



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

CSE318 - Algorithm Design Strategies & Analysis

Topic

Unit-2

Shri B. Srinivasan

*Assistant Professor-III, School of Computing
SASTRA Deemed To Be University*

Topics

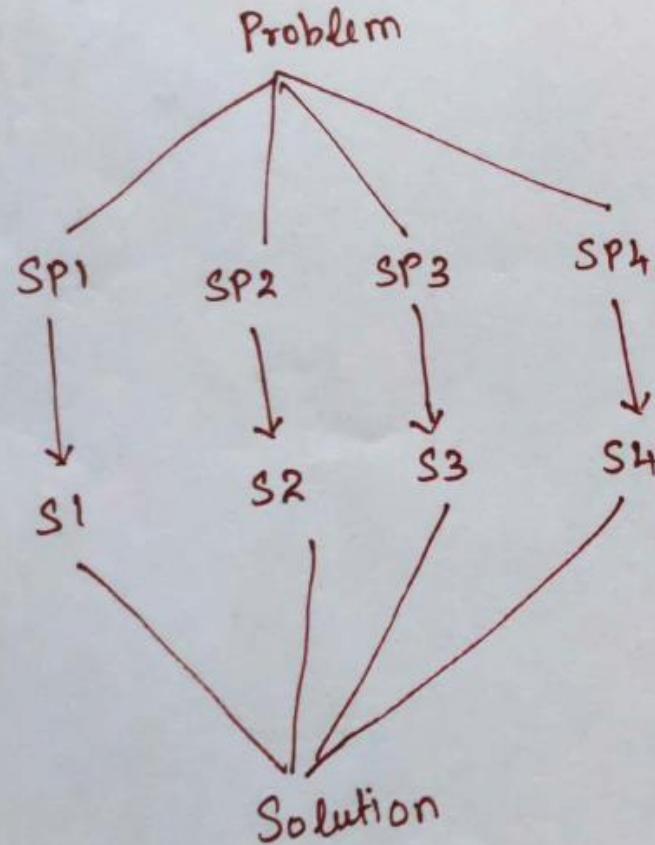
- ✓ **Dynamic Programming Method**
 - ✓ Rod-Cutting Problem
 - ✓ Matrix Chain Ordering
 - ✓ Optimal Binary Search Tree
 - ✓ String Editing Problem
 - ✓ 0/1 Knapsack Problem
 - ✓ Travelling Salesman Problem
- ✓ **Backtracking Method**
 - ✓ n-Queens Problem
 - ✓ Sum of Subsets Problem
 - ✓ Hamiltonian Cycles
 - ✓ Knapsack Problem
- ✓ **Branch and Bound Method**
 - ✓ 0/1 Knapsack Problem
 - ✓ Travelling Salesman Problem

Dynamic Programming Approach

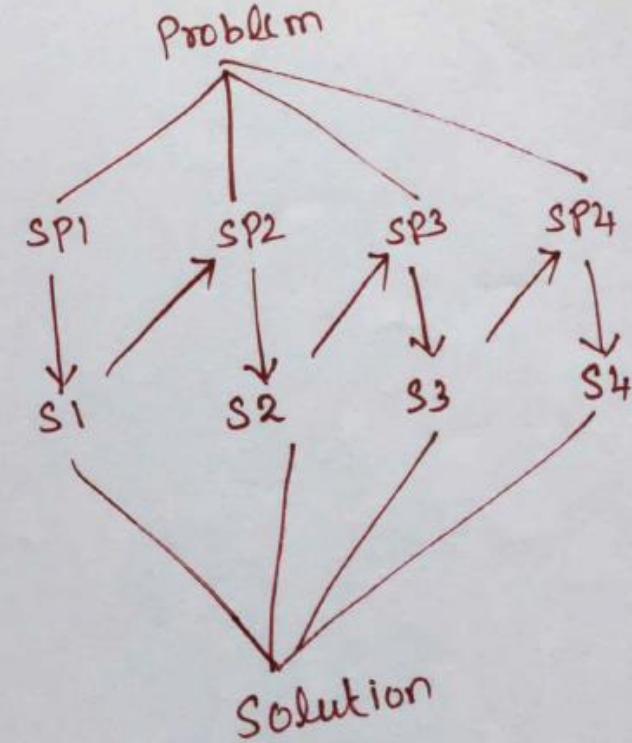
(Memorization / Tabular)

Dynamic Programming:

Divide & Conquer



Dynamic Programming



- Dynamic Programming is like Divide & Conquer method solves the problem by combining the solutions to the sub problems.
- Divide & Conquer algorithm partitions the problem into disjoint subproblems. Solve the sub problems recursively & combine the solutions. But Dynamic programming applies when Sub-problem overlaps.
i.e., when sub problems share sub problem.
- Dynamic Programming algorithm solves each sub problem just once and then saves its answer in a Table, thereby avoiding the work of recomputing the answer every time.

Dynamic Programming Approach

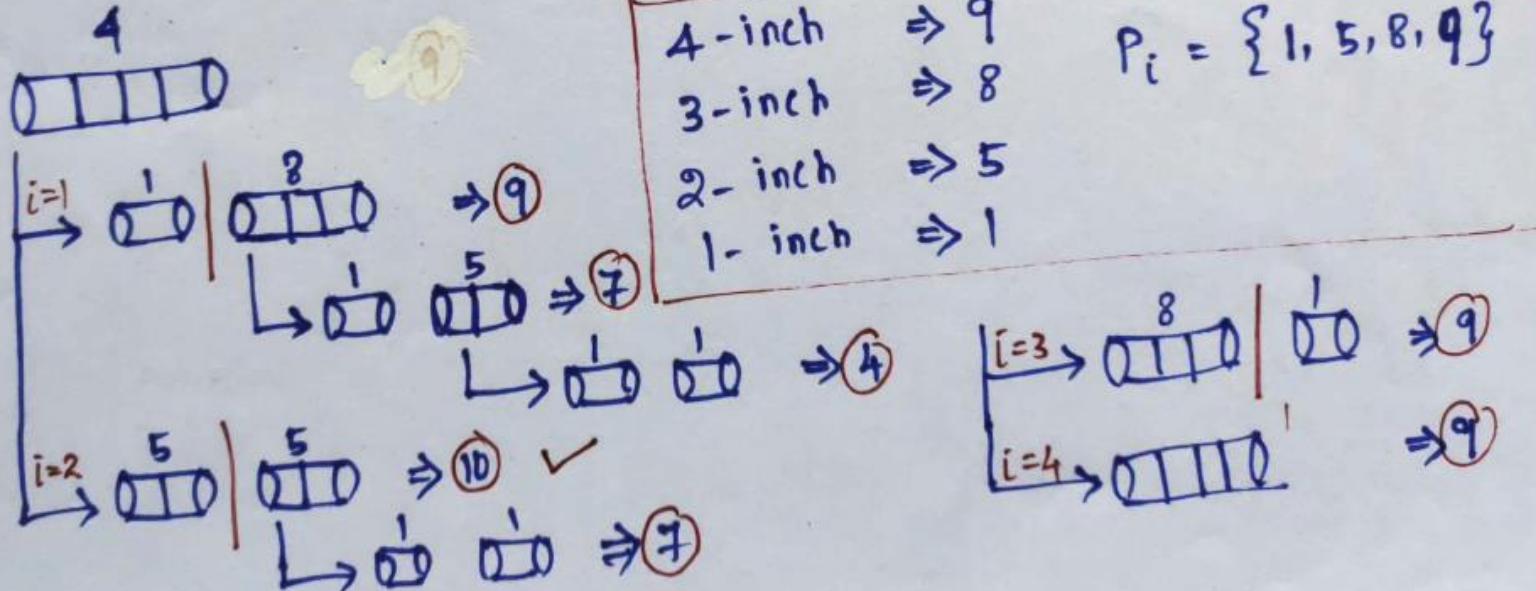
Rod-Cutting Problem

Rod Cutting Problem:

Input: A rod of length ' n ' inches and
 A price list, P_i , $i = 1, 2, \dots, n$

Output: Maximum Revenue, r_n , Obtainable by
 cutting up the rod and selling the pieces.

$$n=4$$



3 Algorithms:

$\rightarrow O(2^n)$

1. Top-down - Recursive - Divide & Conquer
2. Top-down - Memorization
3. Bottom-up.

Dynamic Programming

1. Top-down Recursive:

\rightarrow Return max Rev in

Alg CutRod1 ($P[1..n]$, n)

if $n = 0$ then
return 0.
endif

$q \leftarrow -\infty$

for $i \leftarrow 1$ to n do

$r \leftarrow \text{CutRod1}(P, n-i)$

if $P[i] + r > q$ then

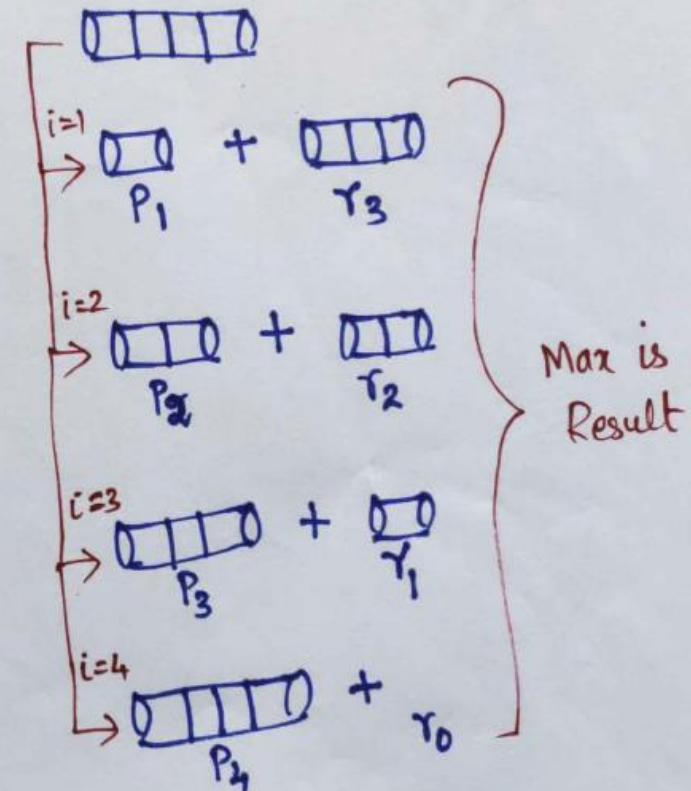
$q \leftarrow P[i] + r$

endif

end for

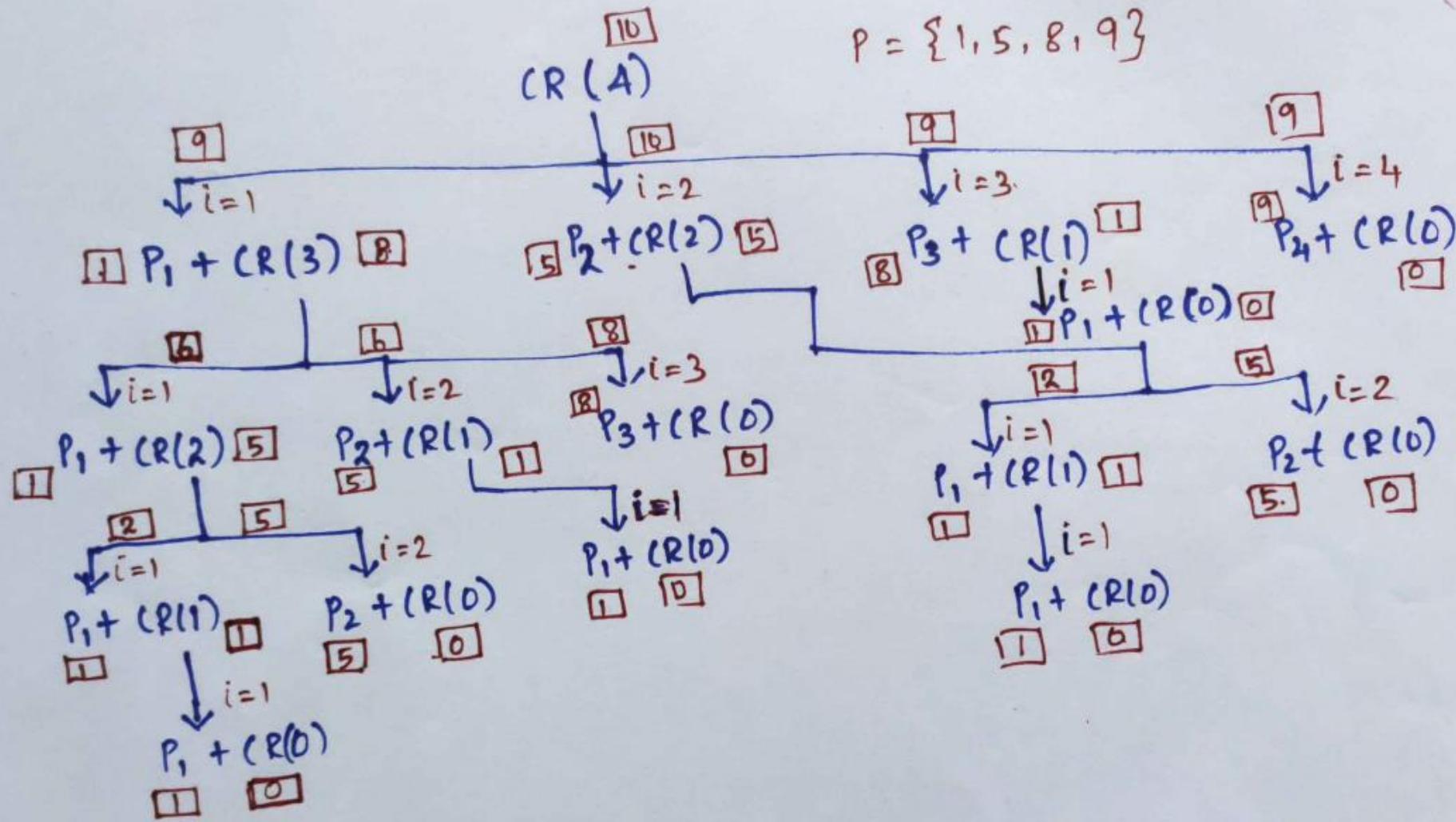
return q

end CutRod1



(21)

$$P = \{1, 5, 8, 9\}$$



2. Top-Down - Memorization : (Rod-cutting Problem)

(220)

Alg CutRod2($P[1..n]$, n)

Let $\gamma[0..n]$ be an array

for $i \leftarrow 0$ to n do

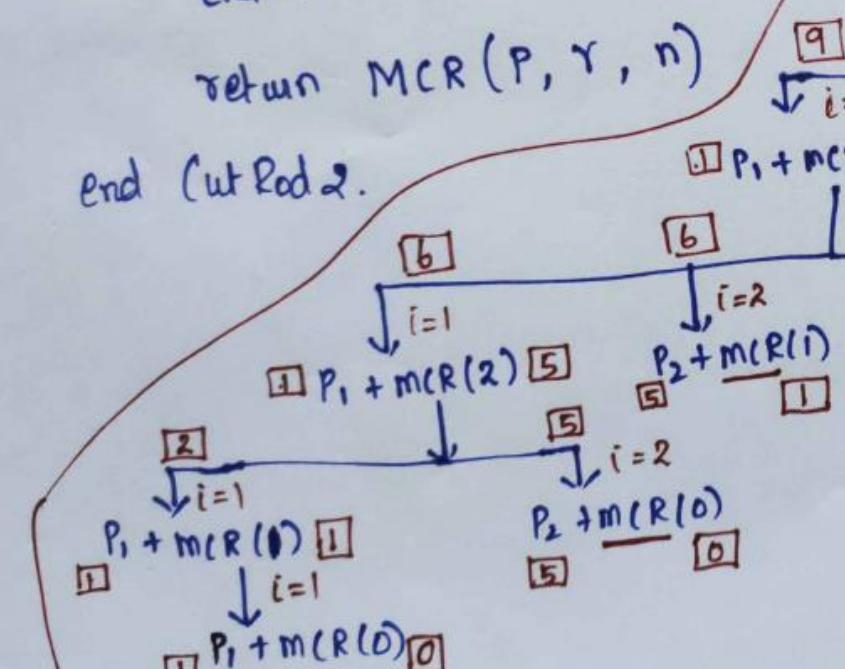
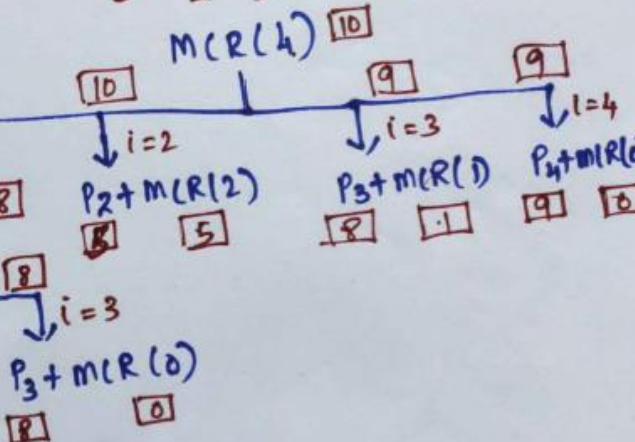
$\gamma[i] \leftarrow -\infty$

end for

return $MCR(P, \gamma, n)$

end CutRod2.

P	0	1	2	3	4
γ	- ∞				
	0	1	5	8	10



Alg. MCR ($P[1..n]$, $r[0..n]$, n)
if $r[n] \geq 0$ then
 return $r[n]$
end if
if $n = 0$ then
 $q \leftarrow 0$
else
 $q \leftarrow -\infty$
 for $i \leftarrow 1$ to n do
 $r \leftarrow MCR(P, r, n-i)$
 if $P[i] + r > q$ then
 $q \leftarrow P[i] + r$
 end if
 end for
end if
 $r[n] \leftarrow q$
return q
end MCR

3. Bottom-up - Dynamic Programming: (Rod-cutting Problem)

Aly CutRod3 ($P[1..n]$, n)

Let $r[0..n]$ be an array

$r[0] \leftarrow 0$

for $j \leftarrow 1$ to n do

$q \leftarrow -\infty$

for $i \leftarrow 1$ to j do

$t \leftarrow r[j-i]$

if $P[i] + t > q$ then
 $q \leftarrow P[i] + t$

end if

end for

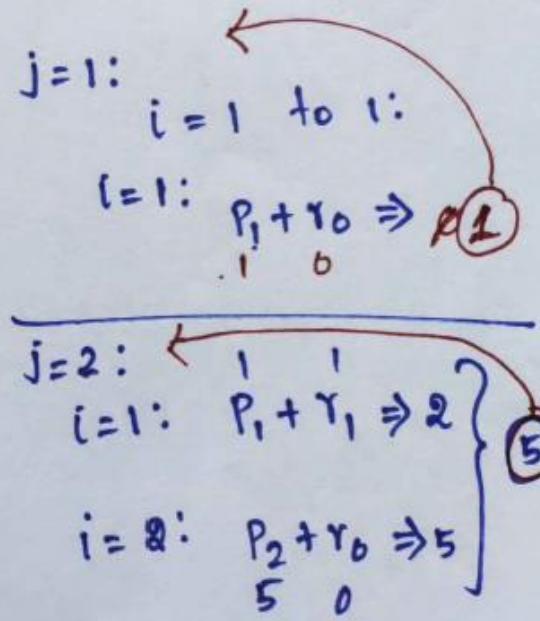
$r[j] \leftarrow q$

end for

return $r[n]$

end CutRod3

	0	1	2	3	4
P		1	5	8	9
r	0	1	5	8	10



j=3:

$$i=1: \quad P_1 + \gamma_2 = 6 \quad \left. \begin{array}{l} 1 \\ 5 \\ \hline 6 \end{array} \right\} 8$$

$$i=2: \quad P_2 + \gamma_1 = 6 \quad \left. \begin{array}{l} 5 \\ 1 \\ \hline 6 \end{array} \right\} 8$$

$$i=3: \quad P_3 + \gamma_0 = 8 \quad \left. \begin{array}{l} 8 \\ 0 \\ \hline 8 \end{array} \right\} 8$$

j=4:

$$i=1: \quad P_1 + \gamma_3 \Rightarrow 9 \quad \left. \begin{array}{l} 1 \\ 8 \\ \hline 9 \end{array} \right\} 9$$

$$i=2: \quad P_2 + \gamma_2 = 10 \quad \left. \begin{array}{l} 5 \\ 5 \\ \hline 10 \end{array} \right\} 10$$

$$i=3: \quad P_3 + \gamma_1 = 9 \quad \left. \begin{array}{l} 8 \\ 1 \\ \hline 9 \end{array} \right\} 9$$

$$i=4: \quad P_4 + \gamma_0 = 9 \quad \left. \begin{array}{l} 9 \\ 0 \\ \hline 9 \end{array} \right\} 9$$

Dynamic Programming Approach

Matrix Chain Ordering

Ordering Matrix Multiplication: (Example for Dynamic Programming)

$A_1 \quad A_2 \quad A_3$
 $5 \times 3 \quad 3 \times 4 \quad 4 \times 2$

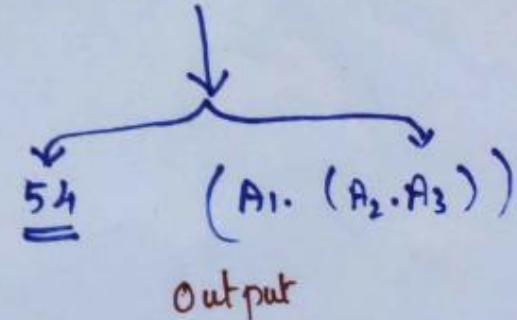
$$\begin{aligned}
 & \left(\left(\begin{array}{c} 5 \times 3 \\ A_1 \end{array} \cdot \begin{array}{c} 3 \times 4 \\ A_2 \end{array} \right) \cdot \begin{array}{c} A_3 \\ 4 \times 2 \end{array} \right) \xrightarrow{5 \times 3 \times 4} 60 \\
 & \left(\begin{array}{c} R_1 \\ 5 \times 4 \end{array} \right) \cdot \begin{array}{c} A_3 \\ 4 \times 2 \end{array} \xrightarrow{5 \times 4 \times 2} 40 \quad \} 100 \\
 & \left(A_1 \cdot \left(\begin{array}{c} 3 \times 4 \\ A_2 \cdot A_3 \end{array} \right) \right) \xrightarrow{3 \times 4 \times 2} 24 \\
 & \left(\begin{array}{c} A_1 \\ 5 \times 3 \end{array} \cdot \left(\begin{array}{c} R_1 \\ 3 \times 2 \end{array} \right) \right) \xrightarrow{5 \times 3 \times 2} 30 \quad \} 54 \quad \checkmark
 \end{aligned}$$

2 $n \times n$ $\frac{O(n^3)}{9}$
 3×3

$m \times n$ $\frac{n \times r}{m \times r} \frac{O(m \cdot n \cdot r)}{}$

INPUT

$A_1 \quad A_2 \quad A_3$
 $5 \times 3 \quad 3 \times 4 \quad 4 \times 2$
 $P [\begin{matrix} 5 & 3 & 4 & 2 \end{matrix}]$



(225)

$$\begin{array}{cccc}
 A_1 & A_2 & A_3 & A_4 \\
 5 \times 3 & 3 \times 4 & 4 \times 2 & 2 \times 6 \\
 \left[\begin{matrix} 5 & 3 & 4 & 2 & 6 \\ 0 & 1 & 2 & 3 & 4 \end{matrix} \right] P
 \end{array}$$

$n=4$

$m[i, j] \Rightarrow$ Min. No. of multiplications
needed to multiply the
matrices A_i, A_{i+1}, \dots, A_j

i^{th} Matrix $\begin{cases} \text{Row} \rightarrow i-1 \\ \text{Column} \rightarrow i \end{cases}$

$$m[i, j] = ?$$

$l=4$

$$\begin{array}{c}
 K=1 \rightarrow \begin{array}{c|ccccc}
 & 5 \times 3 & & 3 \times 6 & & \\
 A_1 & A_2 & A_3 & A_4 & & \\
 \hline
 m[1, 1] & & m[2, 4]
 \end{array} \rightarrow
 \end{array}$$

$$m[1, 1] + m[2, 4] + P_0 P_1 P_4$$

$\underset{l=1}{?} \quad \underset{l=3}{?} \quad \underset{l=4}{5 \cdot 3 \cdot 6}$

$$\begin{array}{c}
 K=2 \rightarrow \begin{array}{c|ccccc}
 & 5 \times 4 & & 4 \times 6 & & \\
 A_1 & A_2 & A_3 & A_4 & & \\
 \hline
 m[1, 2] & & m[3, 4]
 \end{array} \rightarrow
 \end{array}$$

$$m[1, 2] + m[3, 4] + P_0 P_2 P_4$$

$\underset{l=2}{?} \quad \underset{l=2}{?} \quad \underset{l=4}{5 \cdot 4 \cdot 6}$

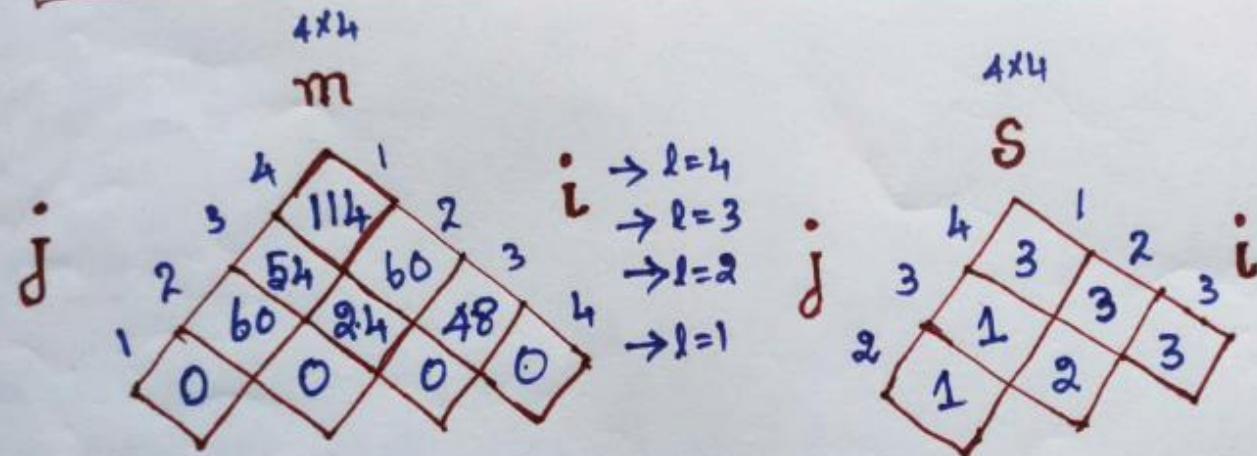
$$\begin{array}{c}
 K=3 \rightarrow \begin{array}{c|ccccc}
 & 5 \times 2 & & 2 \times 6 & & \\
 A_1 & A_2 & A_3 & A_4 & & \\
 \hline
 m[1, 3] & & m[4, 4]
 \end{array} \rightarrow
 \end{array}$$

$$m[1, 3] + m[4, 4] + P_0 P_3 P_4$$

$\underset{l=3}{?} \quad \underset{l=1}{?} \quad \underset{l=4}{5 \cdot 2 \cdot 6}$

