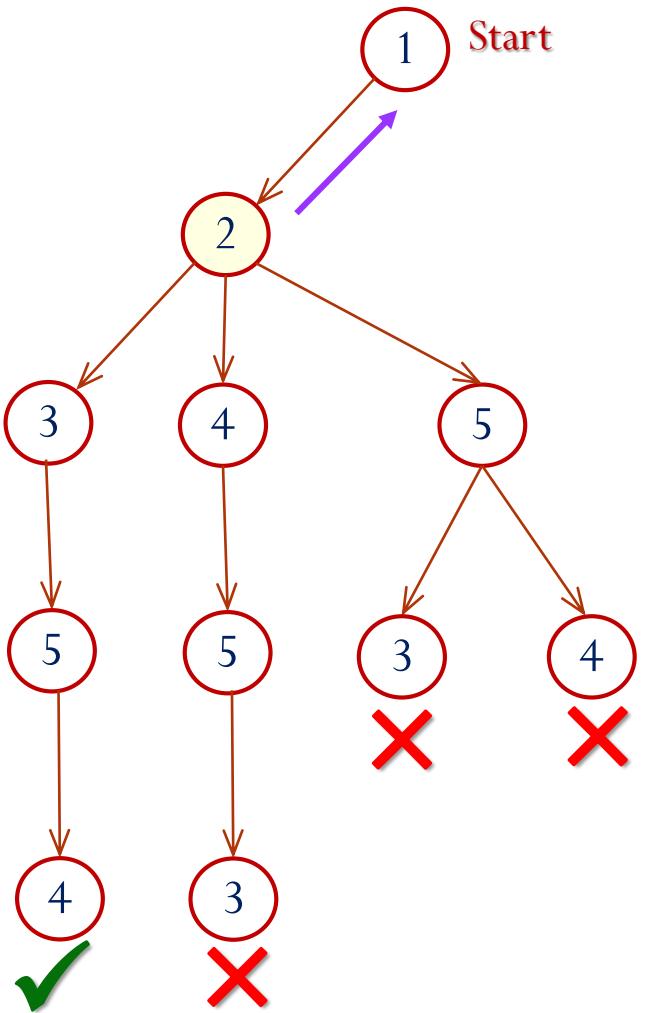


- ✓ For vertex 2, no possible vertex
- ✓ So, backtrack

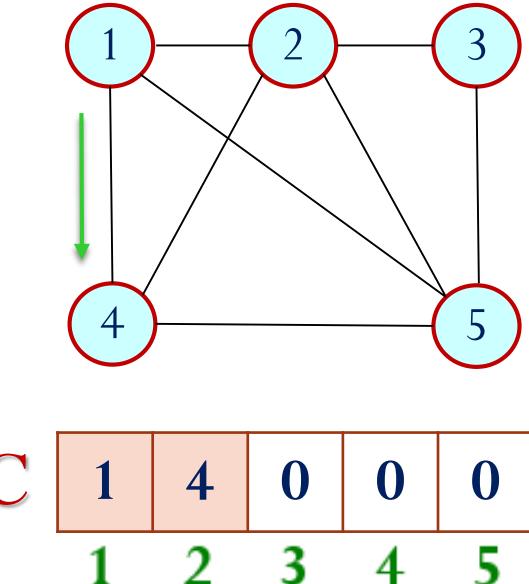
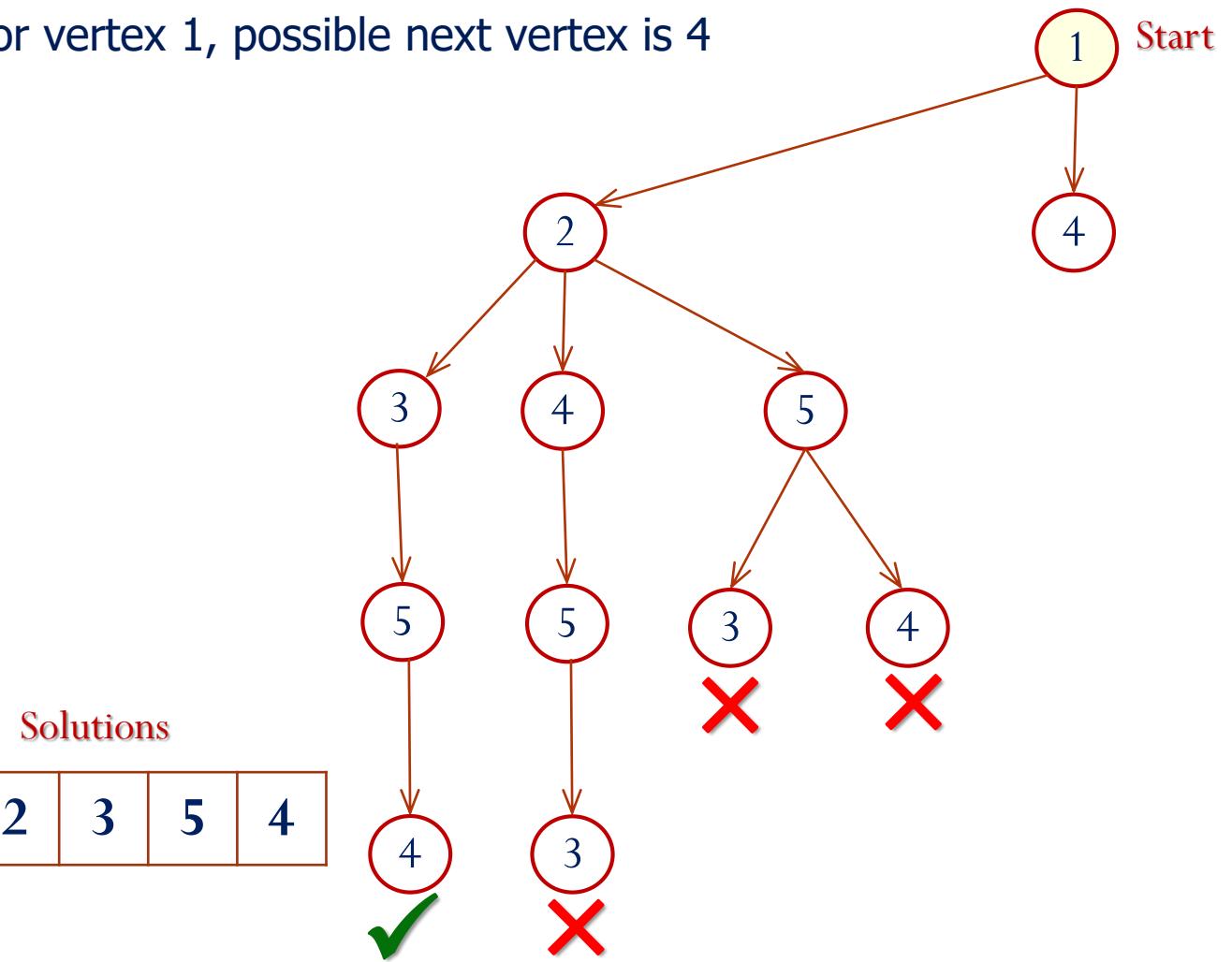


Solutions

1	2	3	5	4
---	---	---	---	---

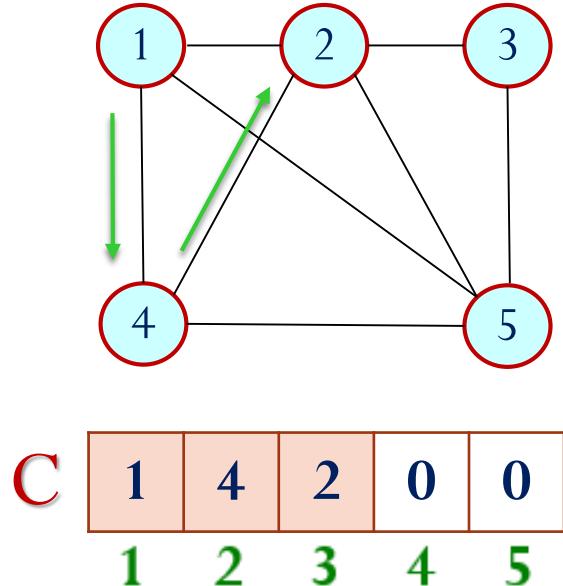
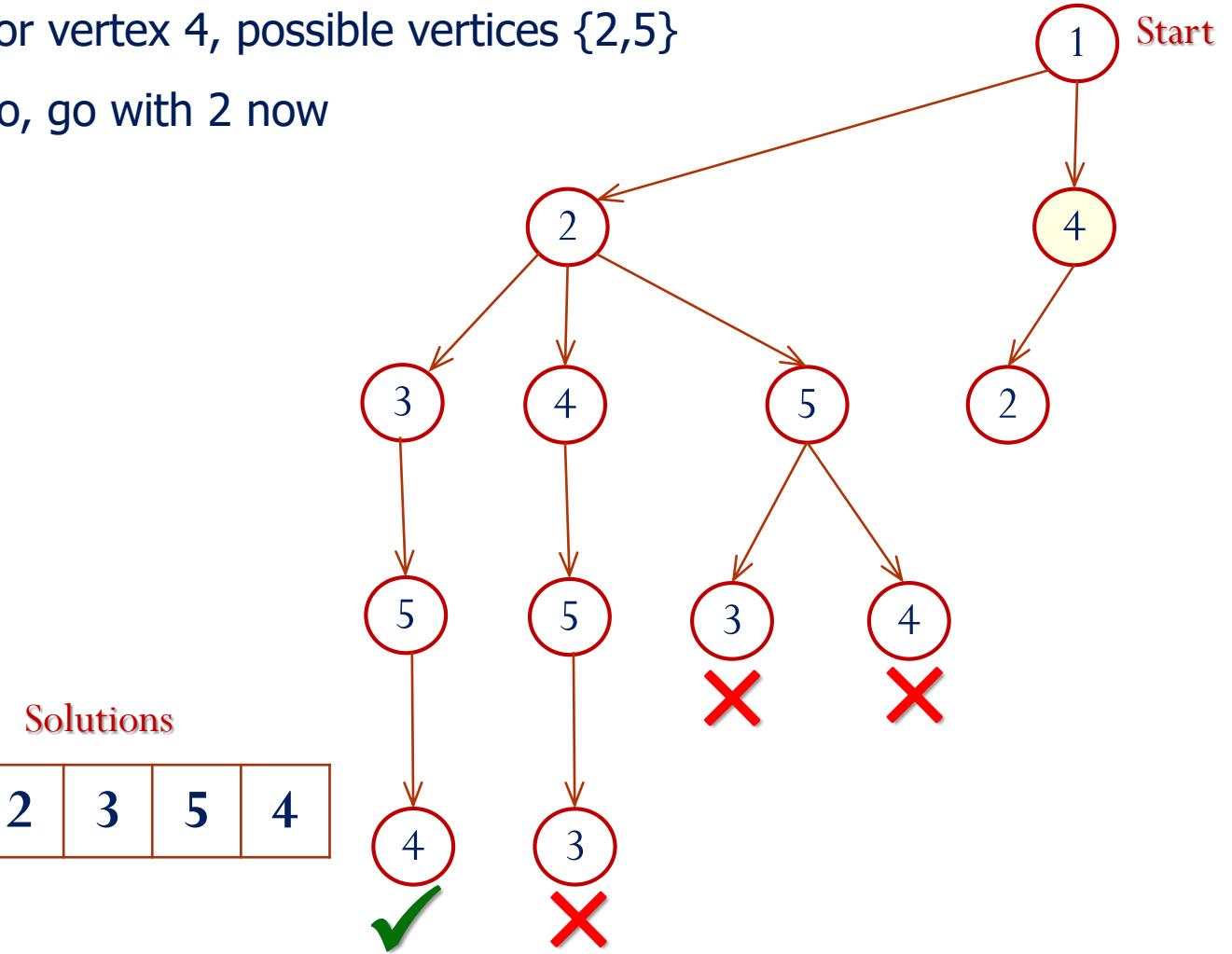


- ✓ For vertex 1, possible next vertex is 4



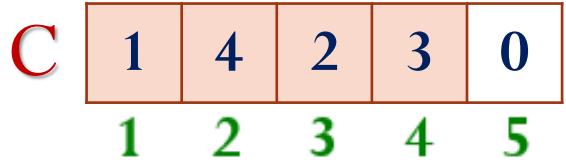
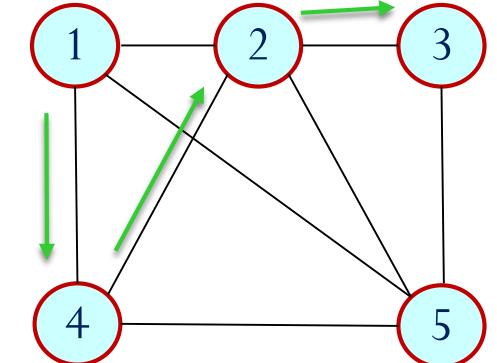
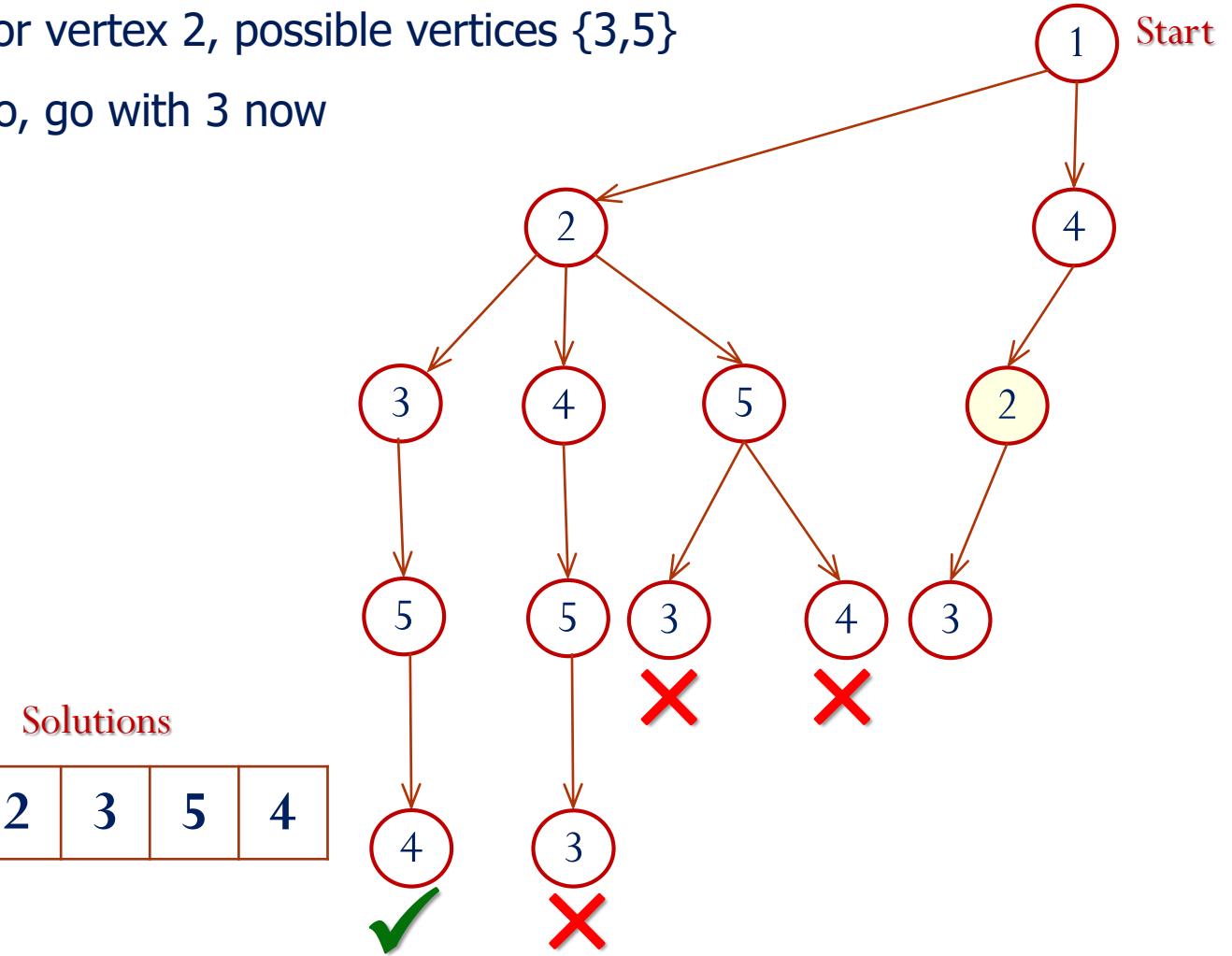
✓ For vertex 4, possible vertices {2,5}

✓ So, go with 2 now



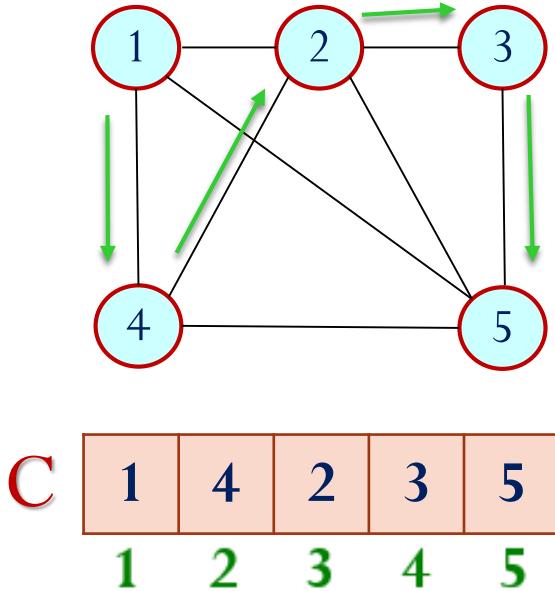
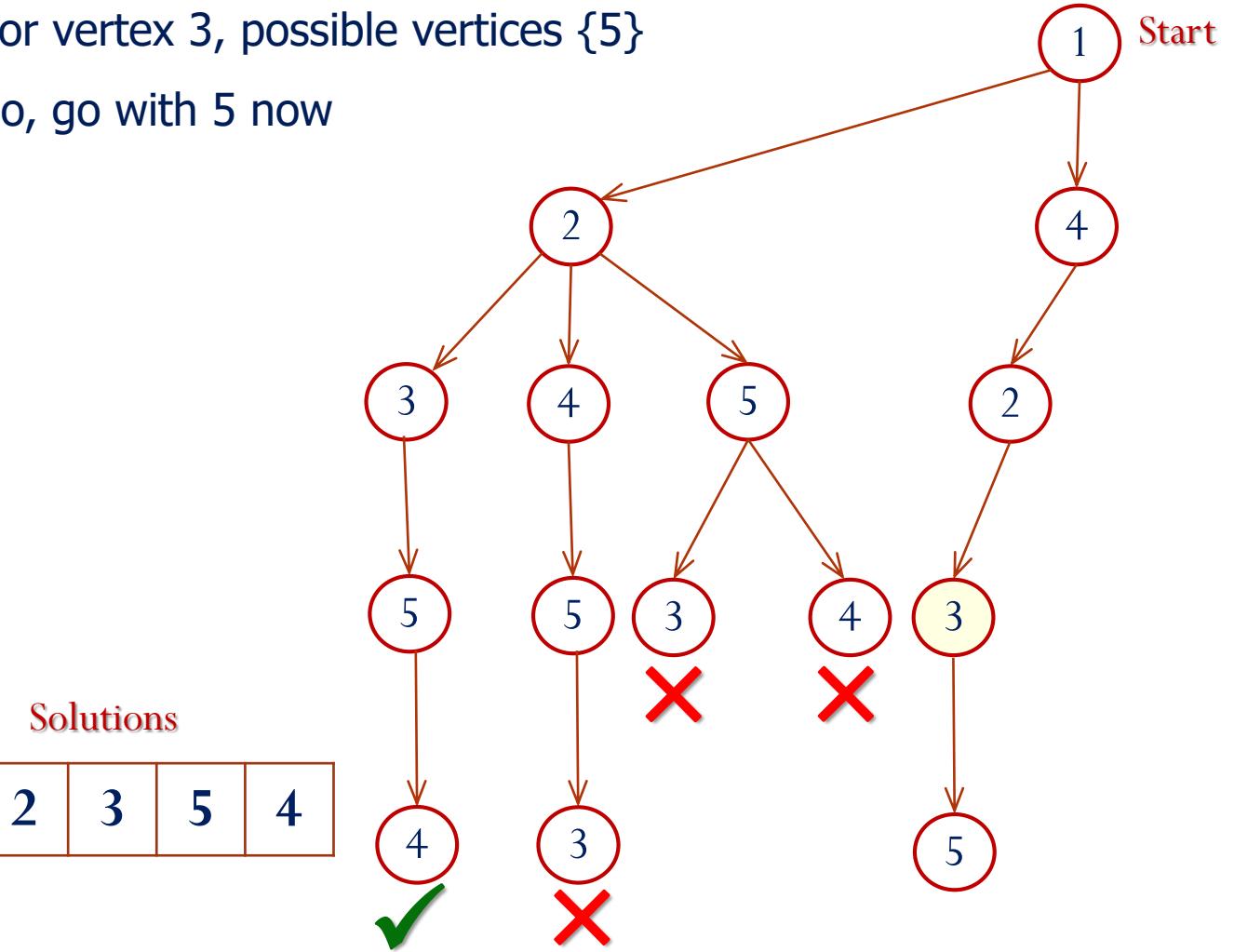
✓ For vertex 2, possible vertices {3,5}

✓ So, go with 3 now

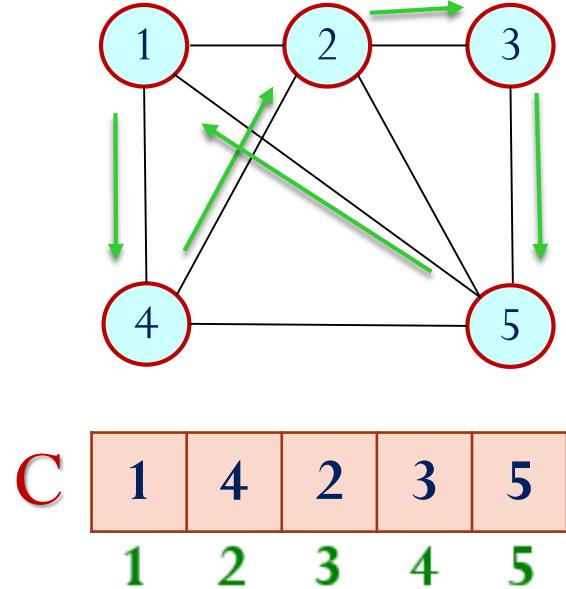
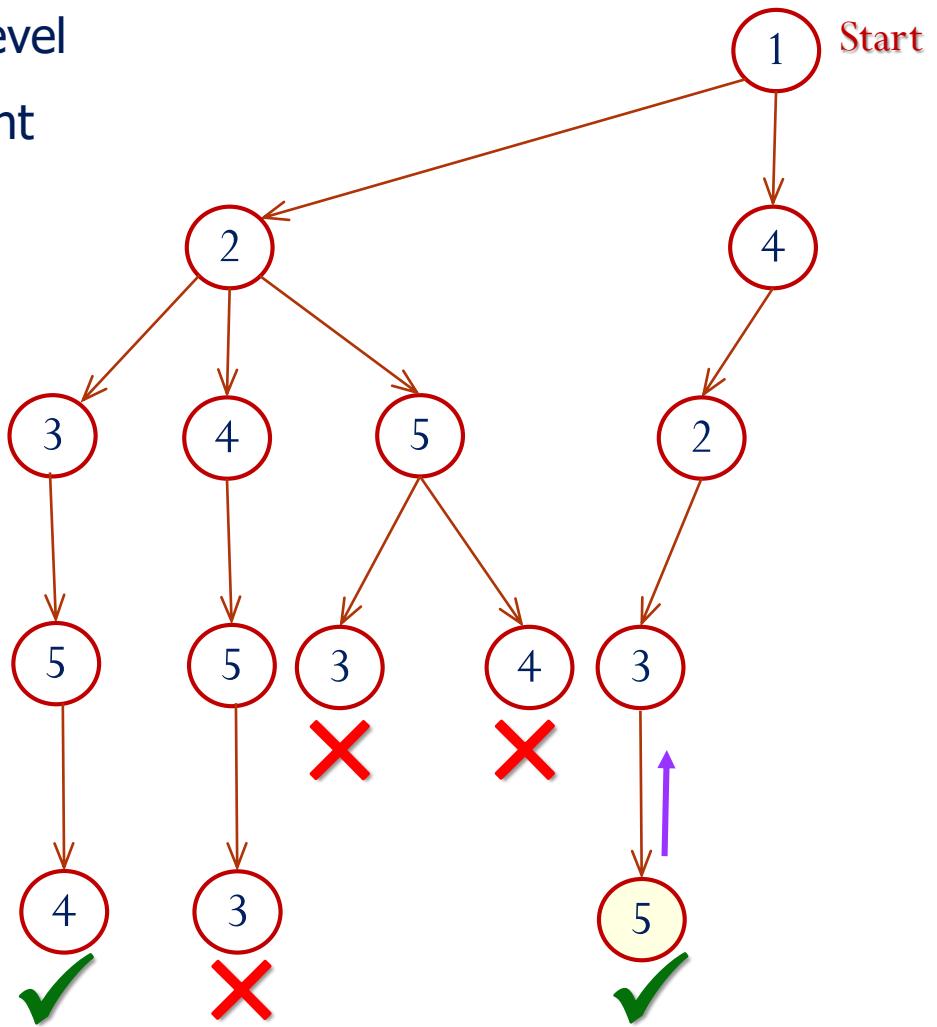


✓ For vertex 3, possible vertices {5}

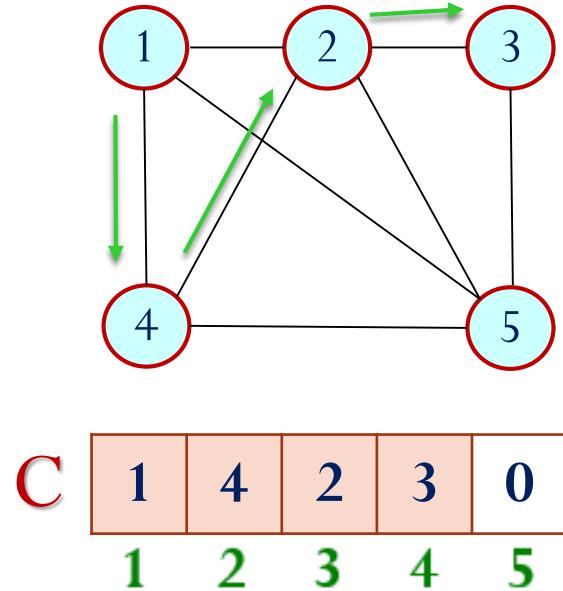
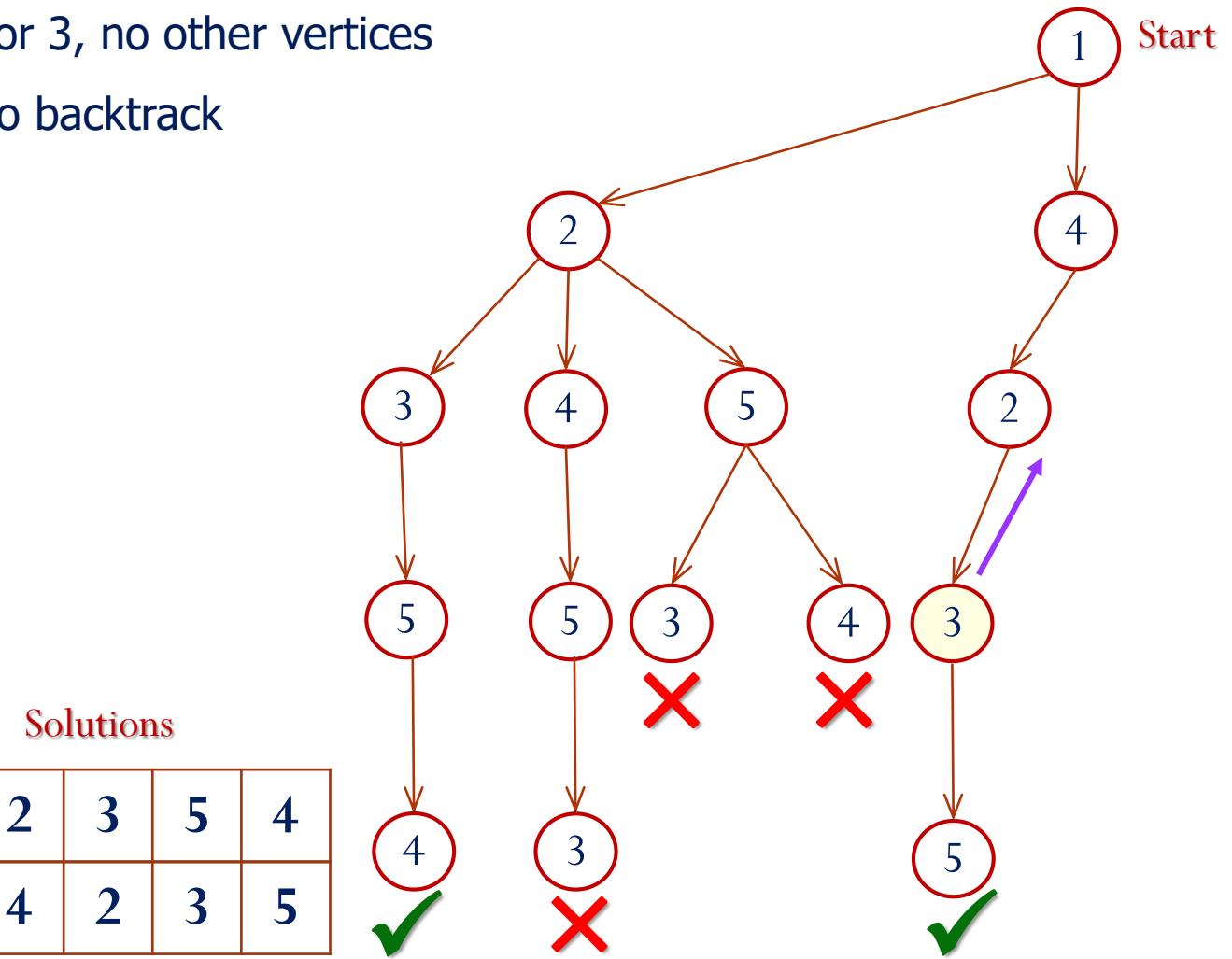
✓ So, go with 5 now



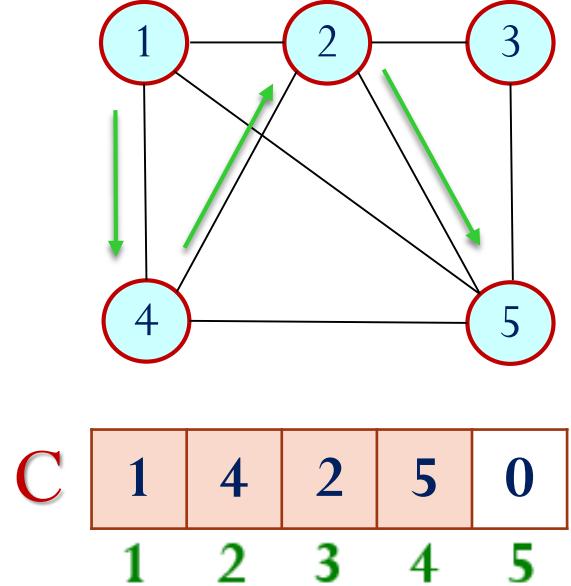
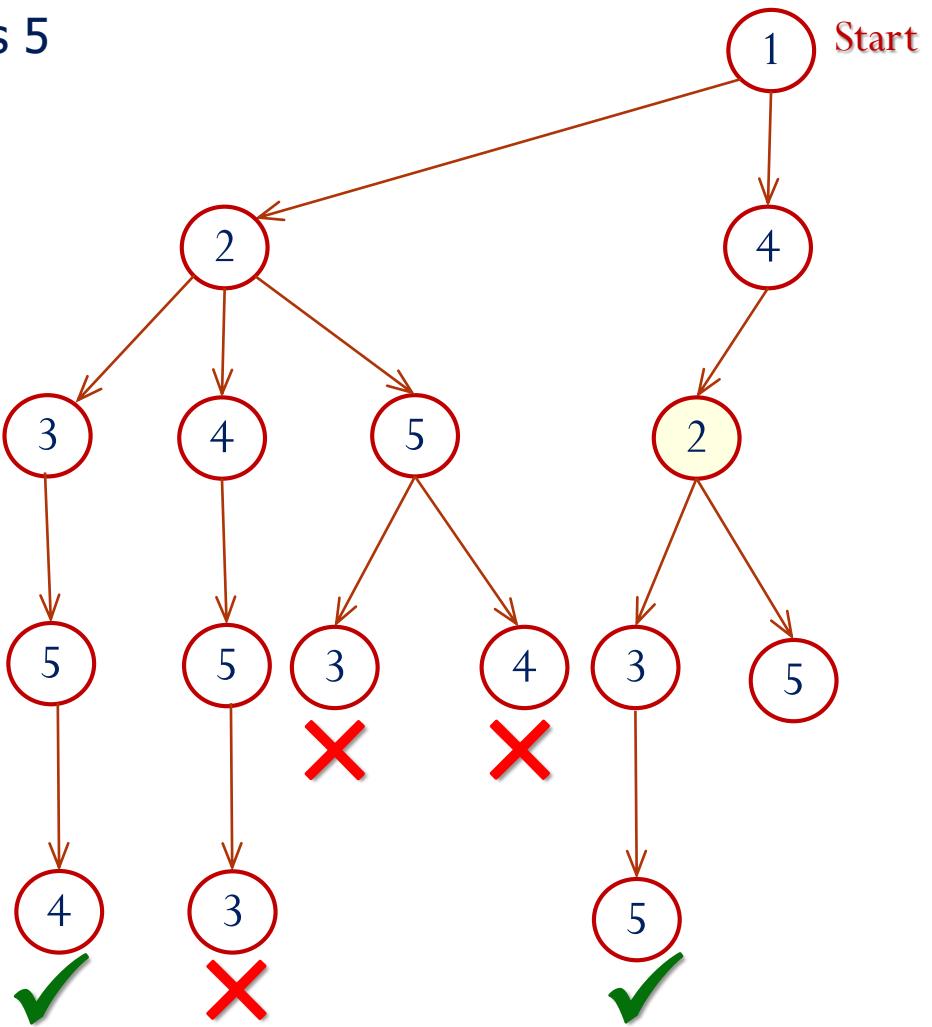
- ✓ Now reached last level
- ✓ (5,1) edge is present
- ✓ So, record solution
- ✓ And backtrack



- ✓ For 3, no other vertices
- ✓ So backtrack

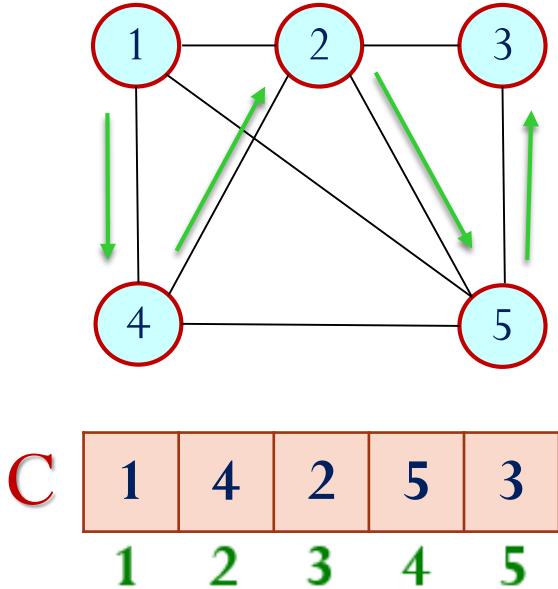
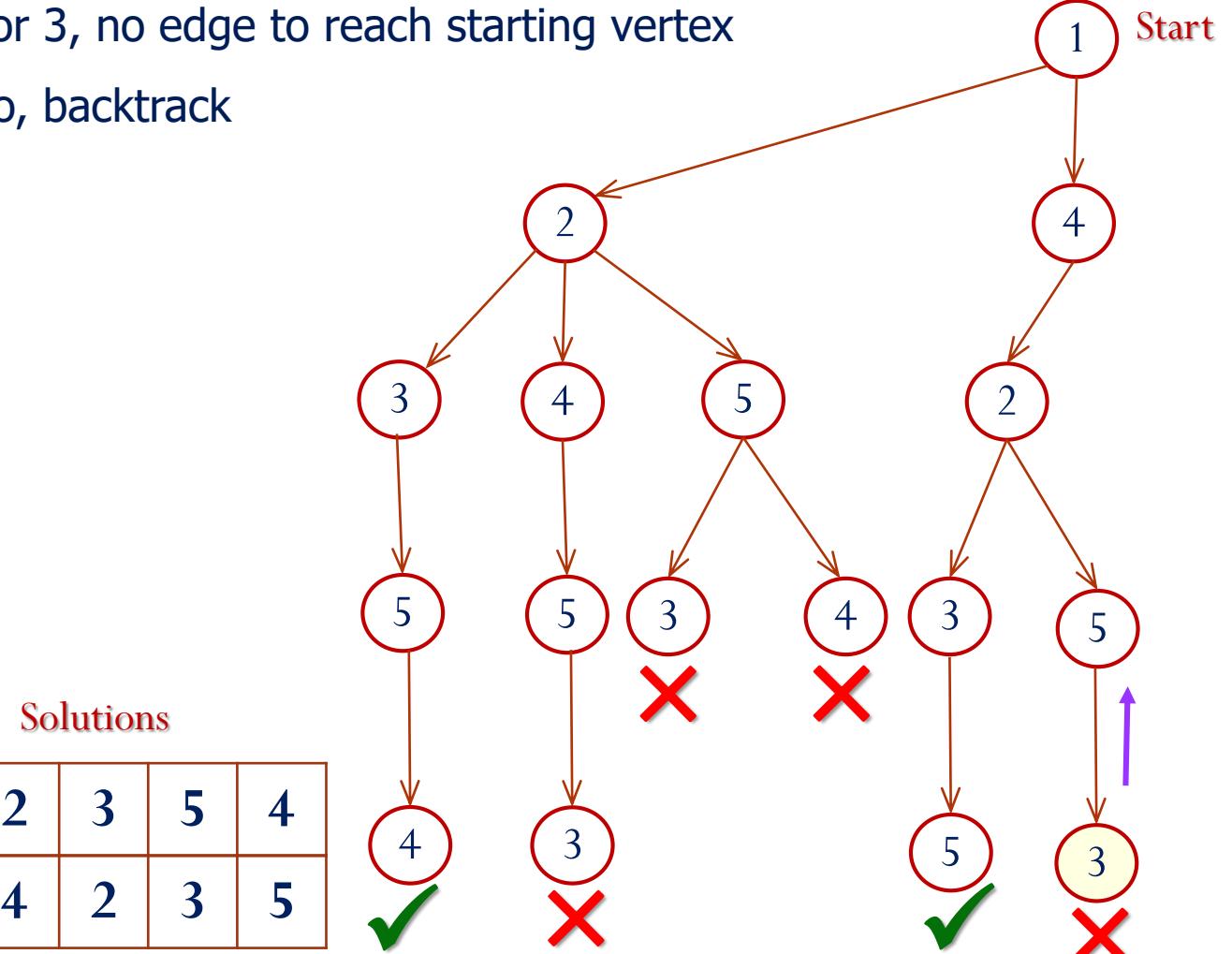


✓ For 2, next vertex is 5



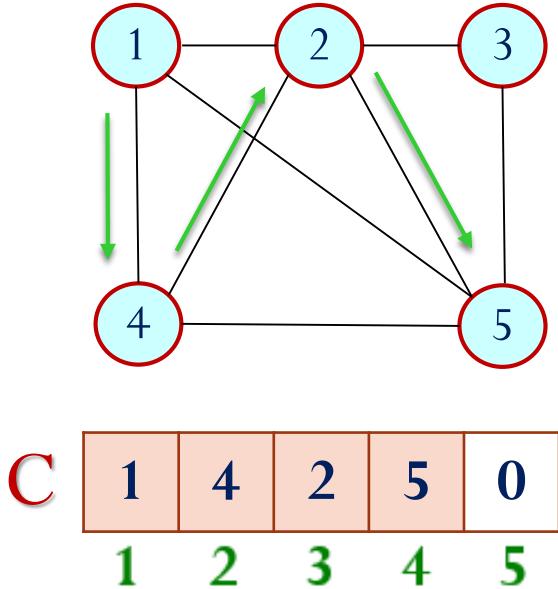
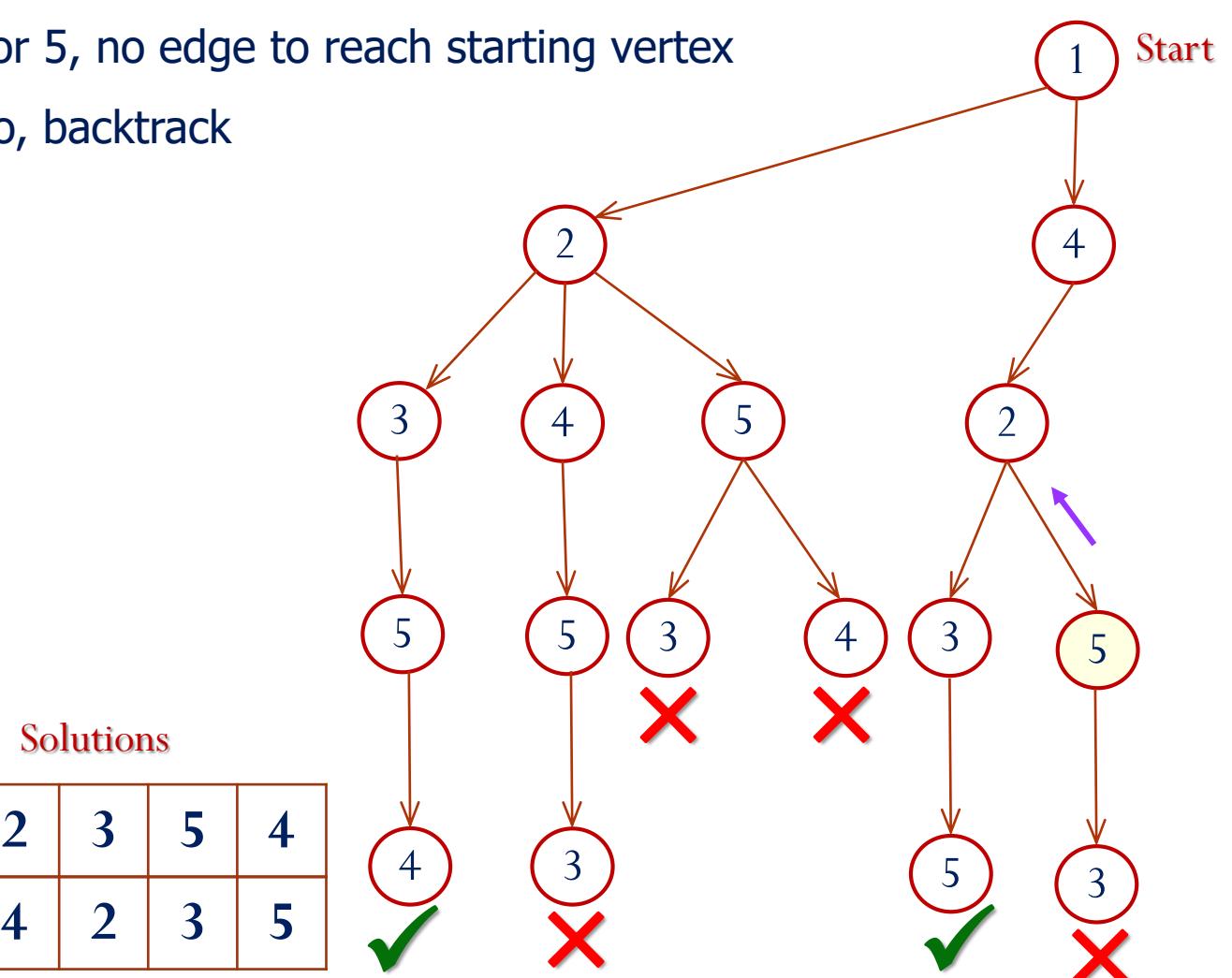
✓ For 3, no edge to reach starting vertex

✓ So, backtrack



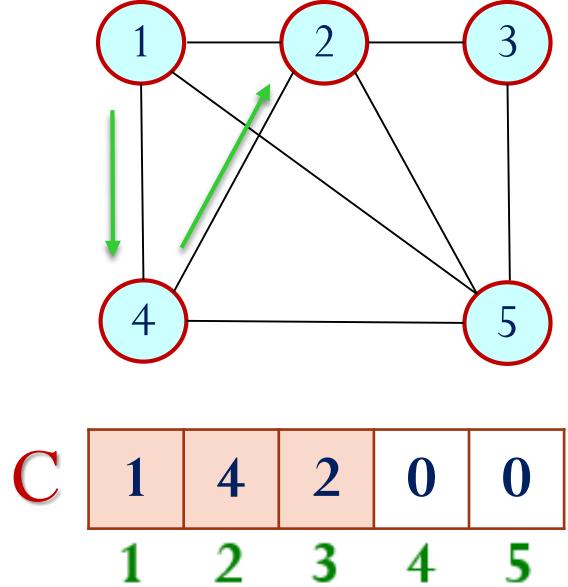
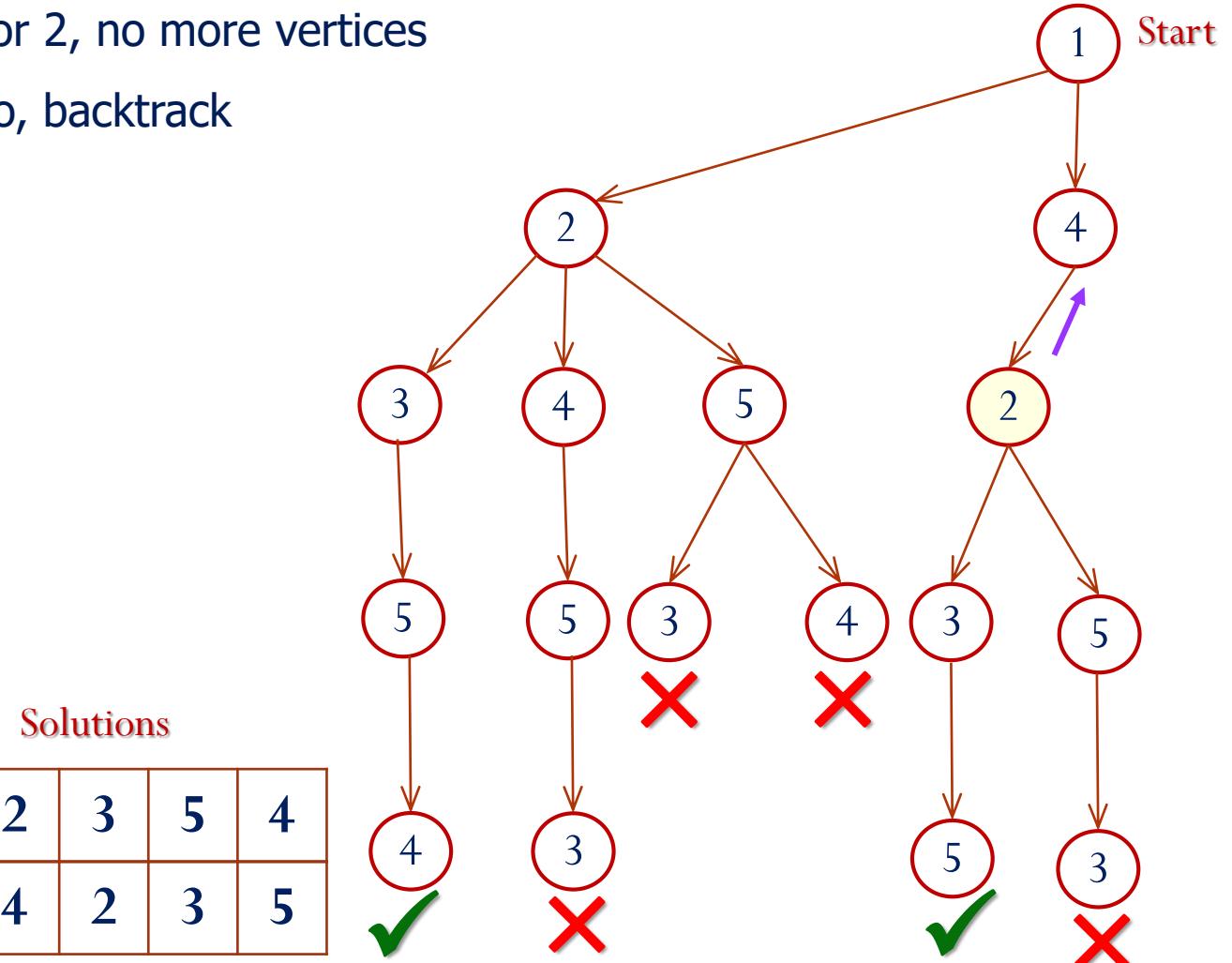
✓ For 5, no edge to reach starting vertex

✓ So, backtrack

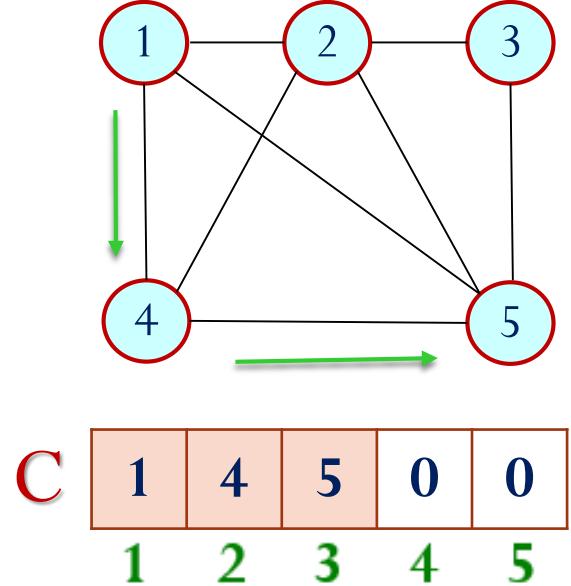
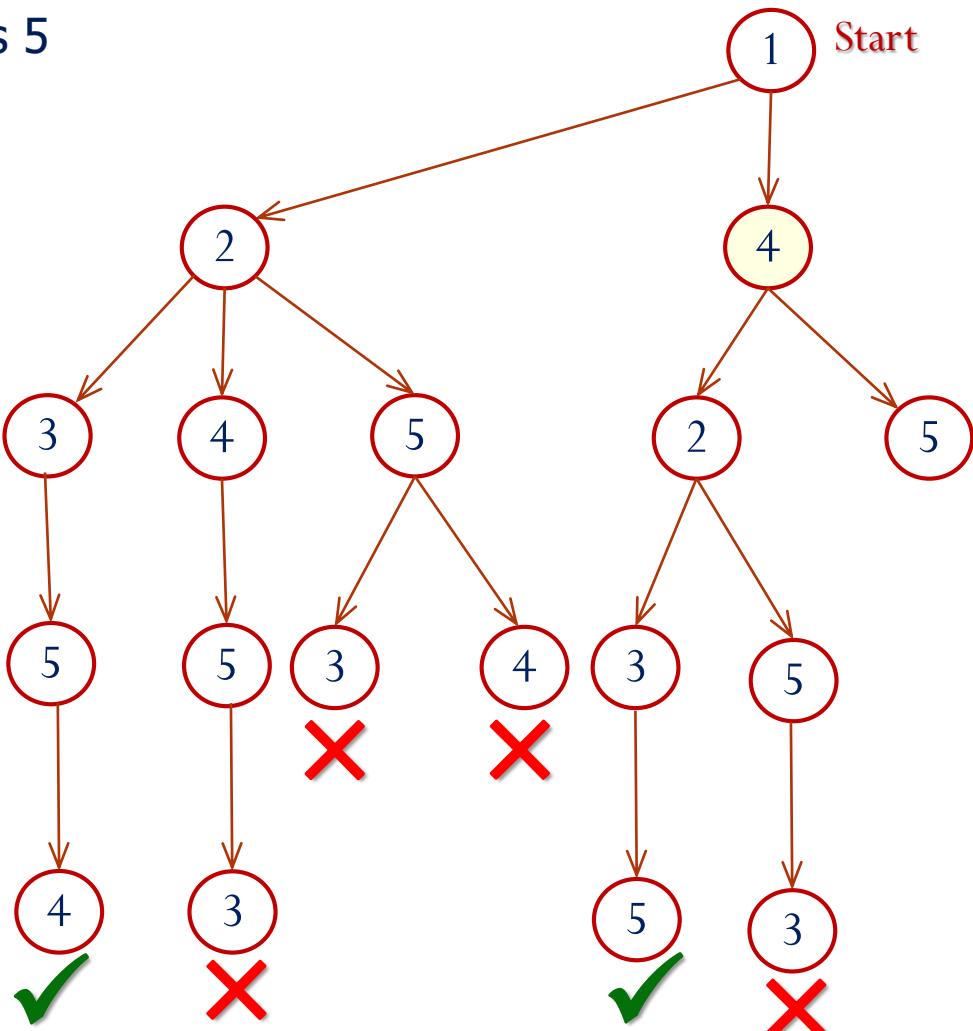


✓ For 2, no more vertices

✓ So, backtrack



✓ For 4, next vertex is 5

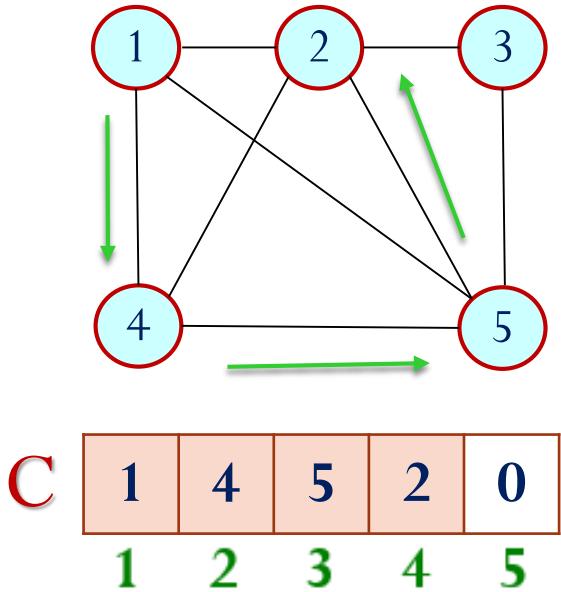
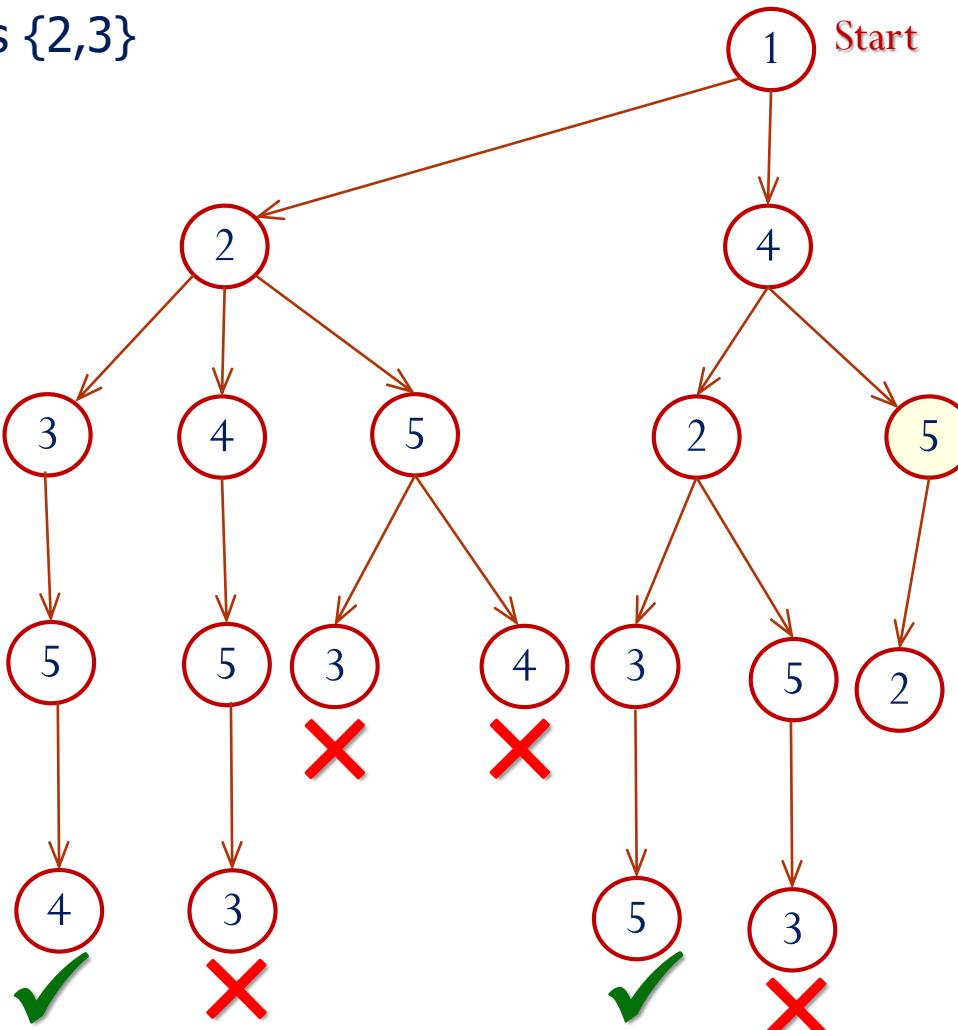


✓ For 5, next vertex is {2,3}

✓ So, proceed with 2

Solutions

1	2	3	5	4
1	4	2	3	5

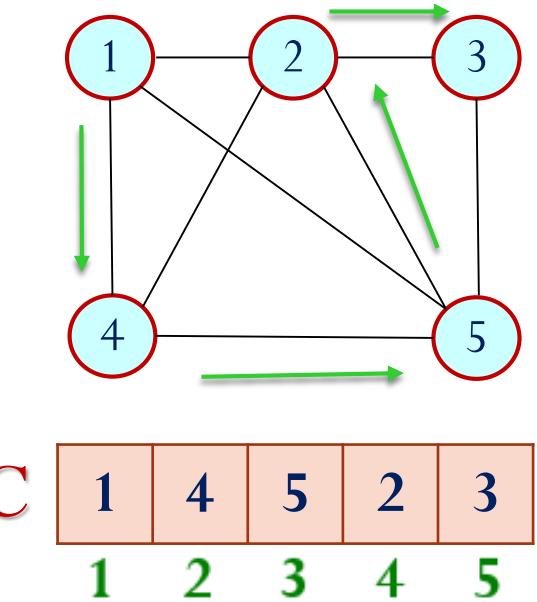
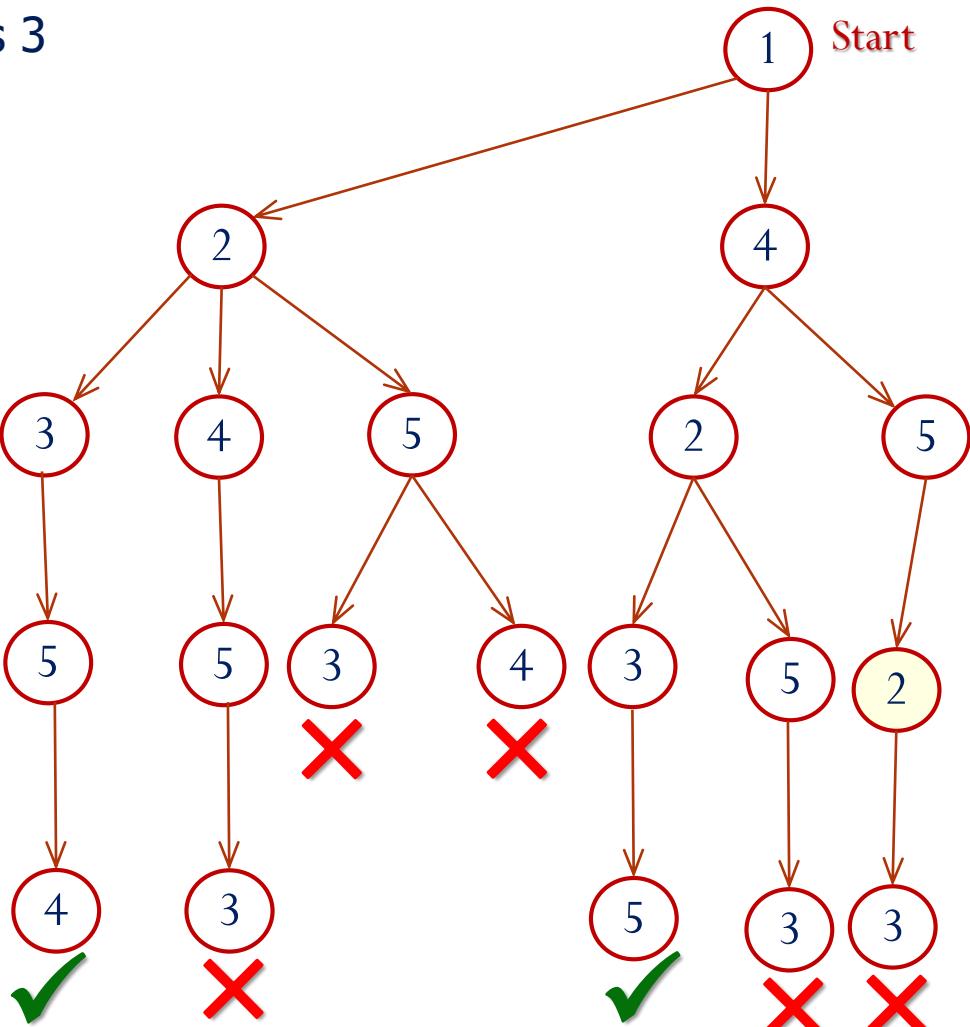


✓ For 2, next vertex is 3

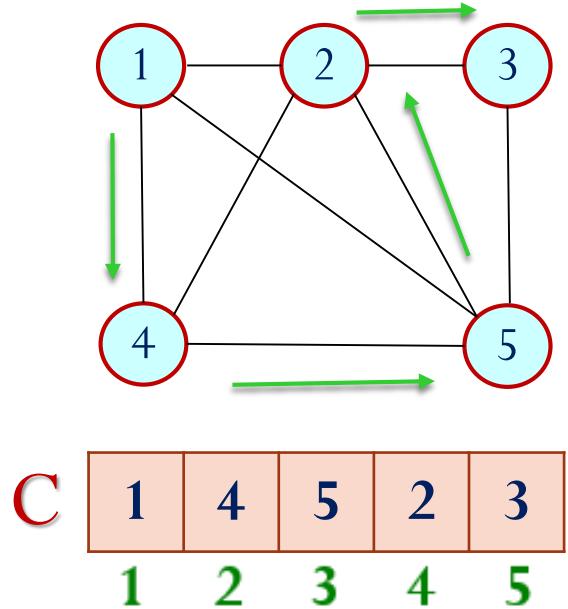
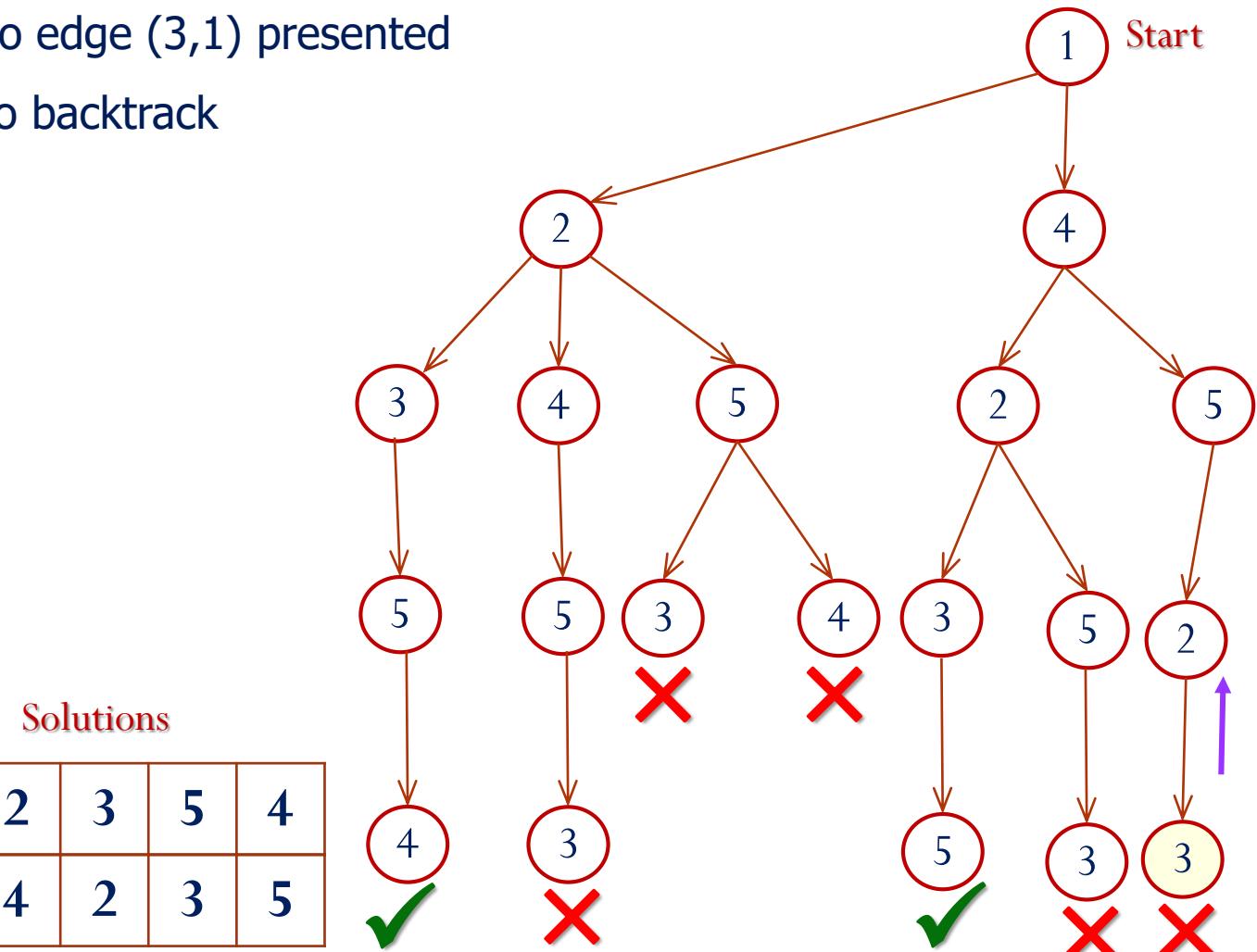
✓ So, proceed with 3