Min ?
↓
Answer

$$m[i, j] = \min_{k=i, i+1, \dots, j-1} \left\{ m[i, k] + m[k+1, j] + p_{i-1}, p_k, p_j \right\}$$



$l=1$:

$$m[1, 1] = 0$$

$$m[2, 2] = 0$$

$$m[3, 3] = 0$$

$$m[4, 4] = 0$$

$l = 2$:

$\overline{A_1 A_2}$	$\overline{A_2 A_3}$	$\overline{A_3 A_4}$	(22)
i	i	i	

$$m[i, j] \xrightarrow{k=1} m[1, 1] + m[2, 2] + P_0 P_1 P_2 \longrightarrow 60$$

$$m[2, 3] \xrightarrow{k=2} m[2, 2] + m[3, 3] + P_1 P_2 P_3 \longrightarrow 24$$

$$m[3, 4] \xrightarrow{k=3} m[3, 3] + m[4, 4] + P_2 P_3 P_4 \longrightarrow 48$$

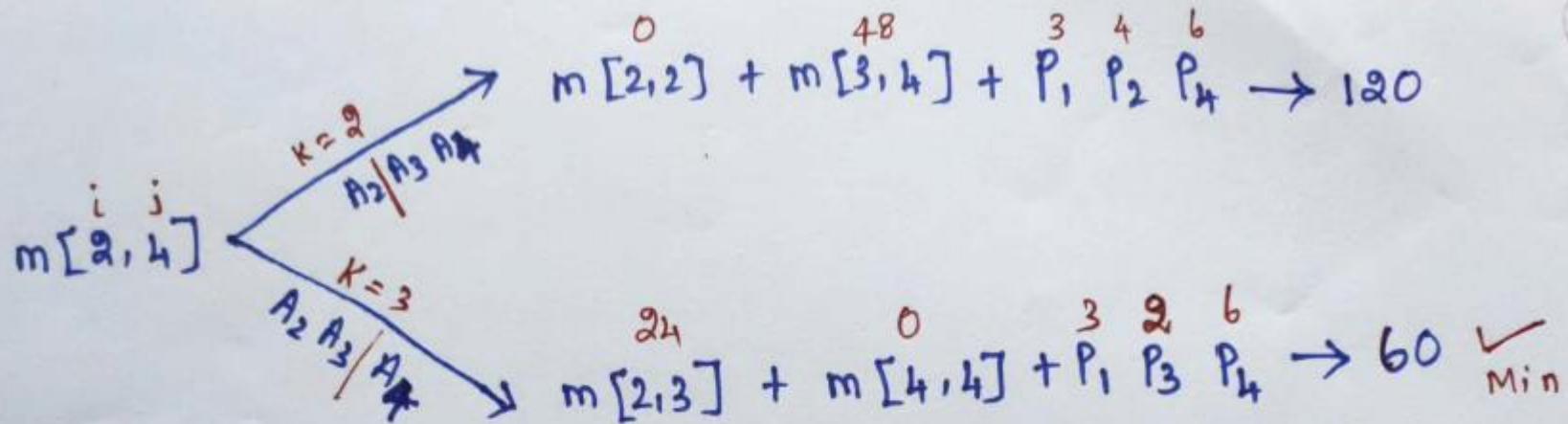
$l = 3$:

$\overline{A_1 A_2 A_3}$	$\overline{A_2 A_3 A_4}$	
i	i	

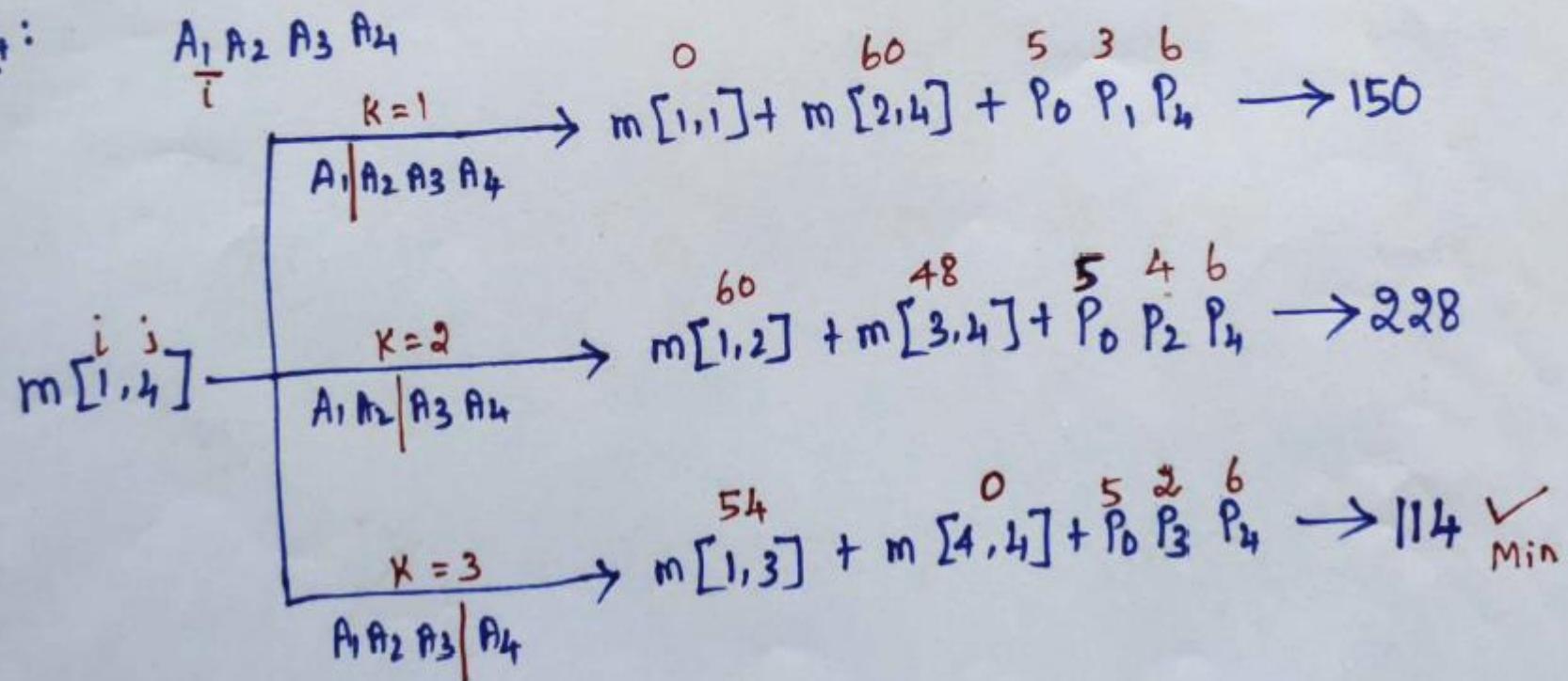
$$m[i, j] \xrightarrow{k=1} A_1 | A_2 A_3 \longrightarrow m[1, 1] + m[2, 3] + P_0 P_1 P_3 \longrightarrow 54 \checkmark_{\text{Min}}$$

$$\xrightarrow{k=2} A_1 A_2 | A_3 \longrightarrow m[1, 2] + m[3, 3] + P_0 P_2 P_3 \longrightarrow 100$$

228



$l=4$:



Alg MCO (P[0..n])

Let $m[i..n, i..n]$ and
 $s[i..n, i..n]$ be matrices

for $i \leftarrow 1$ to n do

$m[i,i] \leftarrow 0$

end for

for $\lambda \leftarrow 2$ to n do

 for $i \leftarrow 1$ to $n-\lambda+1$ do

$j \leftarrow i+\lambda-1$

$m[i,j] \leftarrow \alpha$

 for $k \leftarrow i$ to $j-1$ do

$q \leftarrow m[i,k] + m[k+1,j] + P[i-1]*P[k]*P[j]$

 if $q < m[i,j]$ then

$m[i,j] \leftarrow q$

$s[i,j] \leftarrow k$

 end if

 end for

 end for

End for

Alg PrintMCO (s, i, j)

if $i = j$ then

 print 'A'i

else

 print '('

 PrintMCO (s, i, s[i,j])

 PrintMCO (s, s[i,j]+1, j)

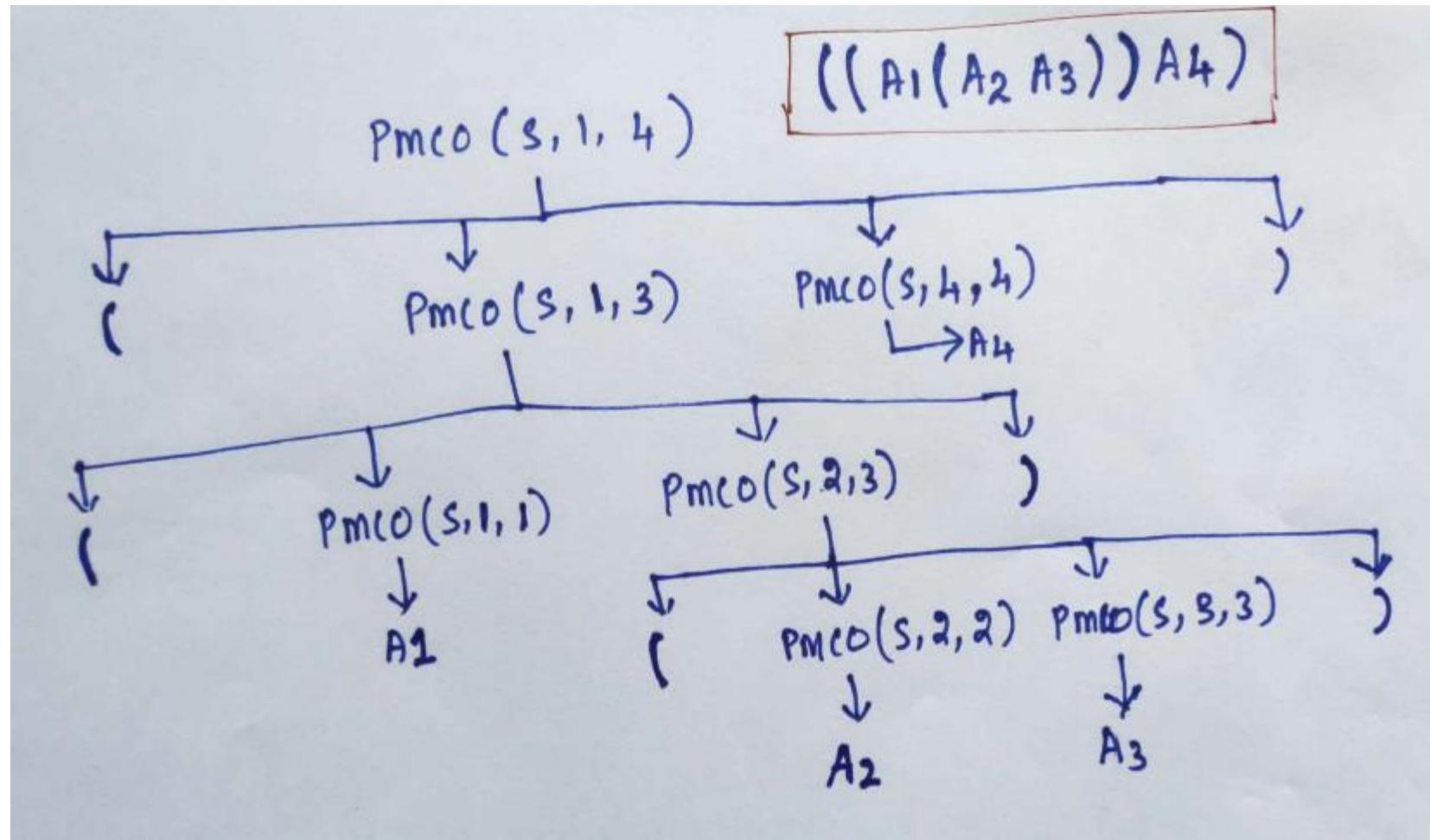
 print ')'

endif

end PrintMCO

return m & s

end MCO



Dynamic Programming vs Greedy Algorithm

Dynamic Programming first solve the smaller sub problems and store the solutions in a table. so that the rest of the sub problems with larger size may use this stored solution.

But in Greedy approach, the smaller sub problems solutions will not be recorded. Instead, at each step, it will choose the Best Solution (Greedy choice). This solution will be used in

constructing the solution for the main problem. At the end of the last step, the optimal solution will be obtained.

Greedy Algorithms:

A	B	C	D
5x3	3x4	4x2	2x6

$\ell=2$: AB BC CD

$\ell=2$: AB BC CD

$\ell=3$: ABC BCD

```

    ABC      BCD
   / \     / \
  (AB)C (A(B)) (BC)D B(CD)
  
```

$\ell=4$: ABCD

$\ell=3$: A(BC) (BC)D

$\ell=3$: (A.(BC)).D

A	B	C	D
101x11	11x9	9x100	100x99

DP
((AB)(CD))

Multiplications = 189090

Greedy
(A · ((BC) · D))

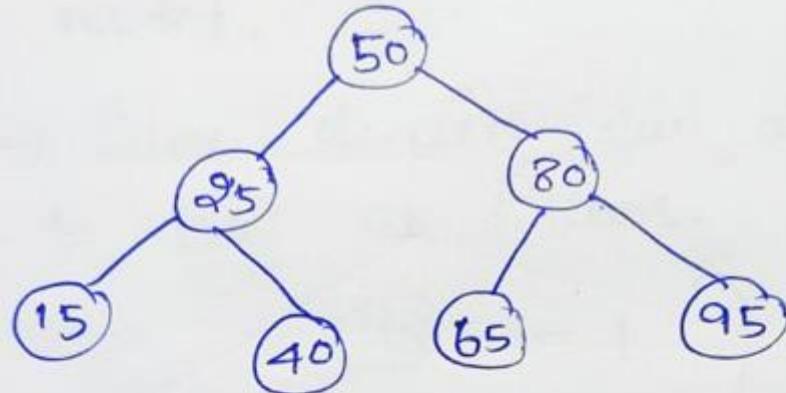
No. of Mult = 227989

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

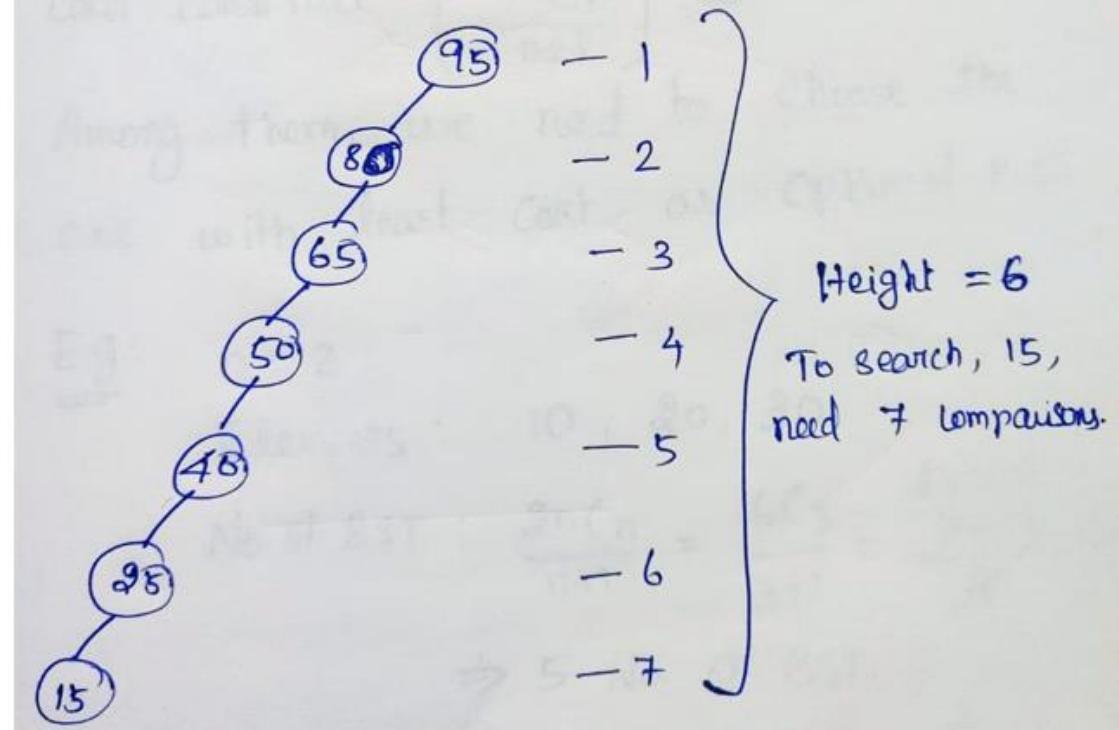
- No. of Comparisons needed for searching an element.
- It depending on the height of the tree.
- If the tree is balanced, then the height is $\log_2 n$.

→ 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.

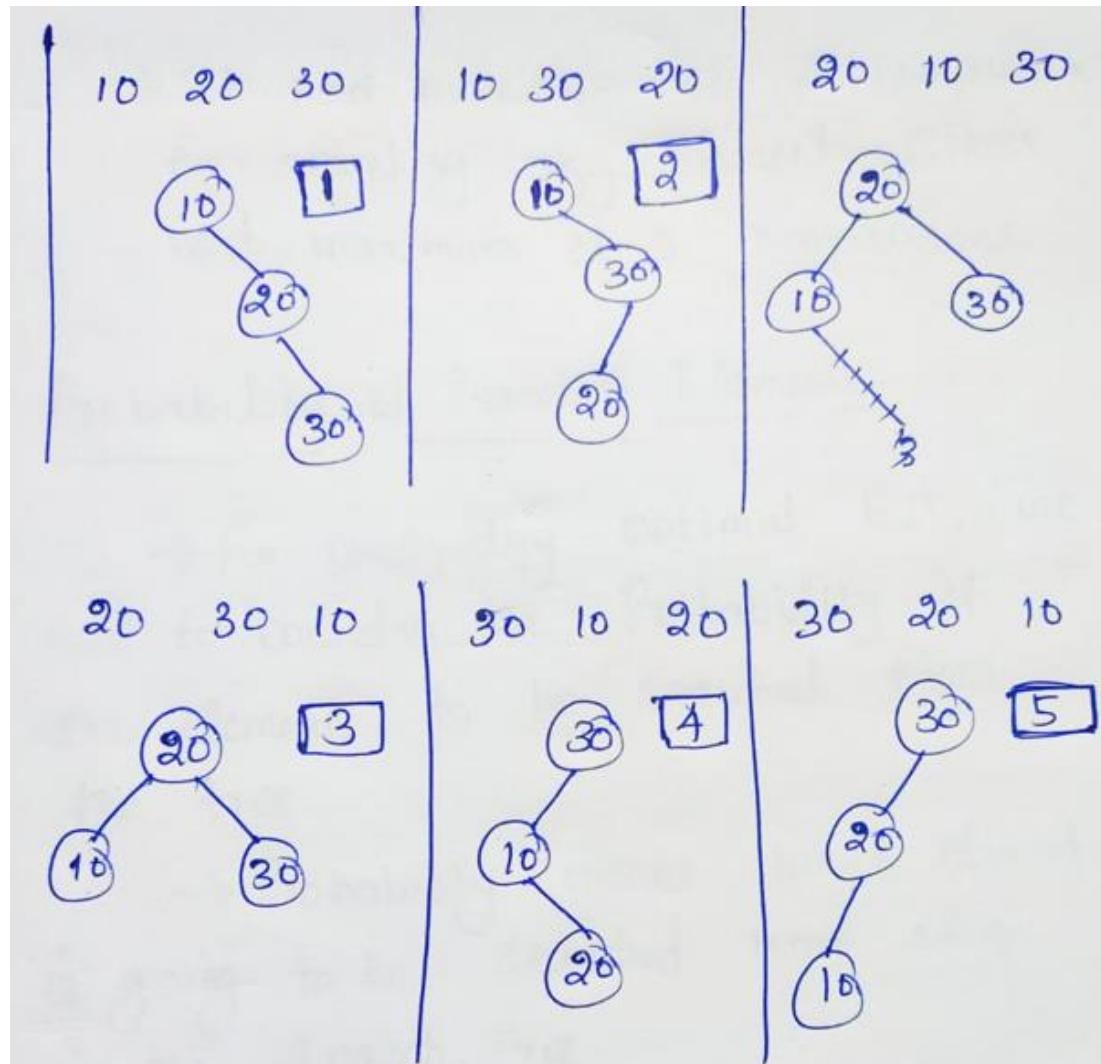
For a given 'n' elements, we can construct $\binom{2^n C_n}{n+1}$ numbers of BST. Among them, we need to choose the one with least cost as Optimal BST.

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.

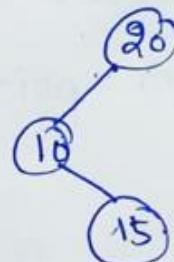


Probability of Searching Elements:

→ 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)

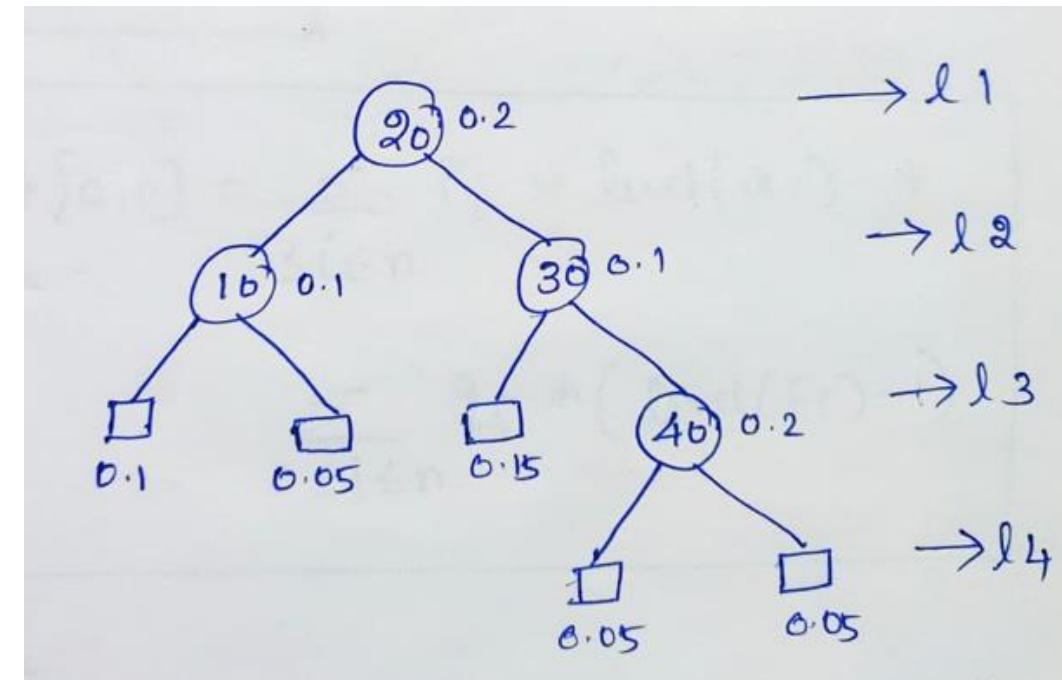
	1	2	3	4
(a) Keys :	10	20	30	40

• 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:

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

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.

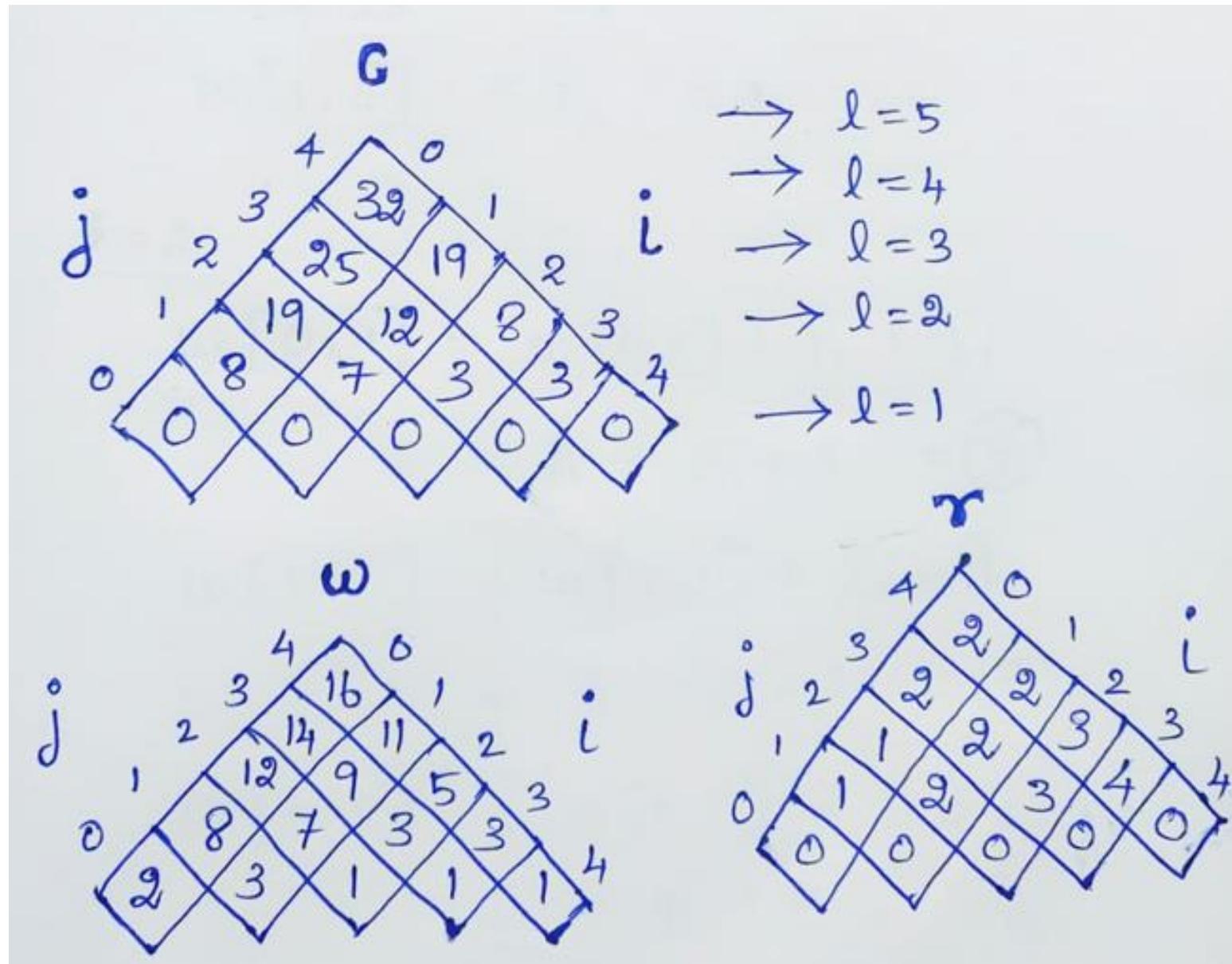


Example For Optimal BST:

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. $C[2,3]$:

$$C[2,3] \xrightarrow{k=3} C[2,2] + C[3,3] + W[2,3]$$
$$0 + 0 + 3$$

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

4. $C[3,4]$

$$C[3,4] \xrightarrow{k=4} C[3,3] + C[4,4] + W[3,4]$$
$$0 + 0 + 3$$

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

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

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

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

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

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

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

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



$$C[2,4] \xrightarrow{\text{min}} \begin{array}{l} K=3 \rightarrow C[2,2] + C[3,4] + w[2,4] \\ 0 + 3 + 5 = 8 \end{array}$$
$$K=4 \rightarrow C[2,3] + C[4,4] + w[2,4]$$
$$3 + 0 + 5 = 8$$

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

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

$$C[0,3] \xrightarrow{\text{min}} \begin{array}{l} K=1 \rightarrow C[0,0] + C[1,3] + w[0,3] \\ 0 + 12 + 14 = 26 \end{array}$$
$$K=2 \rightarrow C[0,1] + C[2,3] + w[0,3] \xrightarrow{\text{min}} 8 + 3 + 14 = 25$$
$$K=3 \rightarrow C[0,2] + C[3,3] + w[0,3] \\ 19 + 0 + 14 = 33$$

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

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

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.