Solutions

1	2	3	5	4
1	4	2	3	5

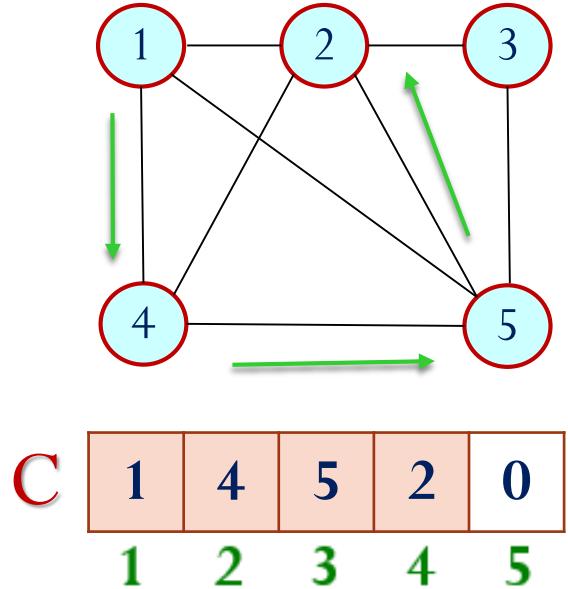
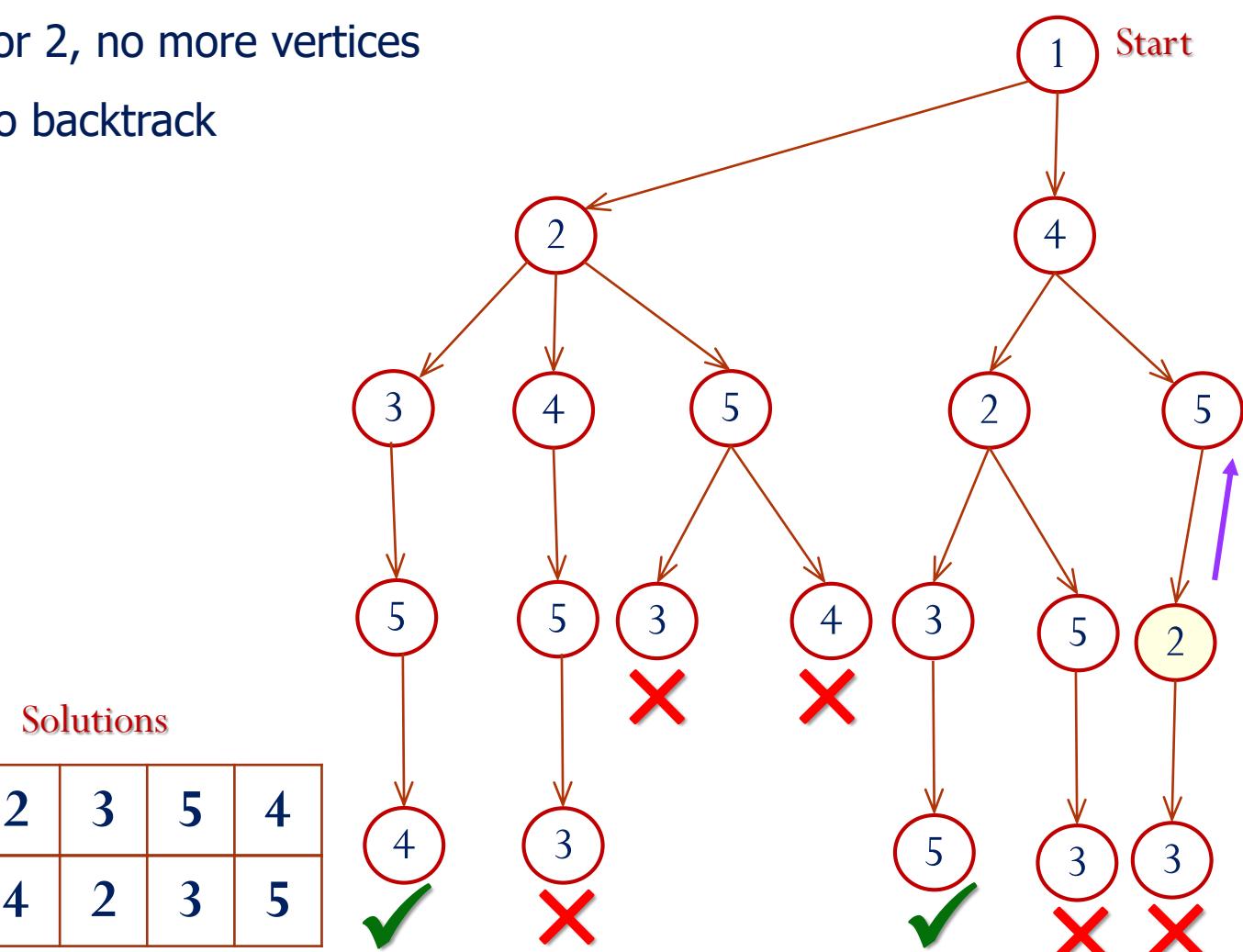


- ✓ No edge (3,1) presented
- ✓ So backtrack



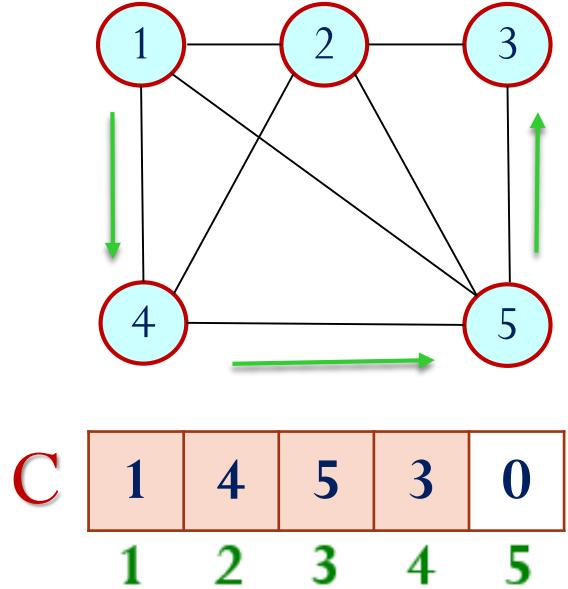
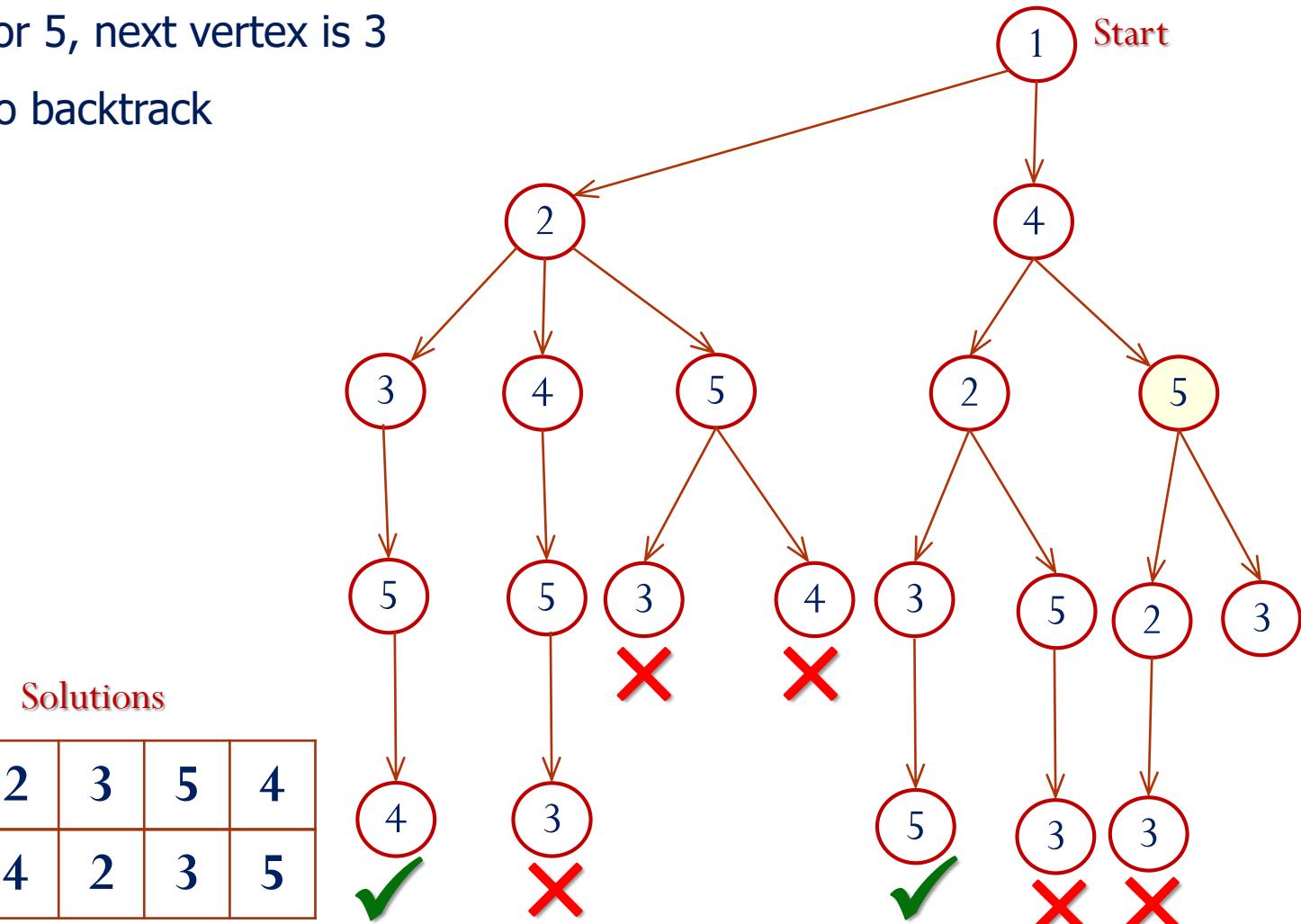
✓ For 2, no more vertices

✓ So backtrack

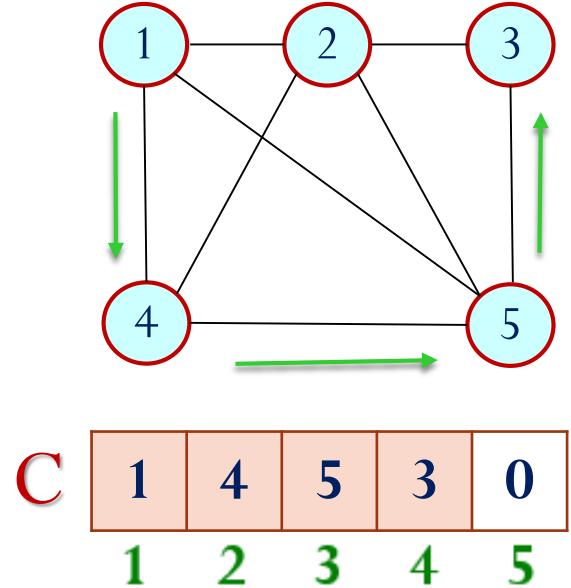
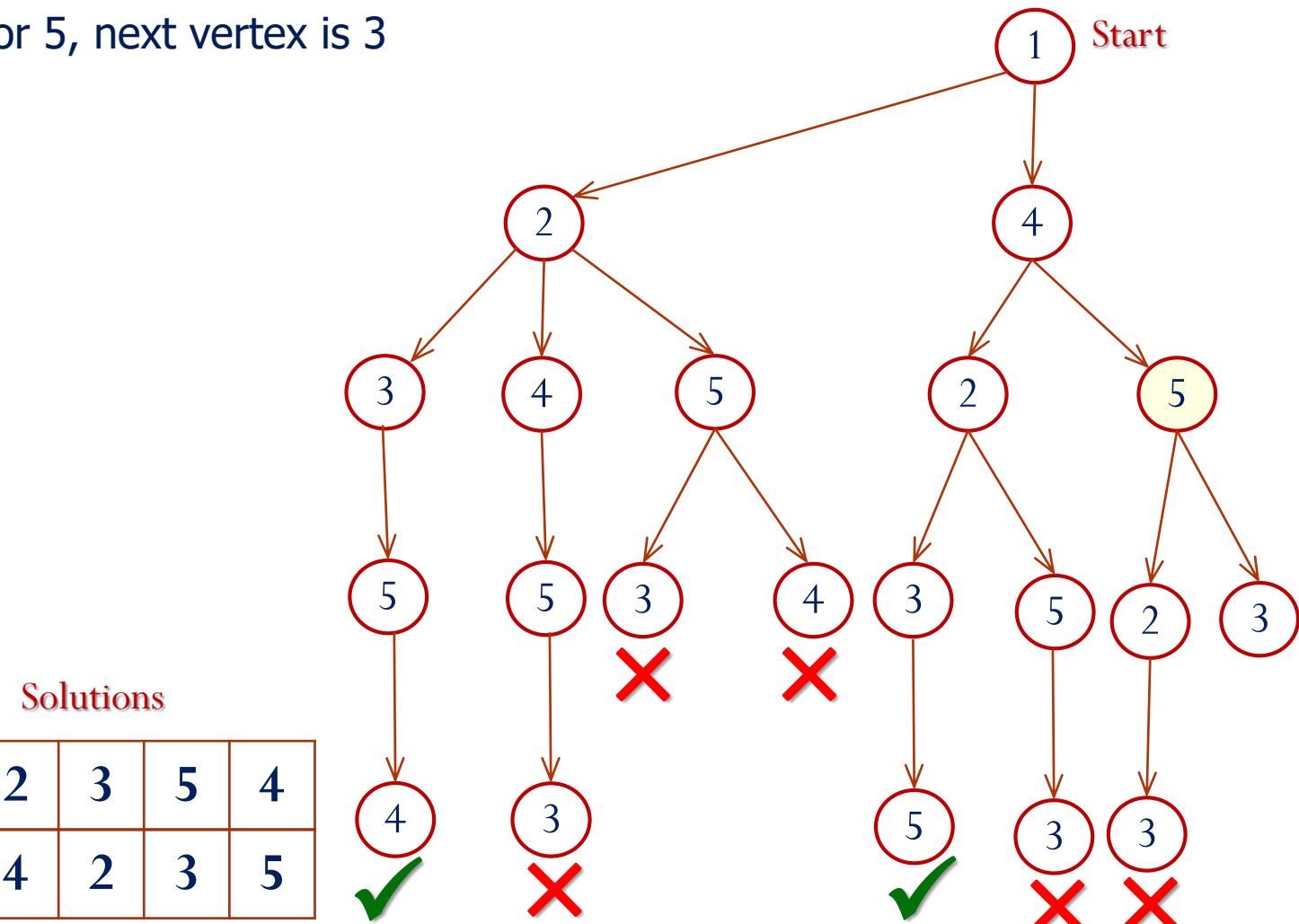


✓ For 5, next vertex is 3

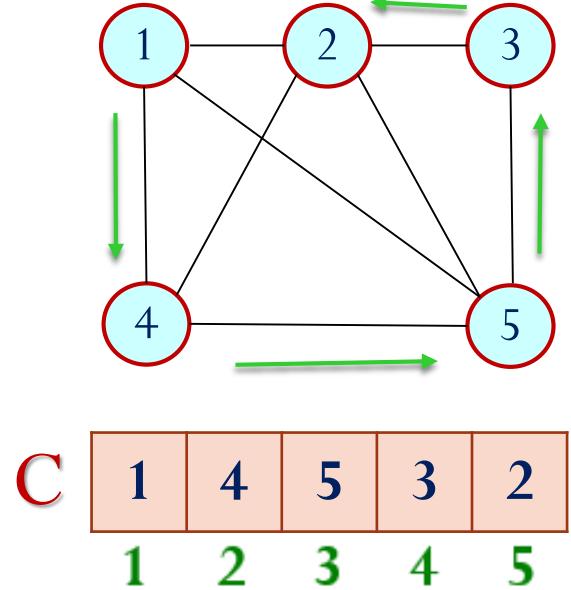
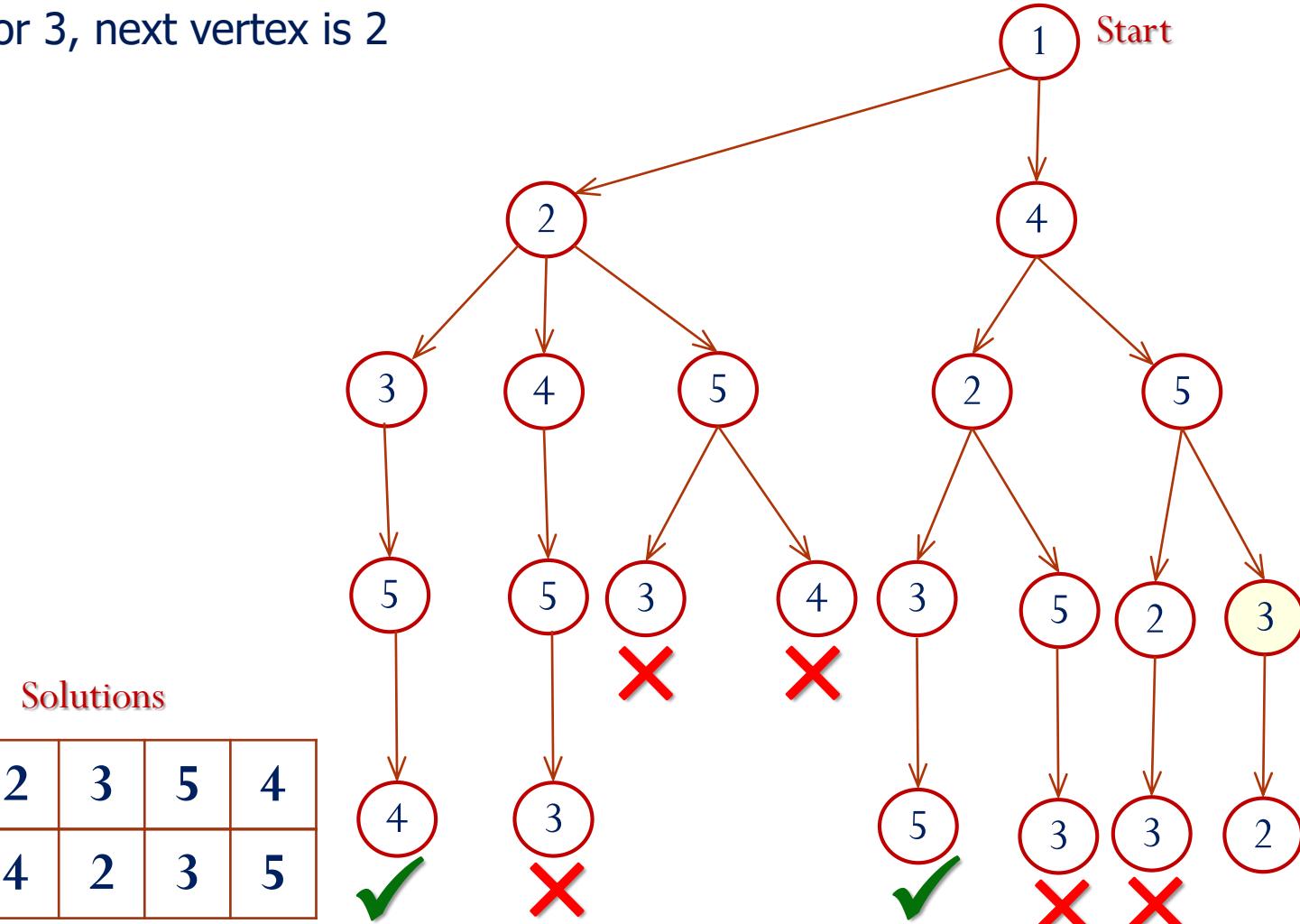
✓ So backtrack



✓ For 5, next vertex is 3



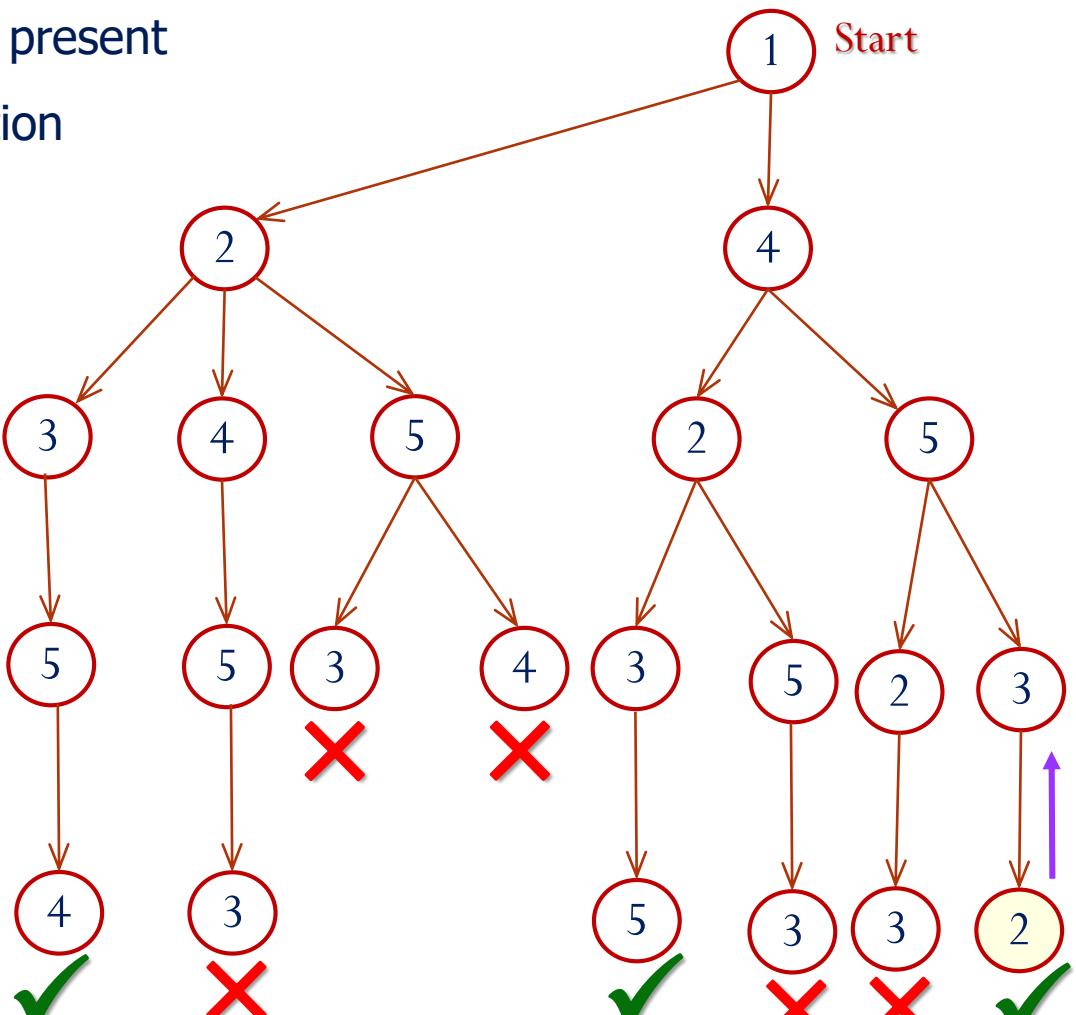
✓ For 3, next vertex is 2



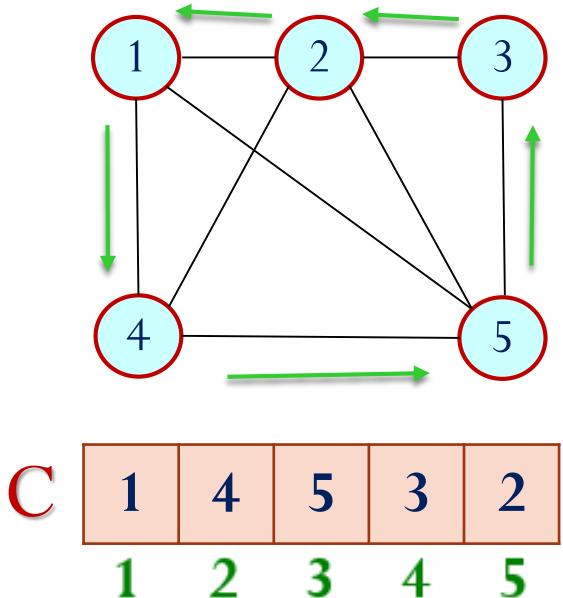
✓ For 2, (2,1) edge is present

✓ So, record the solution

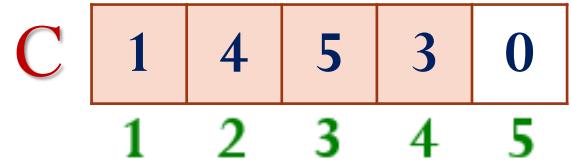
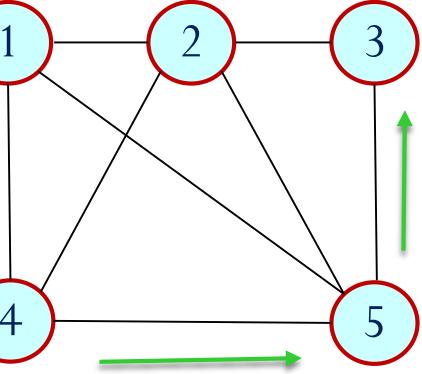
✓ And backtrack



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



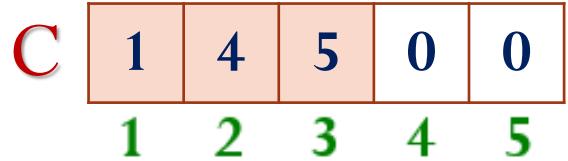
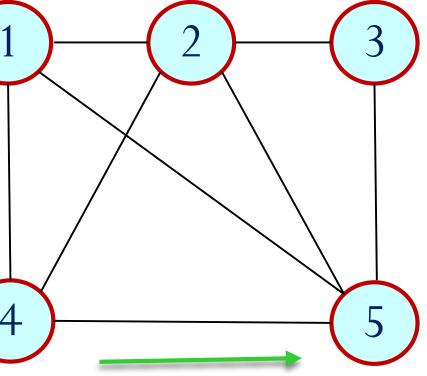
- ✓ For 3, no more choice,
- ✓ So, backtrack



Solutions

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

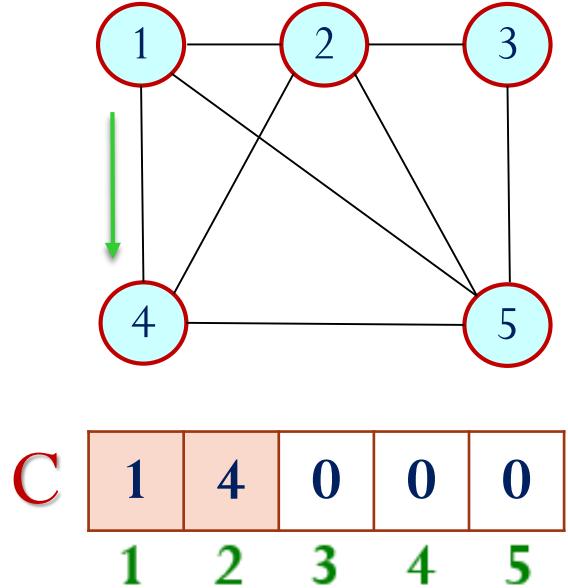
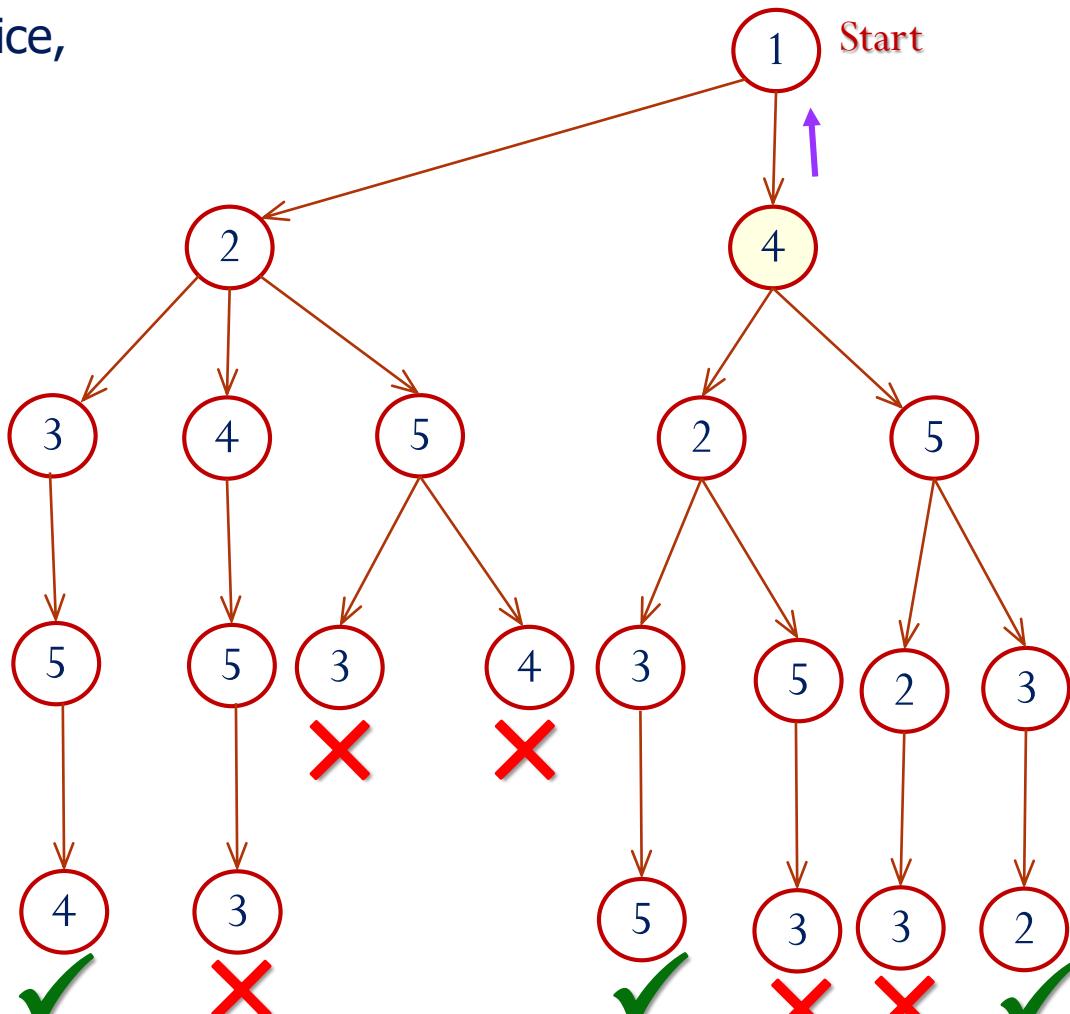
- ✓ For 5, no more choice,
- ✓ So, backtrack



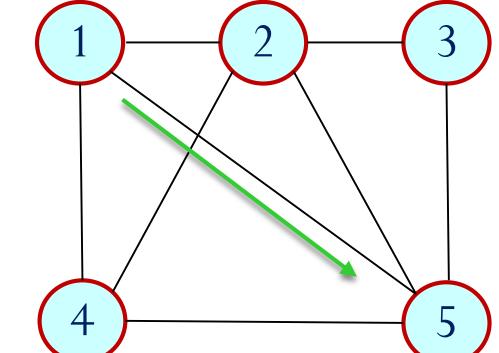
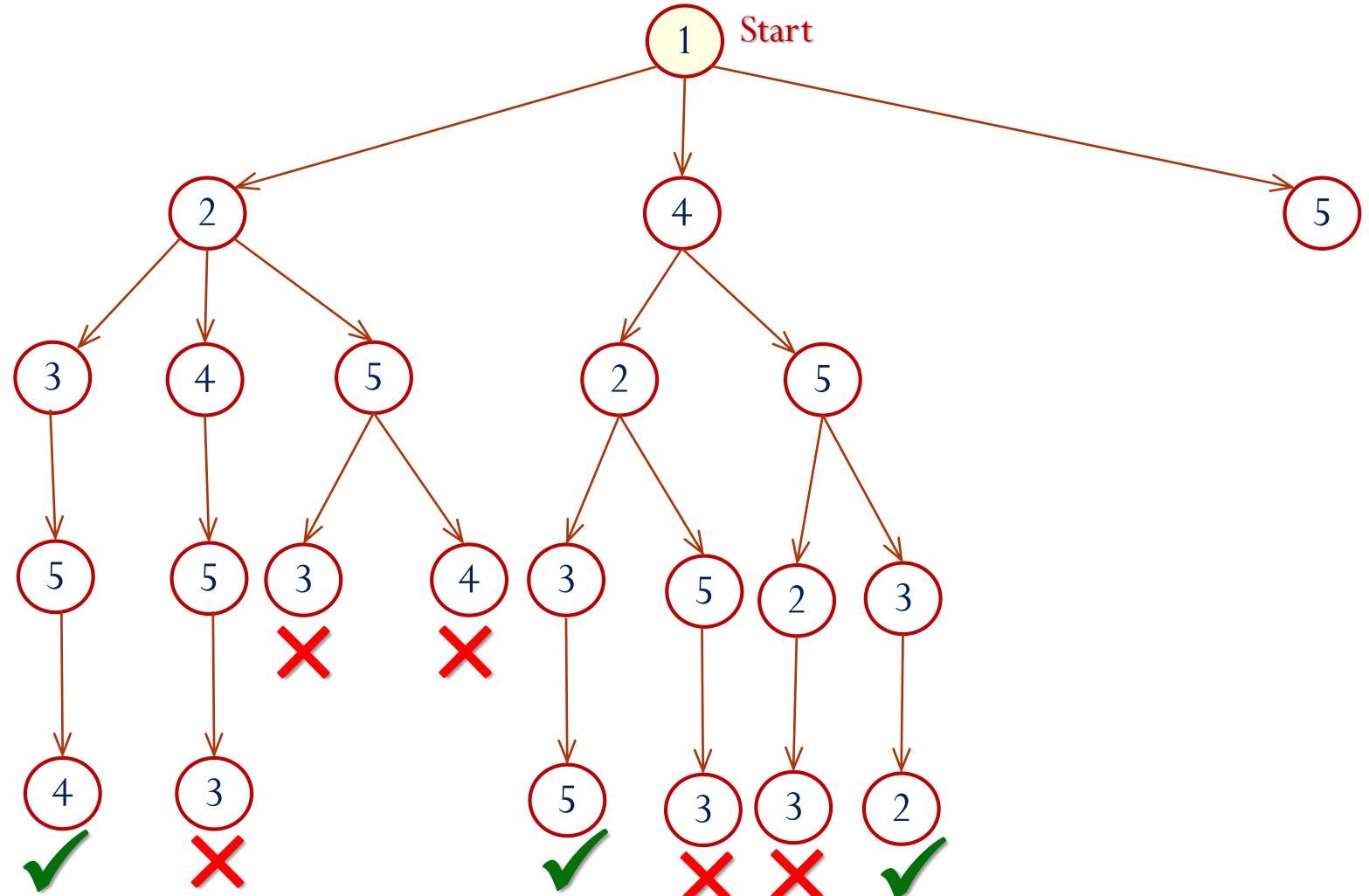
Solutions

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

- ✓ For 4, no more choice,
- ✓ So, backtrack



✓ For 1, next choice is vertex 5,



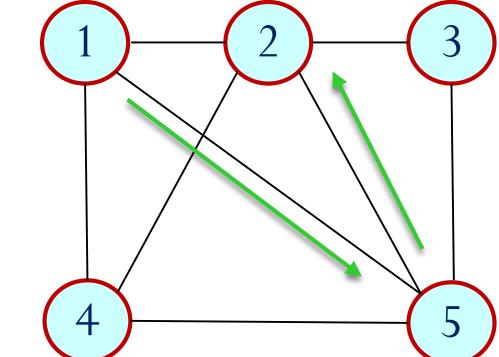
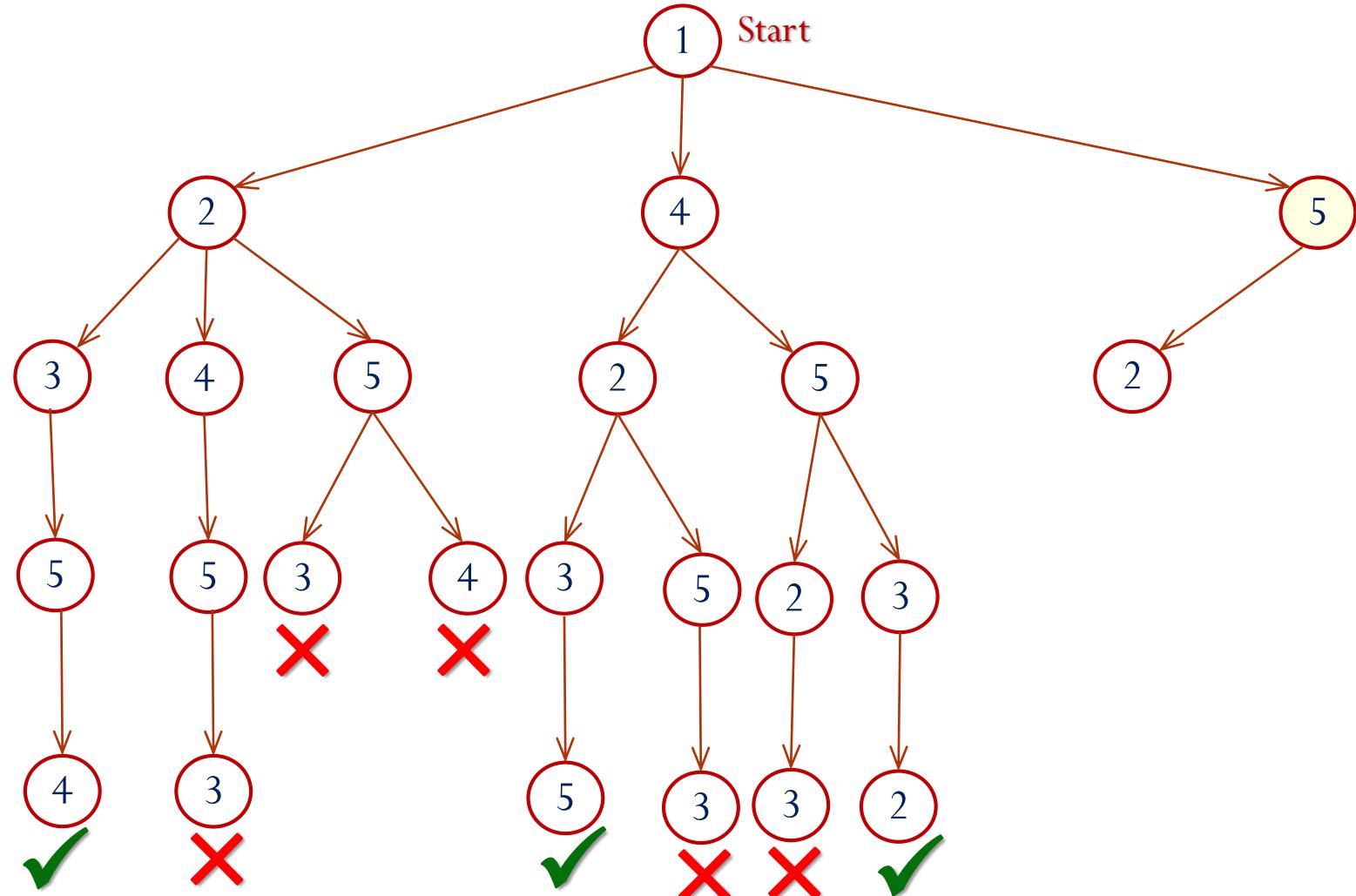
C

1	5	0	0	0
1	2	3	4	5

Solutions

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

✓ For 5, next choice is vertex 2,

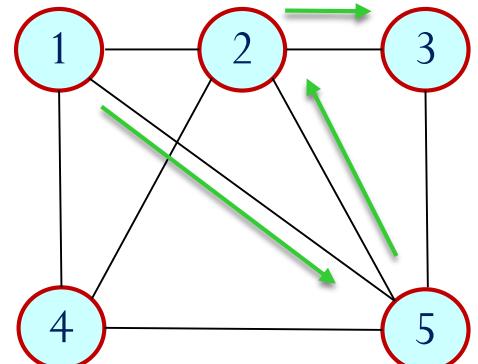
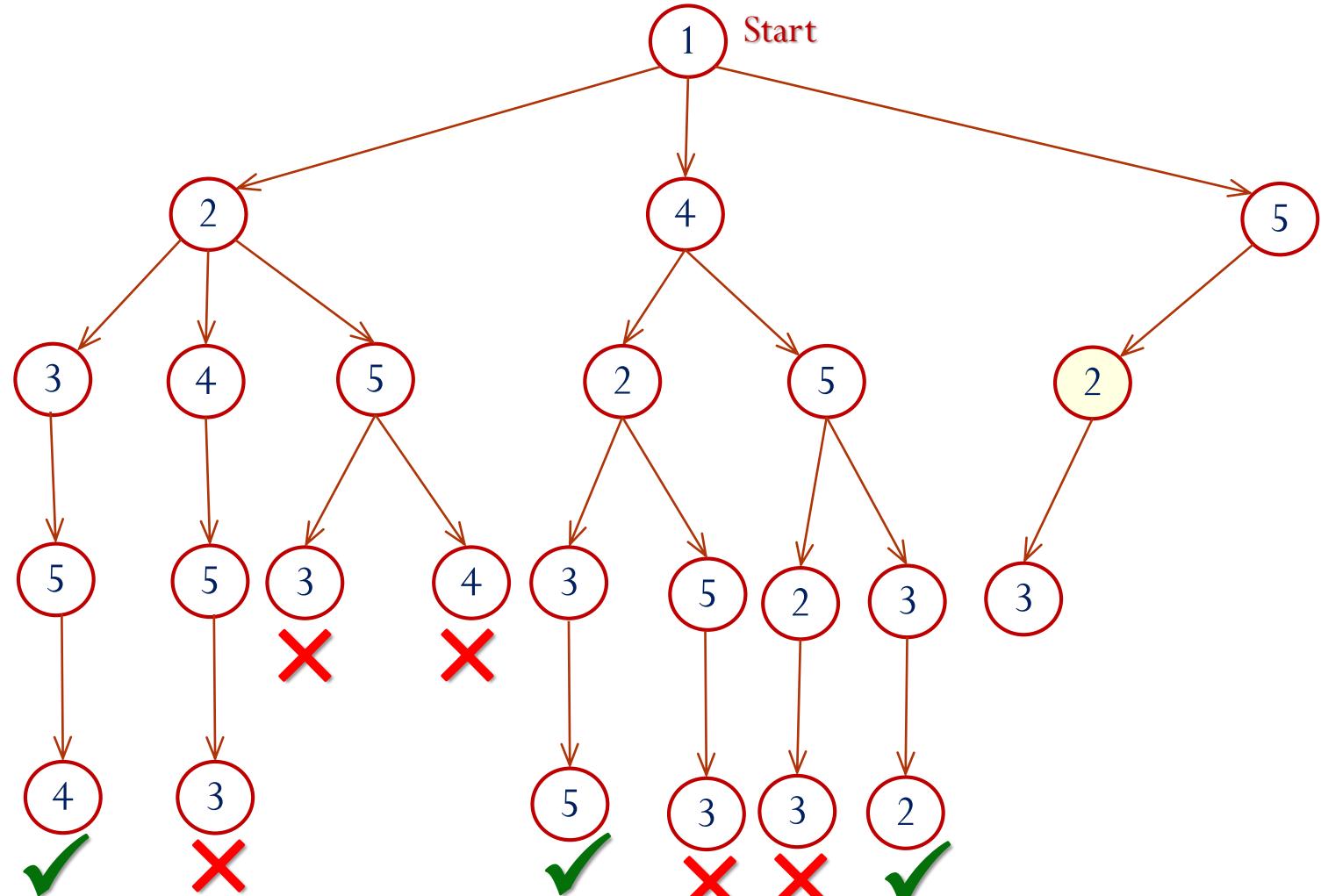


C	1	5	2	0	0
	1	2	3	4	5

Solutions

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

- ✓ For 2, next choice is vertex 3,

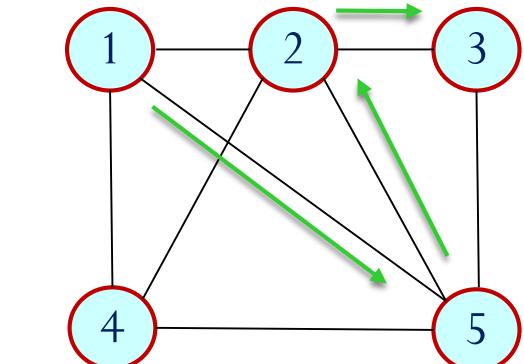
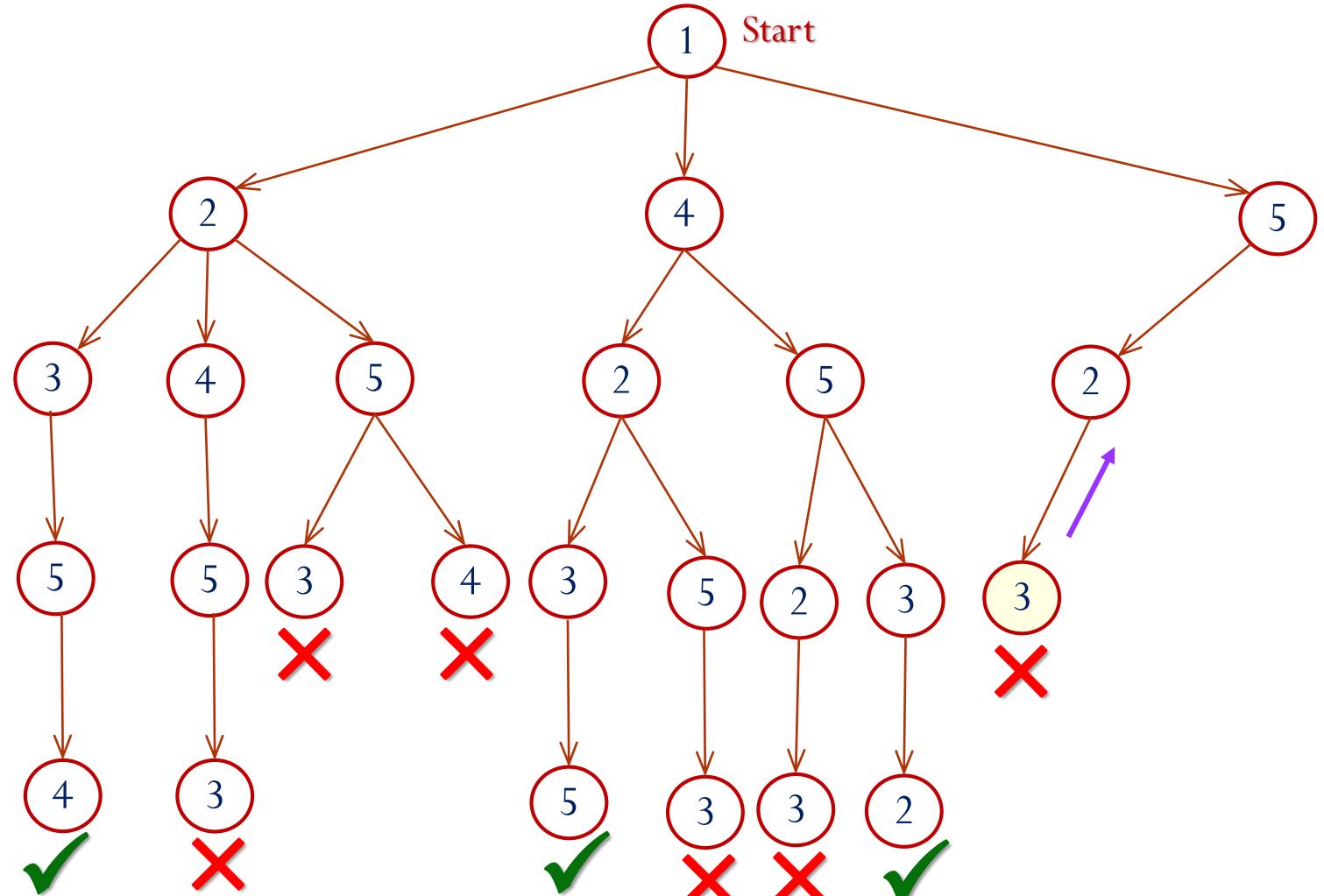


C	1	5	2	3	0
	1	2	3	4	5

## Solutions

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

✓ For 3, no choice, so, backtrack

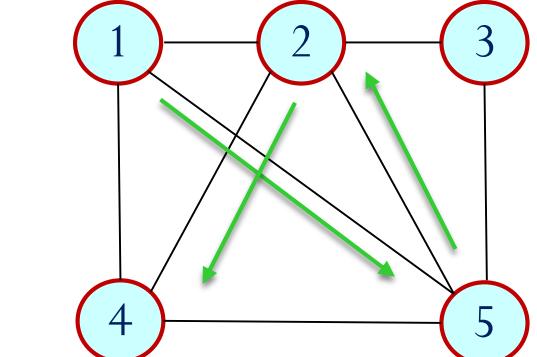
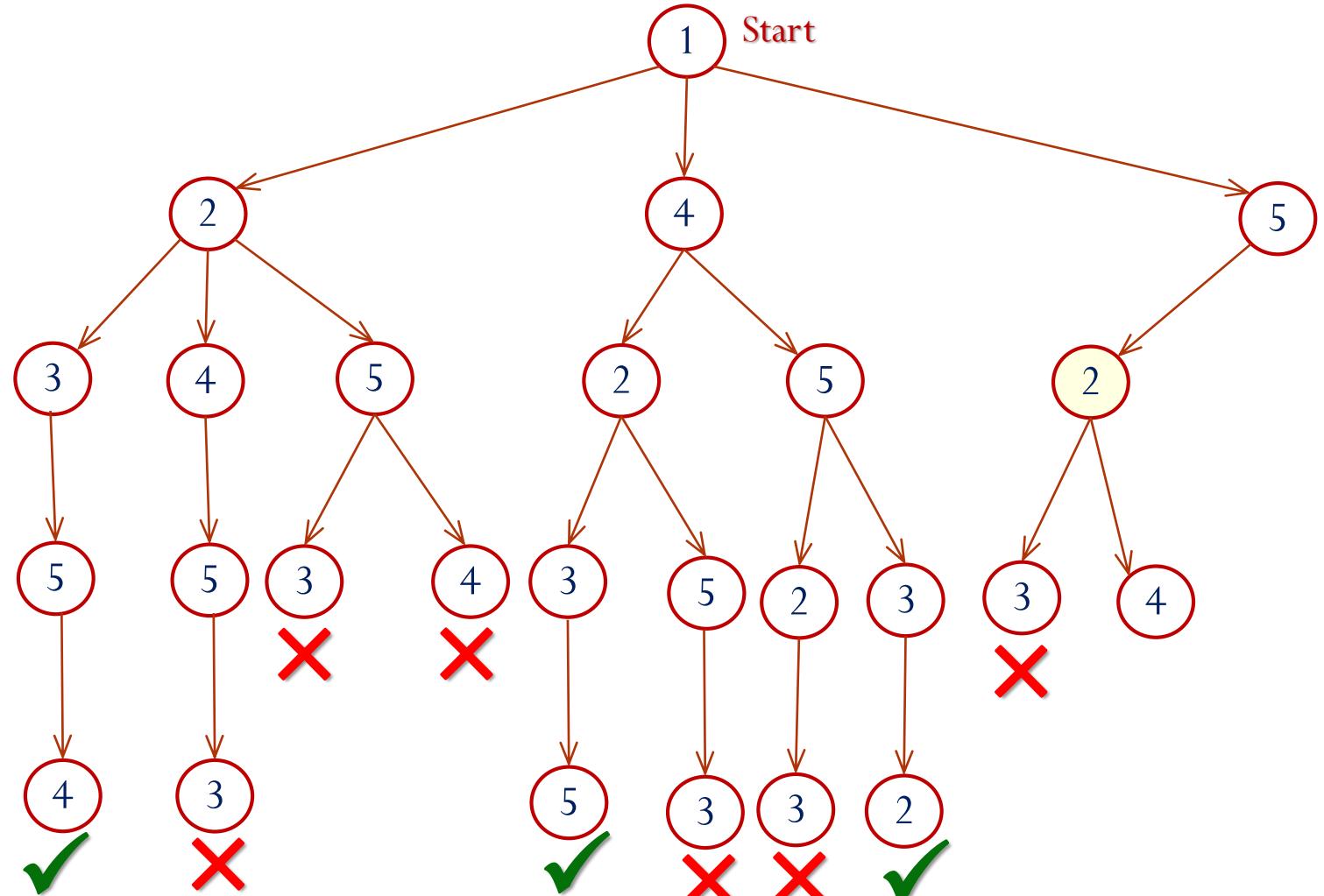


C	1	5	2	3	0
	1	2	3	4	5

Solutions

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

✓ For 2, next choice is vertex 4

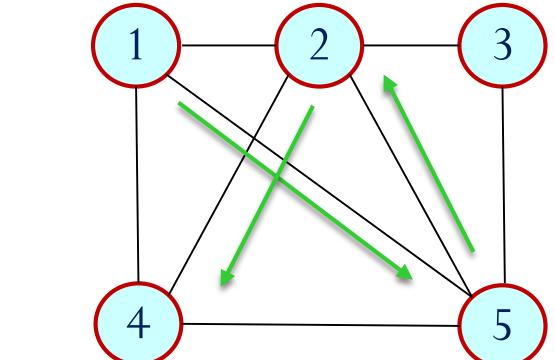
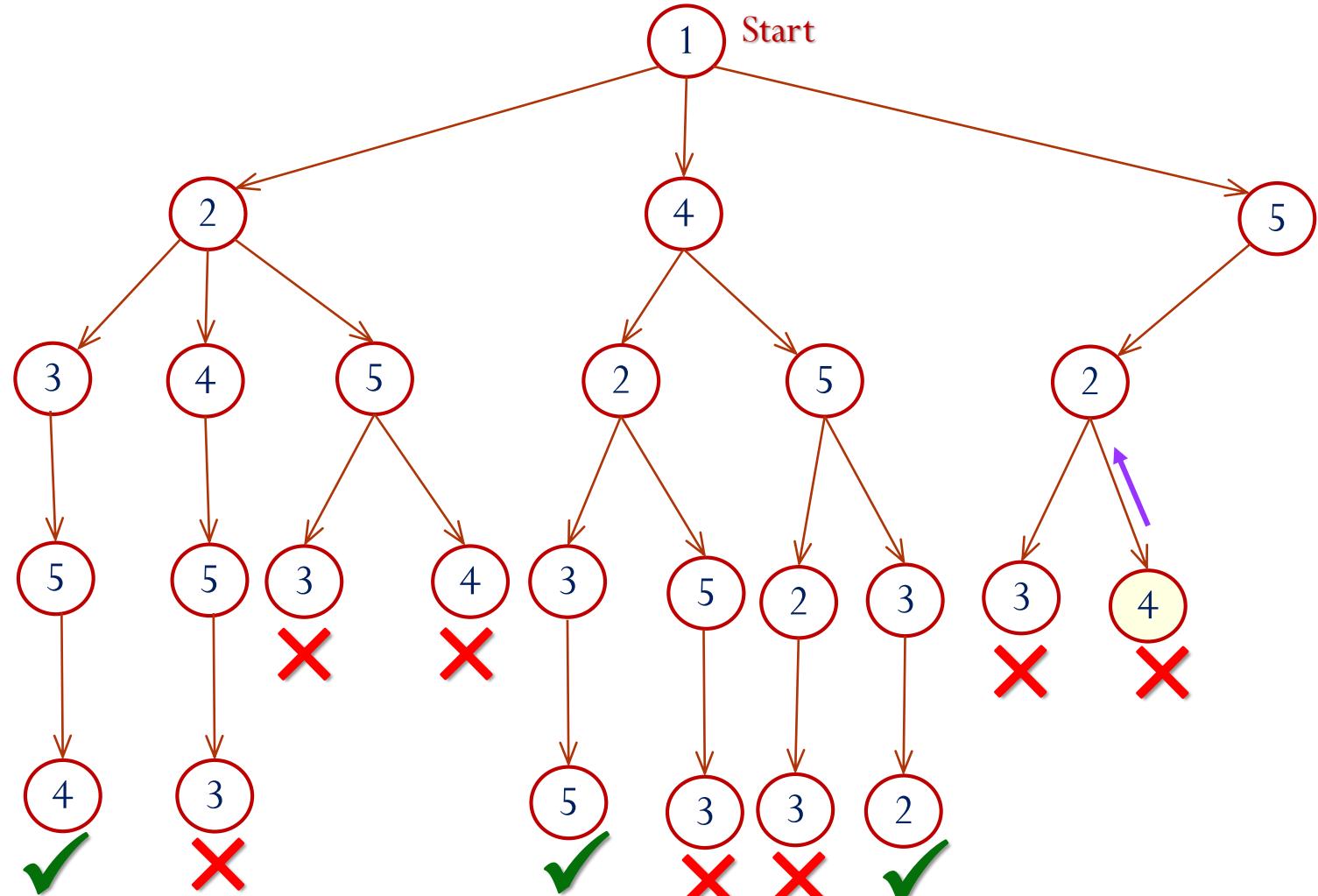


C	1	5	2	4	0
	1	2	3	4	5

Solutions

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

✓ For 4, no choice, so, backtrack

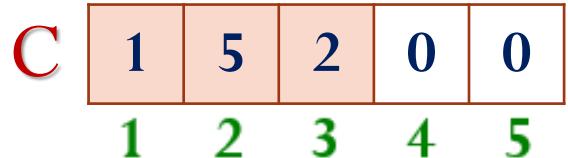
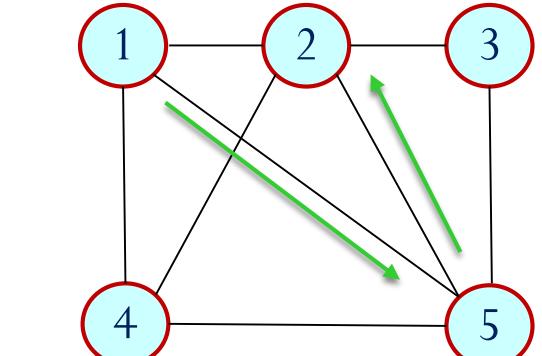
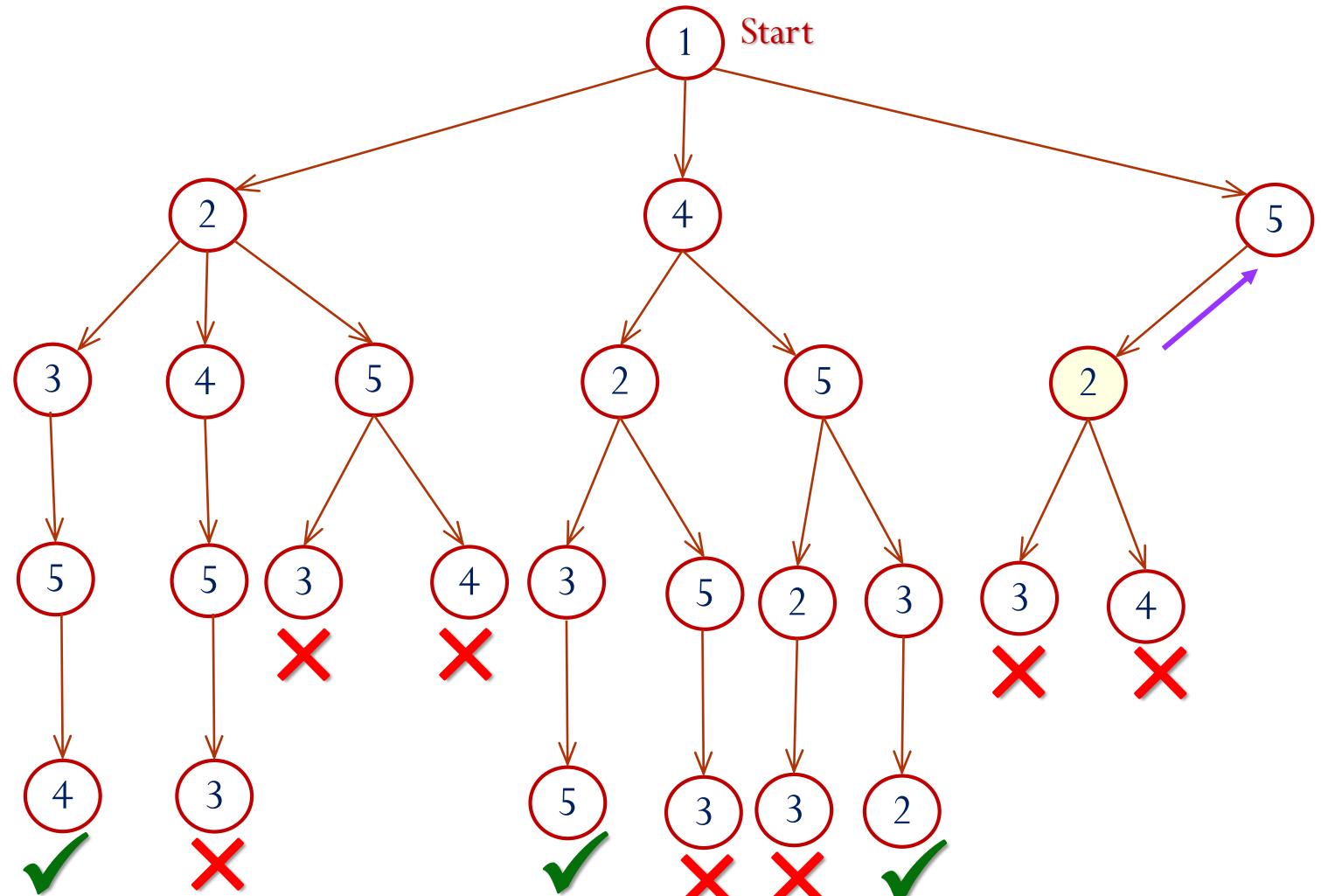


C	1	5	2	4	0
	1	2	3	4	5

Solutions

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

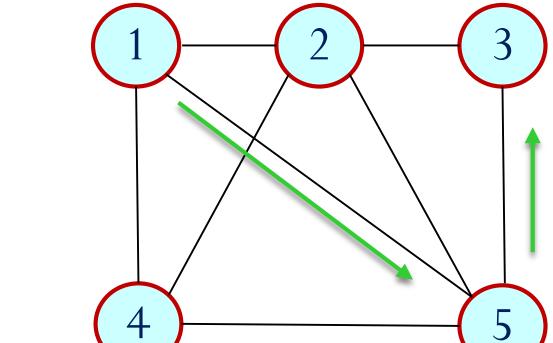
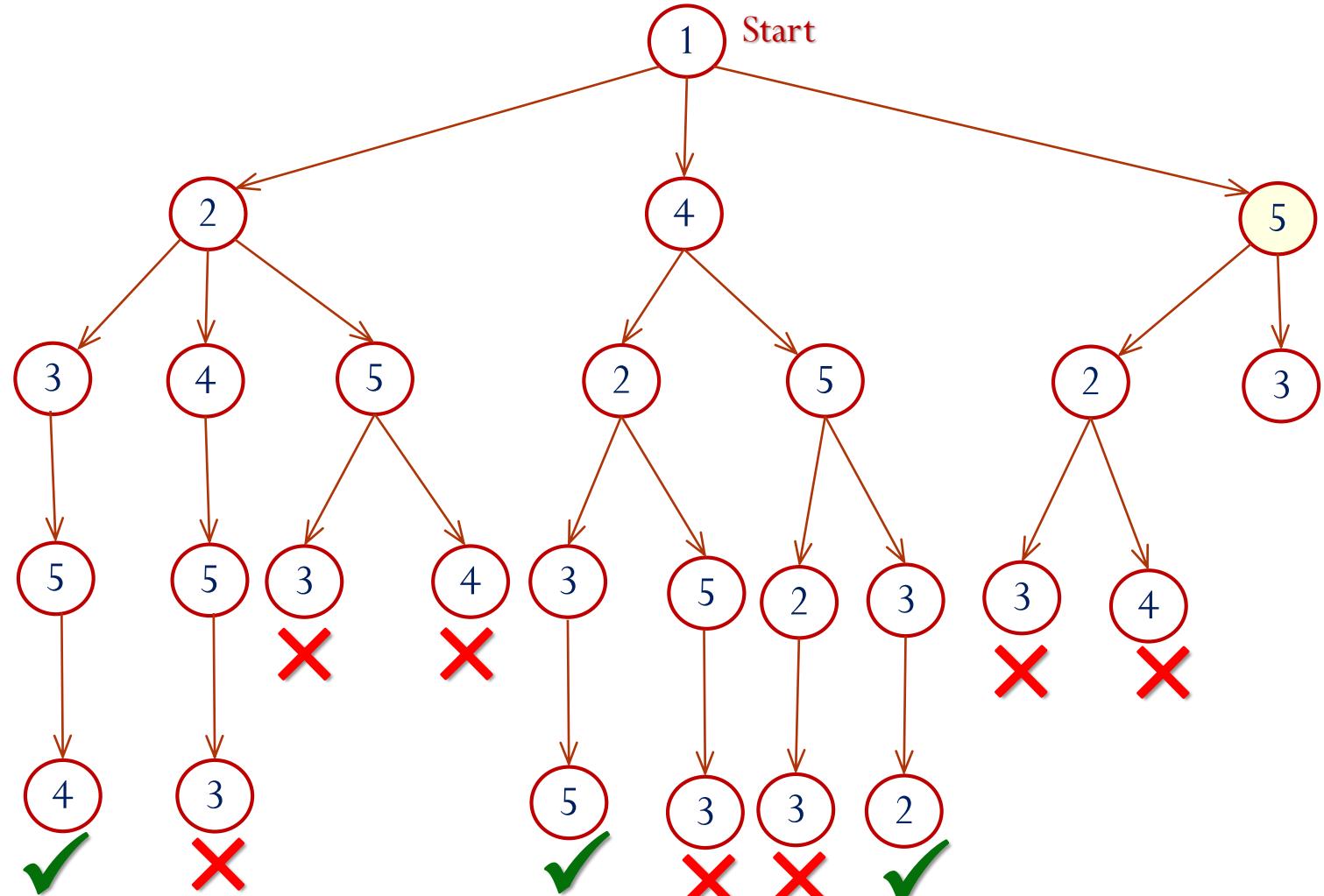
✓ For 2, no choice, so, backtrack



Solutions

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

- ✓ For 5, next possibility is vertex 3

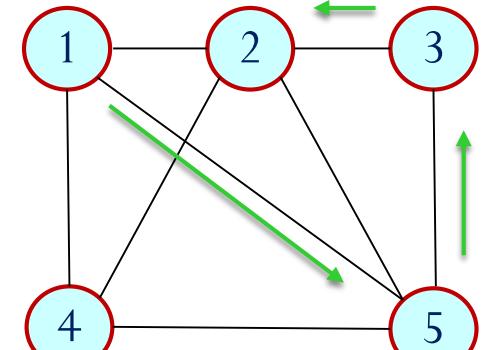
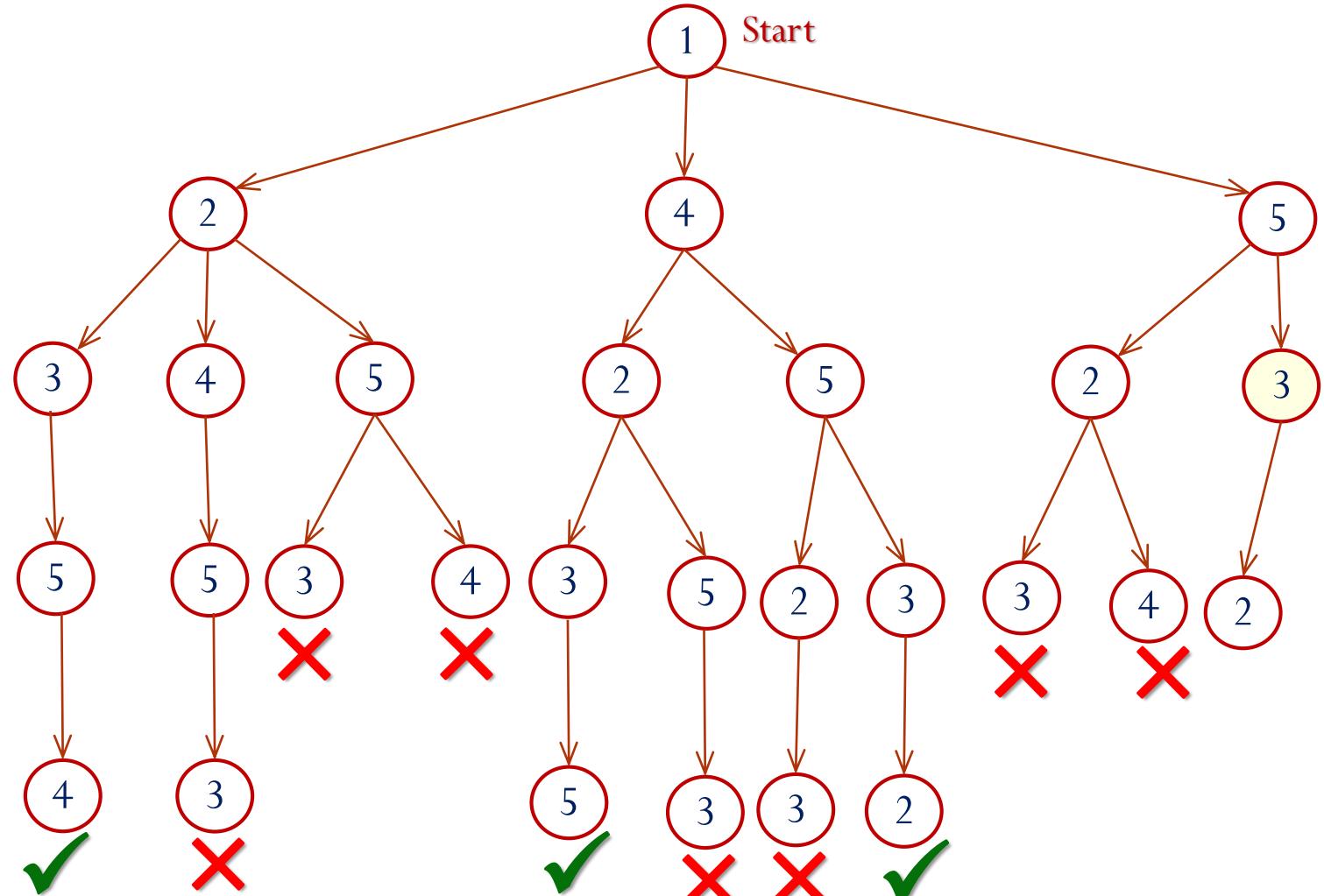