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

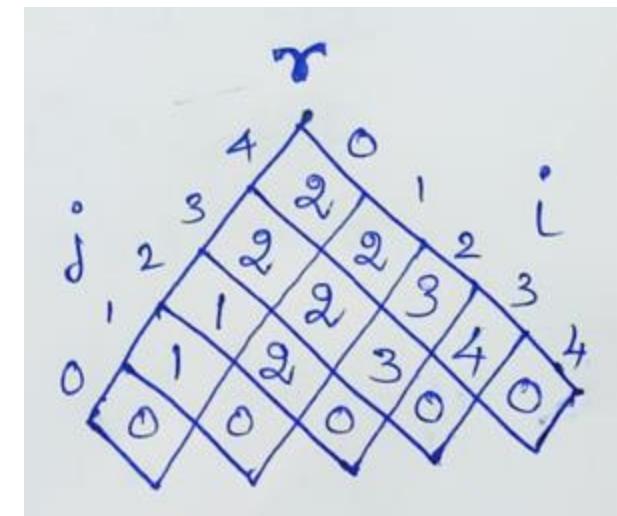
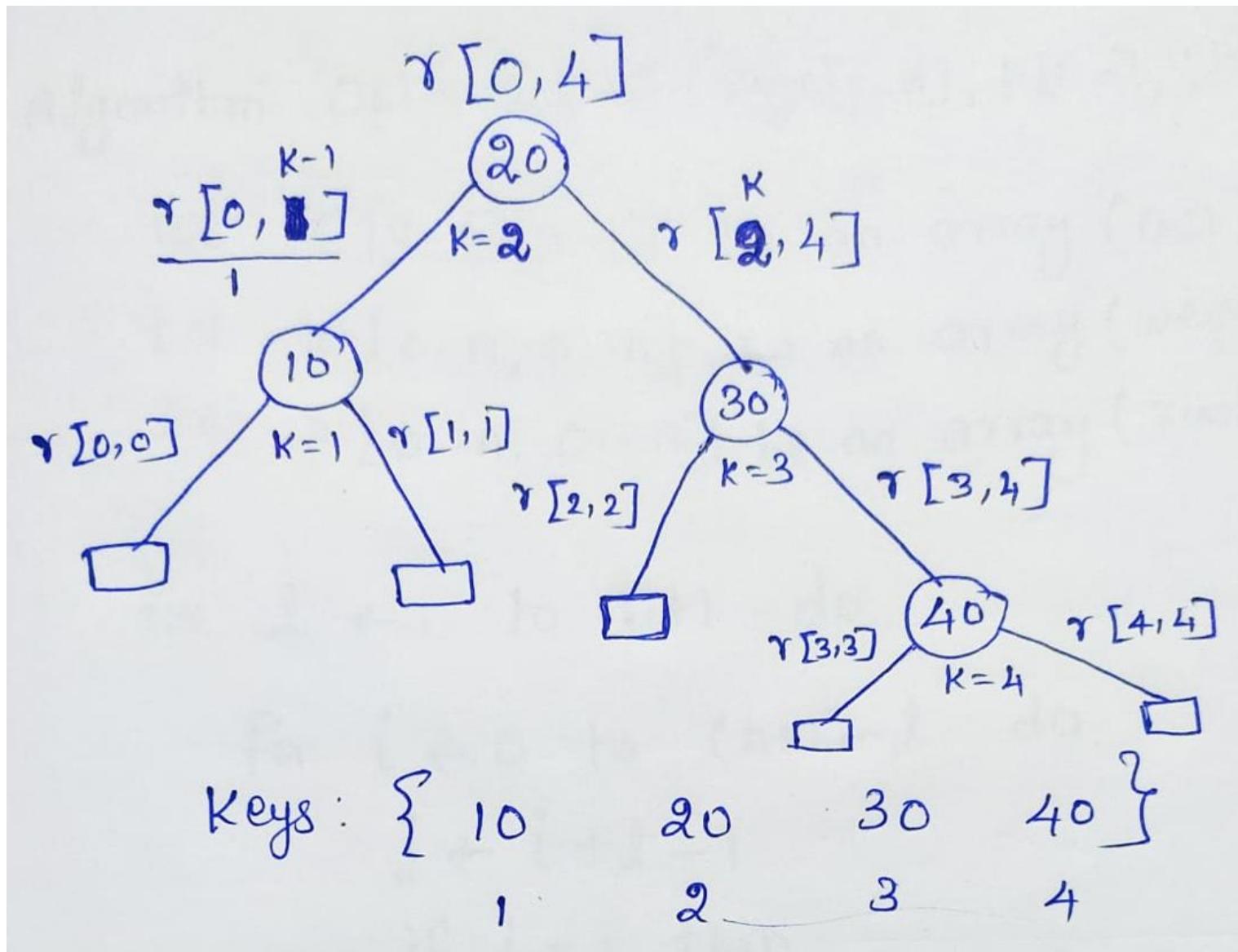
Minimum Cost of BST : $c[0,4] = 2$



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

$$r[0,4] = 2$$

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





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:

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

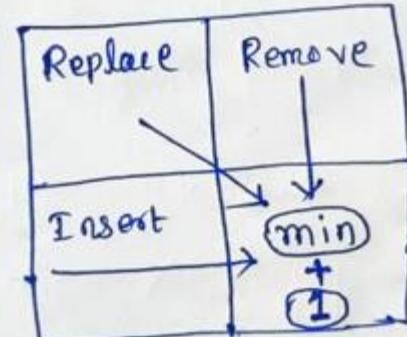
$$Y = abc Fg \quad m=5$$

Symbols:

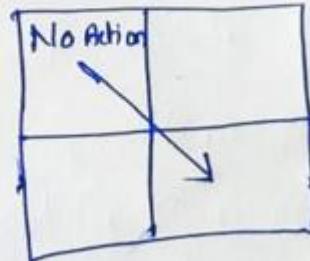
- Insert
- ↓ Remove
- Replace

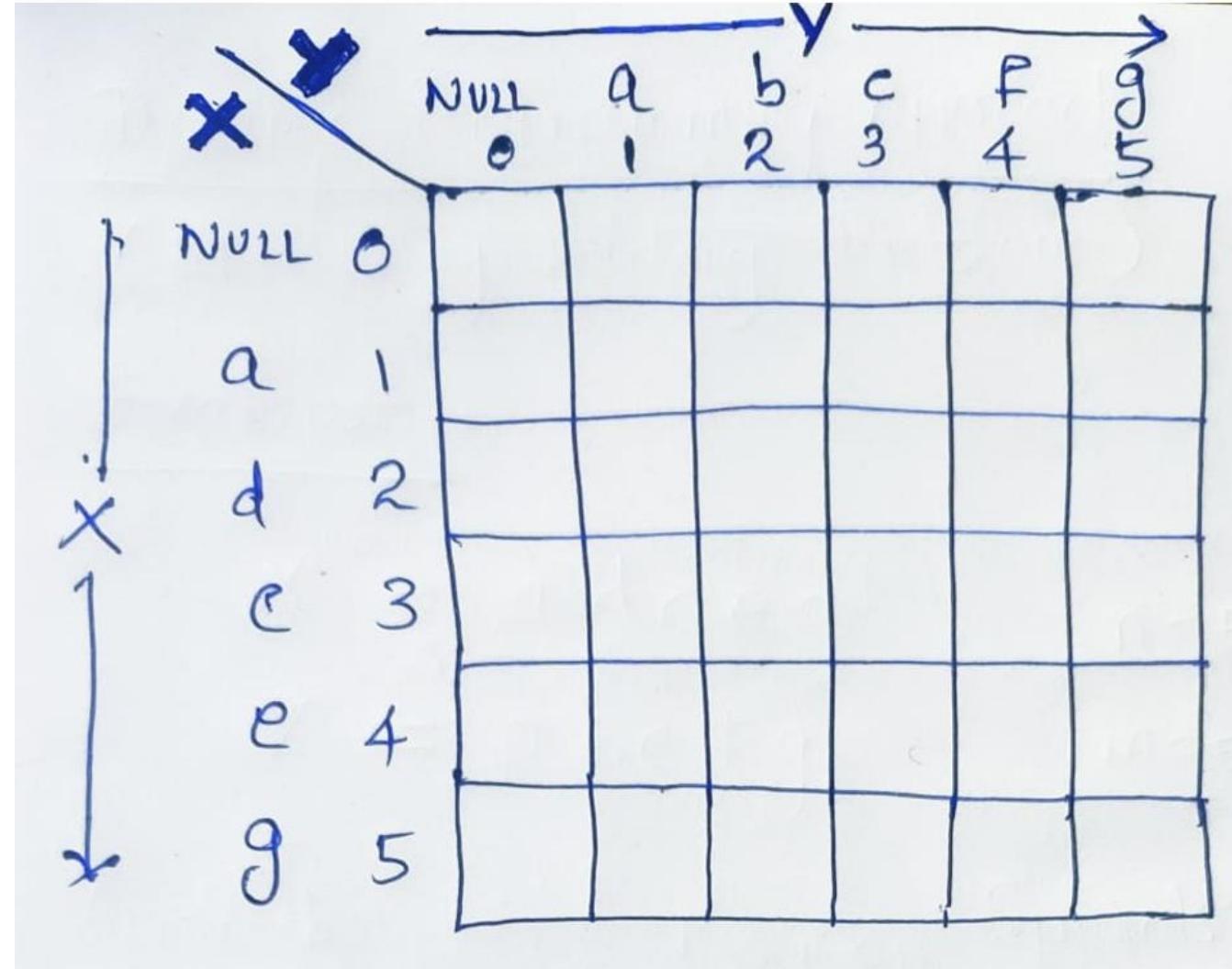
Approach:

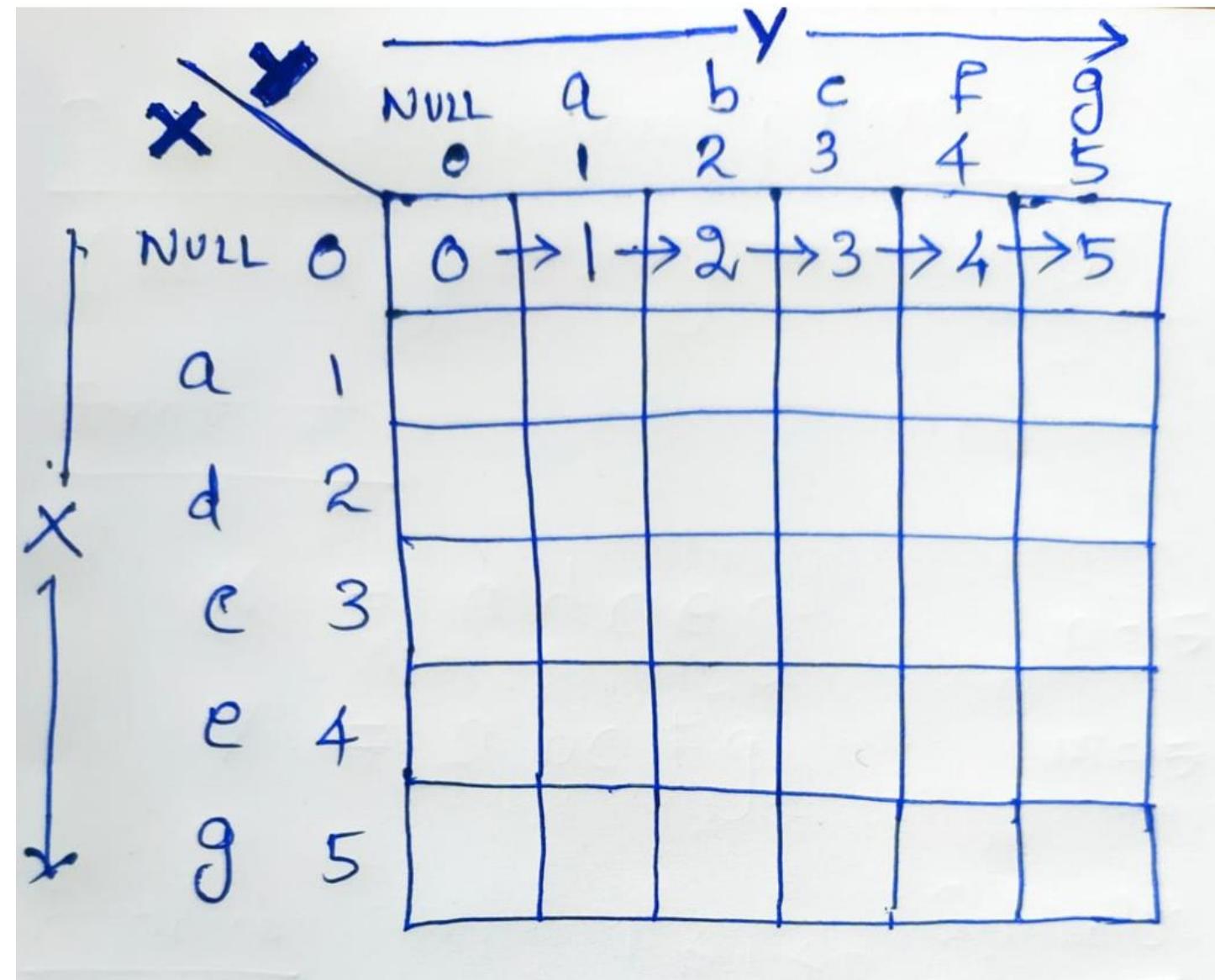
1. If $\gamma \neq c$

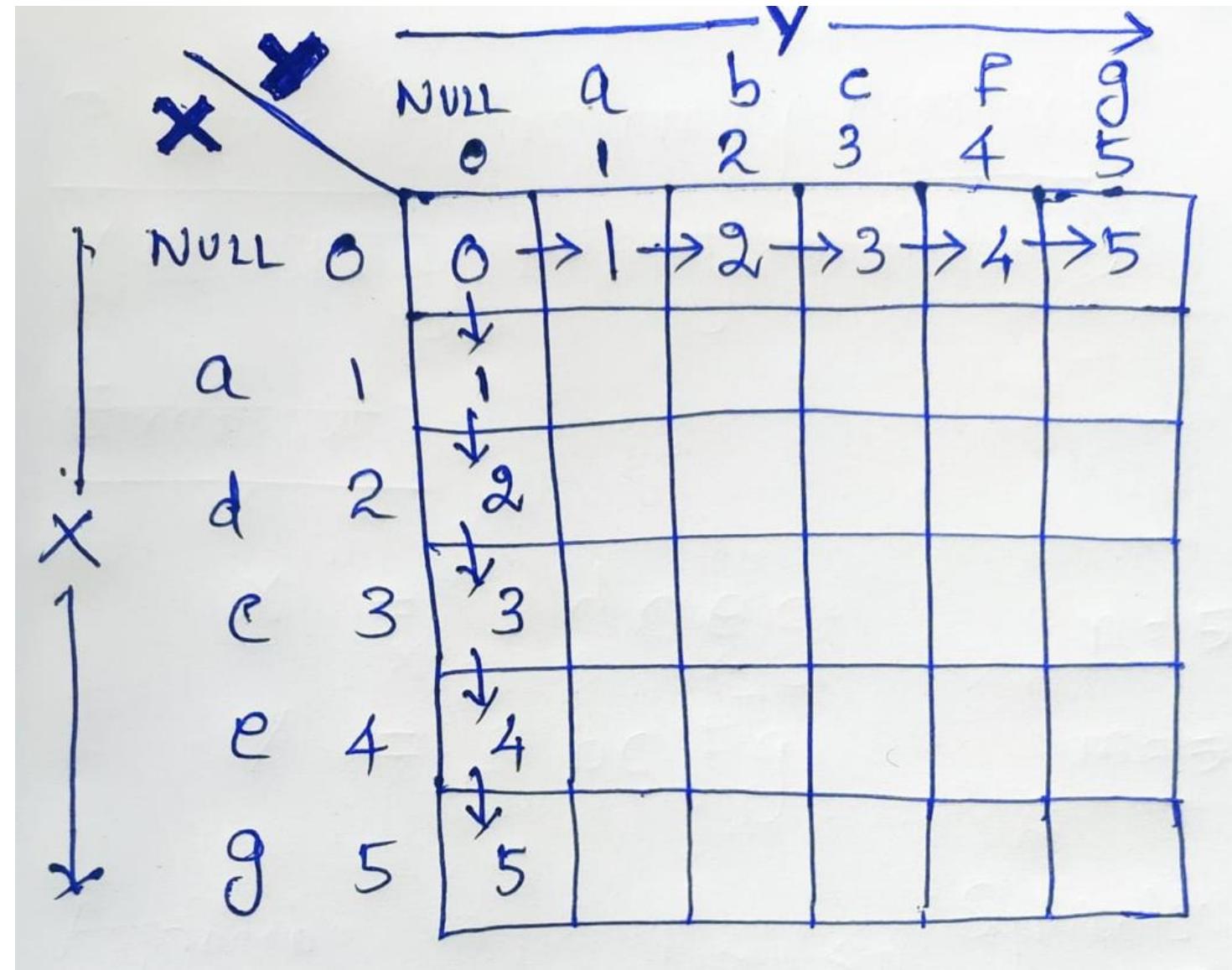


2. If $\gamma = c$









$m[1,1]$: $r = 'a'$ $c = 'a'$

2. $r = c$: No Action Required.

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

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

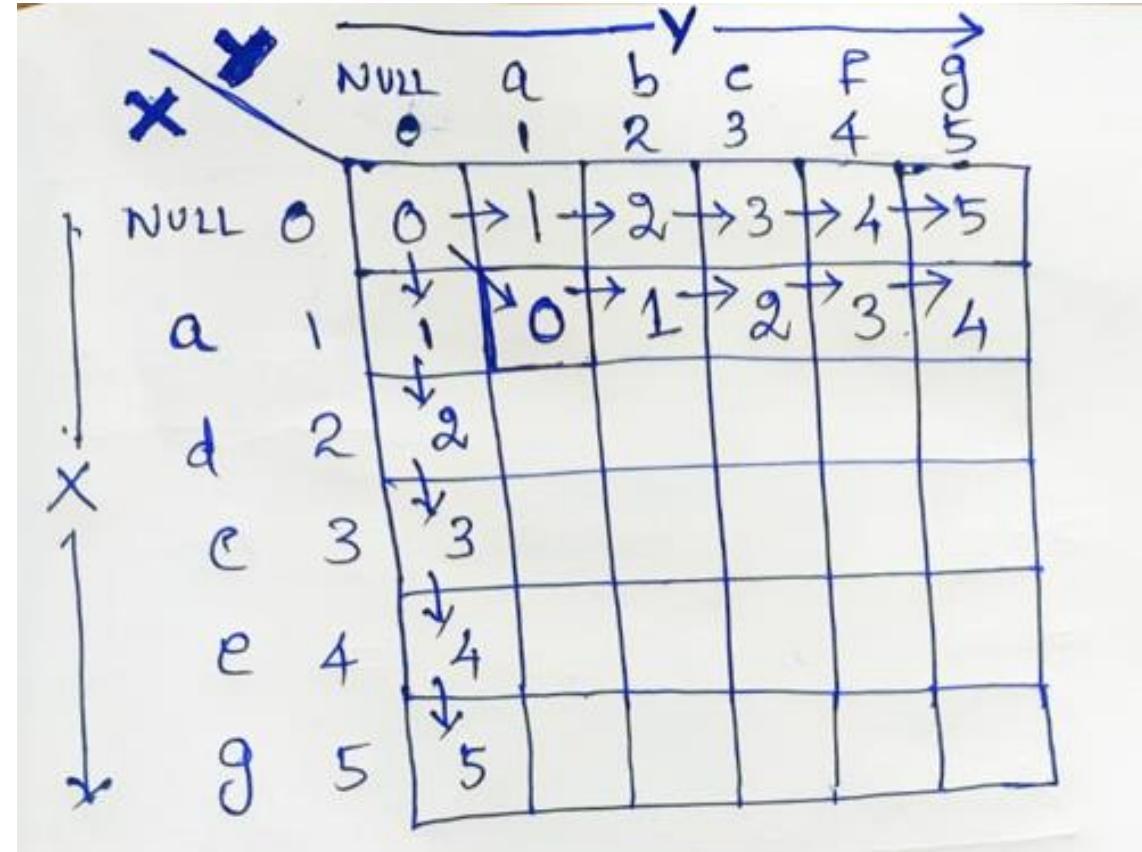
$+1$	1	2	0
------	-----	-----	-----

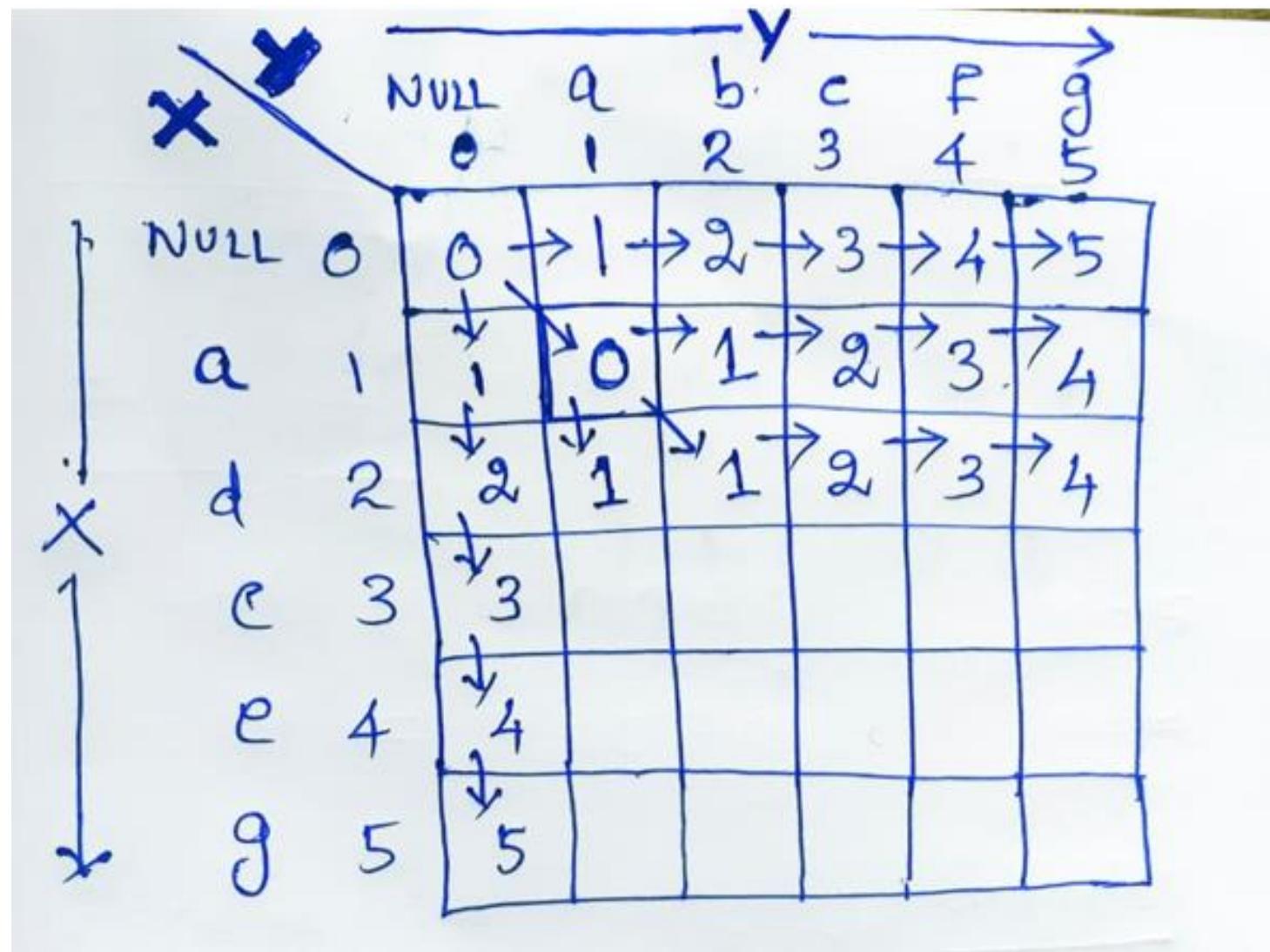
$$\Rightarrow 0 + 1 = 1 \\ (\text{Insert})$$