C	1	5	3	0	0
	1	2	3	4	5

Solutions

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

- ✓ For 3 next possibility is vertex 2

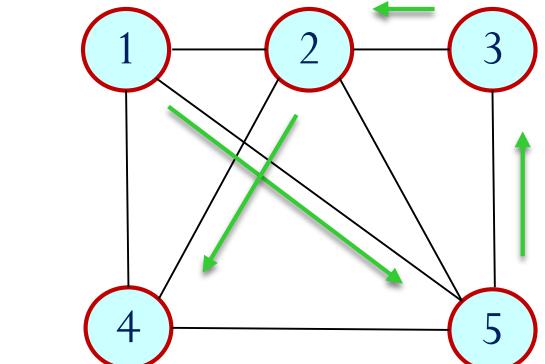
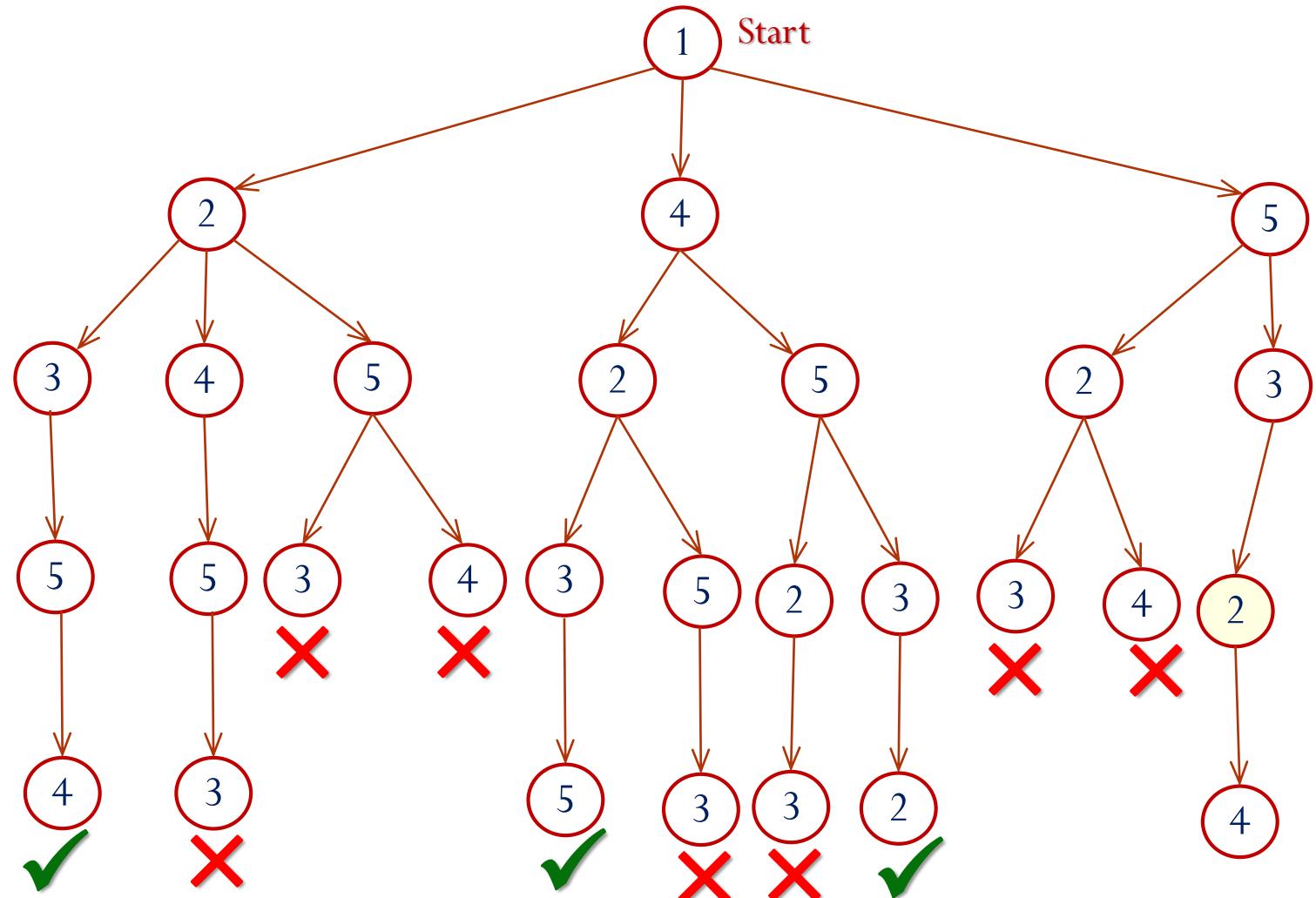


C	1	5	3	2	0
	1	2	3	4	5

## Solutions

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

✓ For 2, next possibility is vertex 4



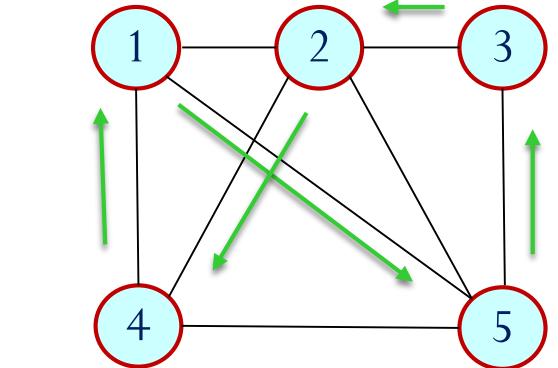
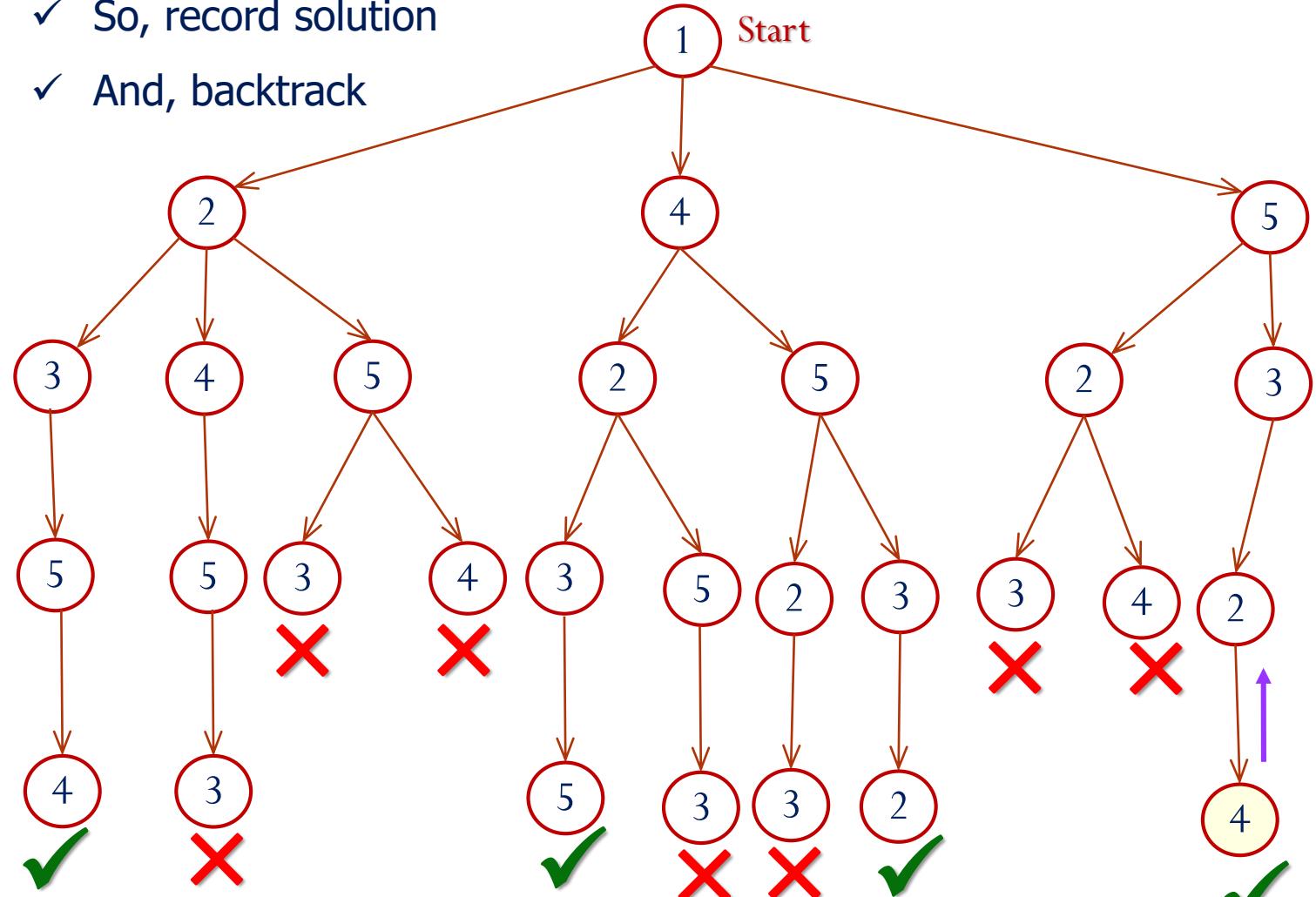
C

1	5	3	2	4
1	2	3	4	5

Solutions

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

- ✓ For 4, edge (4,1) is present
  - ✓ So, record solution
  - ✓ And, backtrack

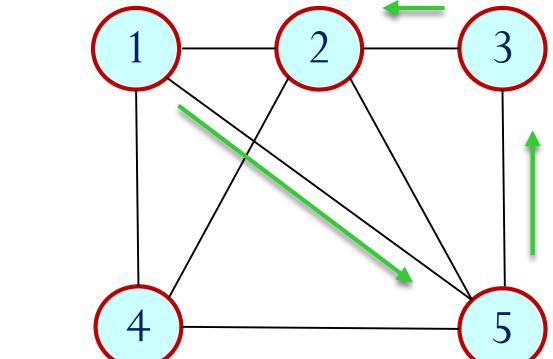
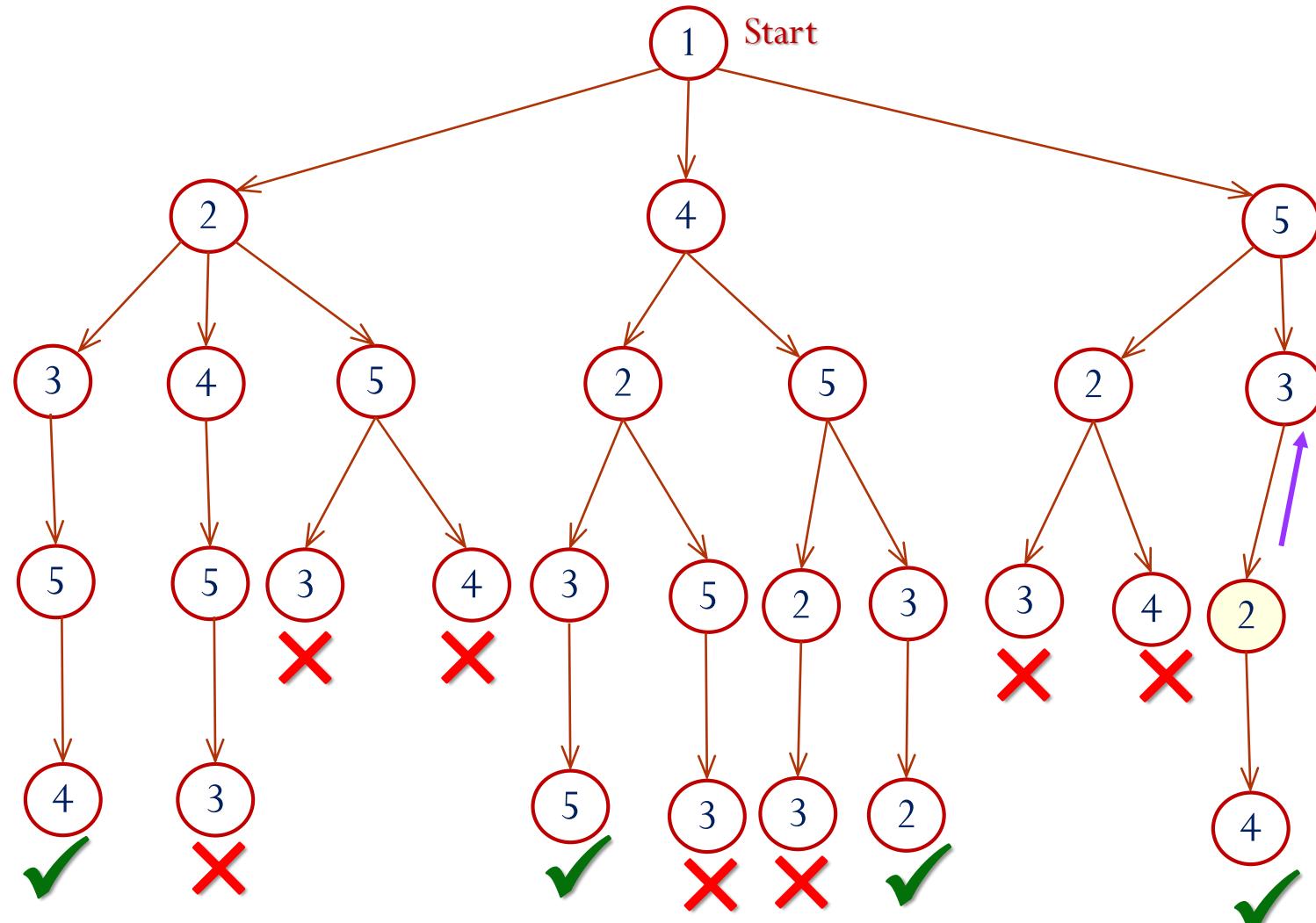


C	1	5	3	2	4
	1	2	3	4	5

## Solutions

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

✓ For 2, no choice, backtrack

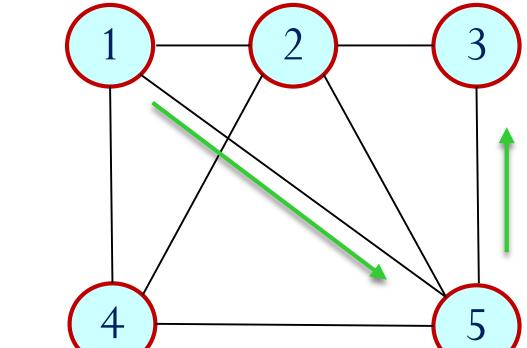
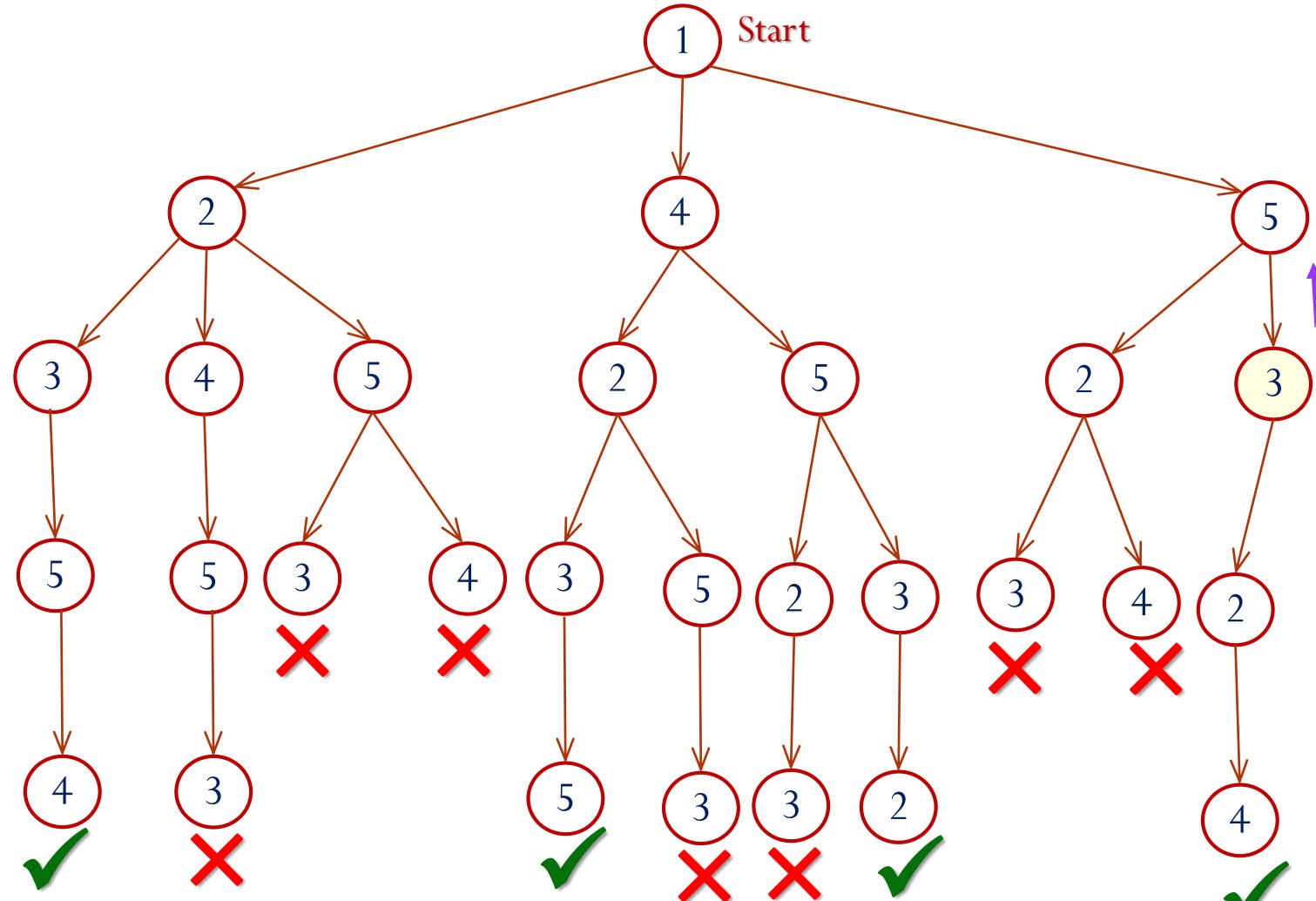


C	1	5	3	2	0
	1	2	3	4	5

Solutions

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

- ✓ For 3, no choice, backtrack

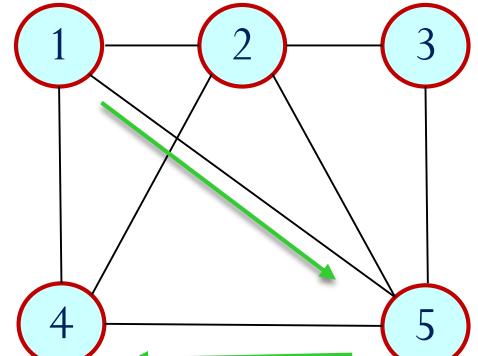
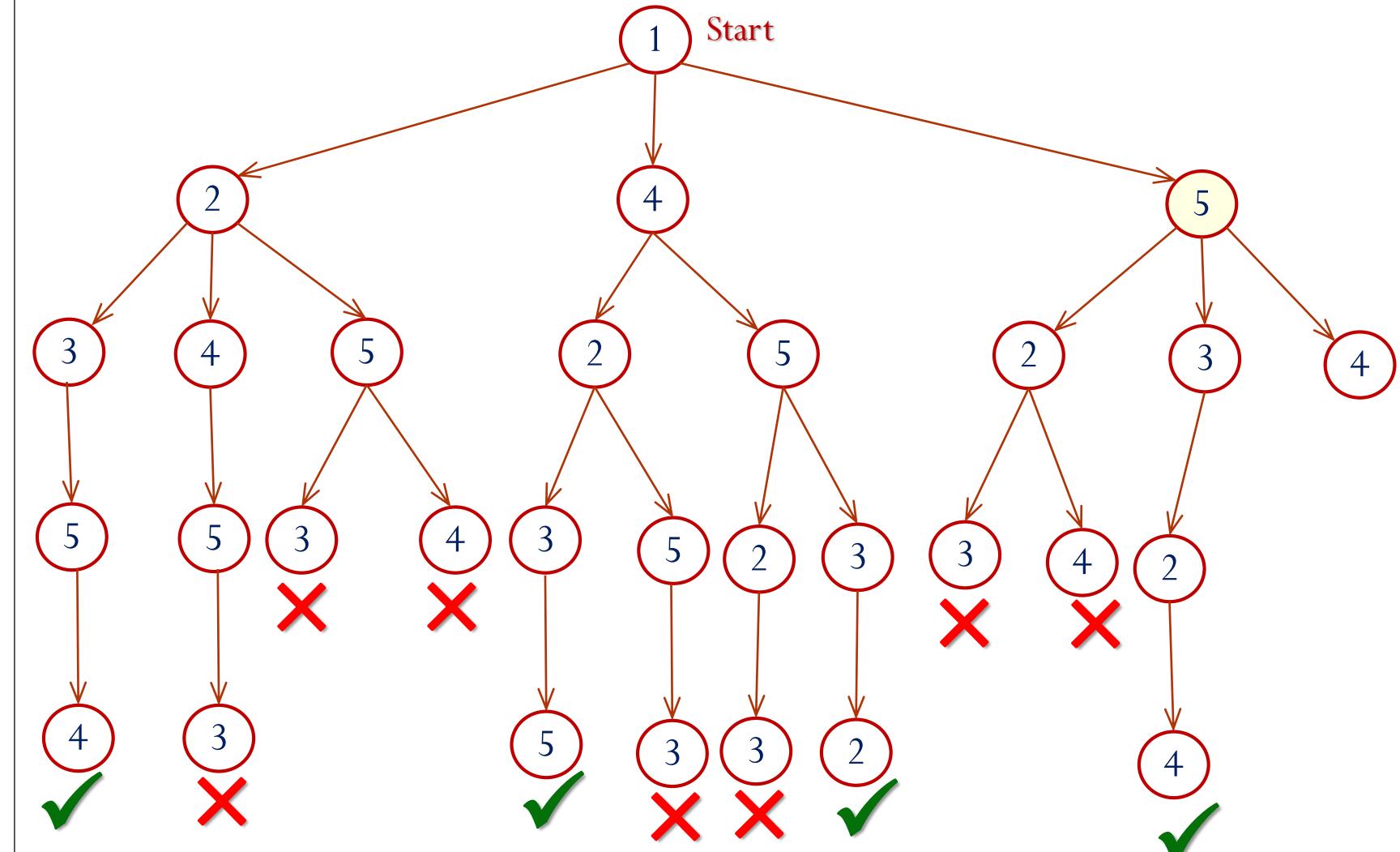


C	1	5	3	0	0
	1	2	3	4	5

## Solutions

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

- ✓ For 5, next choice is 4

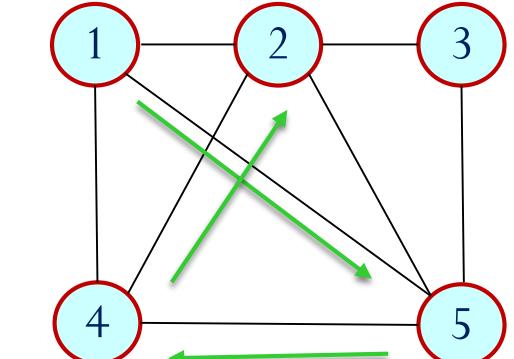
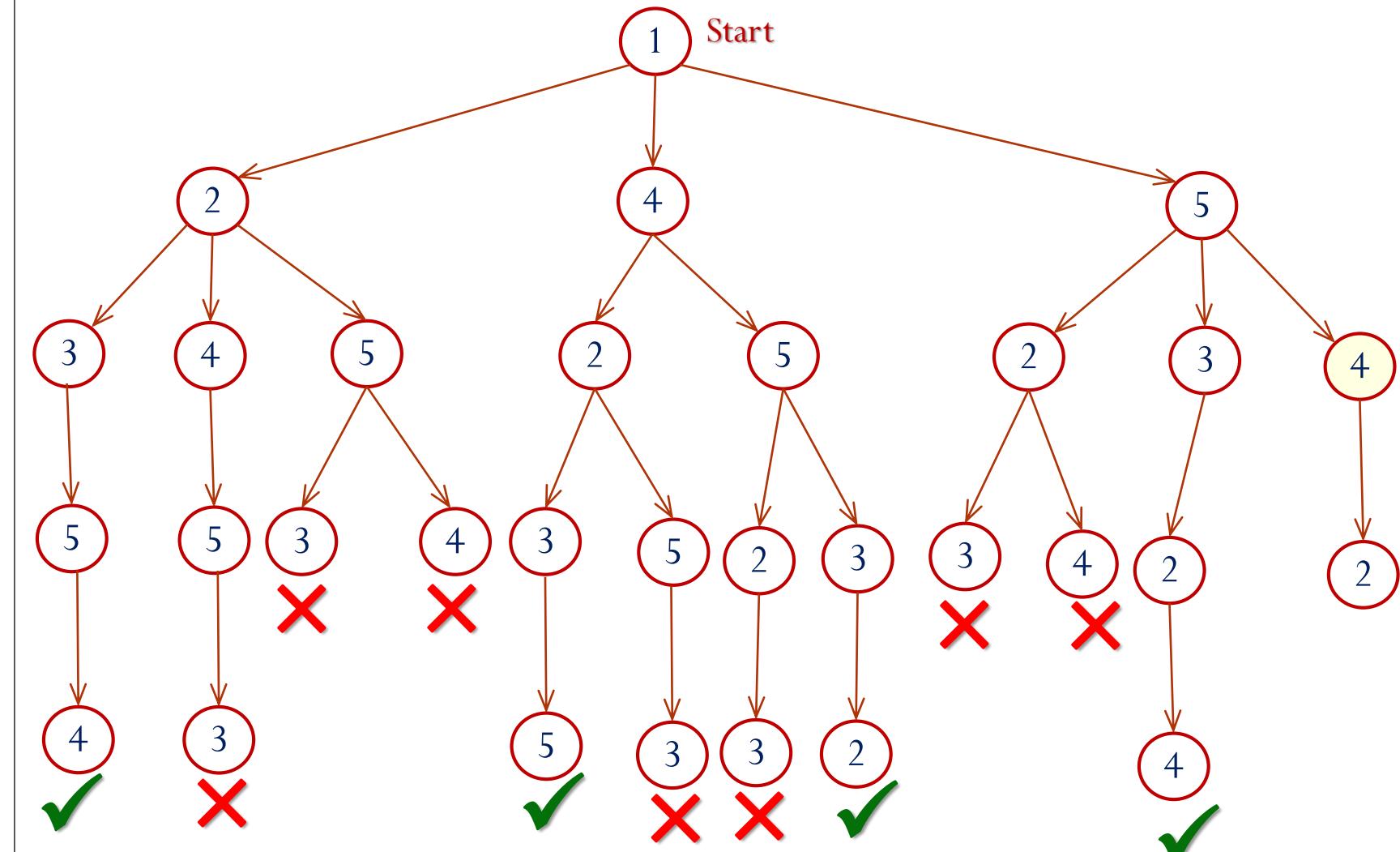


C	1	5	4	0	0
	1	2	3	4	5

# Solutions

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

✓ For 4, next choice is 2



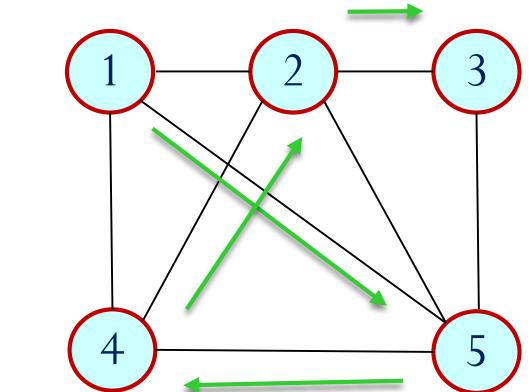
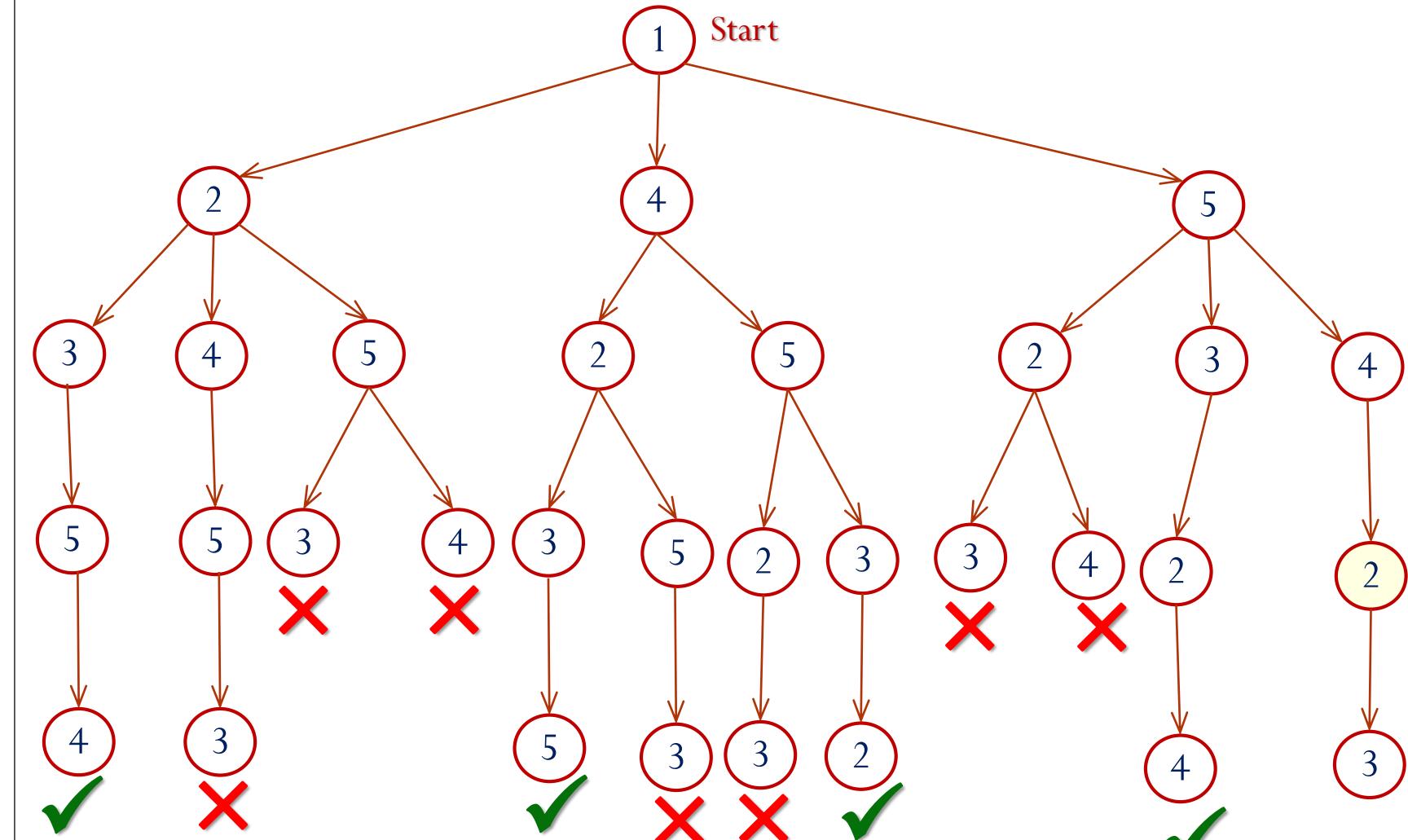
C

1	5	4	2	0
1	2	3	4	5

Solutions

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

✓ For 2, next choice is 3

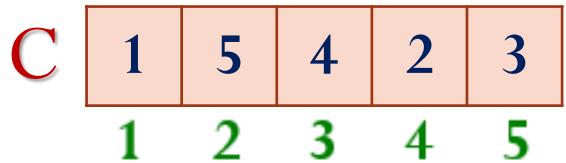
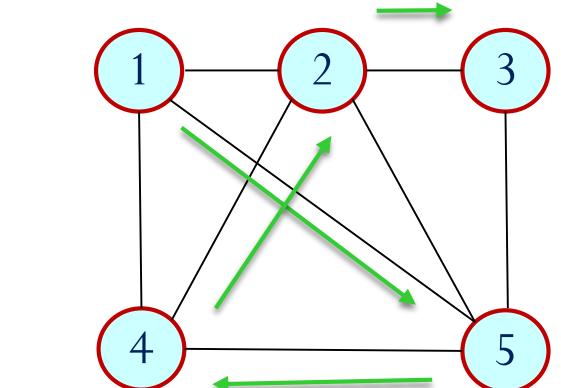
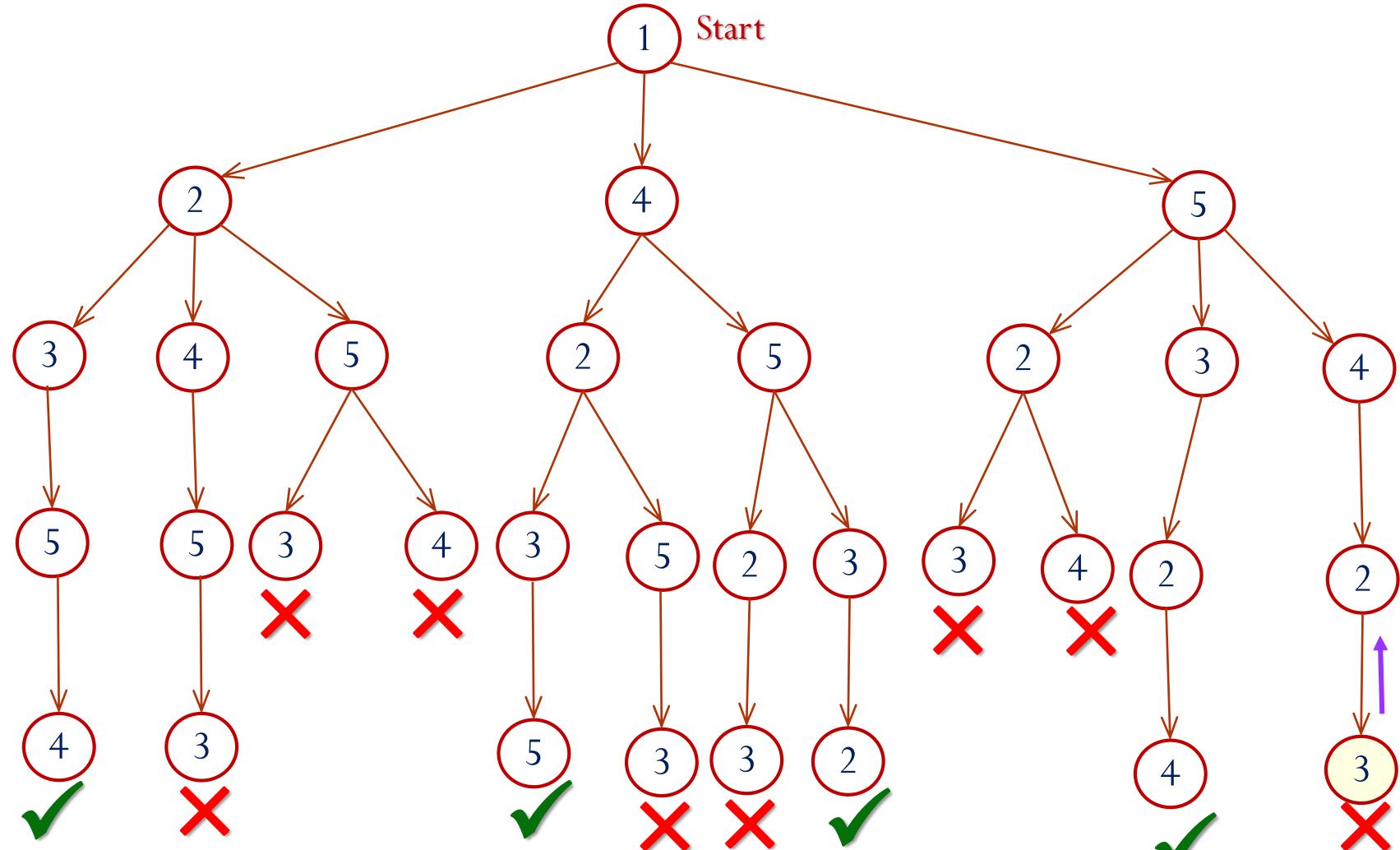


C	1	5	4	2	3
	1	2	3	4	5

Solutions

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

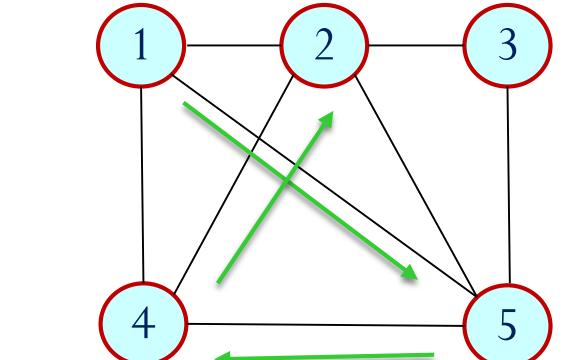
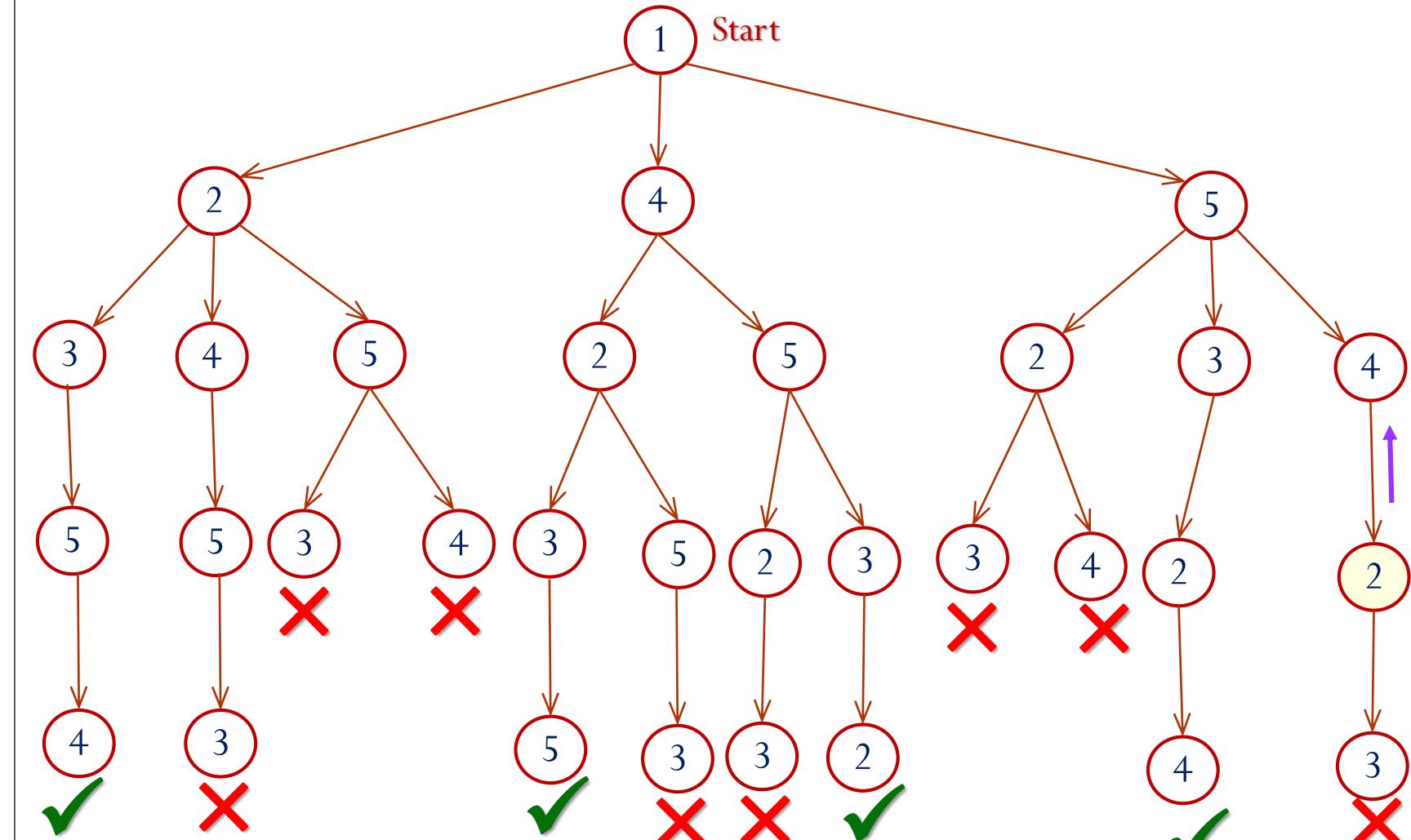
- ✓ For 3, edge (3,1) not present, so backtrack



Solutions

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

- ✓ For 2, no more choice, so backtrack

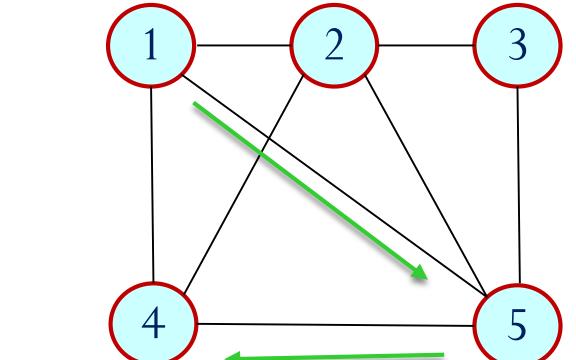
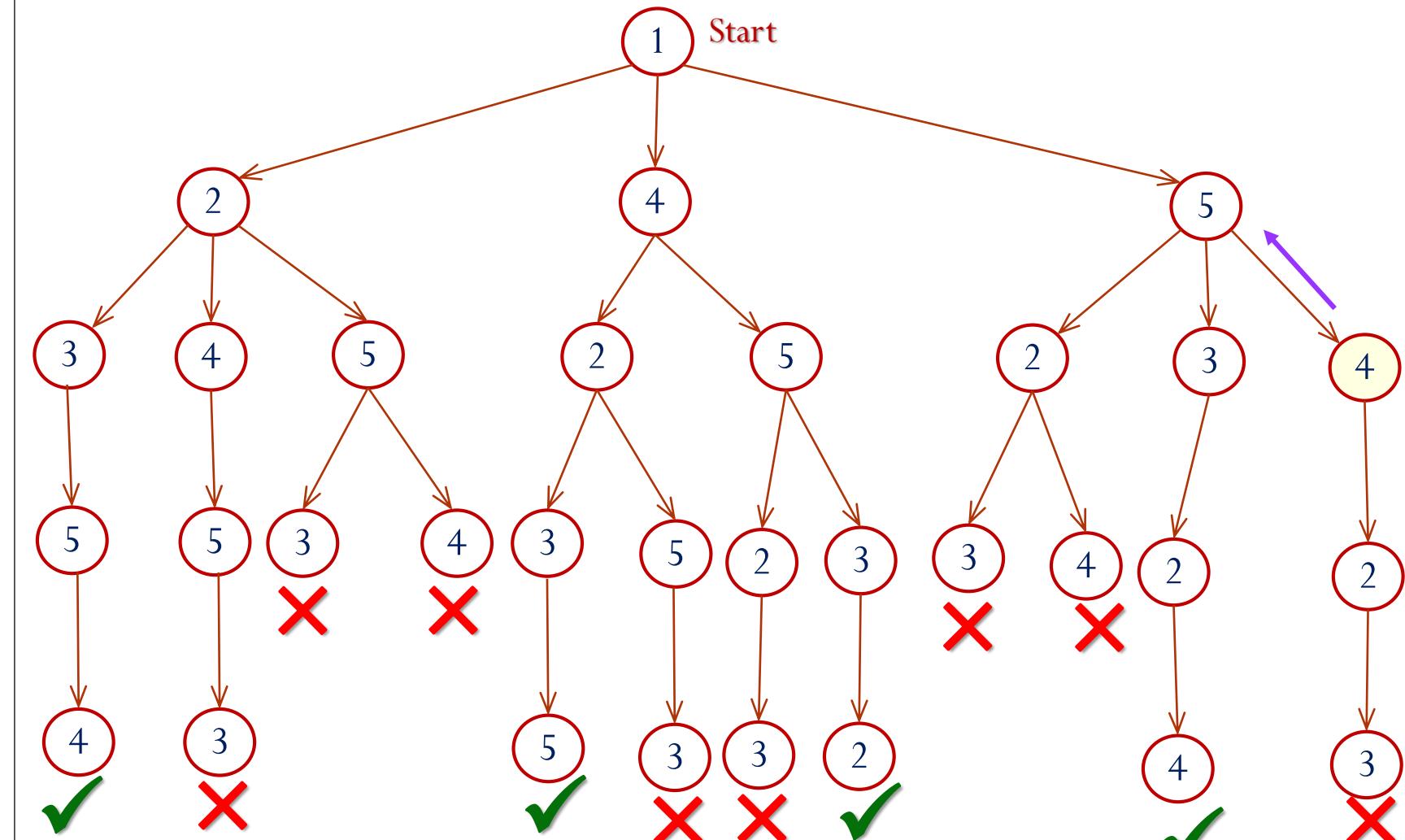


C	1	5	4	2	0
	1	2	3	4	5

# Solutions

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

✓ For 4, no more choice, so backtrack

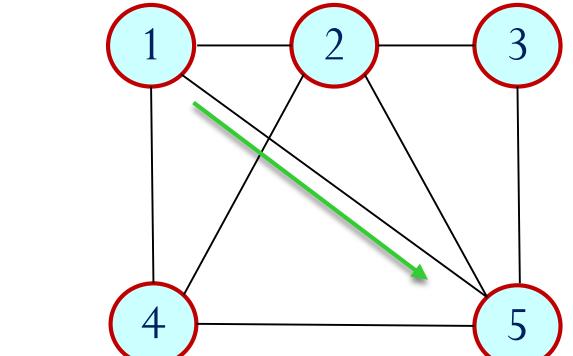
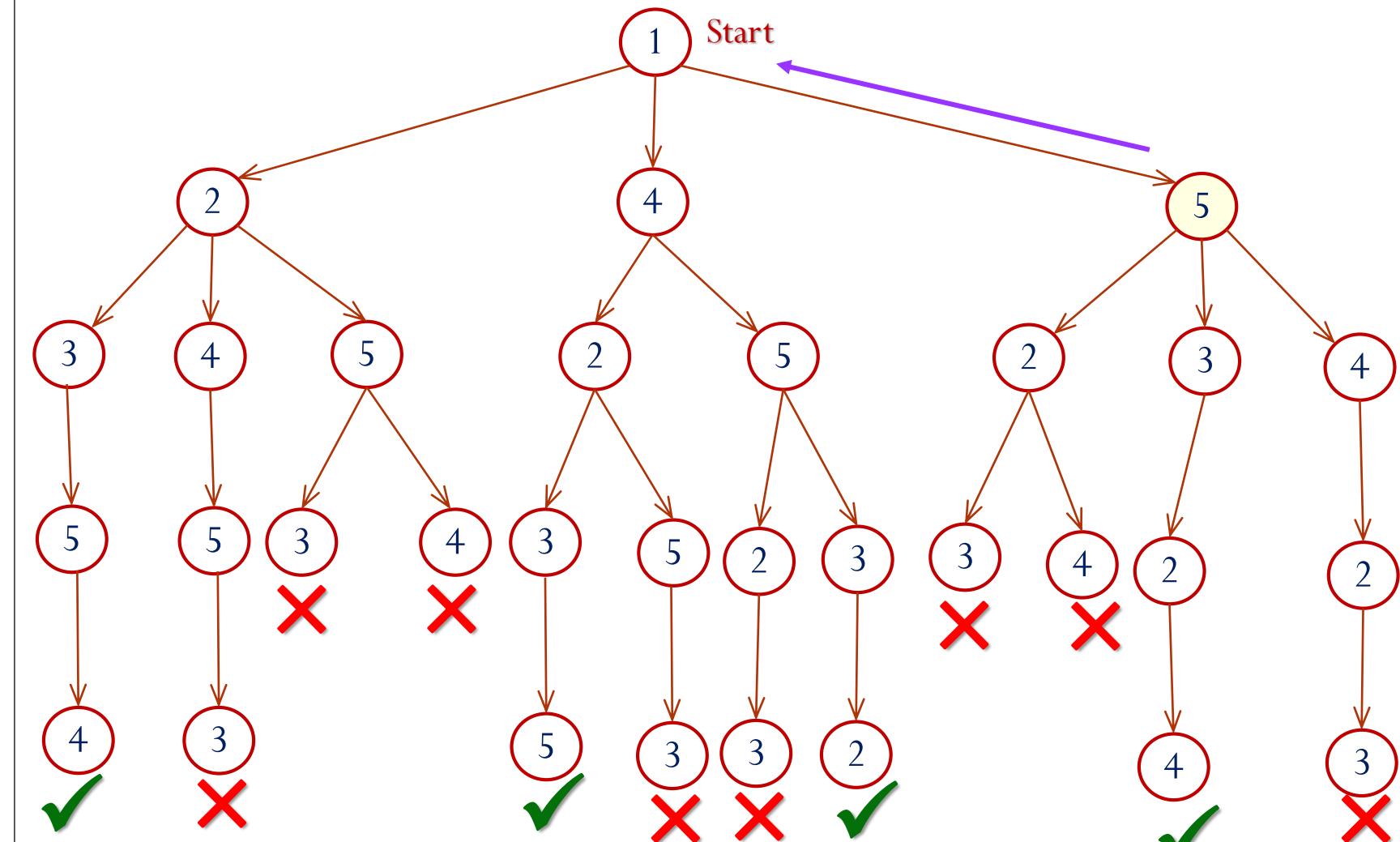


C	1	5	4	0	0
	1	2	3	4	5

Solutions

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

✓ For 5, no more choice, so backtrack



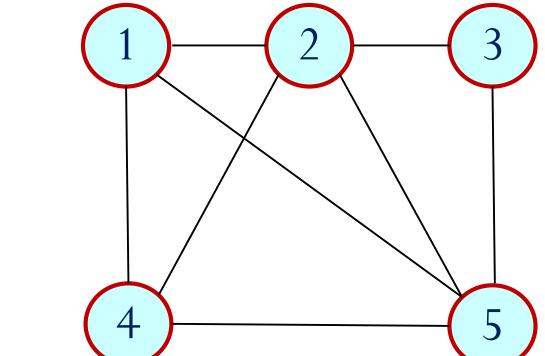
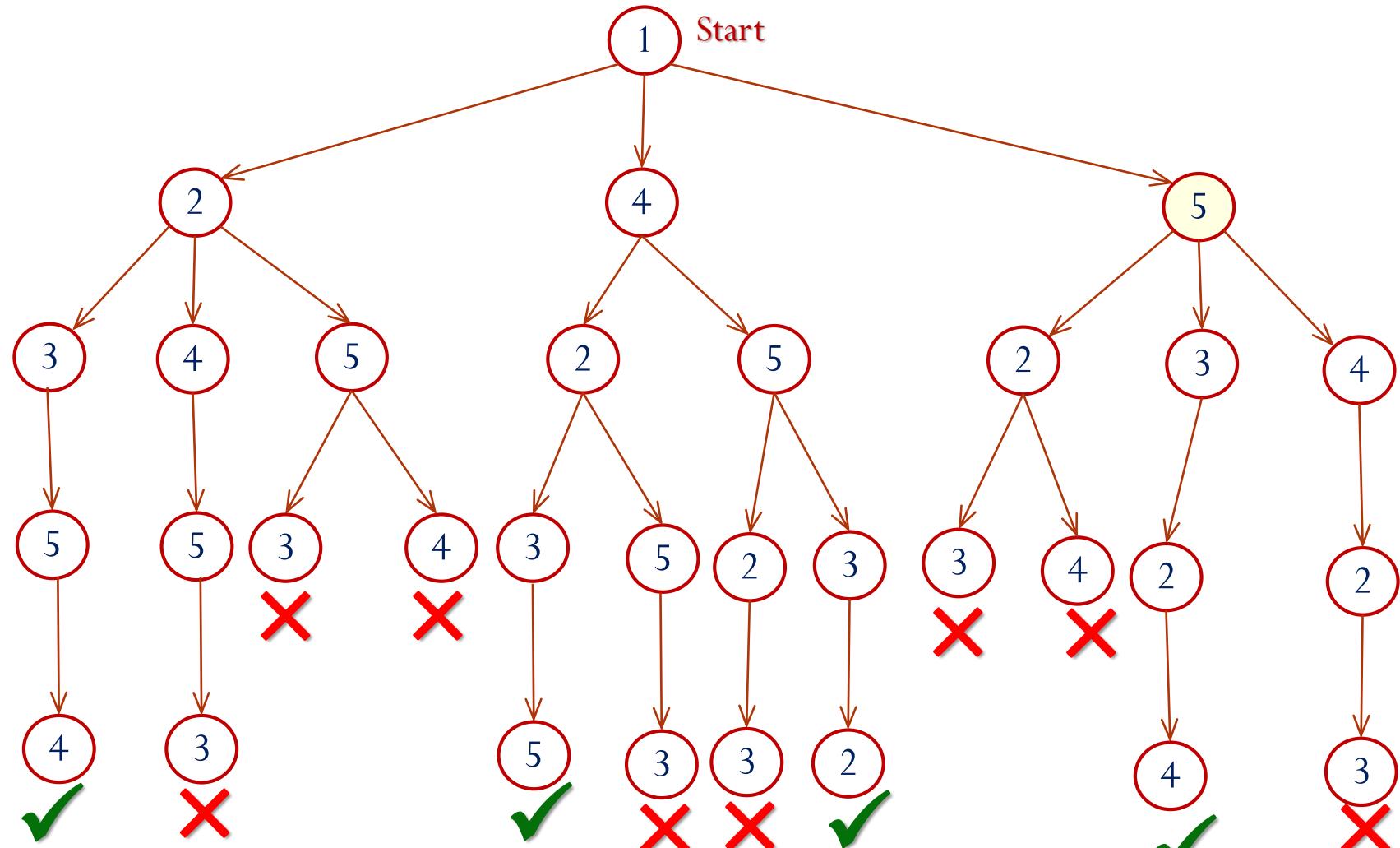
C

1	5	0	0	0
1	2	3	4	5

Solutions

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

✓ For 1, no more choice, so **STOP**

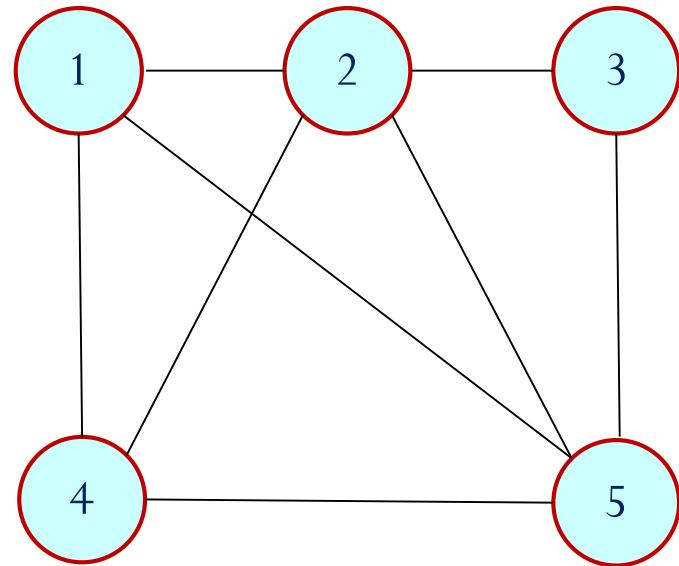


C	1	0	0	0	0
	1	2	3	4	5

Solutions

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

## Graph



## Solutions

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

# Hamiltonian Cycles - Pseudocode

```
Algorithm HamiltonCycles(w[1..n][1..n], n, start)
    //w – Adjacent matrix of input graph – values: 0 or 1
    //n – number of vertices
    //start – Starting vertex number. Usually 1.

    //Let C[1..n] be an Solution Array
    For i←1 to n Do
        C[i] ← 0
    End For

    //Setting first vertex in the cycle as start
    C[1] ← start

    //set current vertex index as 1
    i ← 1

    //set starting vertex as current vertex
    current ← start

    //Lets start selecting vertices one by one
    Cycles(w, n, C, start, current, i)

End HamiltonCycles
```

# Hamiltonian Cycles - Pseudocode



**Algorithm** Cycles( $w[1..n][1..n]$ ,  $n$ ,  $C[1..n]$ , start, current, i)

// $w$  – Adjacent matrix of input graph – values: 0 or 1  
// $n$  – number of vertices  
// $C$  – Solution Array  
//start – Starting Vertex  
//current – Current Vertex  
//i – Level number in solution tree

**If**  $i=n$  and  $w[current][start]=1$  **Then**  
    //Solution found, Print the solution  
    Print  $C[1..n]$

        //Backtrack  
        **Return**

**End If**

**If**  $i=n$  and  $w[current][start]=0$  **Then**  
    //No solution found, Just backtrack  
    **Return**

**End If**

//Let  $j$  be the index of next vertex  
 $j \leftarrow i+1;$

//Choosing next possible vertex

**For**  $\text{next} \leftarrow 1$  to  $n$  **Do**  
    Skip  $\leftarrow$  False  
    **If**  $w[current][\text{next}] = 0$  **Then**  
        Skip  $\leftarrow$  True

**End If**  
    **For**  $k \leftarrow 1$  to  $i$  **Do**  
        **If**  $C[k] = \text{next}$  **Then**  
            Skip  $\leftarrow$  True

**End If**  
    **End For**  
    **If** Skip = False **Then**  
         $C[j] \leftarrow \text{next}$   
        //Go for next level as: next as current  
        Cycles( $w, n, C, \text{start}, \text{next}, j$ )

**End If**

**End For**  
     $C[j] \leftarrow 0$

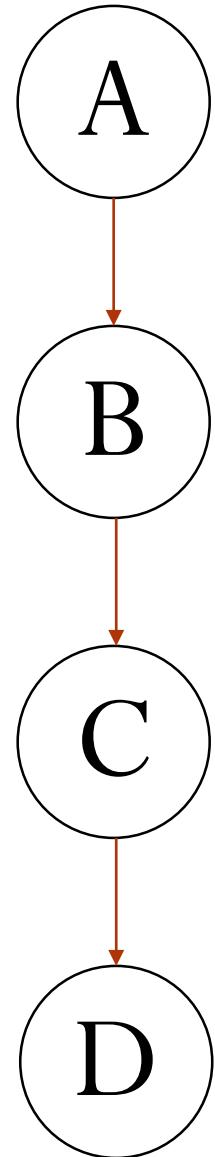
**End Cycles**

# **Graph – Basic Terminologies**

# List

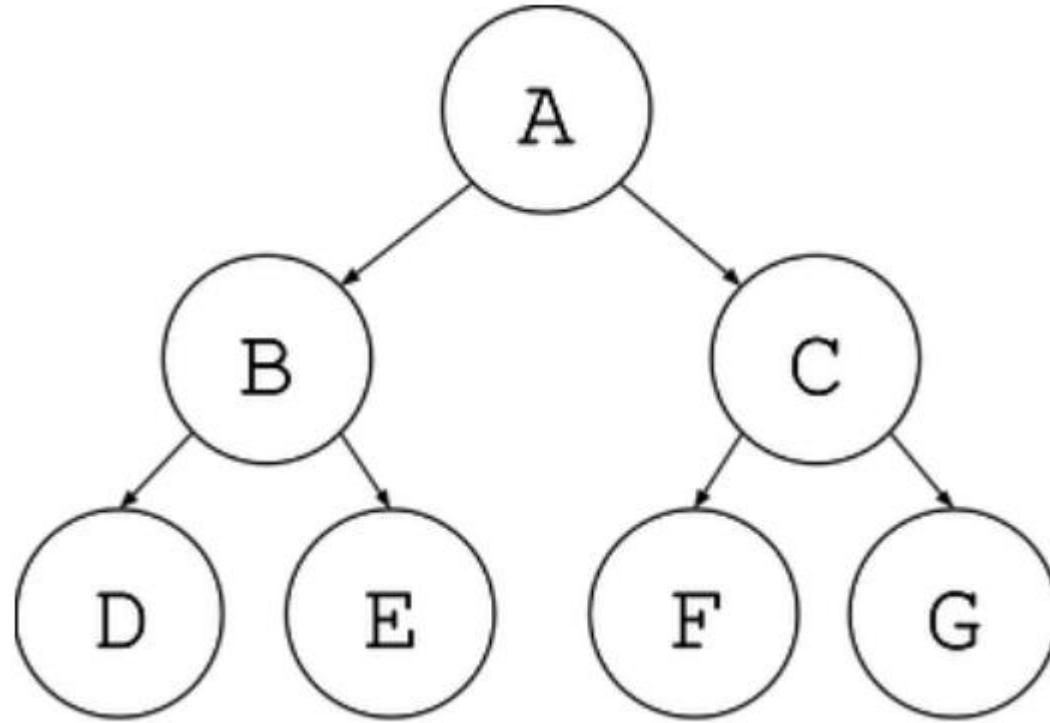


**SASTRA**  
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION  
DEEMED TO BE UNIVERSITY  
(U/S 3 OF THE UGC ACT, 1956)  
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



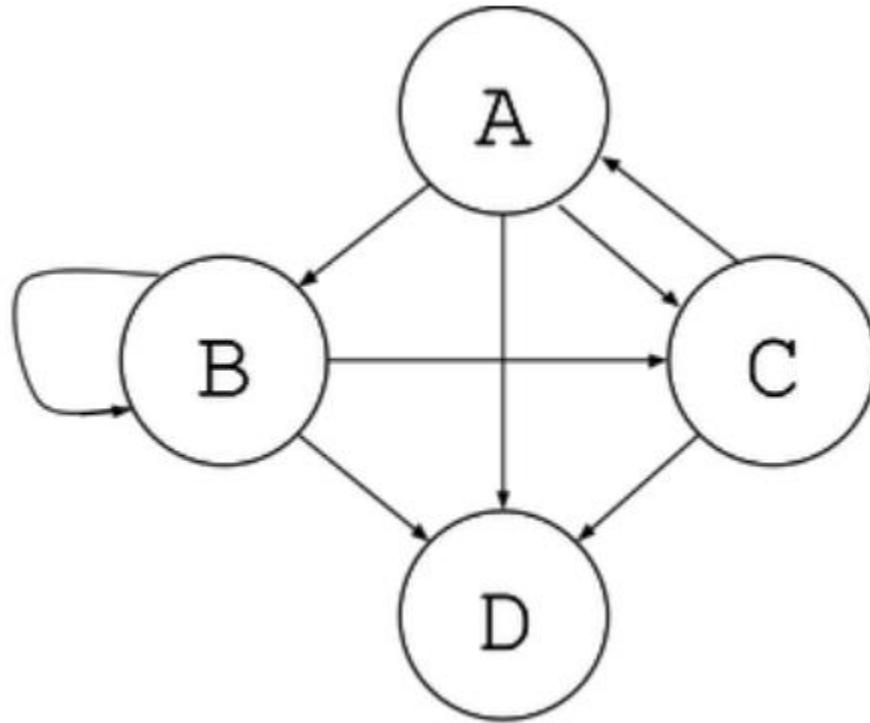
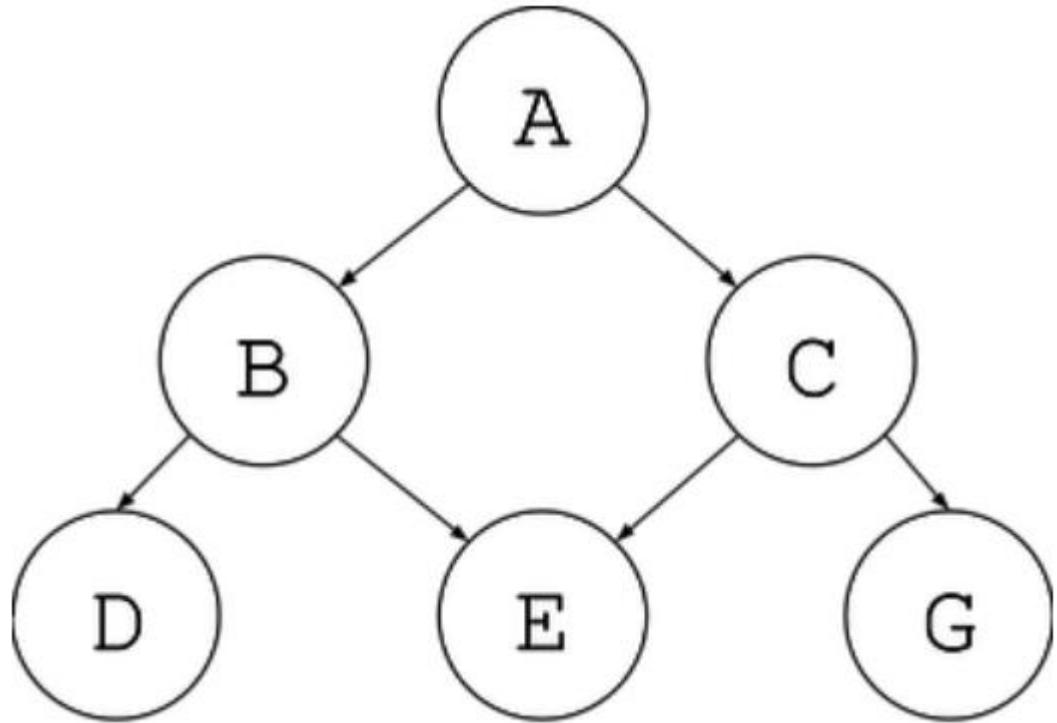
*Node May Have*  
**Single Predecessor**  
**&**  
**Single Successor**

# Tree



*Node May Have*  
**Single Predecessor**  
**&**  
**Multiple Successor**

# Graph



Node May Have

**Multiple Predecessor  
&  
Multiple Successor**

# Graph - Definition

A graph is simply a collection of **Vertices** plus **Edges**.