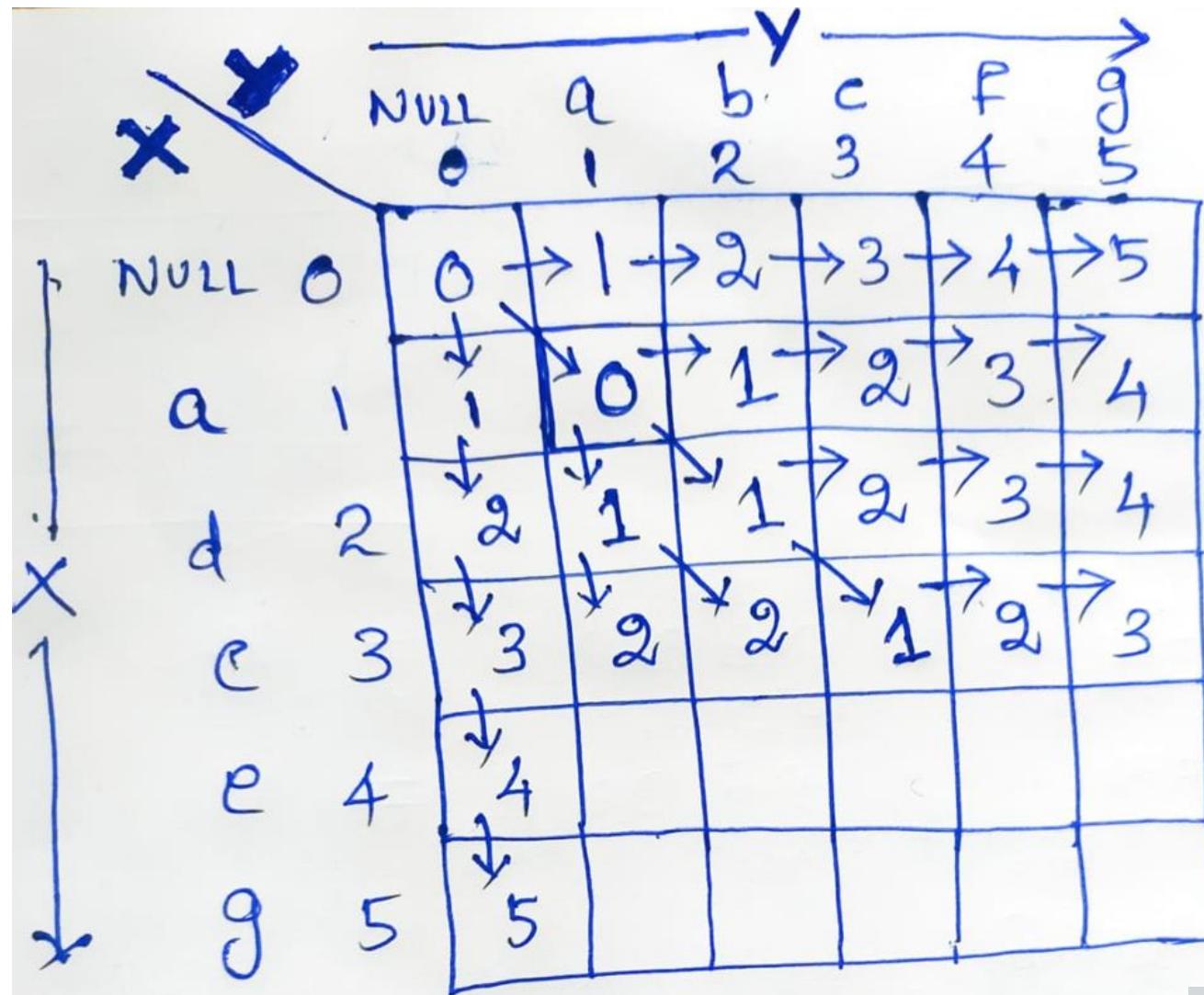
$m[1,3]$: $r = 'a'$ $c = 'c'$

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

$$(I) + 1 \Rightarrow 2$$

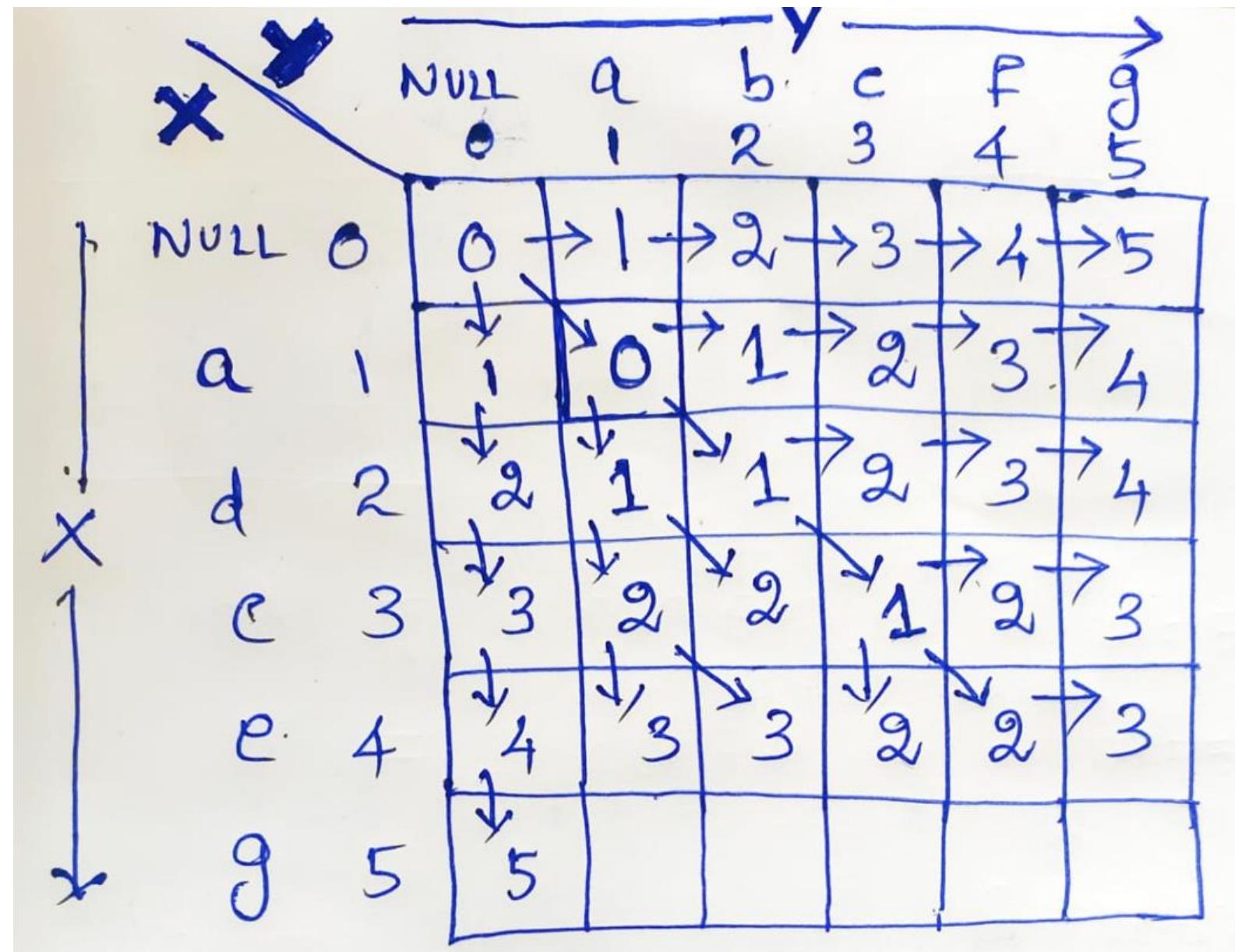


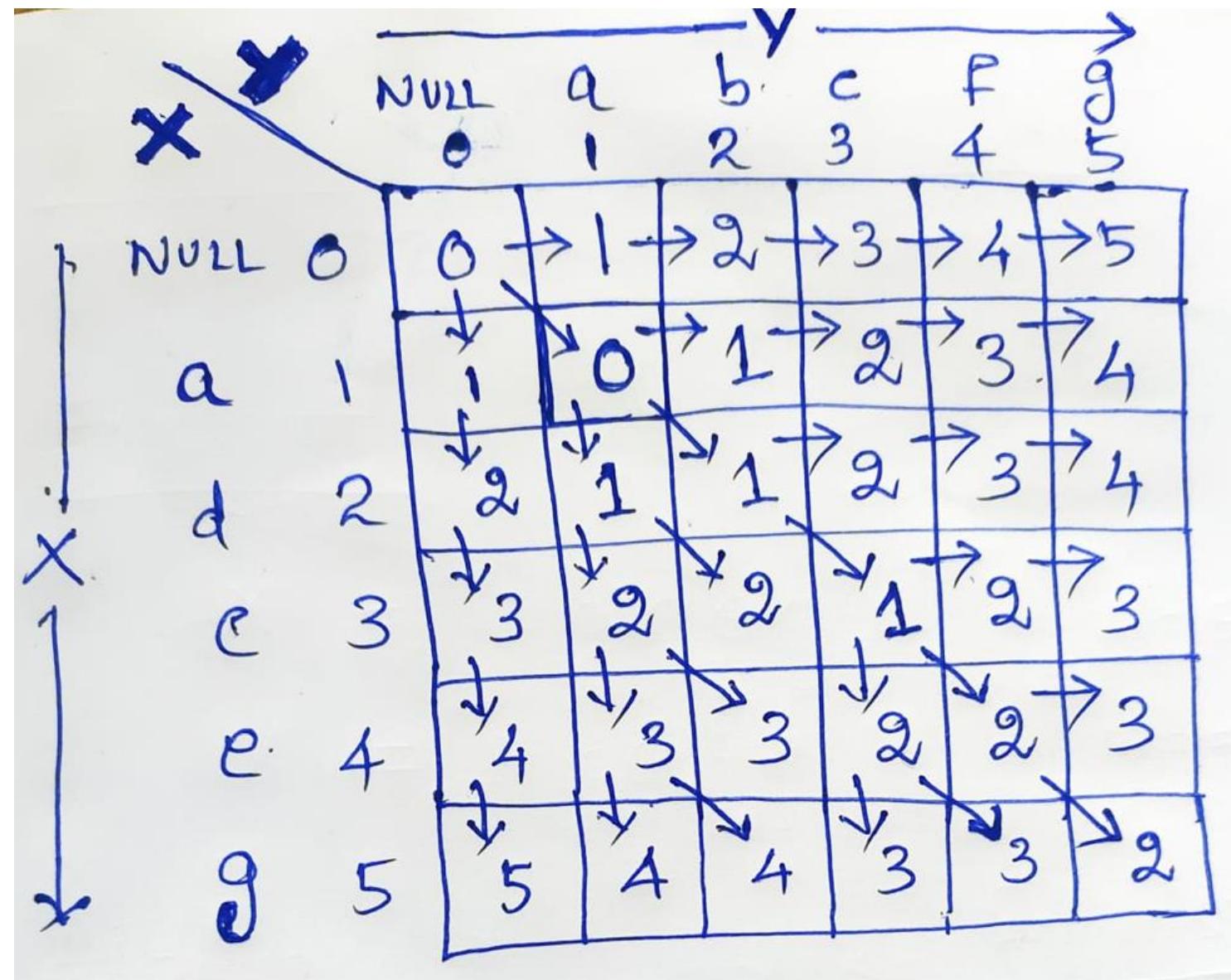




$m[3,3]$: $\tau = 'c'$ $e = 'c'$

$\tau = c$: No Action Required.







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:

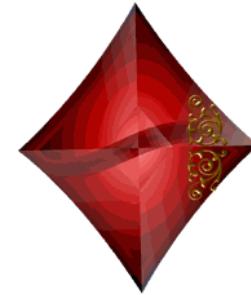
Need to carry maximum no. of items with maximum profit in a bag. without exceeding its capacity.



**Wt. = 5
Value = 10**



**Wt. = 3
Value = 20**



**Wt. = 8
Value = 25**



**Wt. = 4
Value = 8**



→ **Maximum wt. = 13**



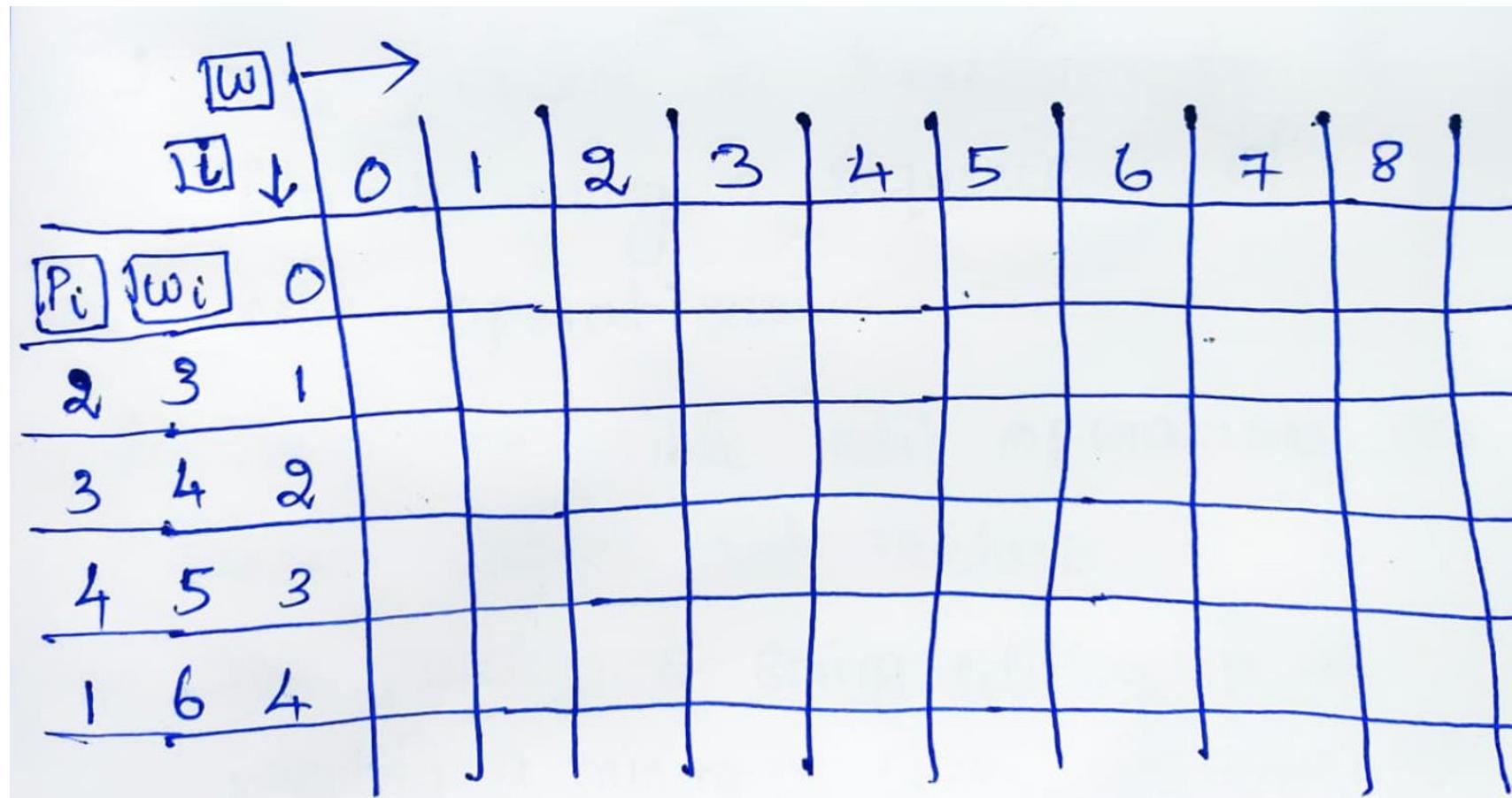
Example:

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	5

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



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

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

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

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

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

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

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

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

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

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

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

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



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

Dynamic Programming Approach

Travelling Salesman Problem

Travelling Salesman Problem: [TSP]

Problem:

→ Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visit every city exactly ^{once} and returns to the starting point.

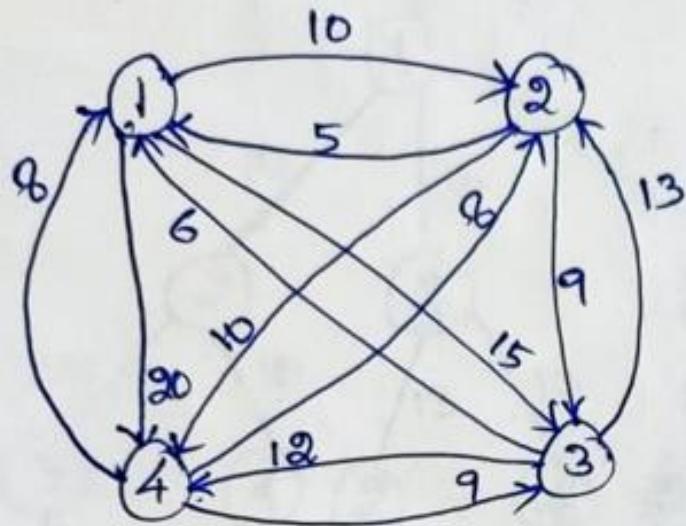
→ In other words, the TSP problem is to find a minimum weight Hamiltonian Cycle.



Hamiltonian Cycle:

↳ Problem is to find if there exist a tour that visits every city exactly once.

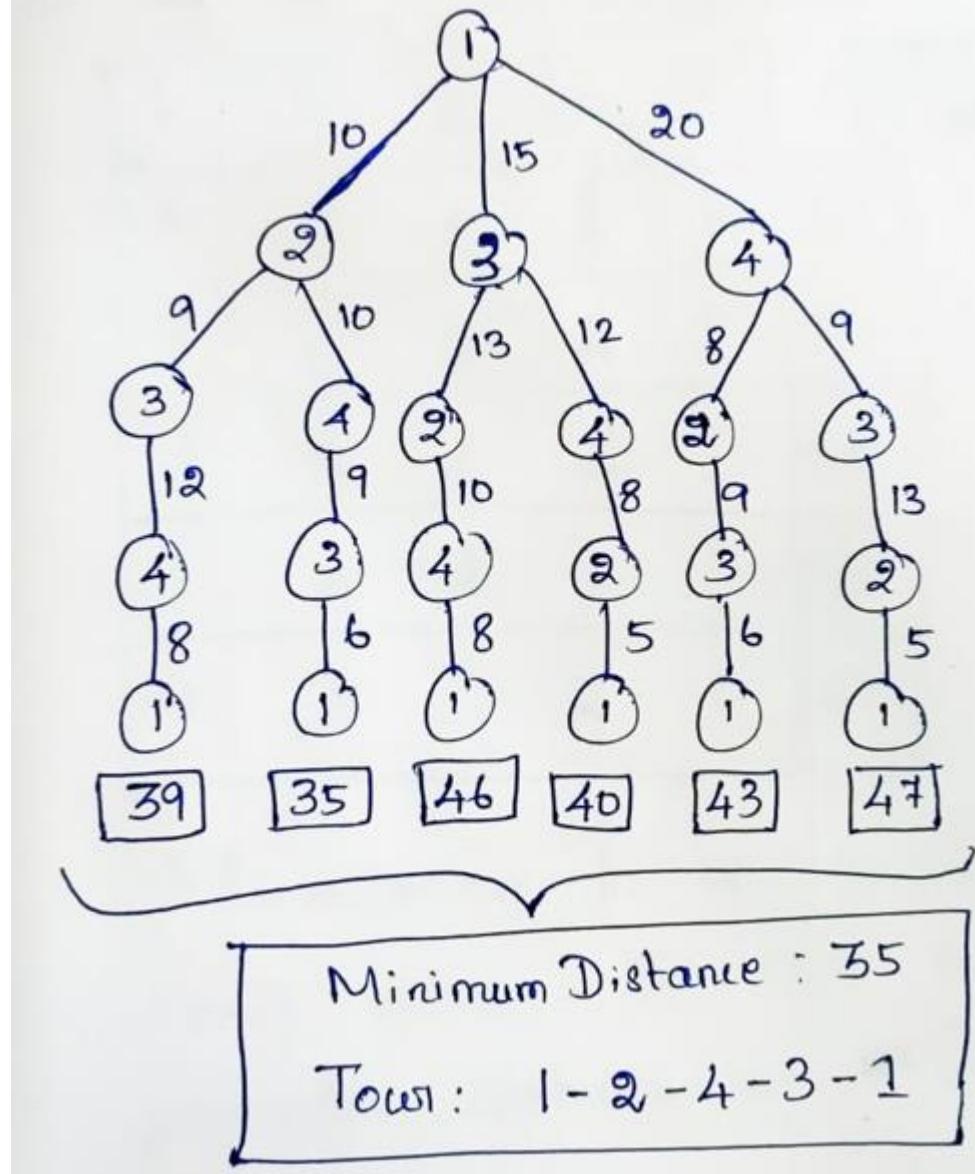
Example:



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

If a salesman starts from city 1 and he has to visit all other cities (2, 3 and 4) exactly once and he has to return to city 1.

Bruteforce Method:





→ BruteForce : Finding all the possible tours and their total distances. and selecting a tour with minimum distance

→ Need to generate permutations on all the cities, as follows. (except starting city)

→ Permutation to be generated for $(n-1)$ cities. So the number of possible Solutions is ~~$n!$~~ $(n-1)!$

→ So need to generate $(n-1)!$ Solution and choose one among them.

→ So complexity is $O(n!)$

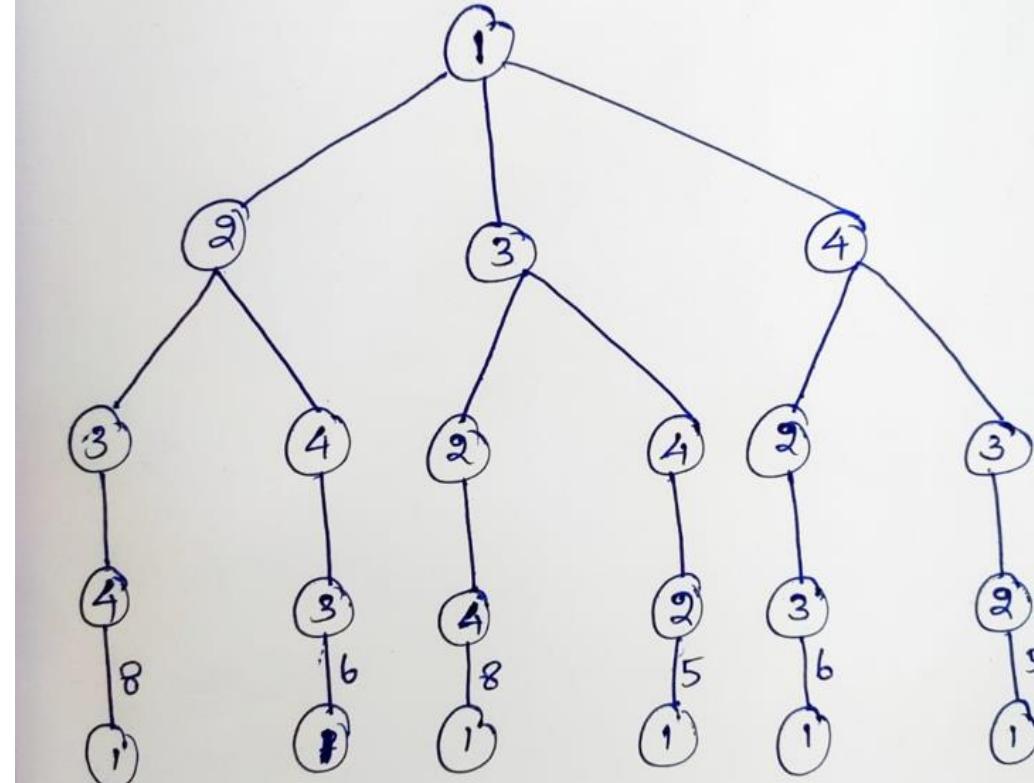
Dynamic Programming Approach: (For Travelling Salesman Problem)

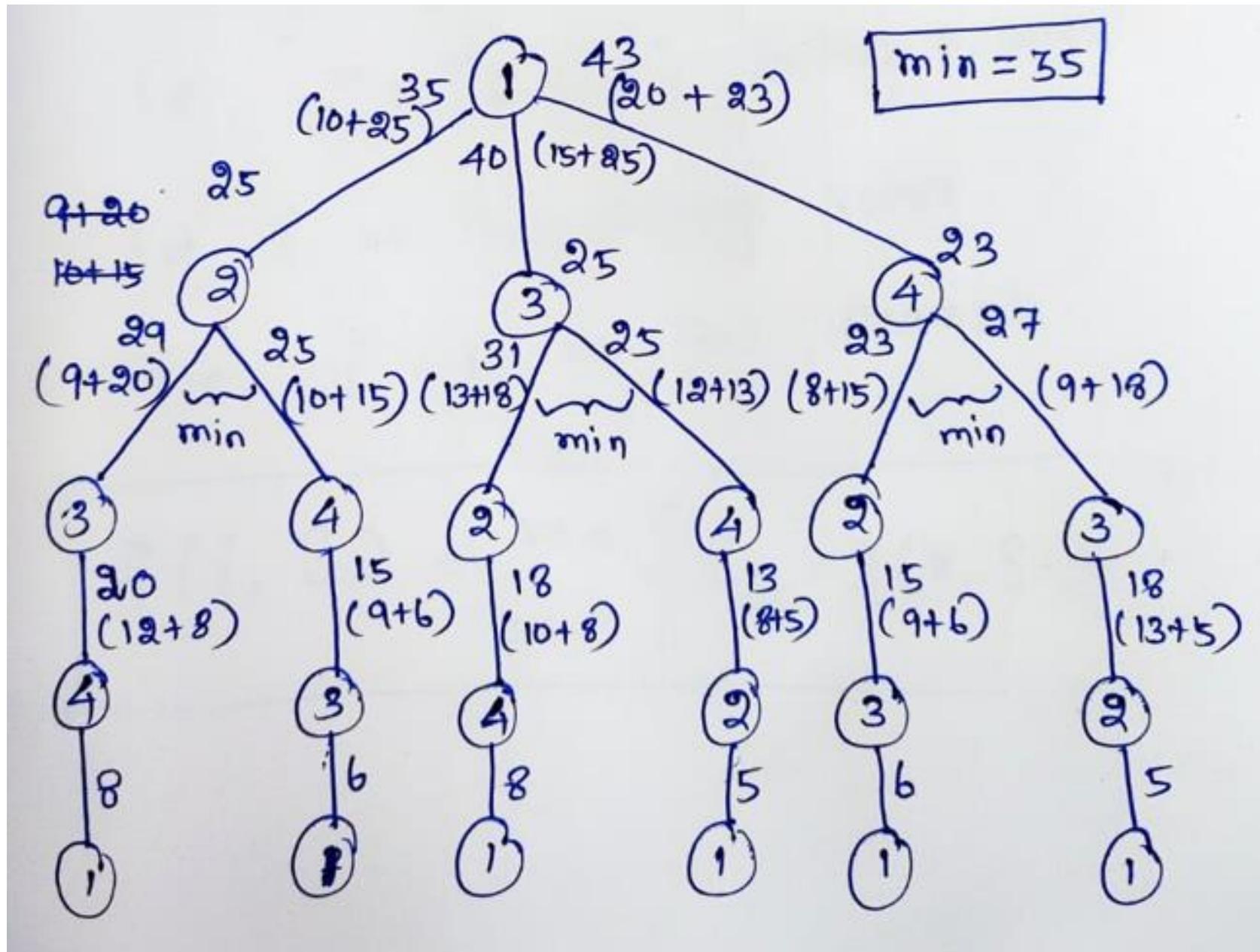


SASTRA
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Derivation of Formula:

Example: (Consider the same example
the one taken for brute force method)







$$g(1, \{2, 3, 4\}) = \min_{k=2, 3, 4} \left\{ c_{1k} + g(k, \{2, 3, 4\} - k) \right\}$$

Let S be a set of cities ~~except~~
containing all the cities except the
starting city.

i.e., $S = \{2, 3, 4\}$ in this example

Let i be the starting vertex.

i.e., $i = 1$ in this example.