A graph  $G$  is a pair  $(V, E)$  where

- ✓  $V$  is a set of **vertices** (singular “Vertex”) or **nodes**
- ✓  $E$  is a set of **edges** that connect vertices
- ✓ A node in a graph – Vertex
- ✓ A line that connect 2 vertices – Edge
- ✓ Each edge is represented as a pair  $(v_1, v_2)$ , where  $v_1, v_2$  are vertices in  $V$

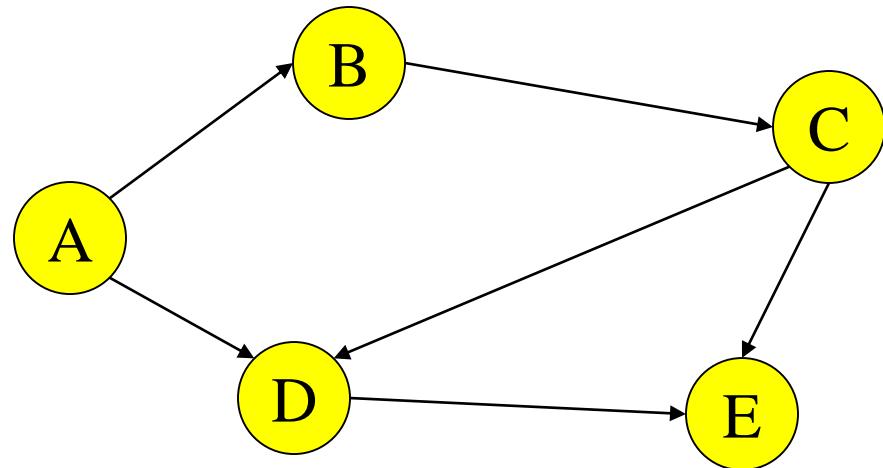
**Note:** Linked lists and Tree are special cases of graphs

# Graph - Example

Here is a graph  $G = (V, E)$

$$V = \{A, B, C, D, E, F\}$$

$$E = \{(A,B), (A,D), (B,C), (C,D), (C,E), (D,E)\}$$

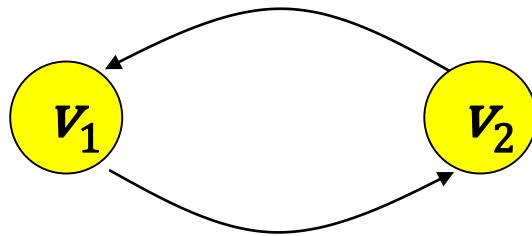


# Graph - Directed Graph vs Undirected Graph

## Directed Graph (also called as “digraph”):

- ✓ Each edge has a Direction. The edge in digraph is usually called as “**ARC**”.
- ✓ If the order of edge pairs  $(v_1, v_2)$  matters, the graph is directed (also called a digraph):

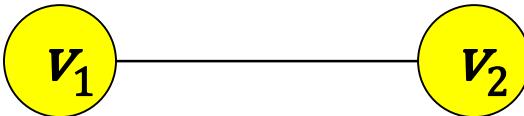
$$(v_1, v_2) \neq (v_2, v_1)$$



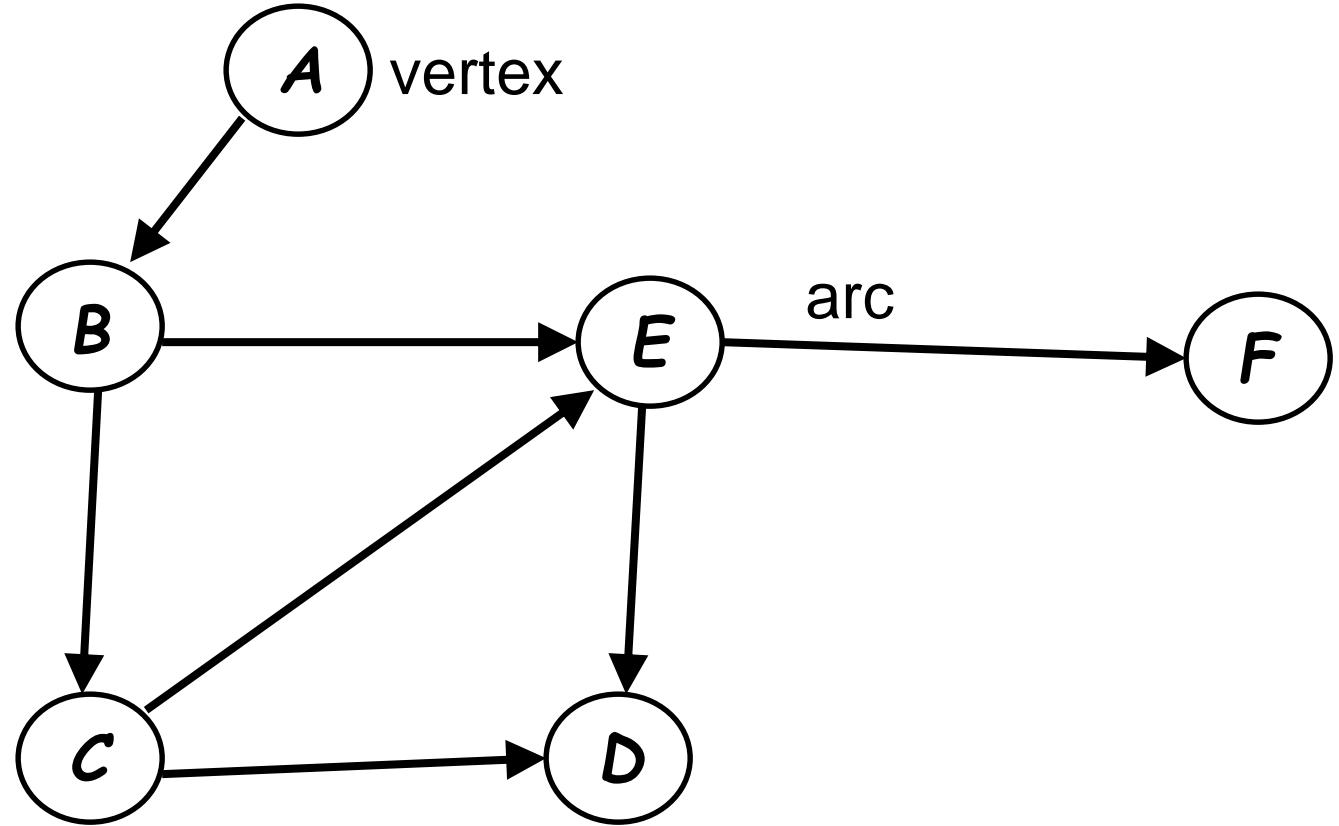
## Undirected Graph:

- ✓ The edges does not have any direction.
- ✓ If the order of edge pairs  $(v_1, v_2)$  does not matter, the graph is called an undirected graph:

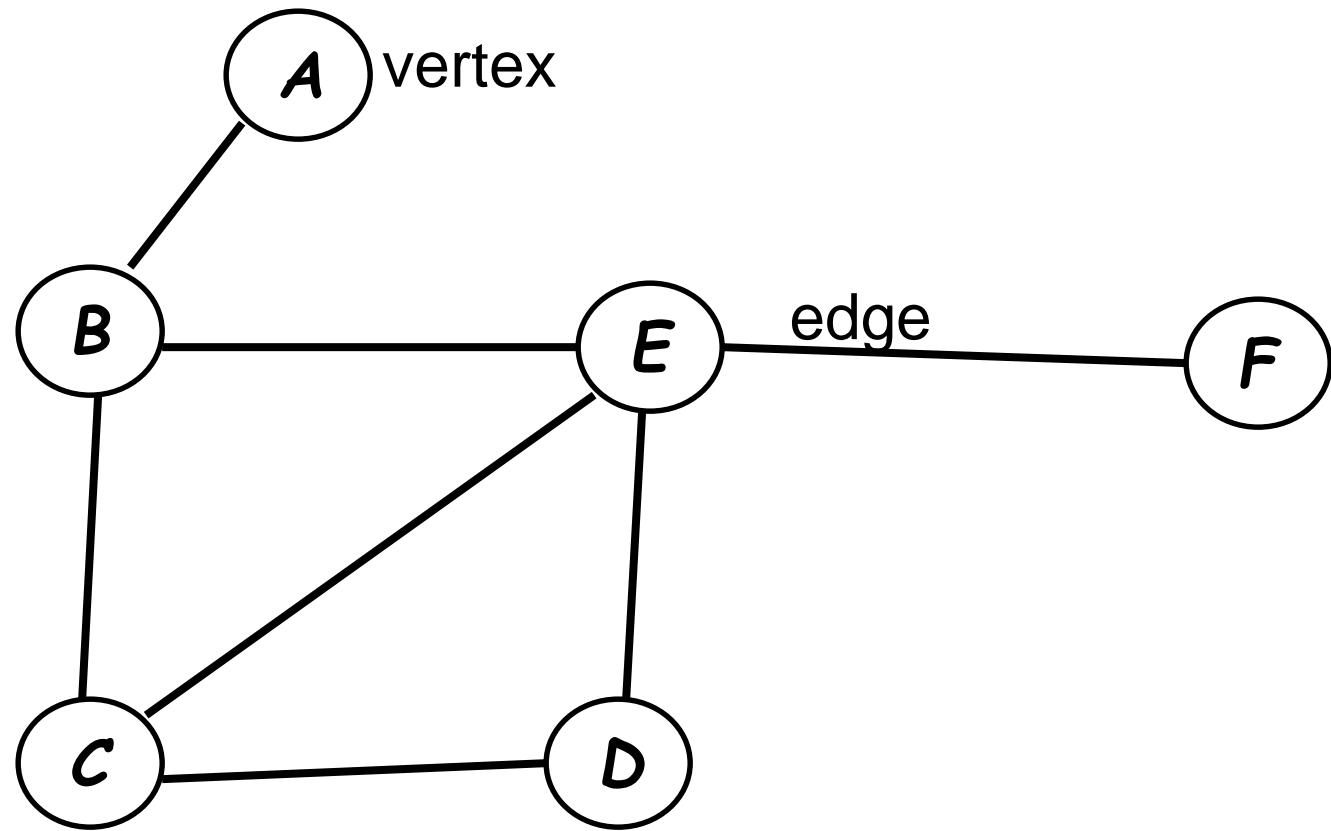
$$(v_1, v_2) = (v_2, v_1)$$



# Digraph - Example



# Undirected Graph - Example



# Graph – Basic Terminology

## Adjacent Vertices (neighbors)

In an **undirected graph G**:

- ✓ Two vertices **u** and **v** are said to be **adjacent**, if there exist an edge, **(u, v)** in the graph **G**.

In a **digraph G**:

- ✓ Vertex **u** is said to be **adjacent to** vertex **v** AND Vertex **v** is said to be **adjacent from** vertex **u**, if there exist an edge, **(u, v)** in the graph **G**.
- ✓ Here, **u** is initial vertex and **v** is terminal vertex.

# Graph – Basic Terminology

## Degree of Vertex

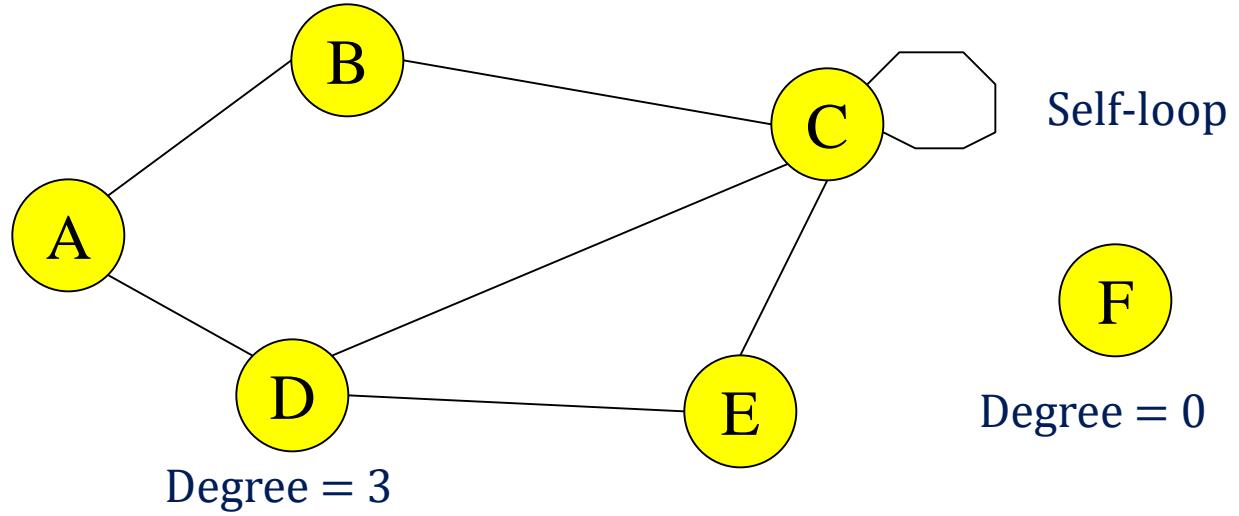
In an **undirected graph** G:

- ✓ The **degree** of the vertex is number of edges associated with it.

In a **digraph** G:

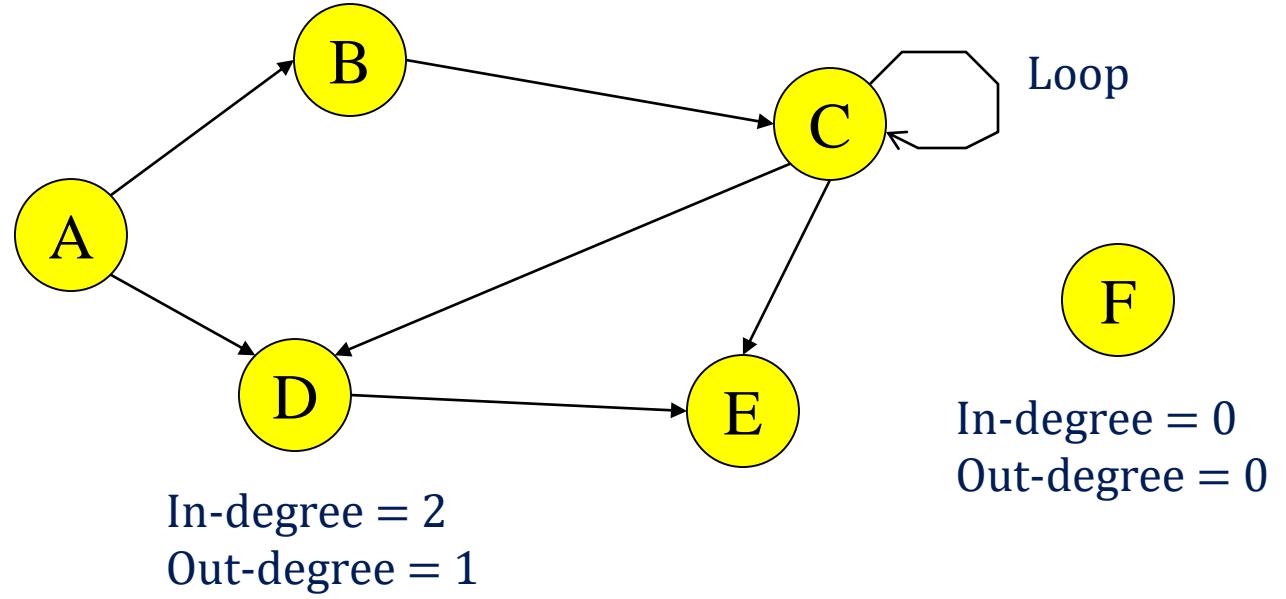
- ✓ The **in-degree** is the number of edges with the vertex as the terminal vertex.
- ✓ The **out-degree** is the number of edges with the vertex as the initial vertex

# Graph – Basic Terminology



Since an edge (A,B) is there, A and B are said to be adjacent to each other. But, the vertices A and C are not adjacent vertices.

# Graph – Basic Terminology



Since an edge (A,B) is there, A adjacent to B and B adjacent from A.

# Graph – Basic Terminology

## Path

A path is a sequence of vertices with the property that each vertex in the sequence is adjacent to the vertex next to it.

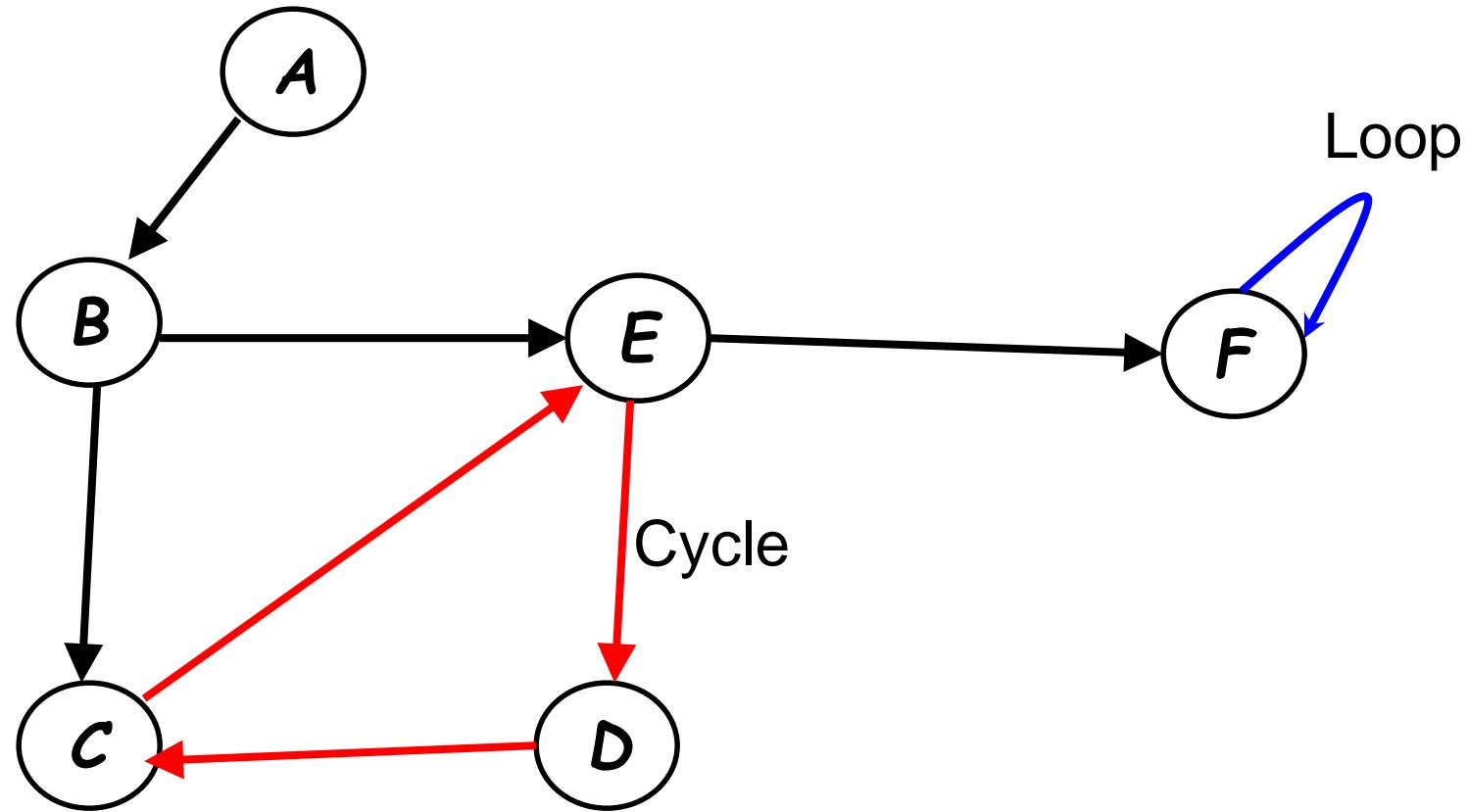
## Cycle

A path consisting of at least three vertices that starts and ends with the same vertex.  
Must follow the proper direction in a digraph

## Loop

A special case of a cycle in which a single arc begins and ends with the same vertex

# Graph – Basic Terminology



Examples:

Path: ABEDC, ABEF, DCEF, EDC, ABCED, etc.,

Cycle: EDCE

Loop: FF

# Graph – Basic Terminology

## Connected Vertices

Two vertices are said to be connected if there exist a path between them

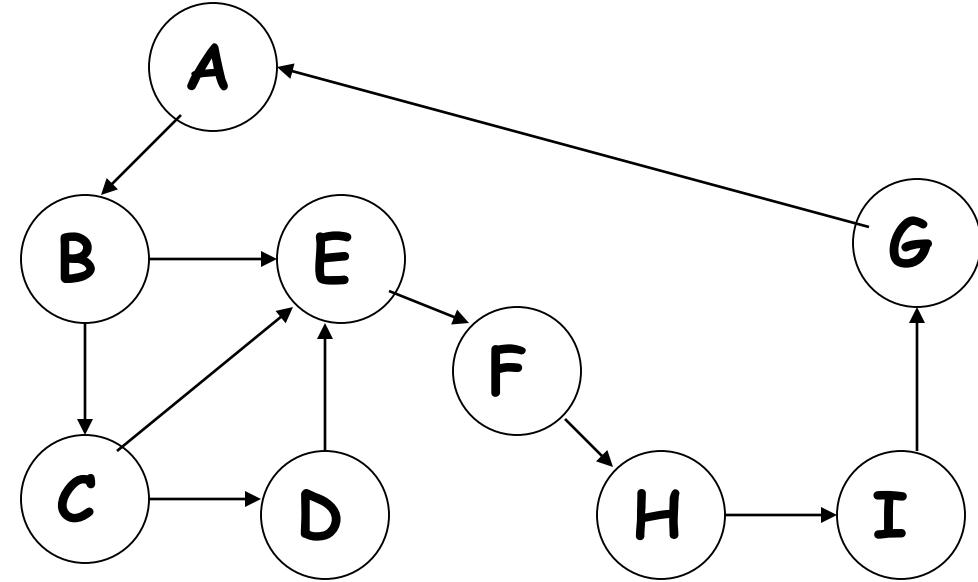
## Strongly Connected Graph

A Directed Graph is said to be strongly connected if there is a path from every vertex to every other vertex in the digraph

## Weakly Connected Graph

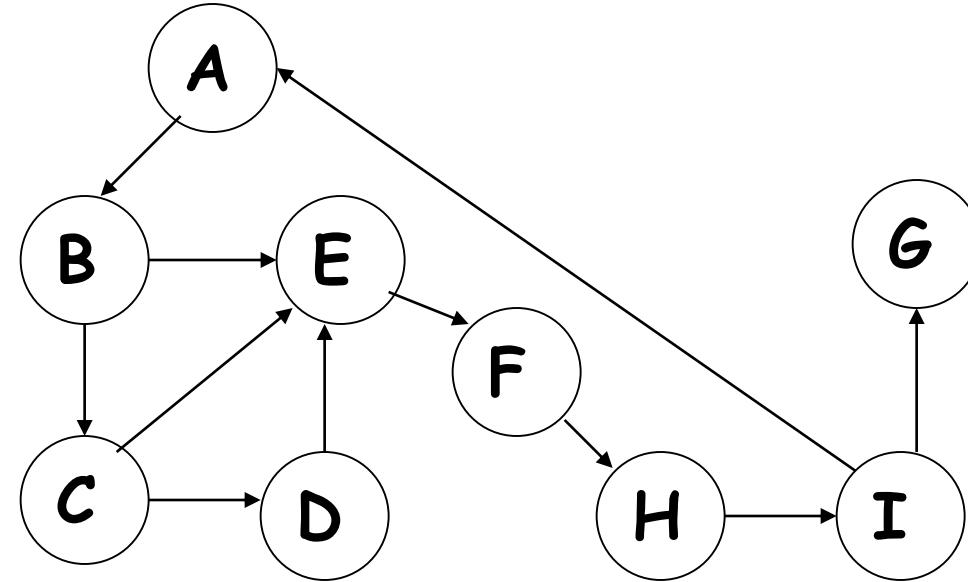
A Directed Graph is said to be weakly connected when there are at least two vertices that are not connected

# Graph – Basic Terminology



**Strongly Connected** - For each and every pair of vertices, there exist a path.

# Graph – Basic Terminology

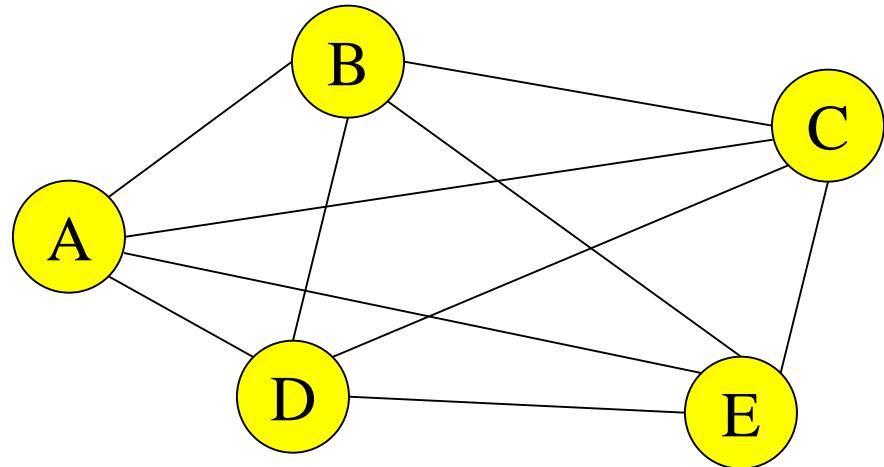


**Weakly Connected** - No path exist from G to all other vertices.

# Graph – Basic Terminology

## Complete Graph

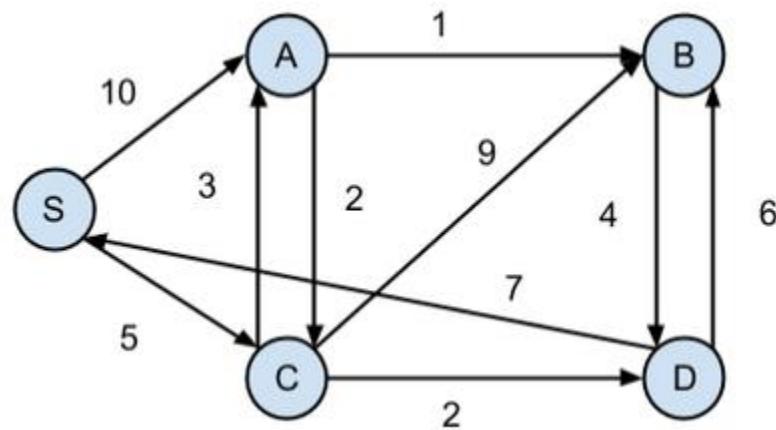
A complete graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge.



# Graph – Basic Terminology

## Weighted Graph

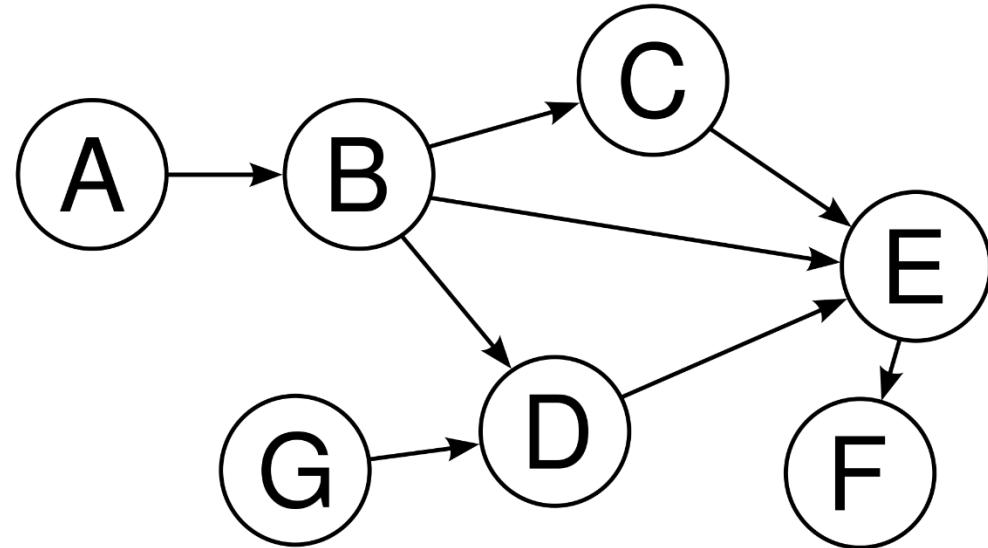
The weighted graph is either directed or undirected. But every edge will have a numerical COST or WEIGHT associated with it. The cost of the edge is not specified, then it will be 1 by default.



# Graph – Basic Terminology

## Directed Acyclic Graph (DAG)

A directed graph is said to be directed acyclic graph if there is no cycle in the graph.



# Graph – Representation

Space and time complexity of graph algorithms are analyzed in terms of:

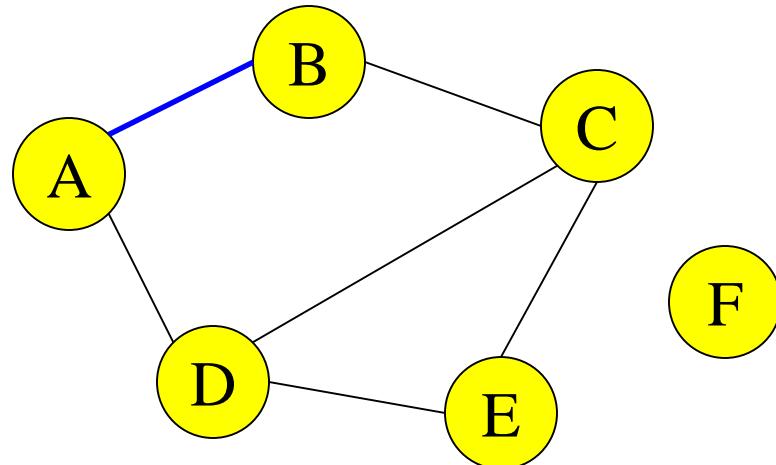
- ✓ Number of vertices =  $|V|$  and
- ✓ Number of edges =  $|E|$

There are **TWO basic representations** for graph.

1. Adjacency Matrix
2. Adjacency List

# Graph – Representation

## Adjacency Matrix – Undirected Graph



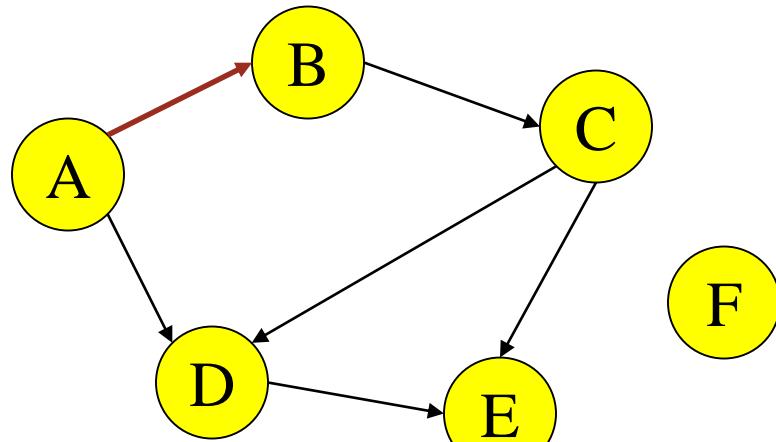
$$M(v, w) = \begin{cases} 1 & \text{if } [v, w] \text{ is in } E \\ 0 & \text{otherwise} \end{cases}$$

	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	1	0
D	1	0	1	0	1	0
E	0	0	1	1	0	0
F	0	0	0	0	0	0

Space =  $|V|^2$

# Graph – Representation

## Adjacency Matrix – Digraph



$$M(v, w) = \begin{cases} 1 & \text{if } [v, w] \text{ is in } E \\ 0 & \text{otherwise} \end{cases}$$

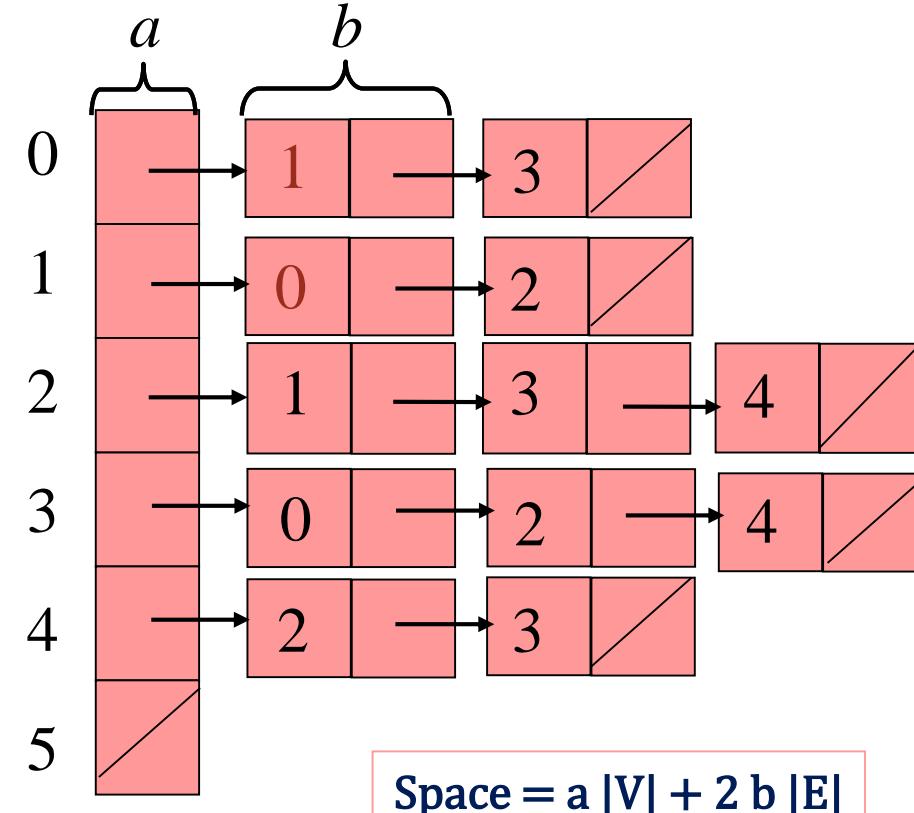
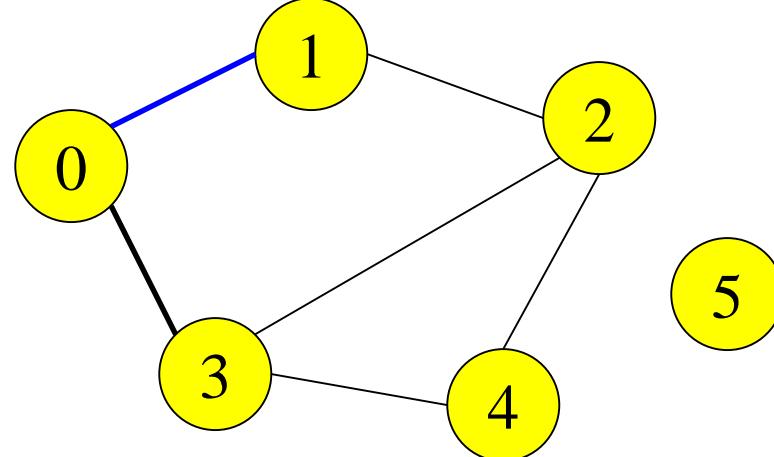
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	1	1	0
D	0	0	0	0	1	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Space =  $|V|^2$

# Graph – Representation

## Adjacency List – Undirected Graph

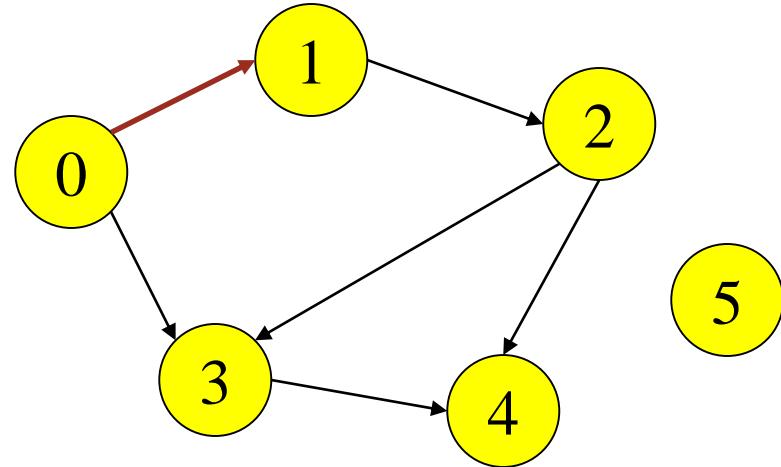
For each  $v$  in  $V$ ,  $L(v) = \text{list of } w \text{ such that } [v, w] \text{ is in } E$



# Graph – Representation

## Adjacency List – Digraph

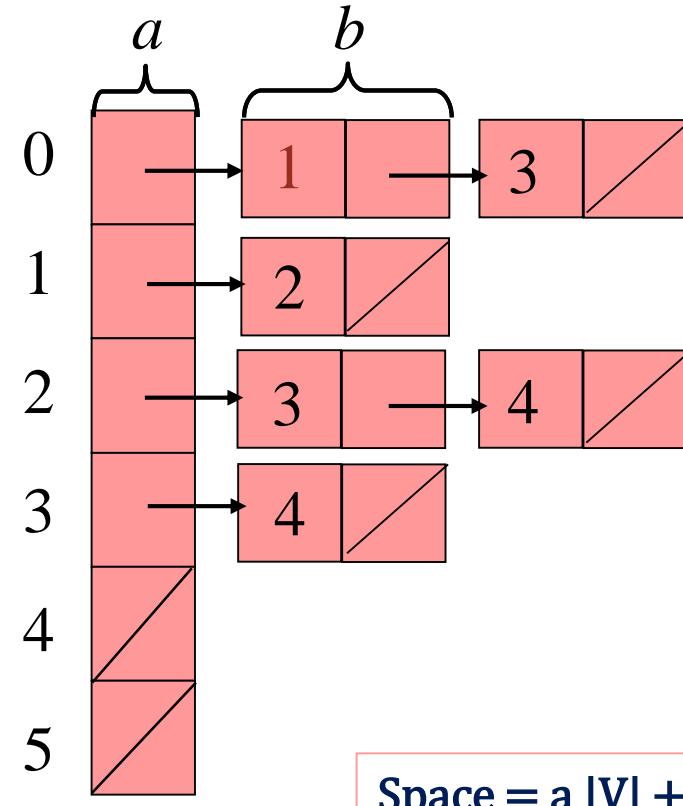
For each  $v$  in  $V$ ,  $L(v) = \text{list of } w \text{ such that } (v, w) \text{ is in } E$



0 is a source

4 is a sink

5 is disconnected from the rest



$$\text{Space} = a |V| + b |E|$$

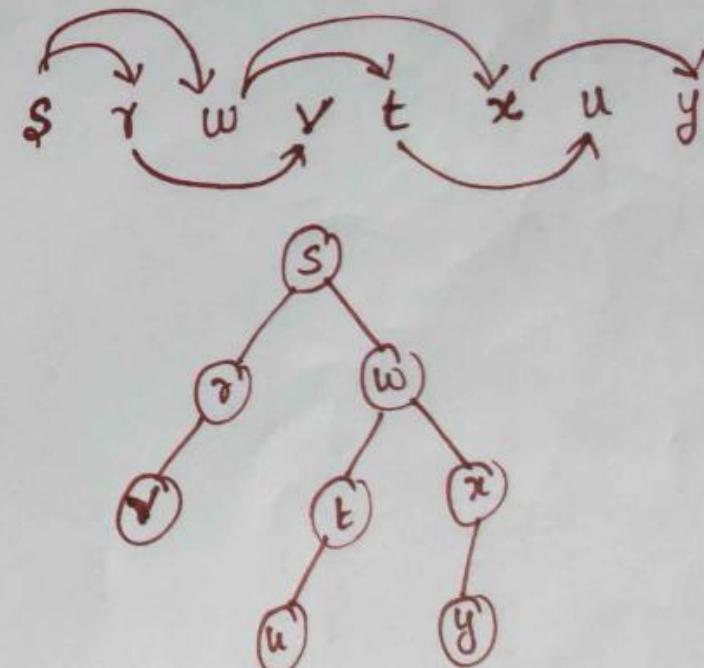
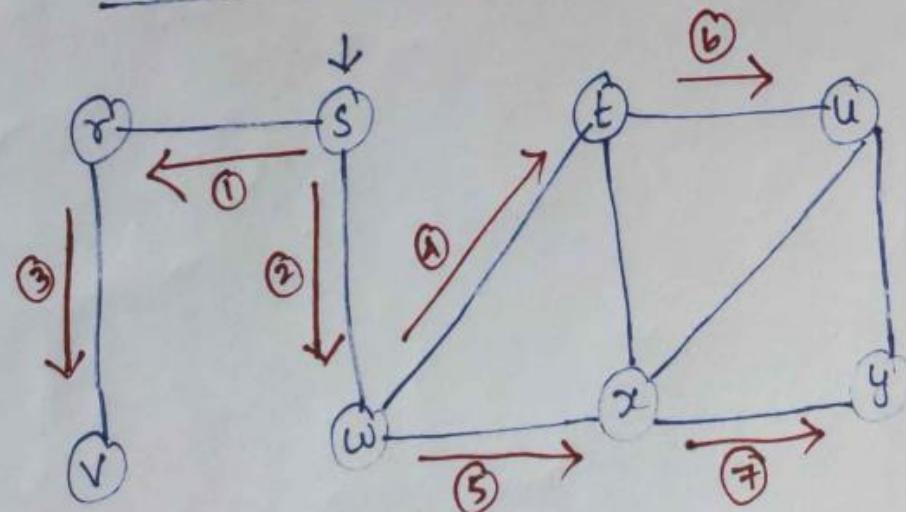
# Traversal Algorithms – Breadth First Search

## Traversal in Graph:

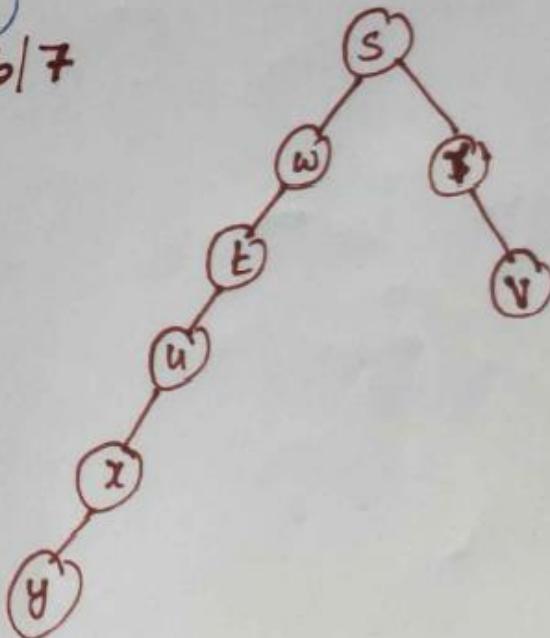
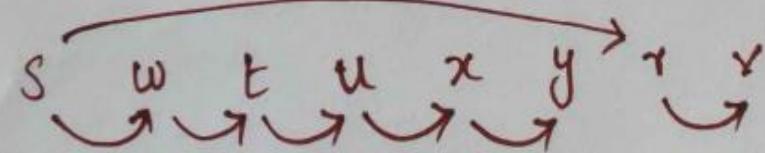
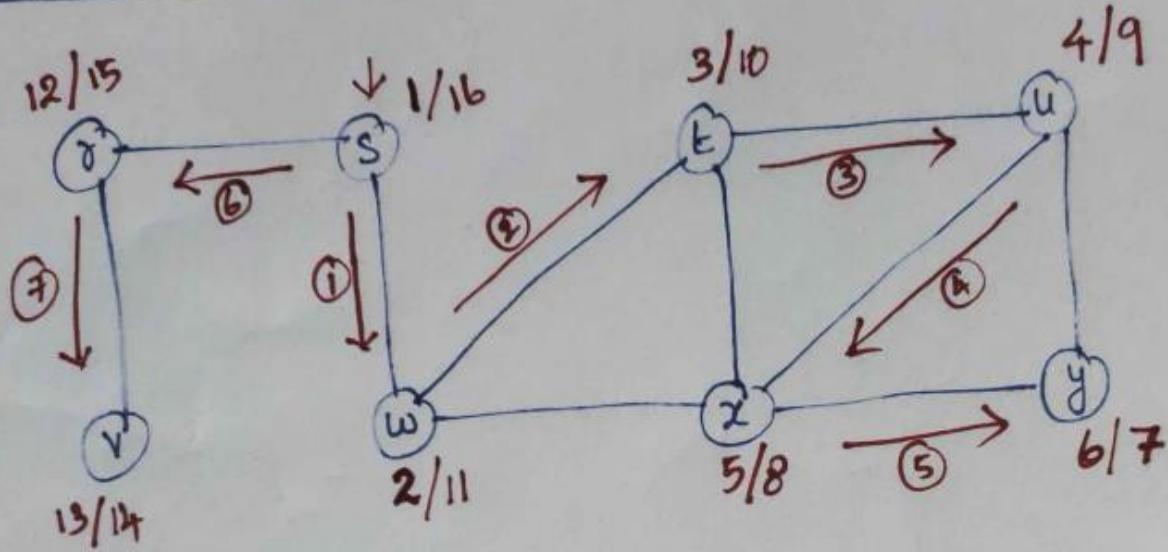
↳ Order of visiting the vertices in Graph.

1. Breadth First Search [BFS]
2. Depth First Search [DFS]

### Breadth First Search: [BFS]

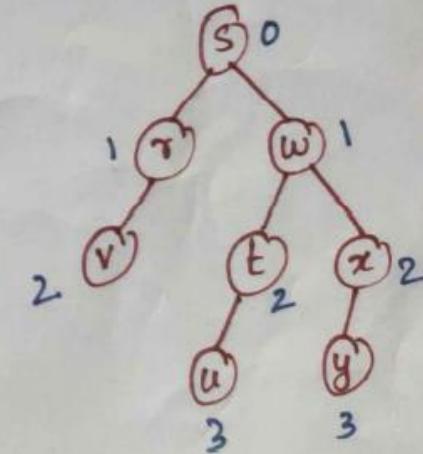
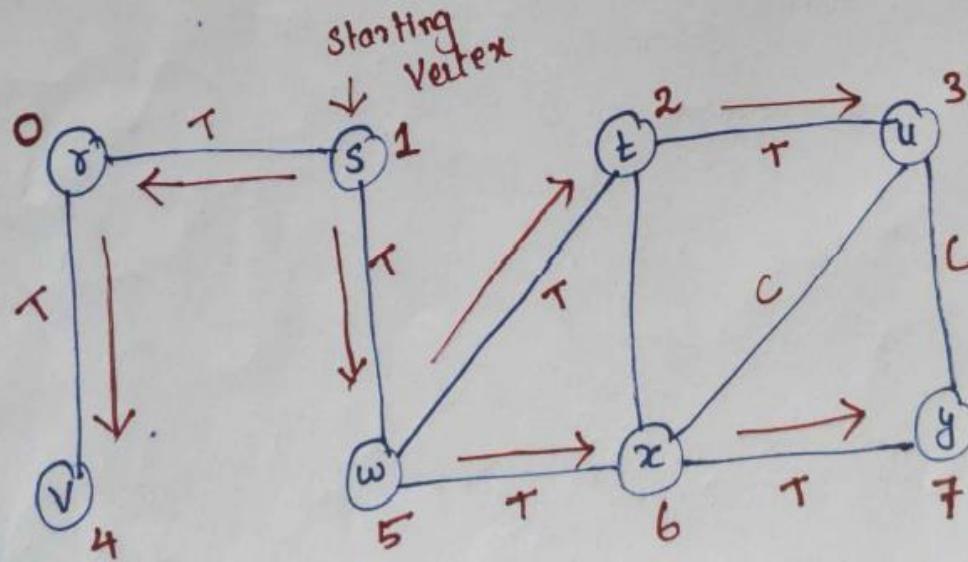
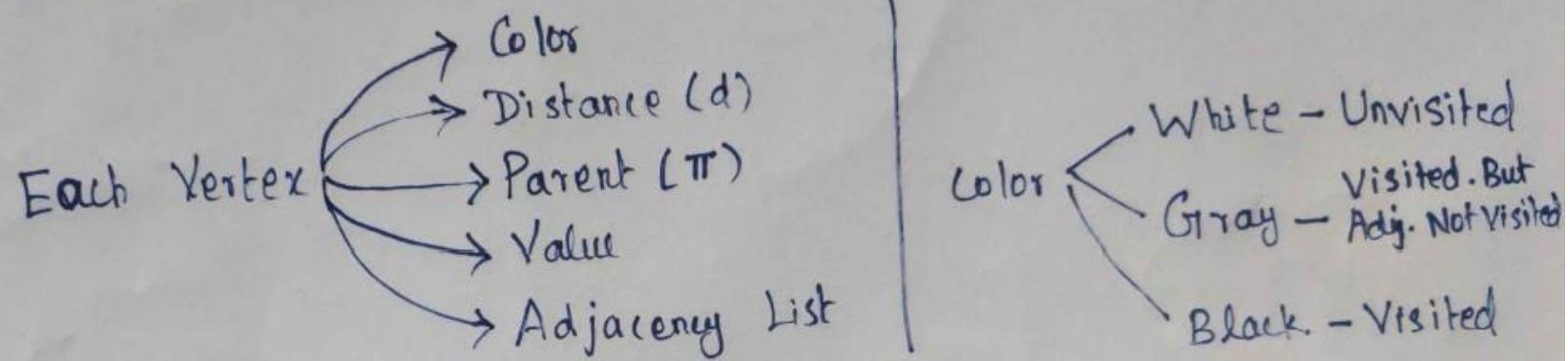


## Depth First Search [DFS]



## Breadth First Search [BFS] - Algorithm

192



$G_i$	Value	Color	$\delta$	$\pi$	$\alpha_i$	
$u \neq 0$	0	WGB	$X_1$	$N_1$		$\rightarrow 1 \rightarrow 4$
$u \neq 1 \Rightarrow 1$	1	WGB	$X_0$	$N_N$		$\rightarrow 0 \rightarrow 5$
$X^2$	2	WGB	$X_2$	$N_5$		$\rightarrow 3 \rightarrow 5 \rightarrow 6$
$X^3$	3	WGB	$X_3$	$N_2$		$\rightarrow 2 \rightarrow 6 \rightarrow 7$
$X^4$	4	WGB	$X_2$	$N_0$		$\rightarrow 0$
$X^5$	5	WGB	$X_1$	$N_1$		$\rightarrow 1 \rightarrow 2 \rightarrow 6$
$X^6$	6	WGB	$X_2$	$N_5$		$\rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7$
$u \neq 1$	7	WGB	$X_3$	$N_6$		$\rightarrow 3 \rightarrow 6$

$$\begin{array}{ccccccccc}
 1 & 3 & 4 & 6 & 8 & 9 & 12 & 14 \\
 \downarrow & \downarrow \\
 \hline
 1 & 0 & 5 & 4 & 2 & 6 & 3 & 7 \\
 \downarrow & \downarrow \\
 2 & 5 & 7 & 10 & 11 & 13 & 15 & 16
 \end{array}$$

s r w v t x u y

Alg BFS( $G, s$ )

for each vertex  $v \in G.V$

Step 1

$v.\text{color} \leftarrow \text{WHITE}$   
 $v.d \leftarrow \infty$   
 $v.\pi \leftarrow \text{NIL}$

end for

Step 2

---

$s.\text{color} \leftarrow \text{GRAY}$   
 $s.d \leftarrow 0$   
 $s.\pi \leftarrow \text{NIL}$

// Let  $Q$  be a Queue to store list of vertices

Step 3

$Q \leftarrow \emptyset$   
 $\text{EnQ}(Q, s)$

Step 4

while  $Q \neq \emptyset$  do

$u \leftarrow \text{DeQ}(Q)$

for each  $v \in G.\text{Adj}[u]$  do

if  $v.\text{color} = \text{WHITE}$  then

$v.\text{color} \leftarrow \text{GRAY}$   
 $v.d \leftarrow u.d + 1$   
 $v.\pi \leftarrow u$   
 $\text{EnQ}(Q, v)$

end if

end for

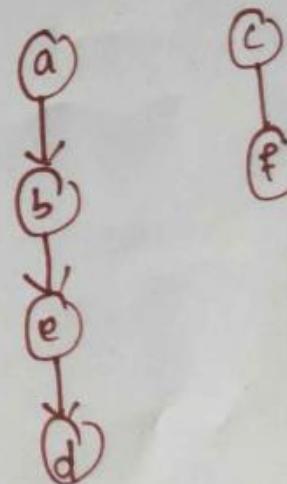
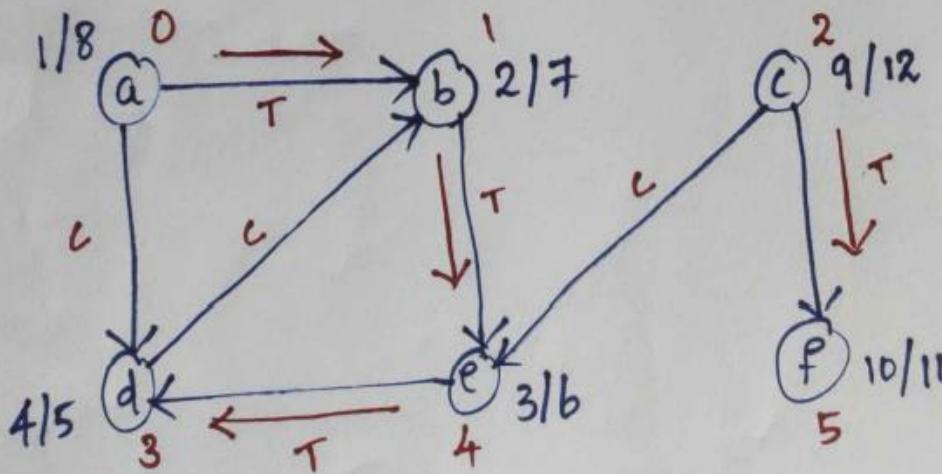
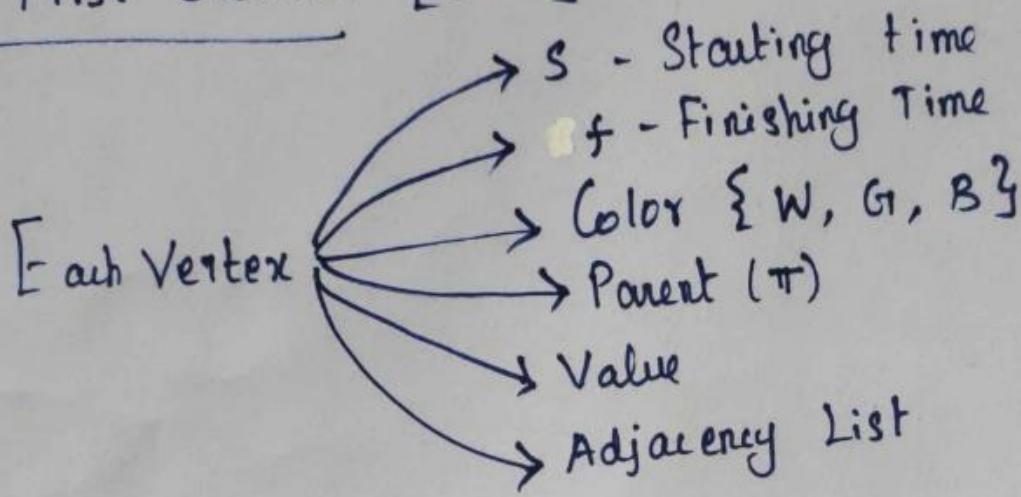
$u.\text{color} \leftarrow \text{BLACK}$

end loop

end BFS

# Traversal Algorithms – Depth First Search

## Depth First Search: [DFS]



		value	$\pi$	$G/b_1$	$\alpha$	$\times$	$A_{ij}$	
$x$	0	N	$WGB$	1	8			$\rightarrow 1 \rightarrow 3$
$\mu$	1	$N_0$	$WGB$	2	7			$\rightarrow 4$
$\chi$	2	N	$WGB$	9	12			$\rightarrow 4 \rightarrow 5$
$\chi$	3	$N_4$	$WGB$	4	5			$\rightarrow 1$
$\chi$	4	$N_1$	$WGB$	3	6			$\rightarrow 3$
$u$	5	$N_2$	$WGB$	10	11		$\times$	

Alg DFS( $G_1$ )

for each  $u \in G_1.V$  do

$u.\pi \leftarrow \text{NIL}$

$u.\text{color} \leftarrow \text{WHITE}$

end for

time  $\leftarrow 0$   
for each  $u \in G_1.V$  do

if  $u.\text{color} = \text{WHITE}$  then

DFS-VISIT( $G_1, u, \text{time}$ )

endif

end for

end DFS

Alg. DFS-VISIT ( $G, u, \text{time}$ )

$u.\text{Color} \leftarrow \text{GRAY}$

$\text{time} \leftarrow \text{time} + 1$

$u.s \leftarrow \text{time}$

for each  $v \in G.\text{Adj}[u]$  do

if  $v.\text{Color} = \text{WHITE}$  then

$v.\pi \leftarrow u$

DFS-VISIT ( $G, v, \text{time}$ )

end if

end for

$u.\text{Color} \leftarrow \text{BLACK}$

$\text{time} \leftarrow \text{time} + 1$

$u.f \leftarrow \text{time}$

end DFS-VISIT

# Topological Sort of Edges

## Topological Sort: - Application of DFS

(20)

→ Linear Ordering of vertices in a Directed Acyclic Graph [DAG]

### Definition:

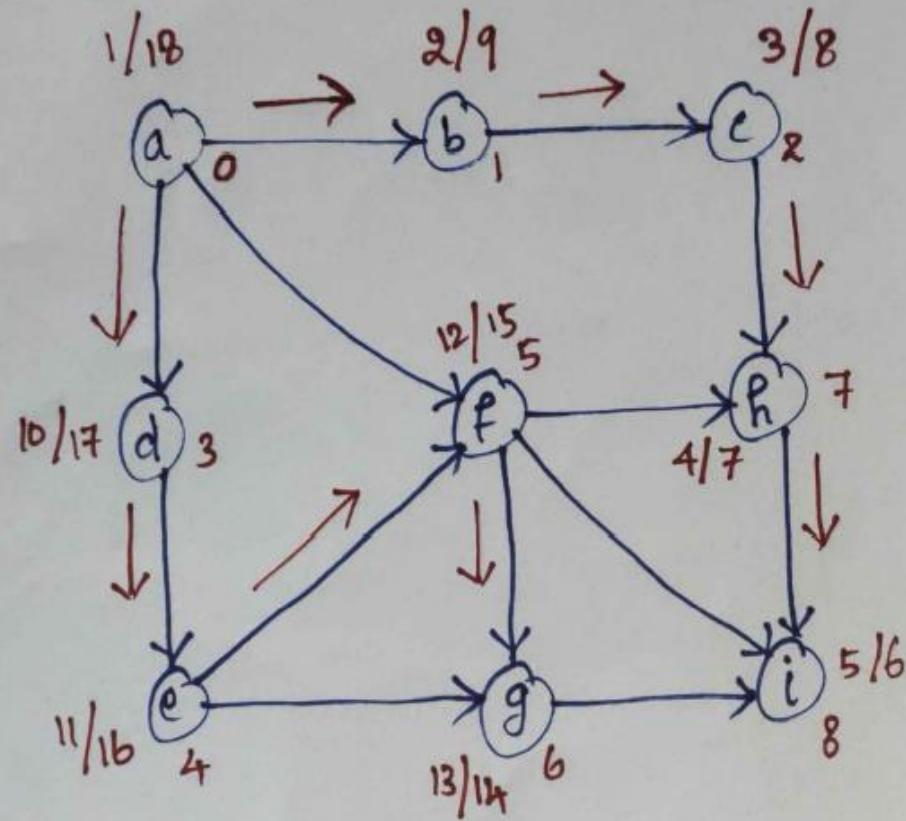
A topological sort of a DAG,  $G = (V, E)$  is a linear ordering of all its vertices such that if  $G$  contains an Edge  $(u, v)$ , then the vertex  $u$  appears before  $v$  in the ordering.

### Algorithm:

Aly. Topological Sort( $G$ )

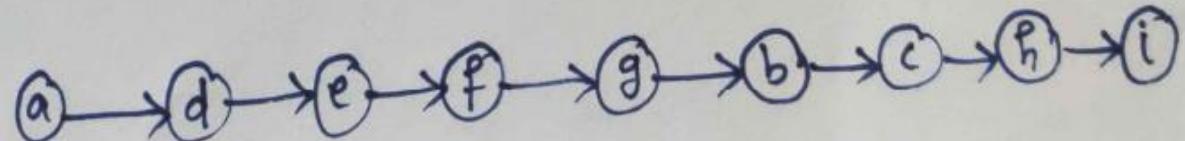
1. Call  $\text{DFS}(G)$  to compute the finishing time for every vertex  $v$ .
2. As each vertex is finished, Insert it onto the front of the linked list.
3. Return the linked List.

### Topological Order Example:



→ Topological order

### List:



## Running Time of Traversal Algorithms:

1. BFS -  $O(|V| + |E|)$

2. DFS -  $O(|V| + |E|)$

3. Topological Sort -  $O(|V| + |E|)$

# Minimum Spanning Tree – Prim's

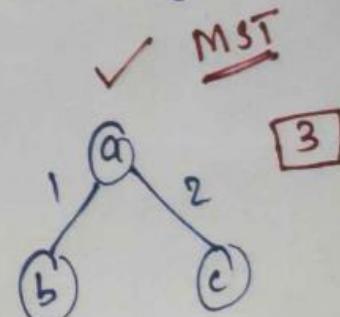
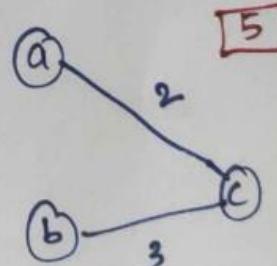
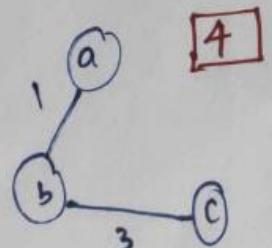
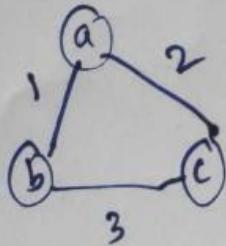
## Minimum Spanning Tree: (of a Graph)

→ is a Tree constructed from a weighted-  
undirected Graph

→ which connects all the vertices of the Graph  
without any cycle.

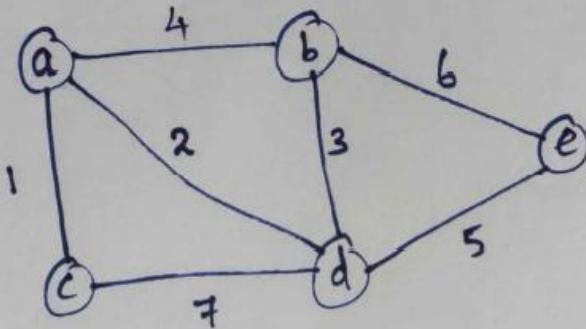
→ The resultant spanning Tree's Cost (Sum of Cost of  
all edges)  
Should be minimum  
Connects 'n' vertices by using ' $n-1$ ' edges.

Example:

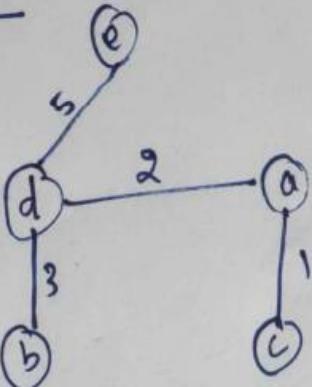


## Algorithms for MST:

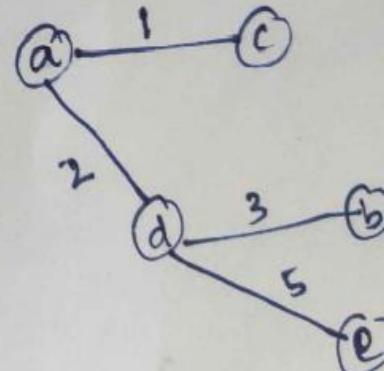
→ Prim's  
→ Kruskal's



Prim's:

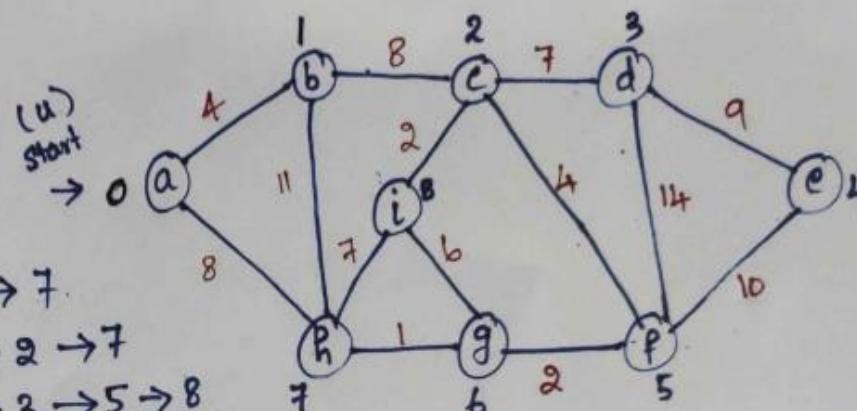


Kruskal's

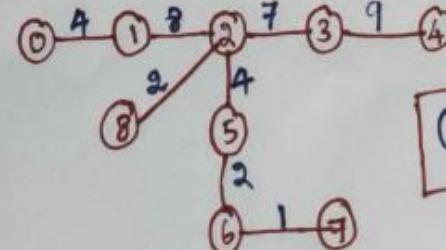


## Prim's Algorithm:

	Value	N	π	Color	Cost	Aj.
1	0	N		W	B	X 0
2	X	0		W	B	X 4
3	X	1		W	B	X 8
4	X	2		W	B	X 7
5	X	3		W	B	X 10
6	X	4		W	B	X 4
7	X	5		W	B	X 62
8	X	6		W	B	X 87
9	X	2		W	B	X 2



205



Cost = 37

W	0	1	2	3	4	5	6	7	8
0	0	4	0	0	0	0	0	8	0
1	4	0	8	0	0	0	0	11	0
2	0	8	0	7	0	4	0	0	2
3	0	0	7	0	9	14	0	0	0
4	0	0	0	9	0	10	0	0	0
5	0	0	4	14	10	0	2	0	0
6	0	0	0	0	0	2	0	1	8
7	8	11	0	0	0	0	1	0	7
8	0	0	2	0	0	0	8	7	0

Alg Prims ( $G_1, w, u^o$ )

for each  $v \in G_1.V$  do

$v.\pi \leftarrow \text{NIL}$   
 $v.\text{color} \leftarrow \text{WHITE}$   
 $v.\text{cost} \leftarrow \infty$

end for

$u.\text{cost} \leftarrow 0$

Let a Min-Priority Queue,  $Q$

$Q \leftarrow G_1.V$

20b

while  $Q \neq \emptyset$  do

$s \leftarrow \text{Extract-Min}(Q)$

for each  $v \in G_1.\text{Adj}[s]$  do

if  $v.\text{color} = \text{WHITE}$  then

if  $w(s, v) < v.\text{cost}$  then

$v.\text{cost} \leftarrow w(s, v)$   
 $v.\pi \leftarrow s$

endif

end if

end for

$s.\text{color} \leftarrow \text{BLACK}$

end while

end Prims

# **Minimum Spanning Tree – Kruskal's**

## Kruskal's Algorithm: [For MST]

Alg. Kruskals ( $G, \omega$ )  $\rightarrow$  Graph ( $V, E$ )  
 weight matrix

$A \leftarrow \emptyset$

for each vertex  $v \in G.V$   
 MAKE-SET( $v$ )

end for

Sort the Edges of  $G$  [ $G.E$ ]  
 into non-decreasing order  
 of its weight

for each edge  $(u,v) \in G.E$  do  
 if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) then

$A \leftarrow A \cup \{(u,v)\}$

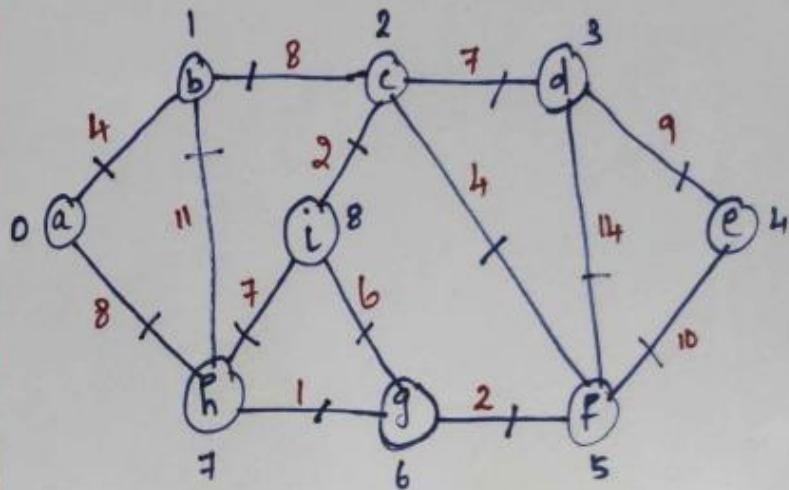
end if UNION( $u,v$ )

end for

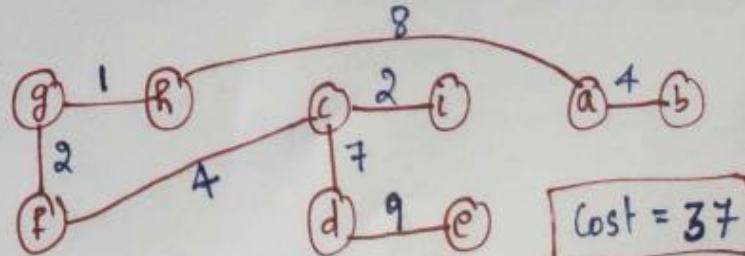
return  $A$

end Kruskals

$(1,2) \rightarrow$  Edge  $A = \{\}$  (207)



$A = \{(g,h), (l,i), (g,f), (a,b), (c,f), (c,d), (a,h), (d,e)\}$



Edge	Connected Components	Decision
.	{a} {b} {c} {d} {e} {f} {g} ✓ {h} ✓ {i}	
(g, h) - 1	{a} {b} {c} {d} {e} {f} {g, h}, {i} *	✓
(l, i) - 2	{a} {b} {c, i}, {d}, {e}, {f} {g, h} *	✓
(g, f) - 2	* {b} {c, i} {d} {e} {f, g, h}	✓
(a, b) - 4	{a, b} {c, i} {d} {e} {f, g, h} *	✓
(c, f) - 4	{a, b} {c, i, f, g, h} {d} {e}	✓
(g, i) - 6	{a, b} {c, i, f, g, h} {d} {e} *	✗
(c, d) - 7	{a, b} {c, i, f, g, h, d} {e} *	✓

Edge	Connected Components	Decision
(h,i) - 7	{a,b} {c,i,f,h,g,d} {e}	X
(a,f) - 8	{a,b,c,i,f,g,h,d} {e}	✓
(b,c) - 8	{a,b,c,i,f,g,h,d} {e}	X
(d,e) - 9	{a,b,c,i,f,g,h,d,e}	✓
(e,f) - 10	{a,b,c,i,f,g,h,d,e}	X
(b,h) - 11	{a,b,c,i,f,g,h,d,e}	X
(d,f) - 14	{a,b,c,i,f,g,h,d,e}	X