$$g(i, S) = \min_{k \in S} \{ c_{ik} + g(k, S - \{k\}) \}$$



$$g(2, \emptyset) = 5$$

$$g(3, \emptyset) = 6$$

$$g(4, \emptyset) = 8$$

$$g(2, \{3\}) = 15$$

$$g(2, \{4\}) = 18$$

$$g(3, \{2\}) = 18$$

$$g(3, \{4\}) = 20$$

$$g(4, \{2\}) = 13$$

$$g(4, \{3\}) = 15$$

$$\boxed{g(2, \{3, 4\}) = 25}$$
$$g(3, \{2, 4\}) = 25$$
$$g(4, \{2, 3\}) = 23$$
$$\boxed{g(1, \{2, 3, 4\}) = 35}$$

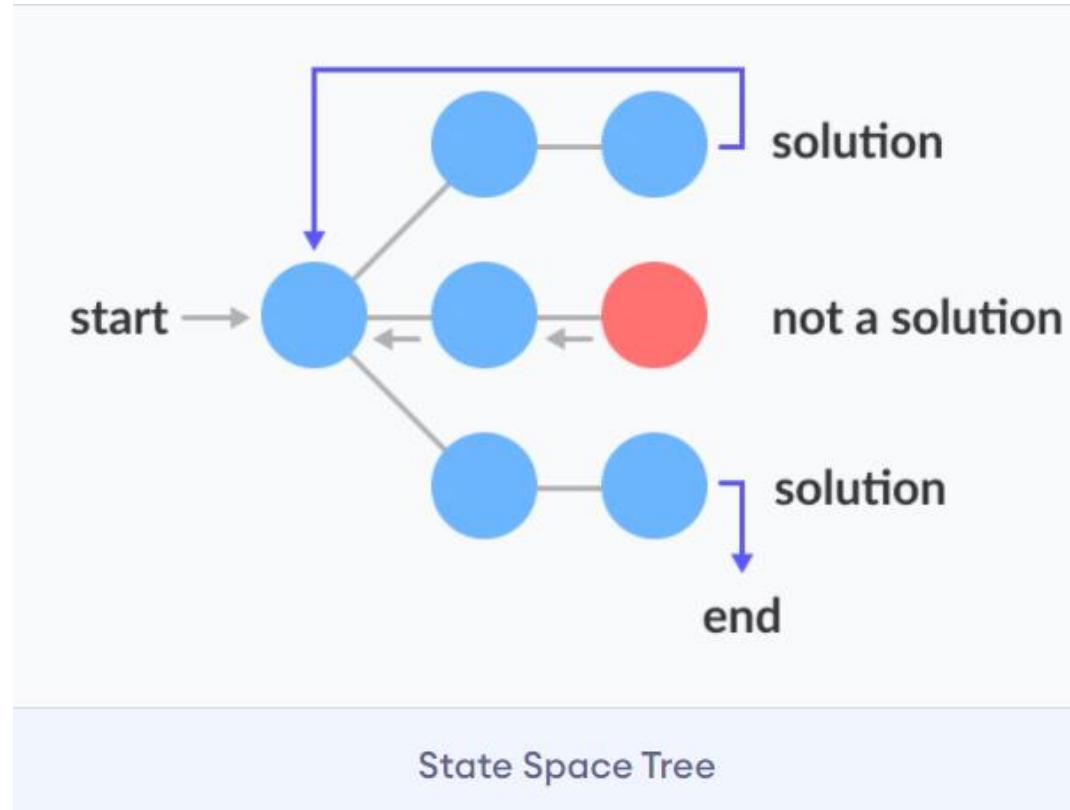
Backtracking Approach

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

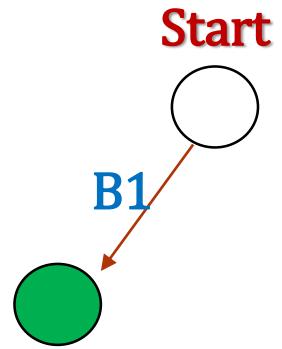
Step-1

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

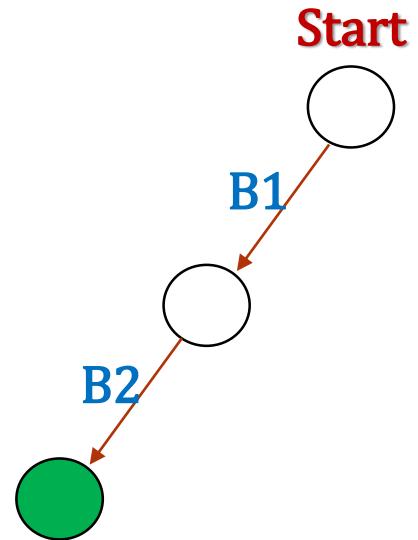
Step-2



Step-3



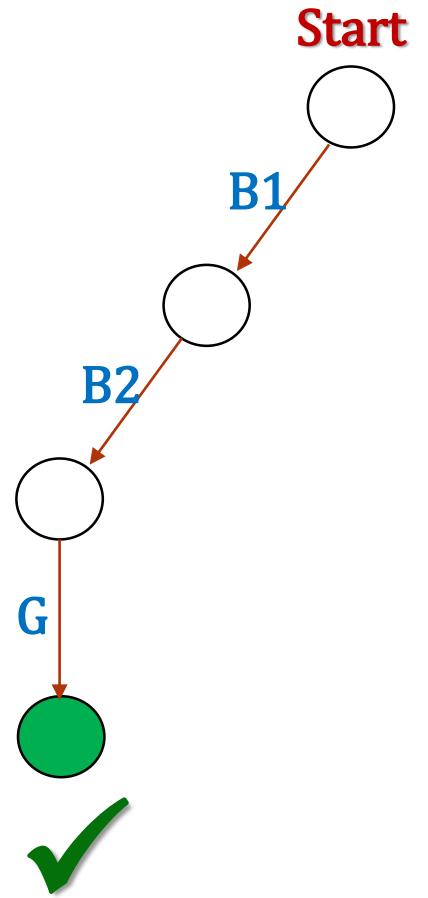
SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



Step-4

Solutions

B1-B2-G

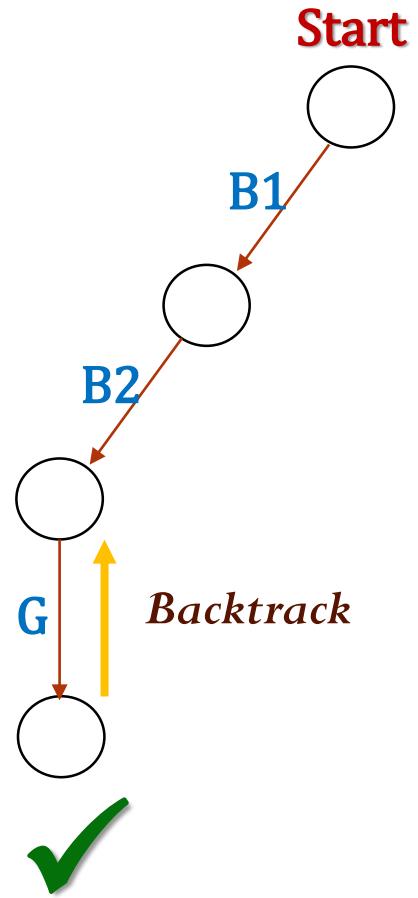


SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Step-5

Solutions

B1-B2-G

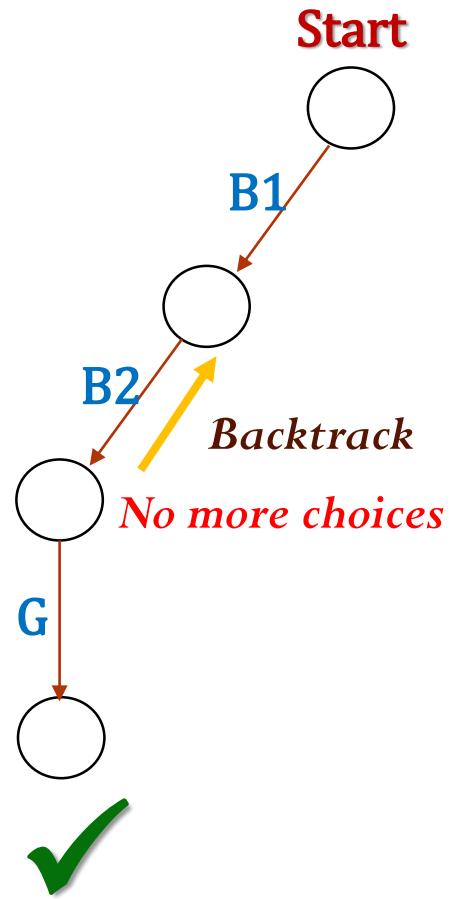


SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Step-6

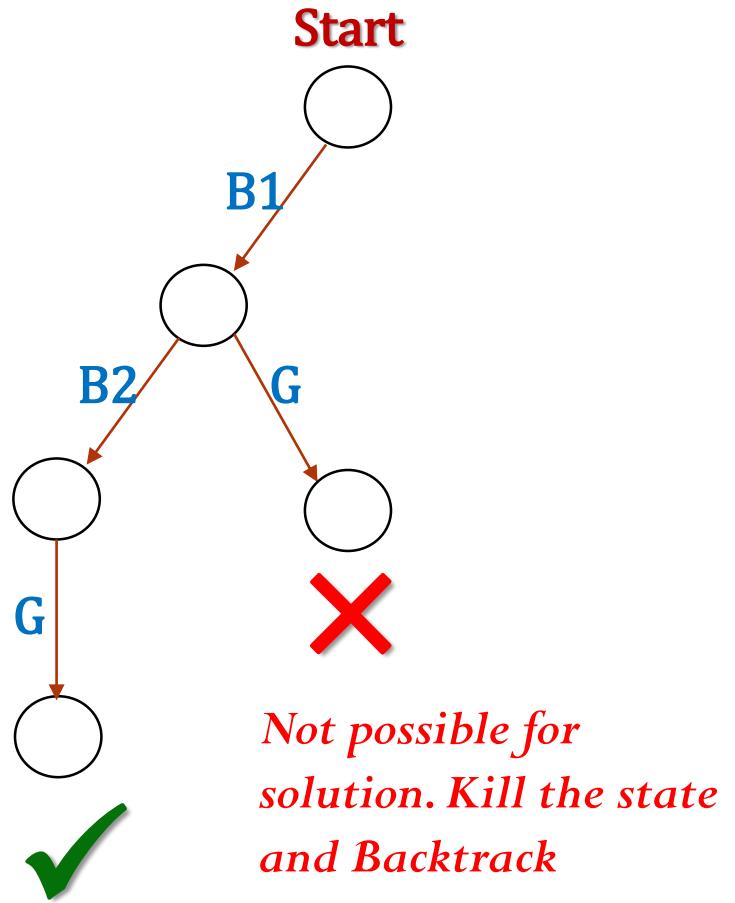
Solutions

B1-B2-G



Solutions

B1-B2-G



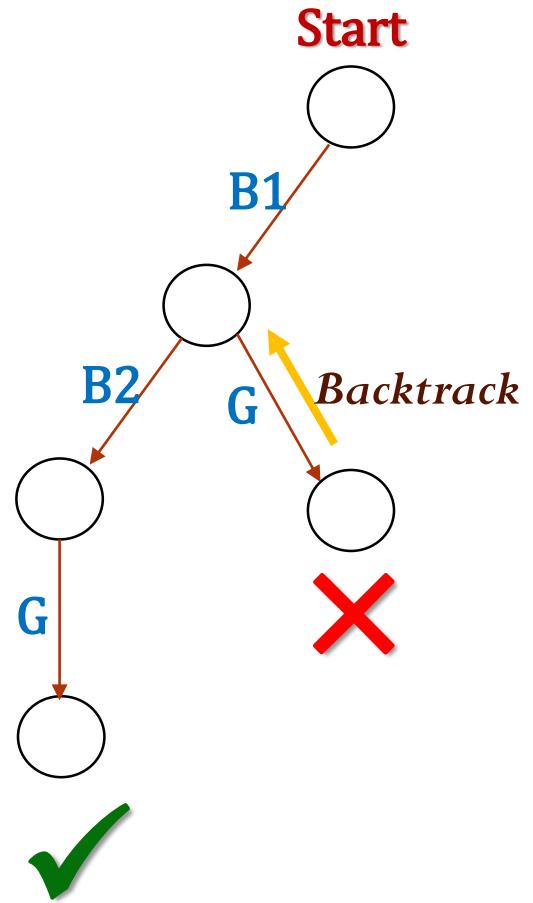
*Not possible for
solution. Kill the state
and Backtrack*

Bounding Condition: Girl should not be on the middle bench

Step-8

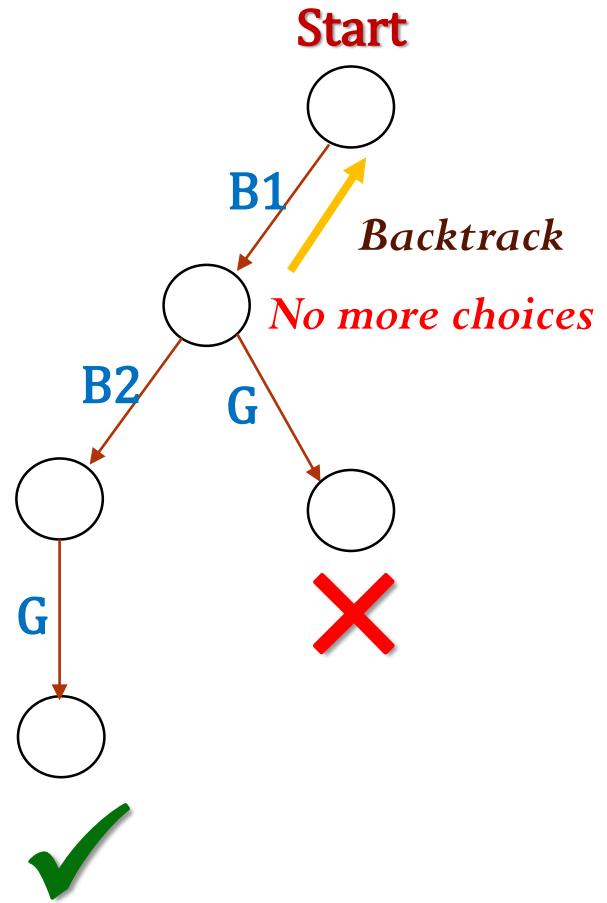
Solutions

B1-B2-G



Solutions

B1-B2-G



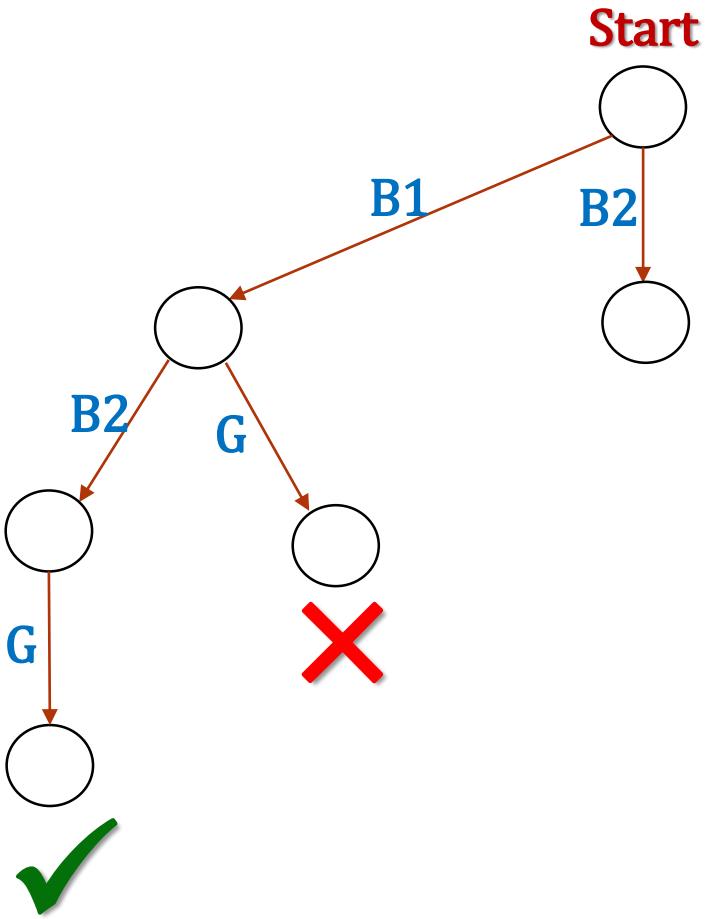
Step-10



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



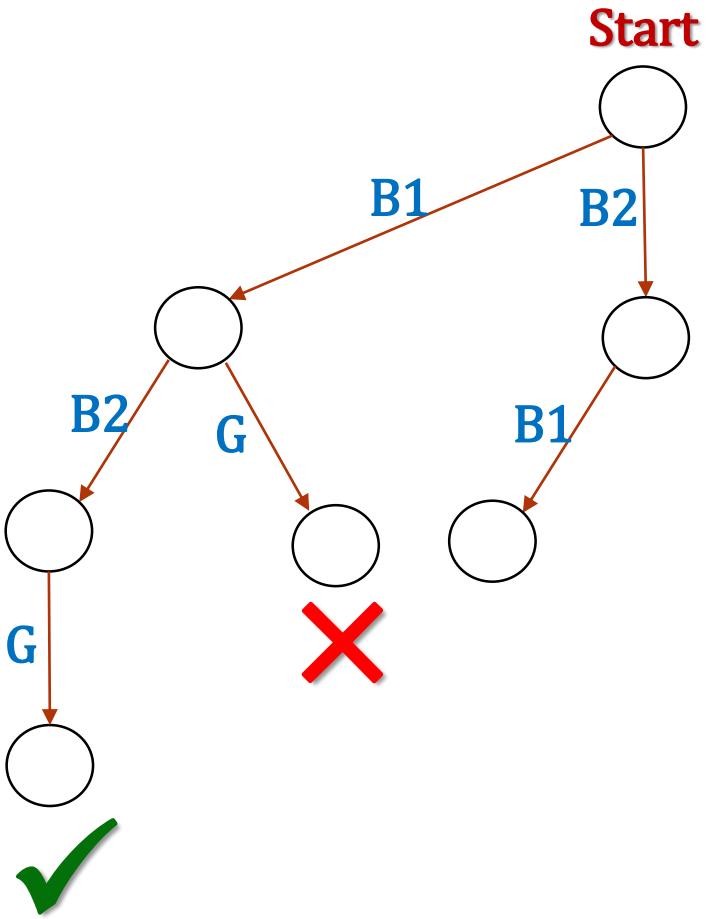
Step-11



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



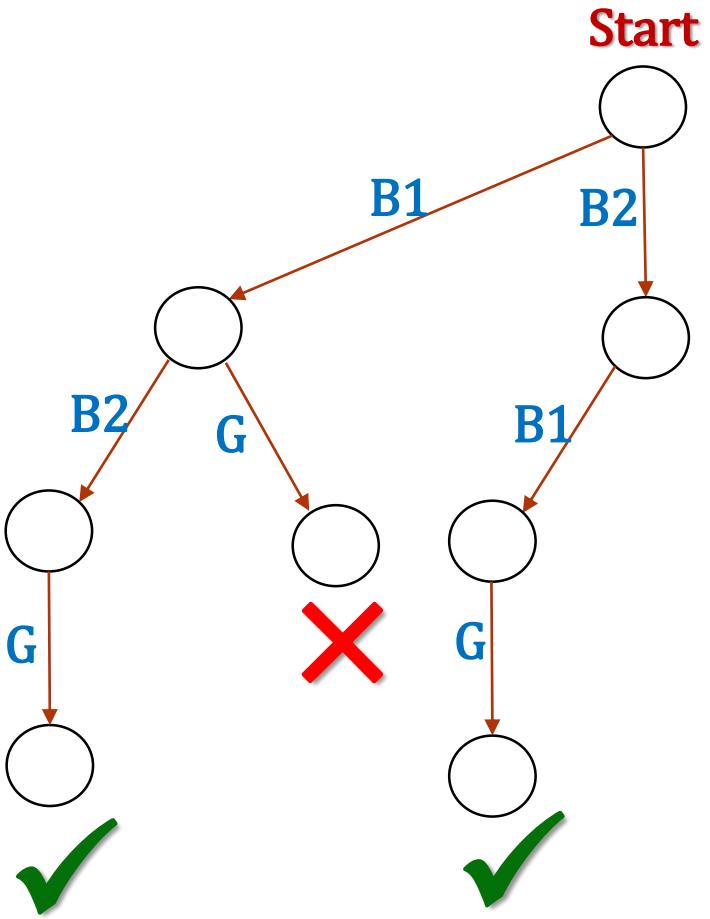
Step-12



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



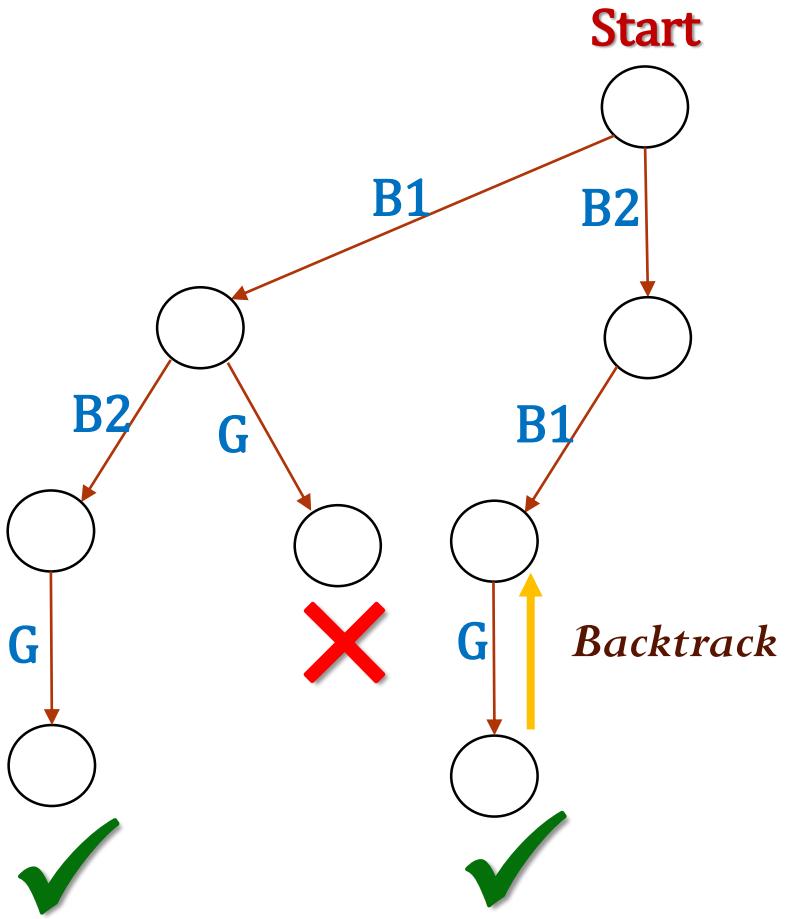
Step-13



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



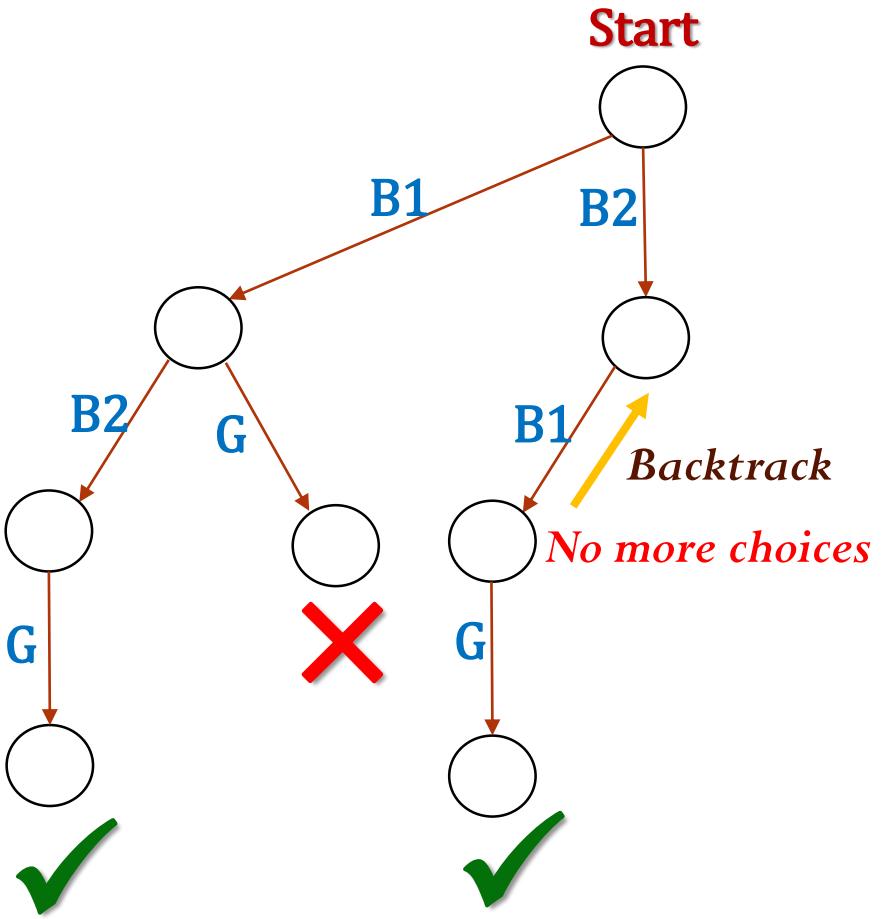
Step-14



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



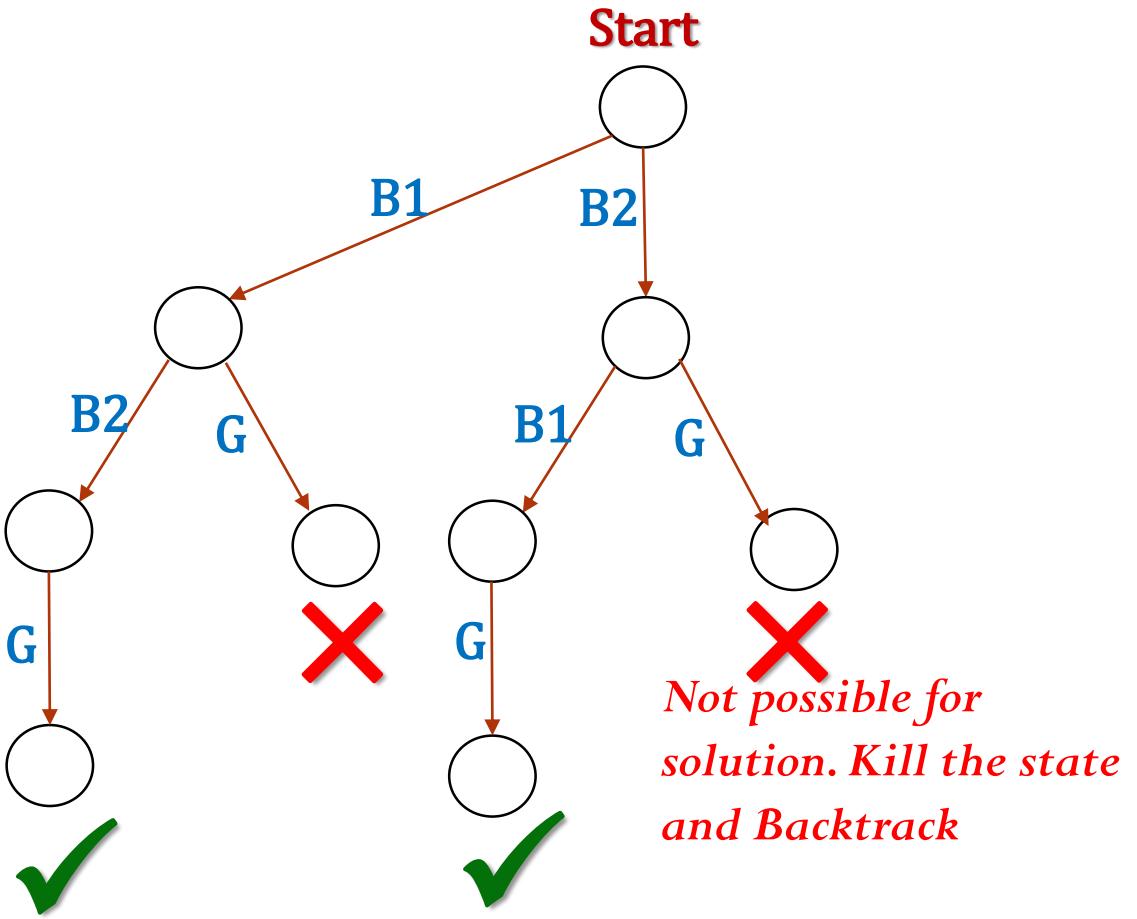
Step-15



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



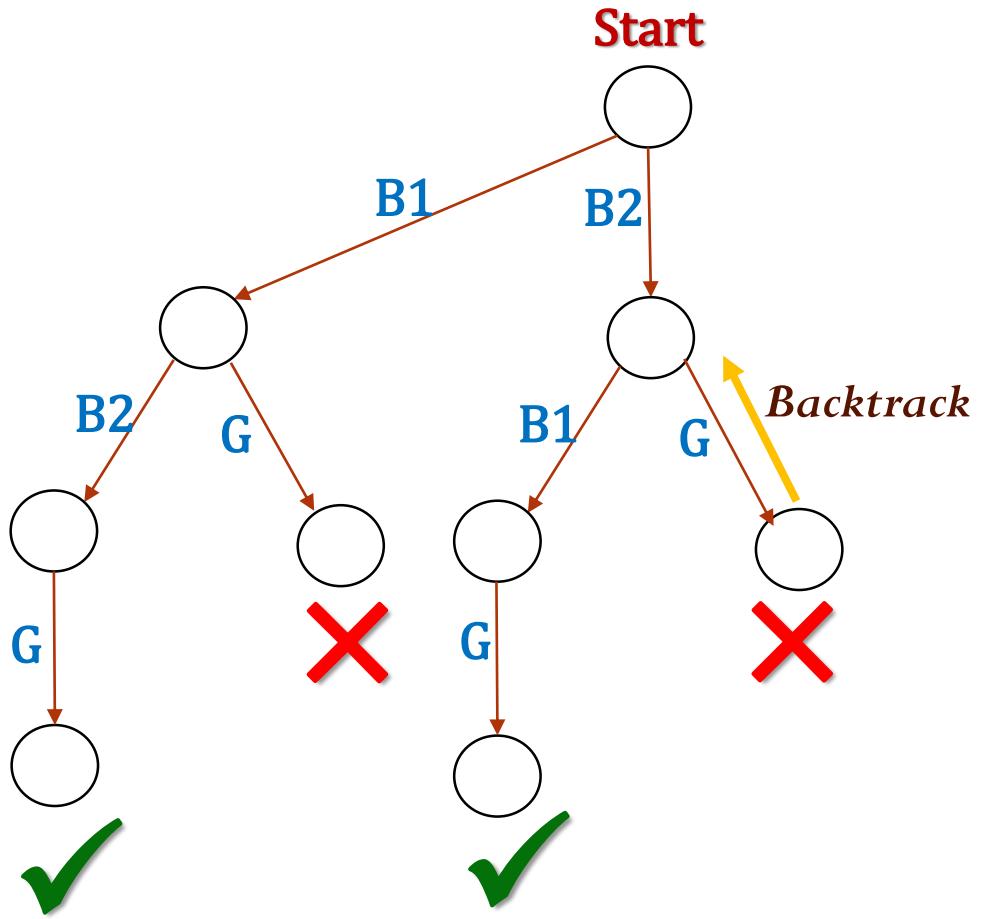
Step-16



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

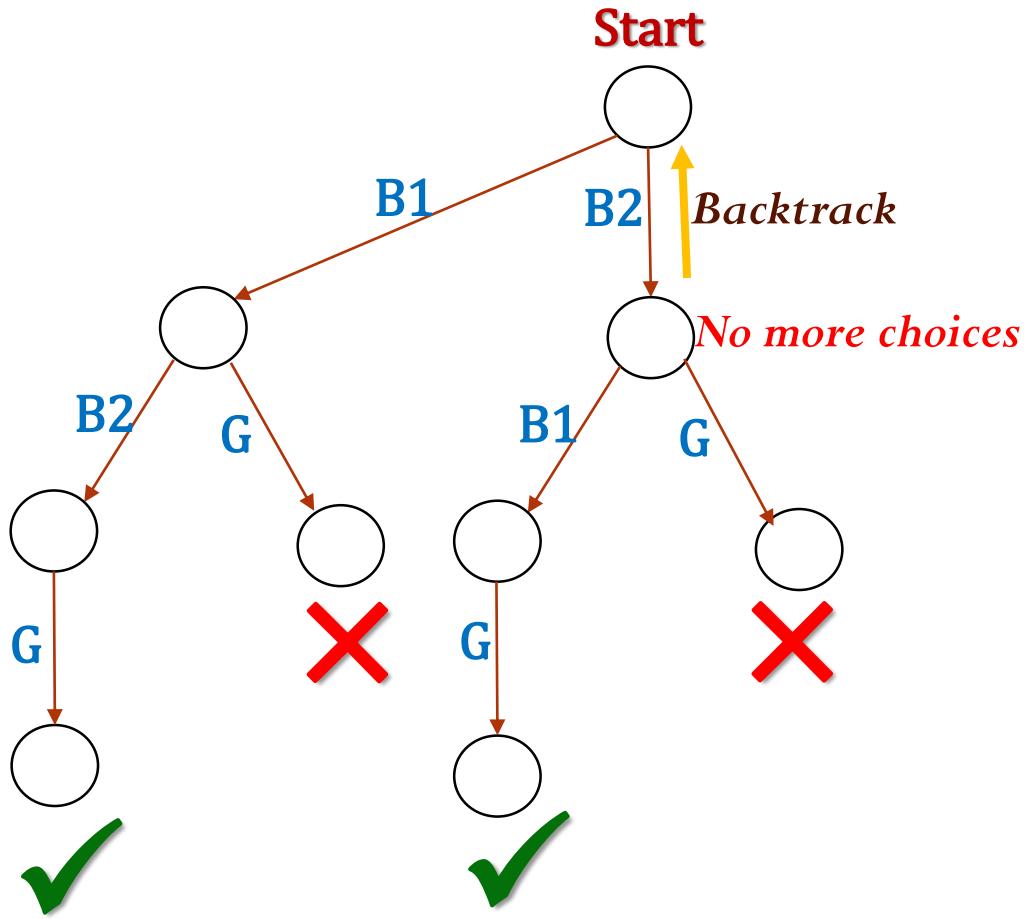
B1-B2-G
B2-B1-G



Step-17

Solutions

B1-B2-G
B2-B1-G

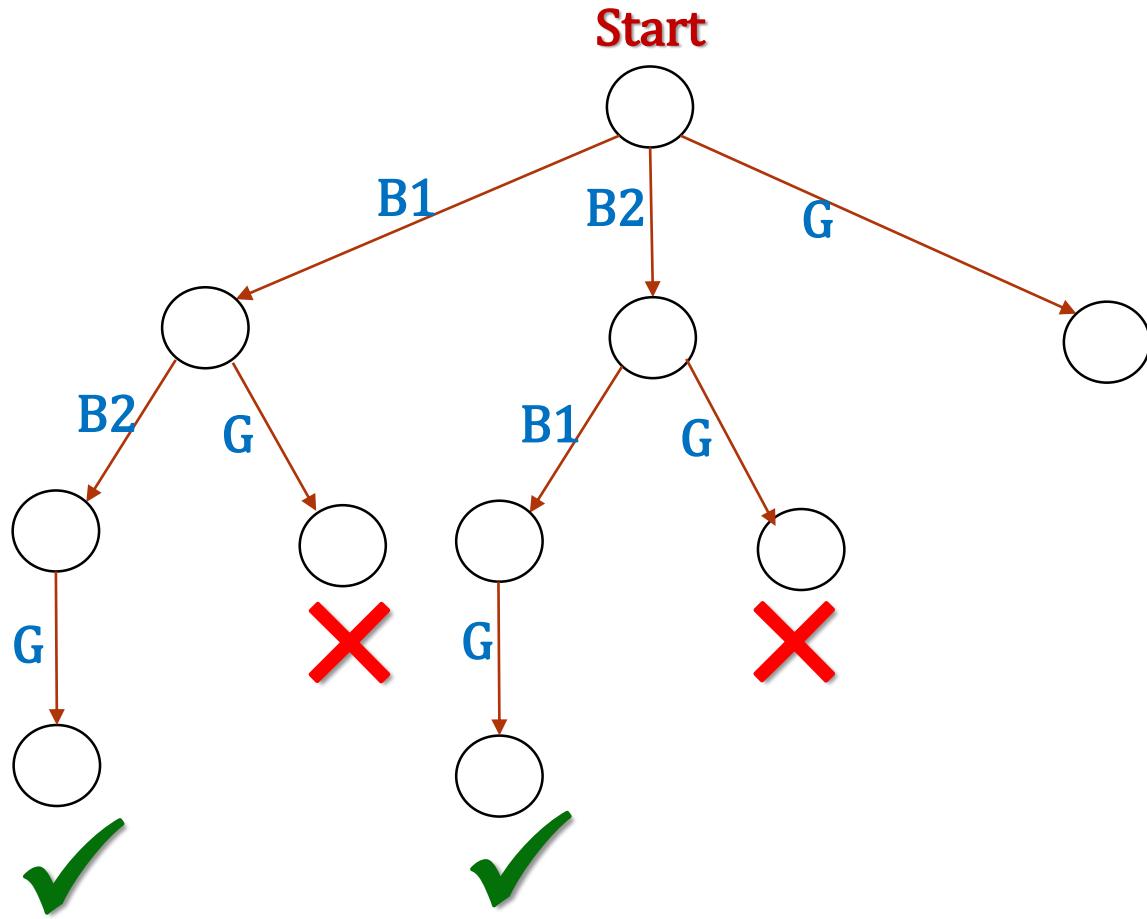


Step-18



Solutions

B1-B2-G
B2-B1-G



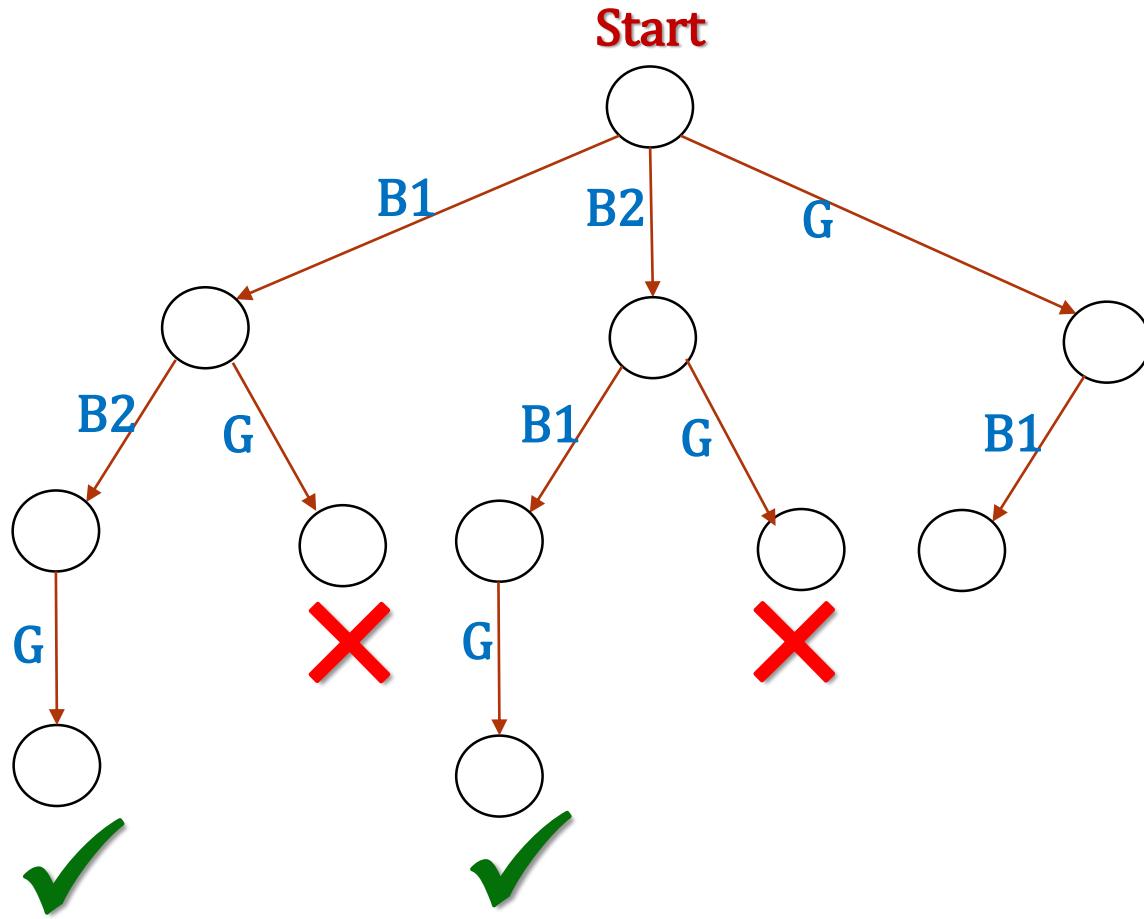
Step-19



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

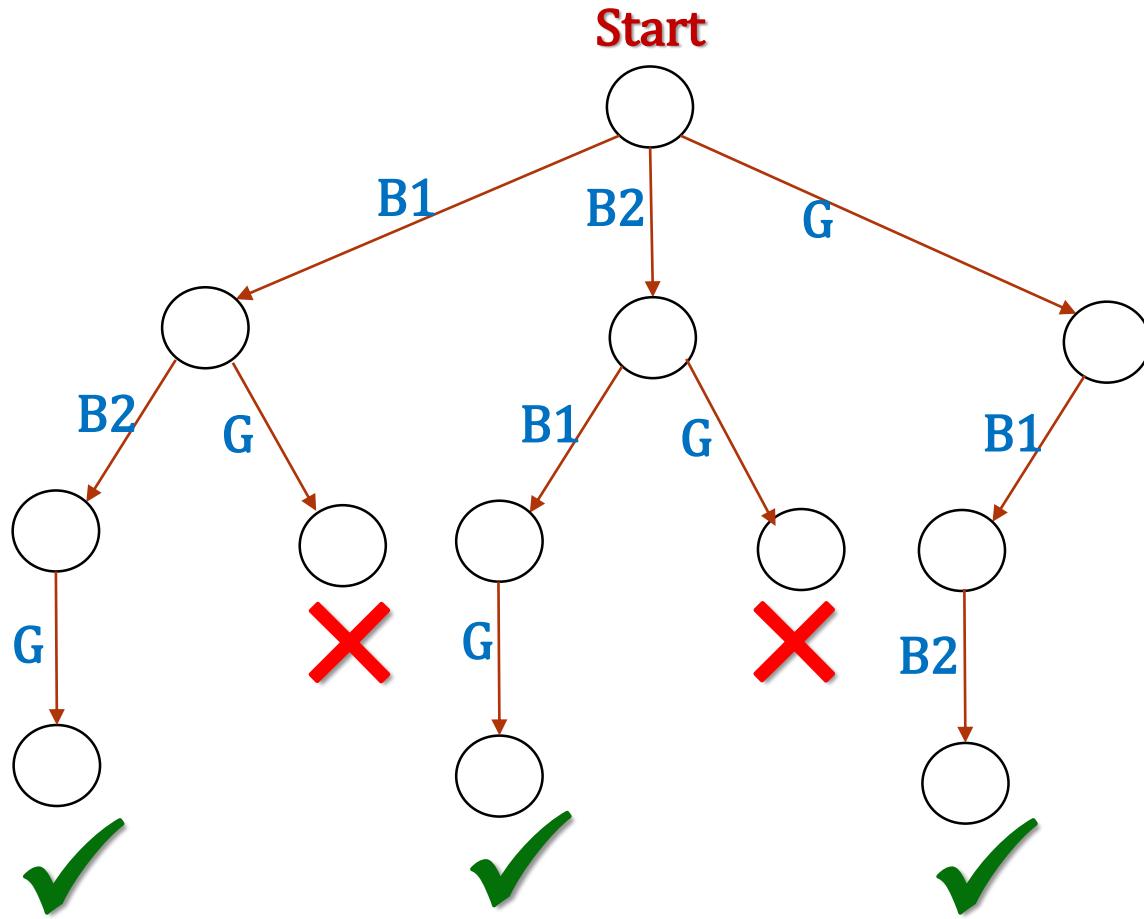


Step-20



Solutions

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



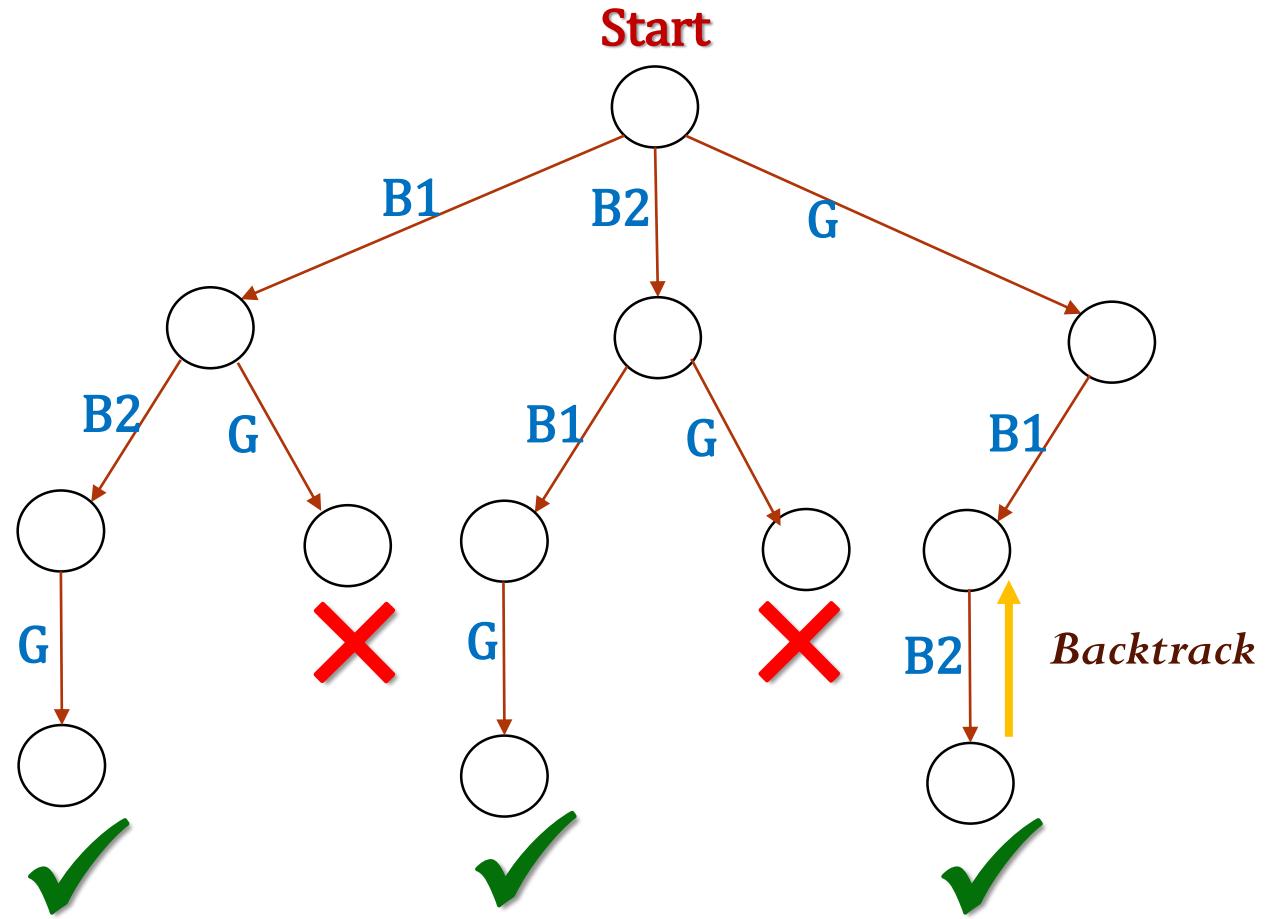
Step-21



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

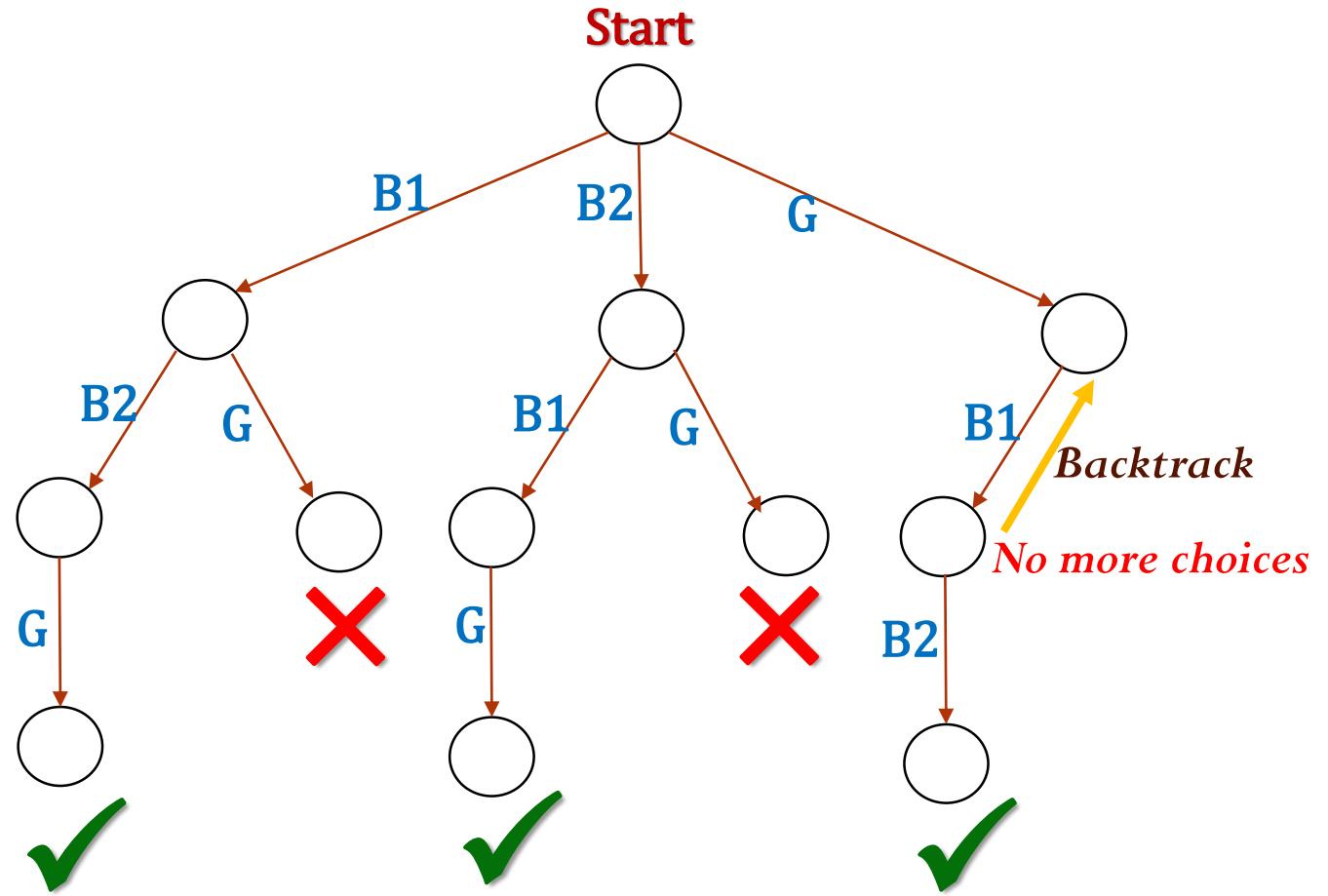


Step-22



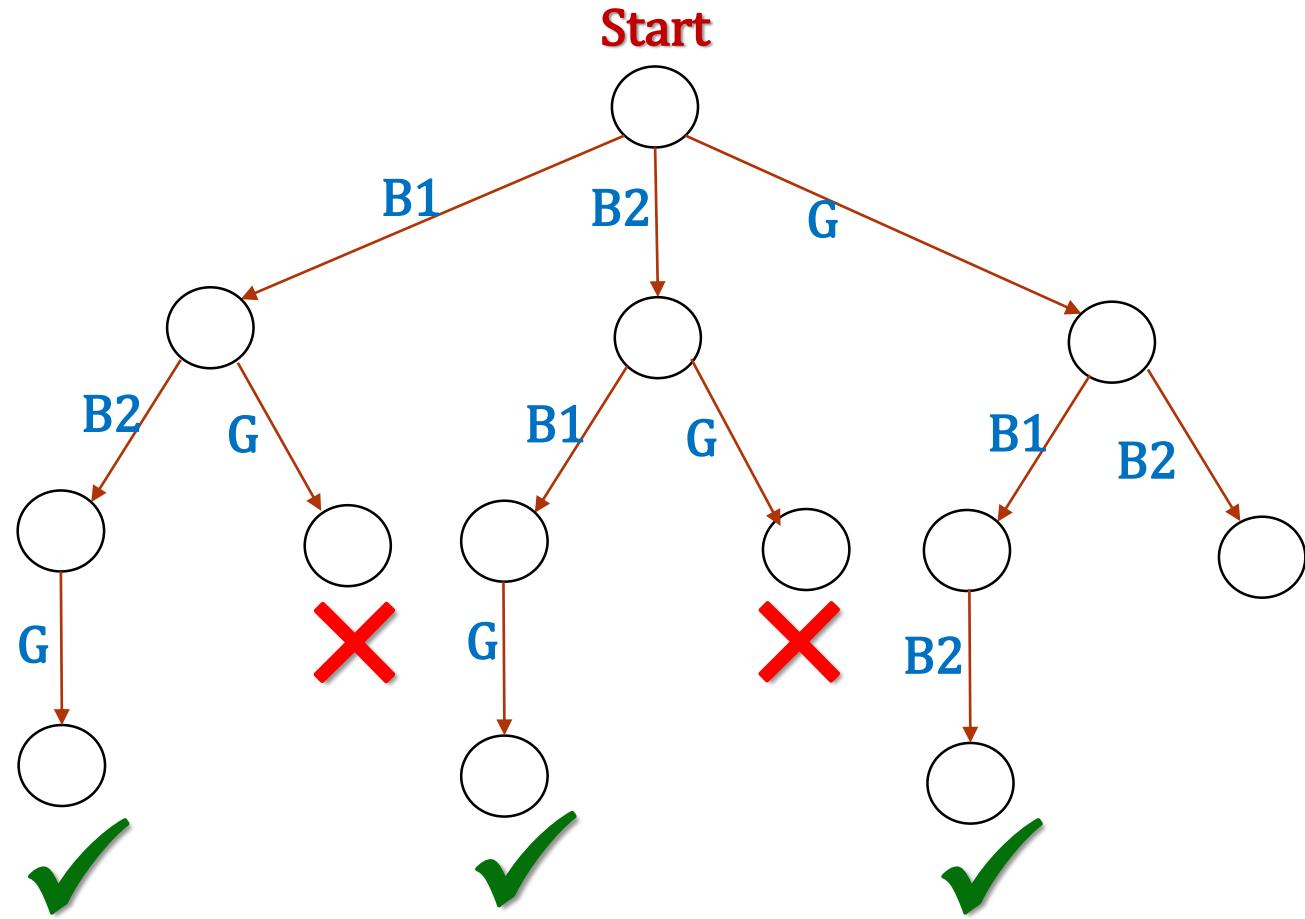
Solutions

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



Solutions

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

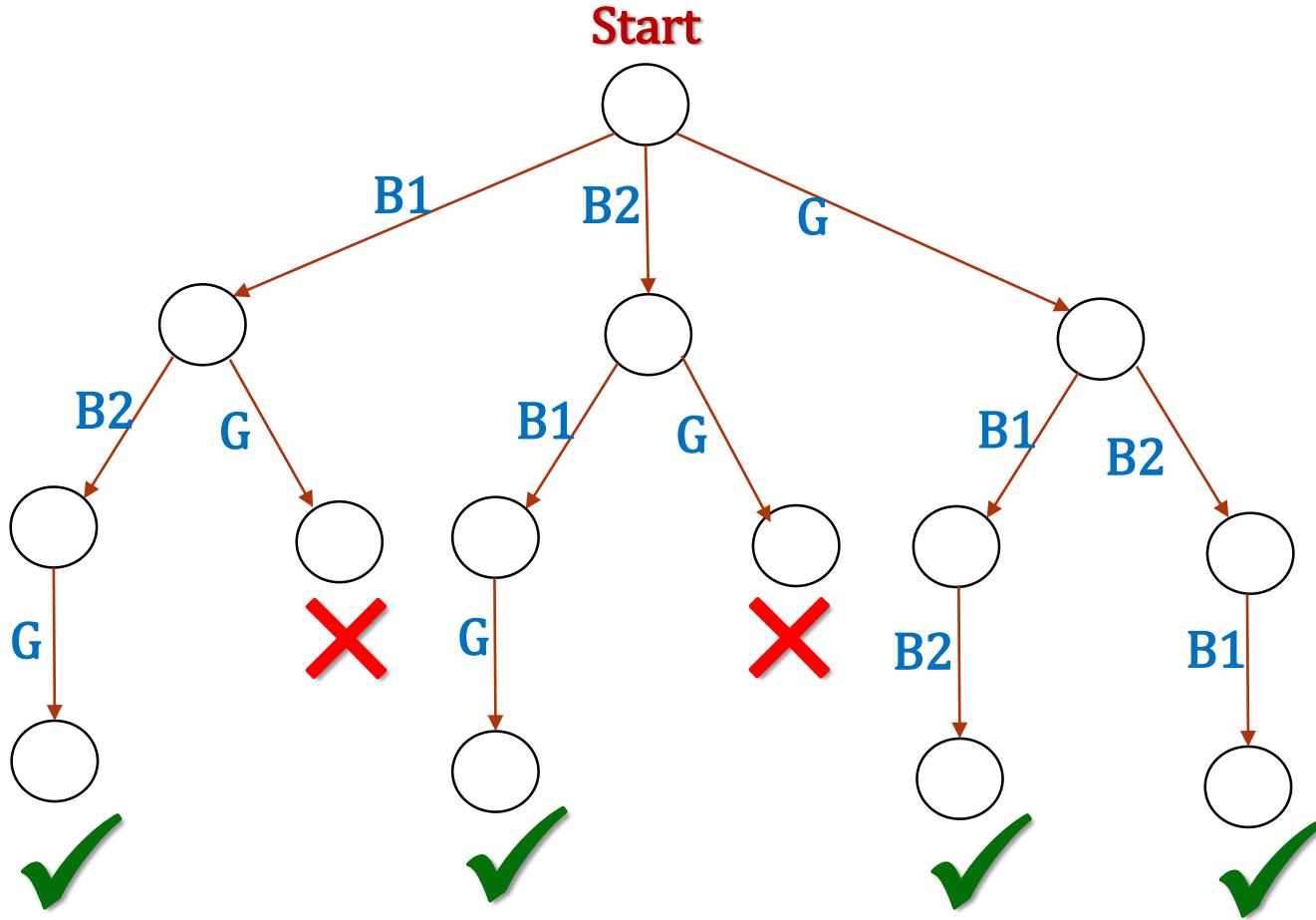


Step-24



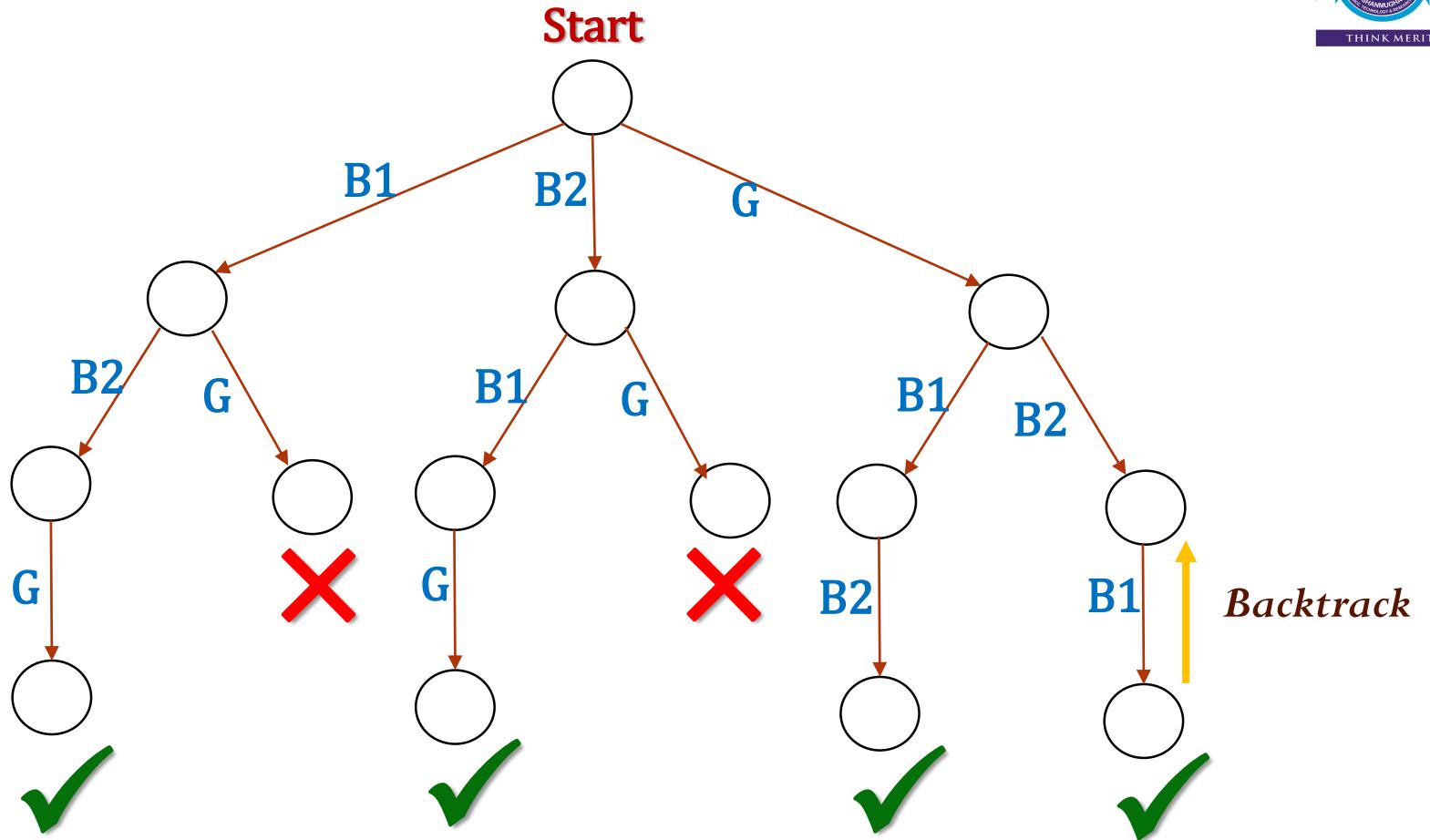
Solutions

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



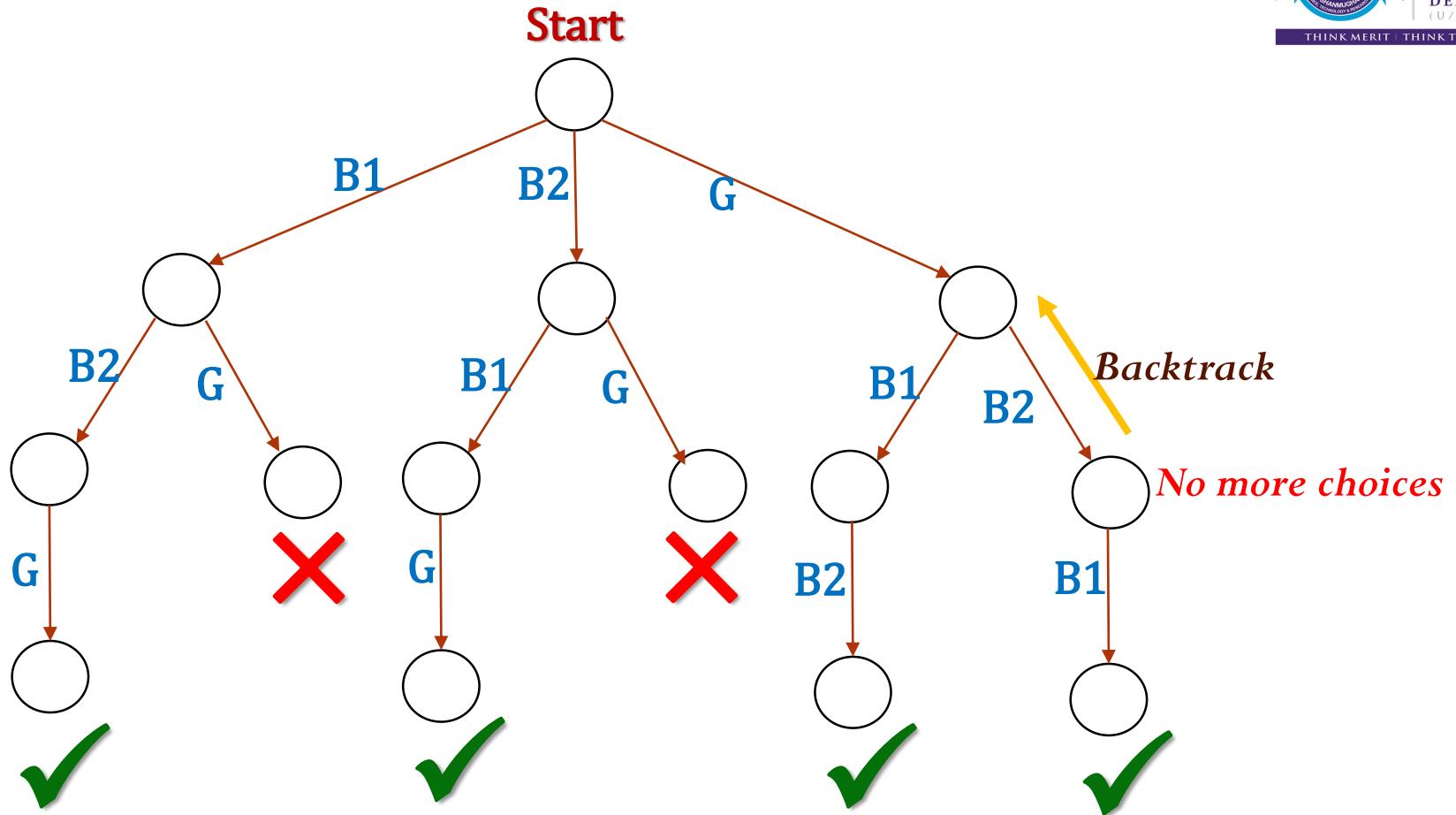
Solutions

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



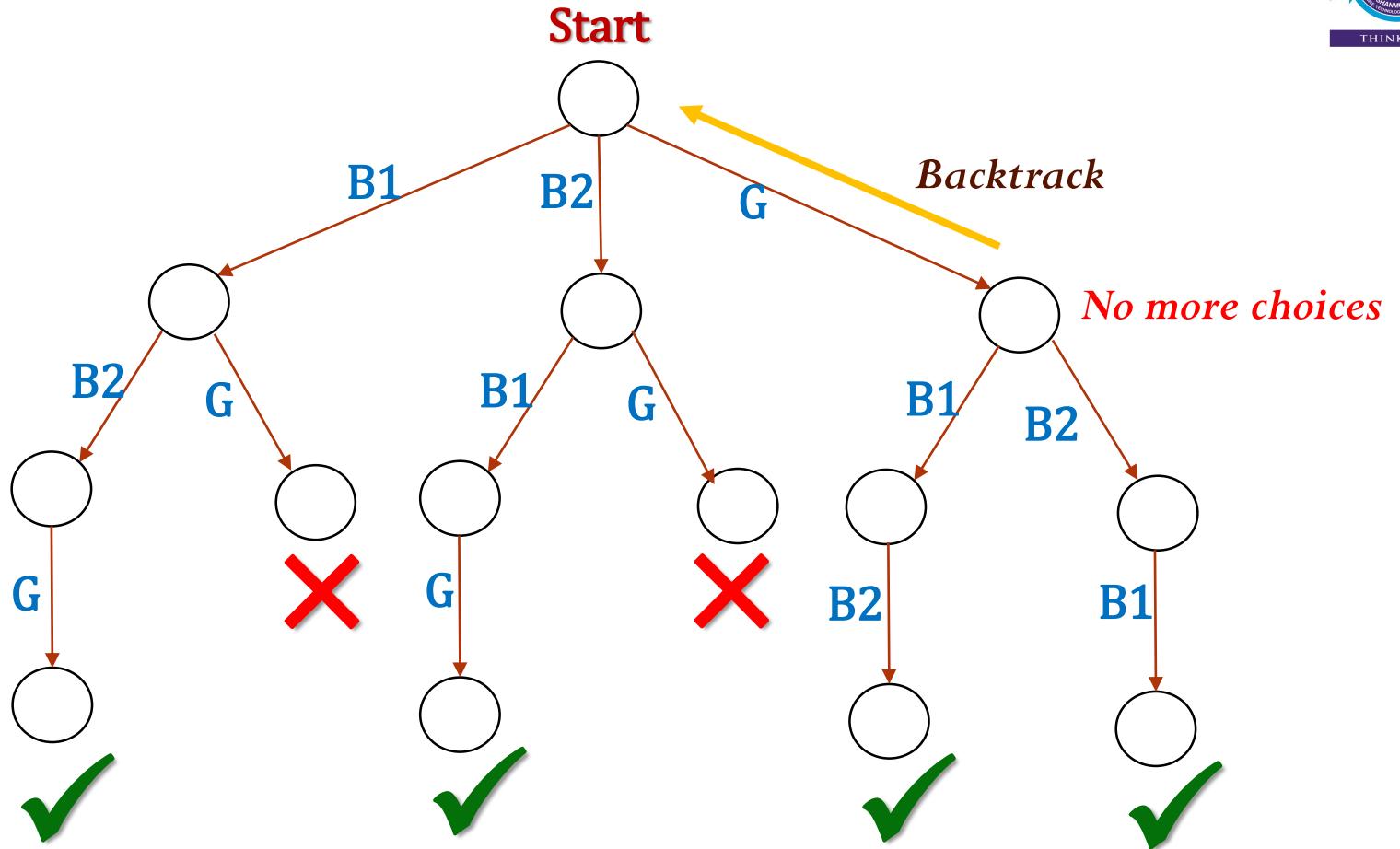
Solutions

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



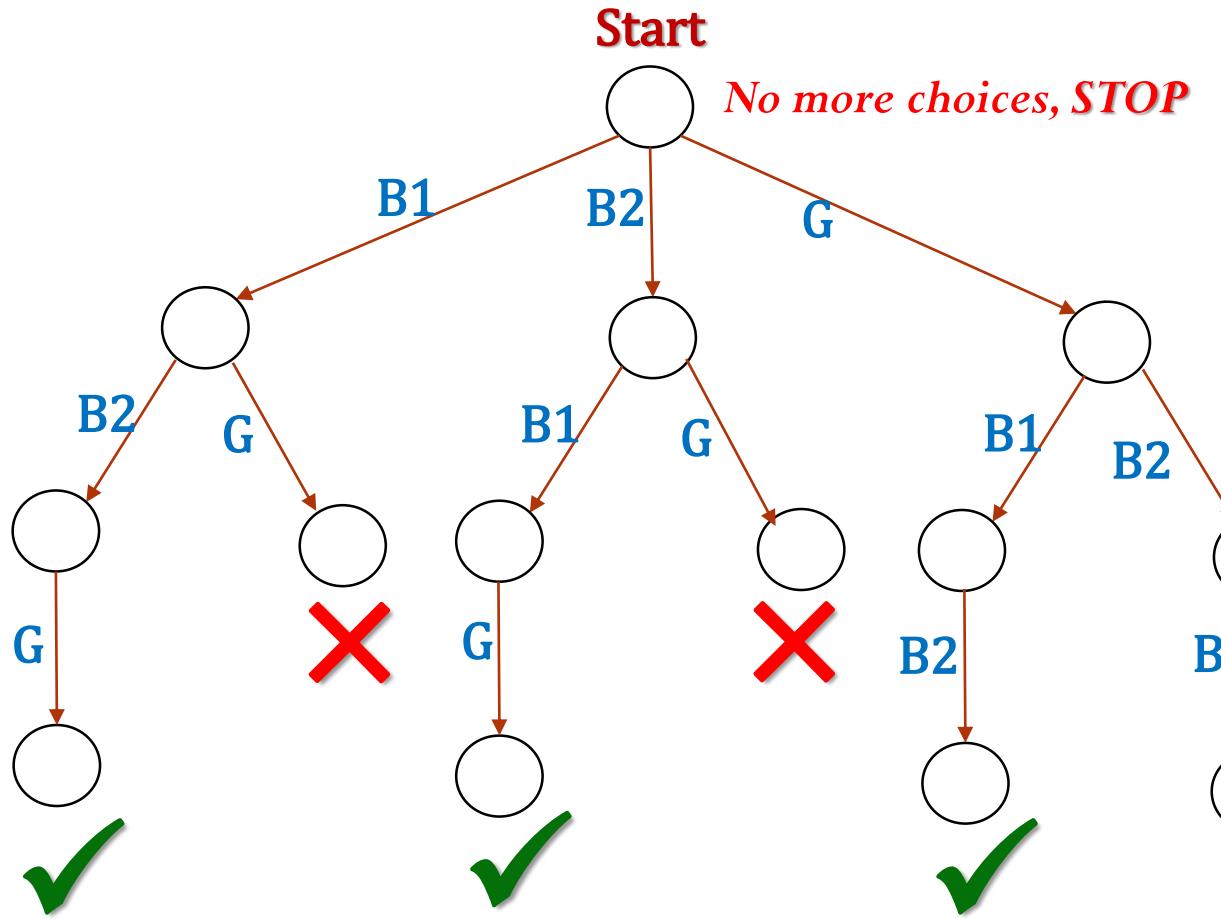
Solutions

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

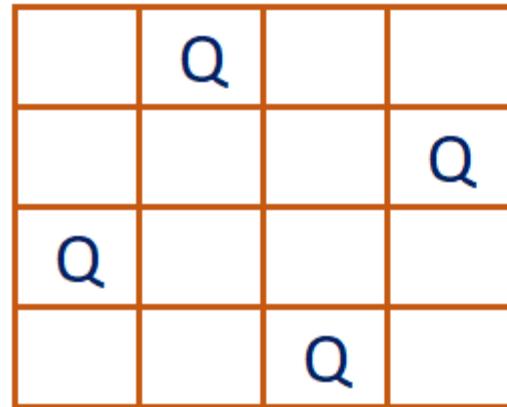
n-Queen Problem

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.

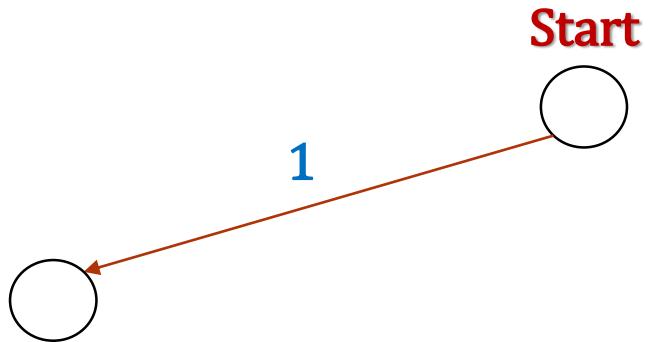
	1	2	3	4
1				
2				
3				
4				

Start



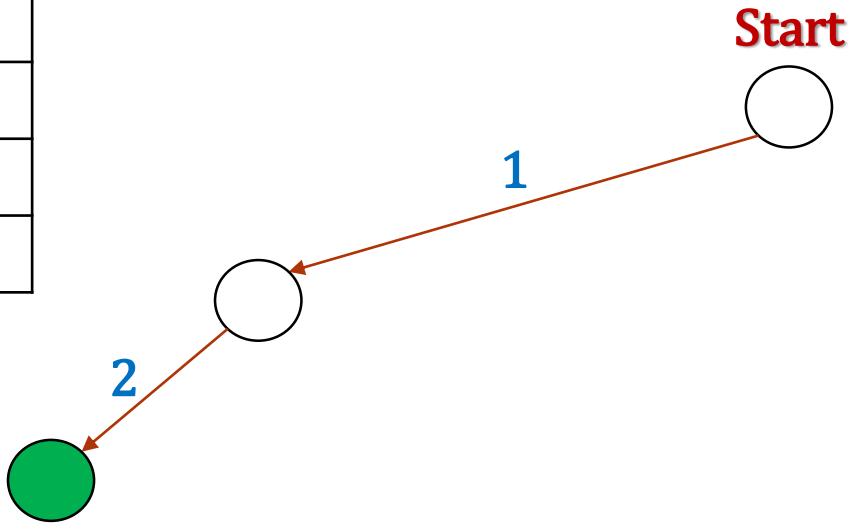
SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

	1	2	3	4
1	Q			
2				
3				
4				

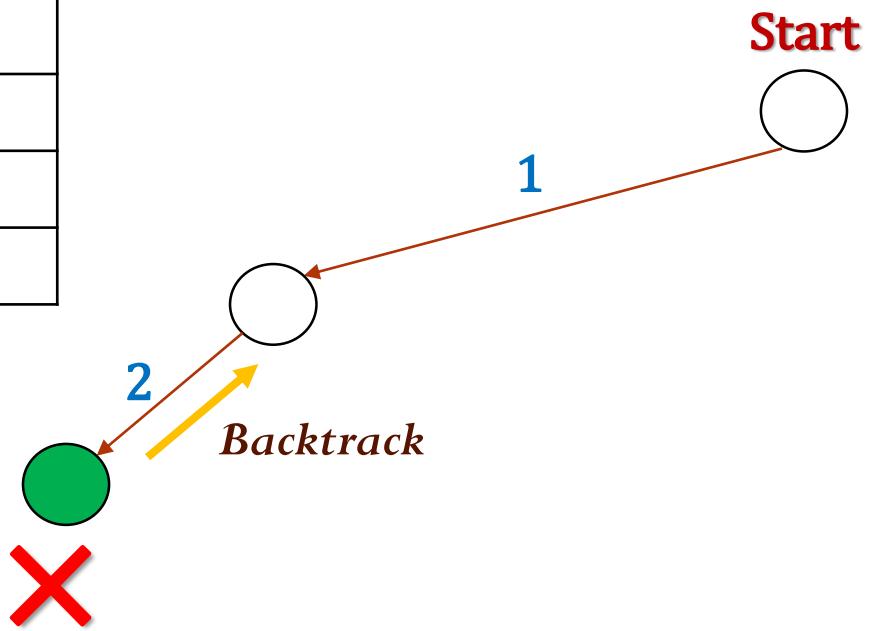


SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

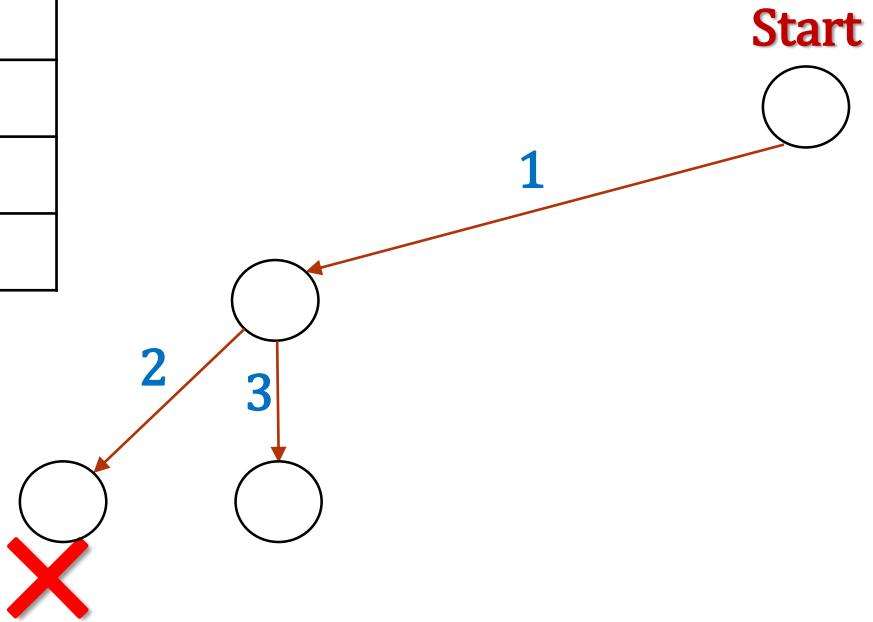
	1	2	3	4
1	Q			
2		Q		
3				
4				



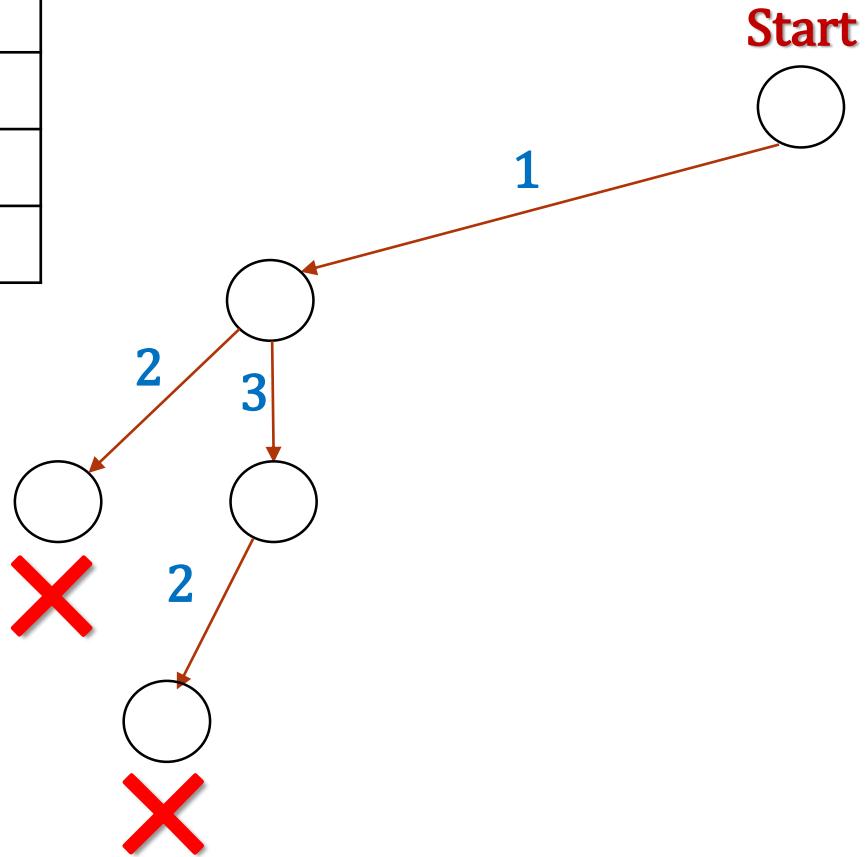
	1	2	3	4
1	Q			
2				
3				
4				



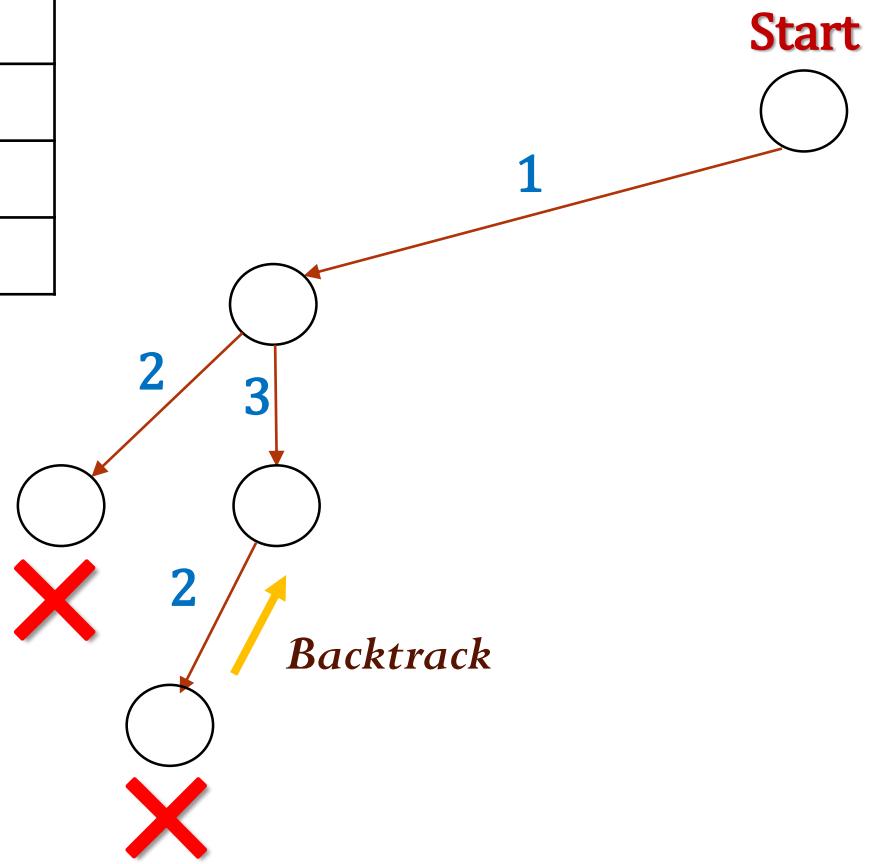
	1	2	3	4
1	Q			
2			Q	
3				
4				



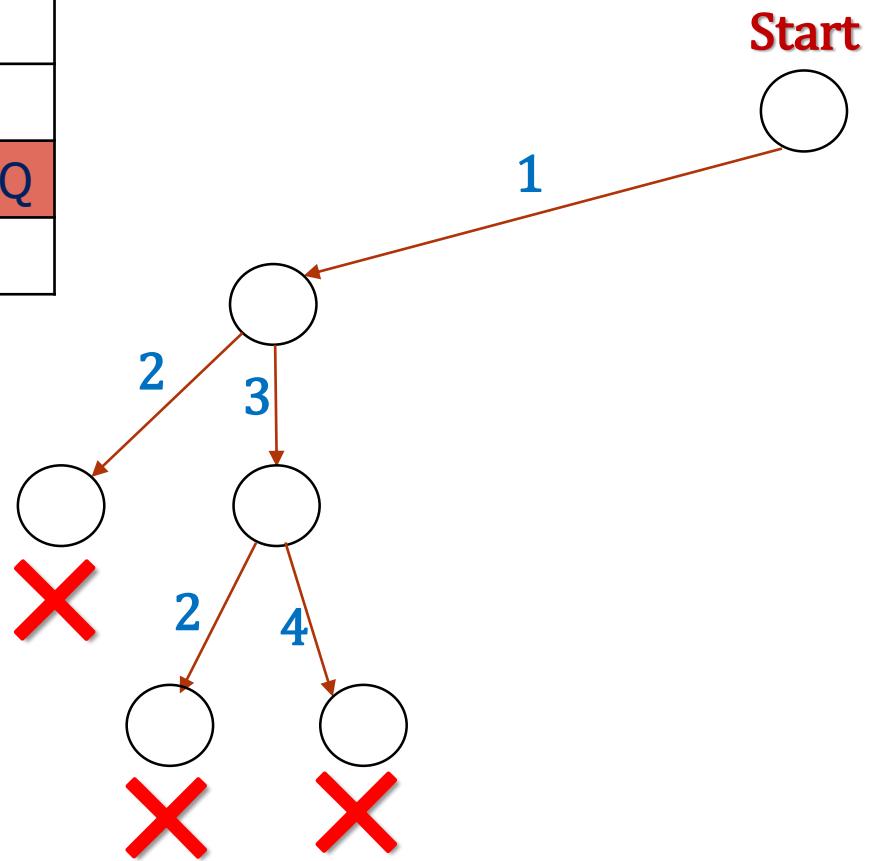
	1	2	3	4
1	Q			
2			Q	
3		Q		
4				



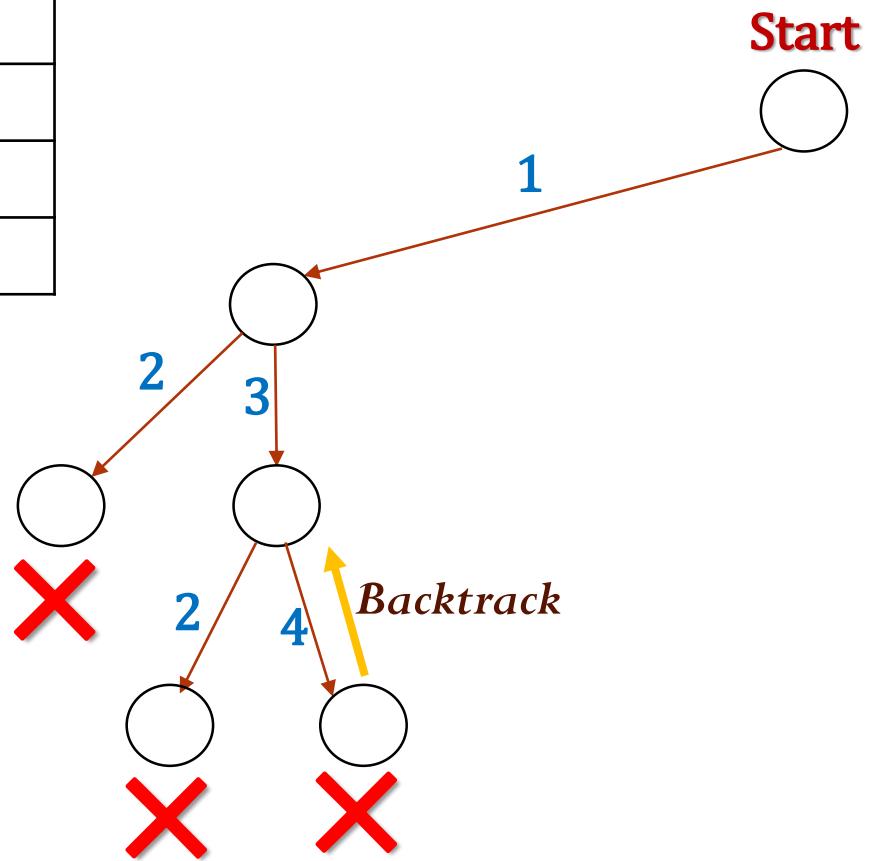
	1	2	3	4
1	Q			
2			Q	
3				
4				



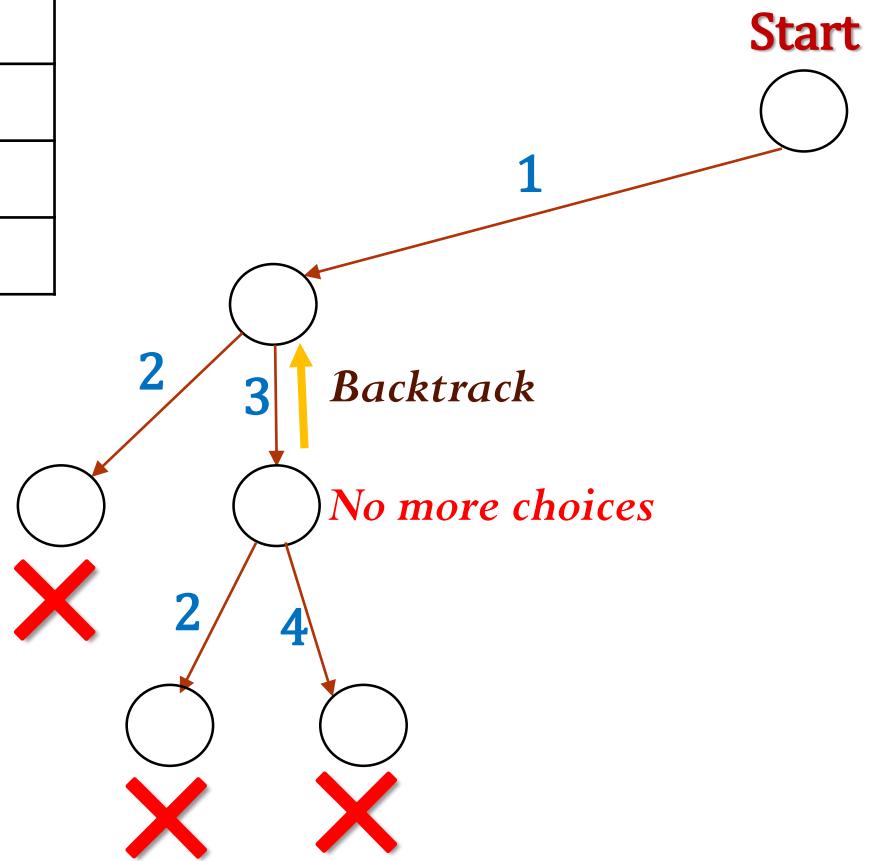
	1	2	3	4
1	Q			
2			Q	
3				Q
4				



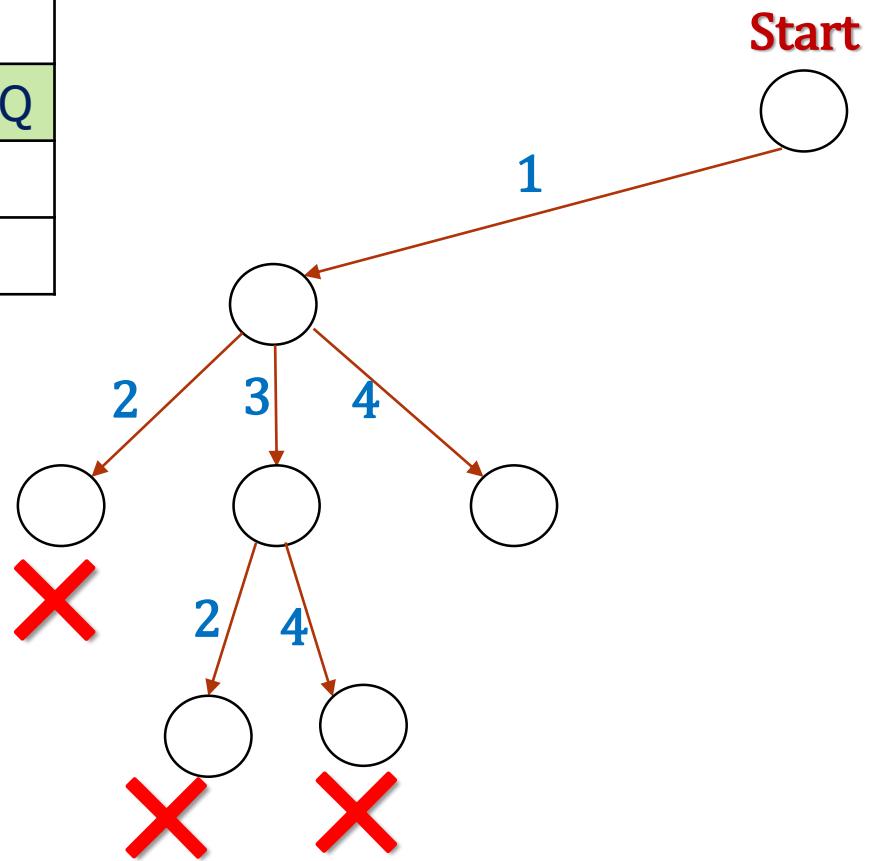
	1	2	3	4
1	Q			
2			Q	
3				
4				



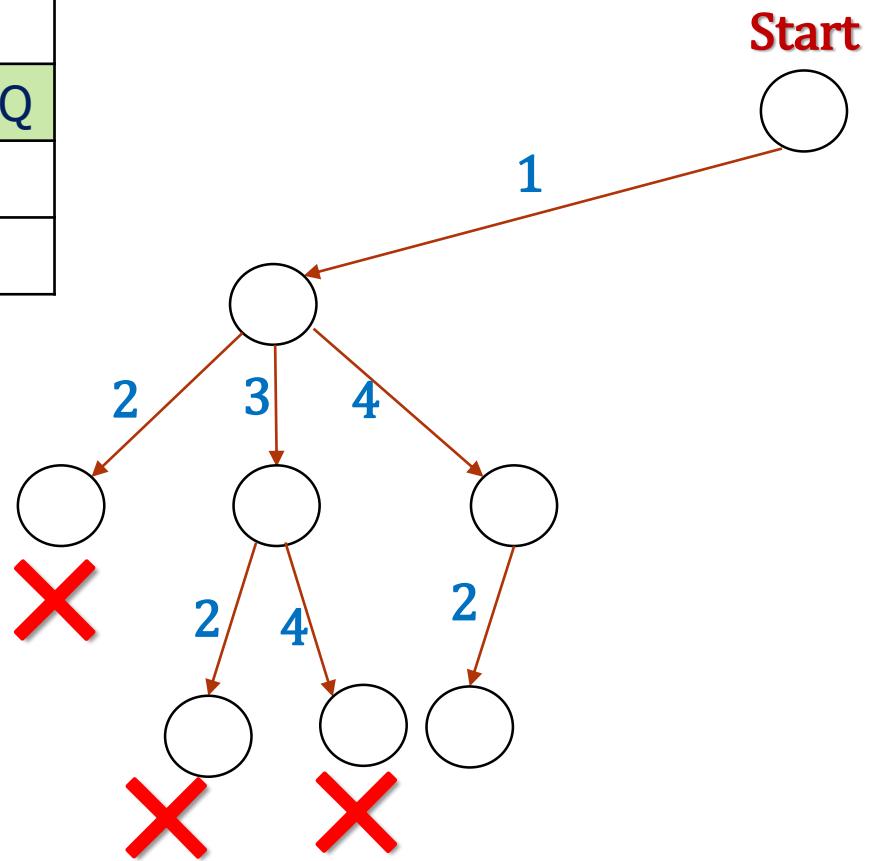
	1	2	3	4
1	Q			
2				
3				
4				



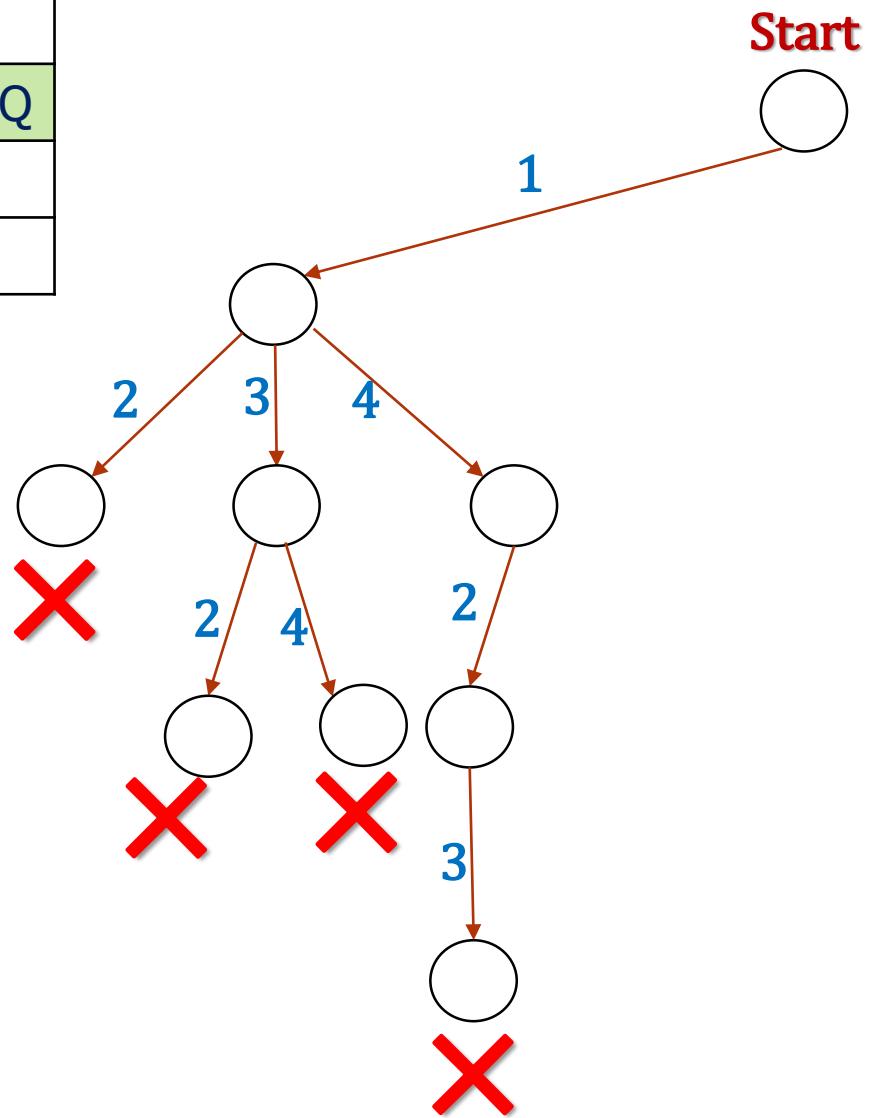
	1	2	3	4
1	Q			
2				Q
3				
4				



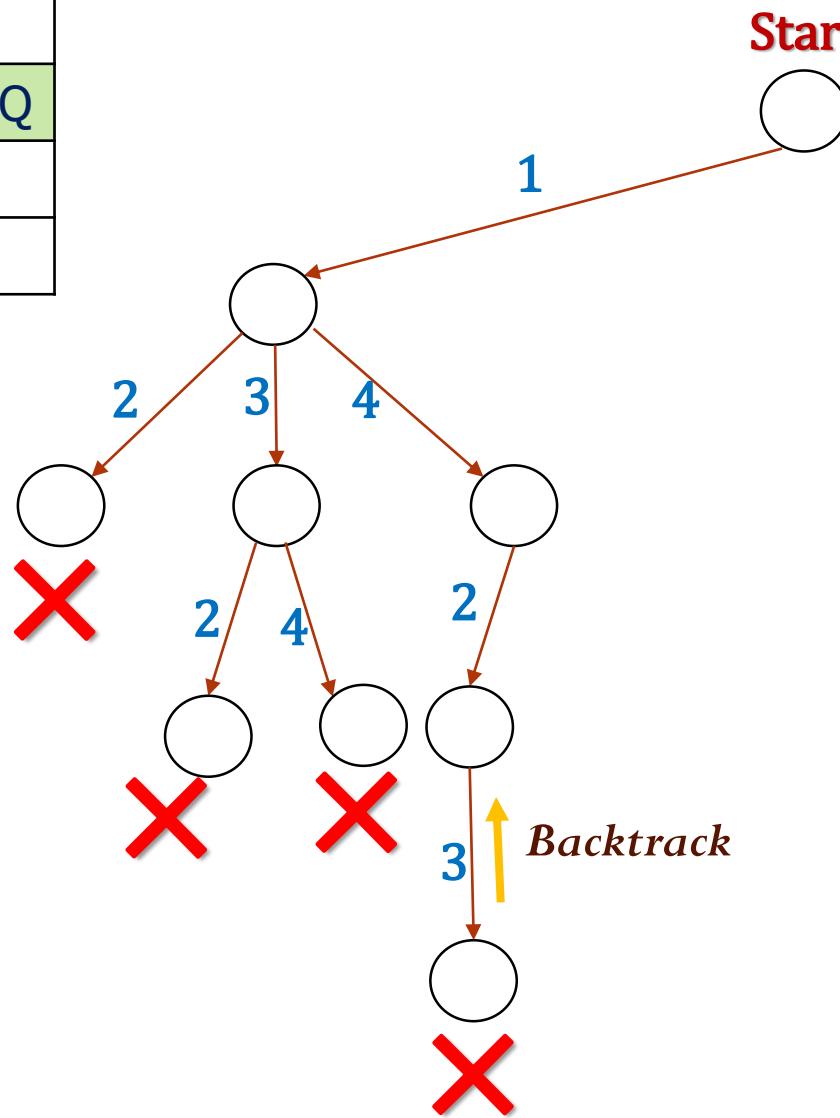
	1	2	3	4
1	Q			
2				Q
3		Q		
4				



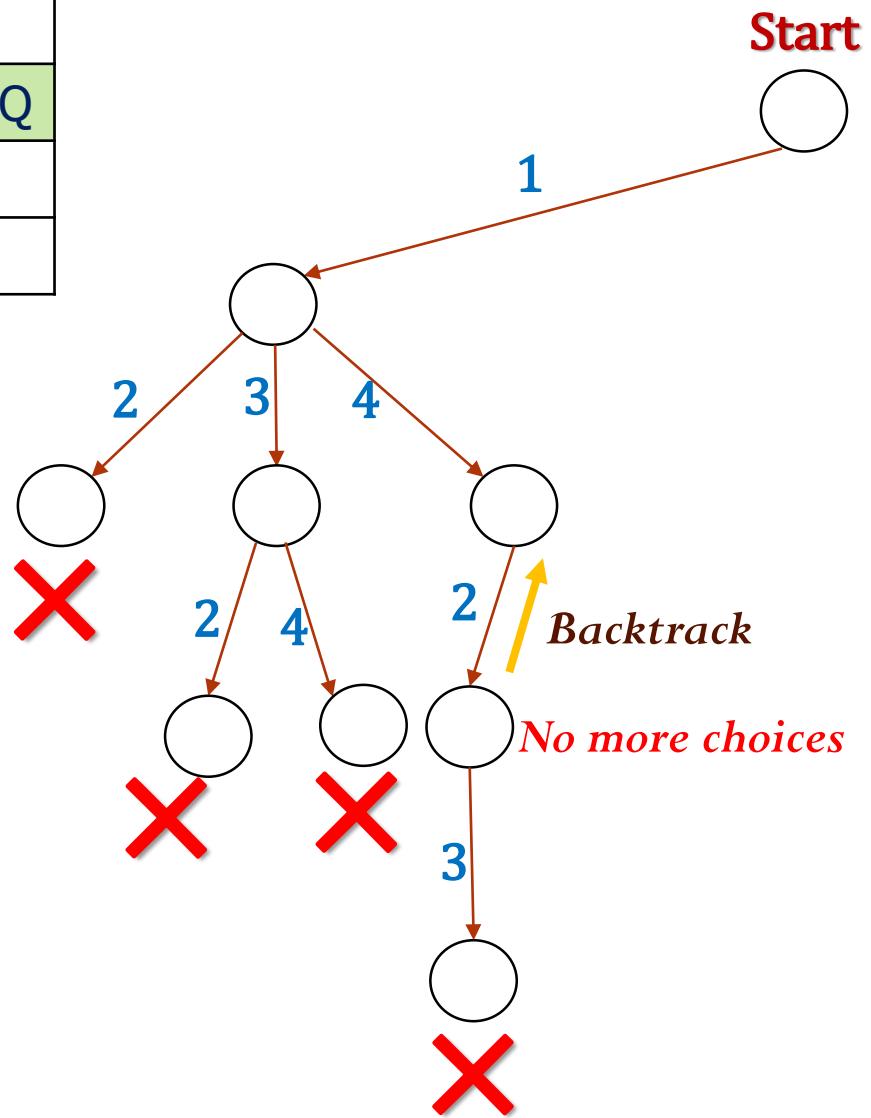
	1	2	3	4
1	Q			
2				Q
3		Q		
4			Q	



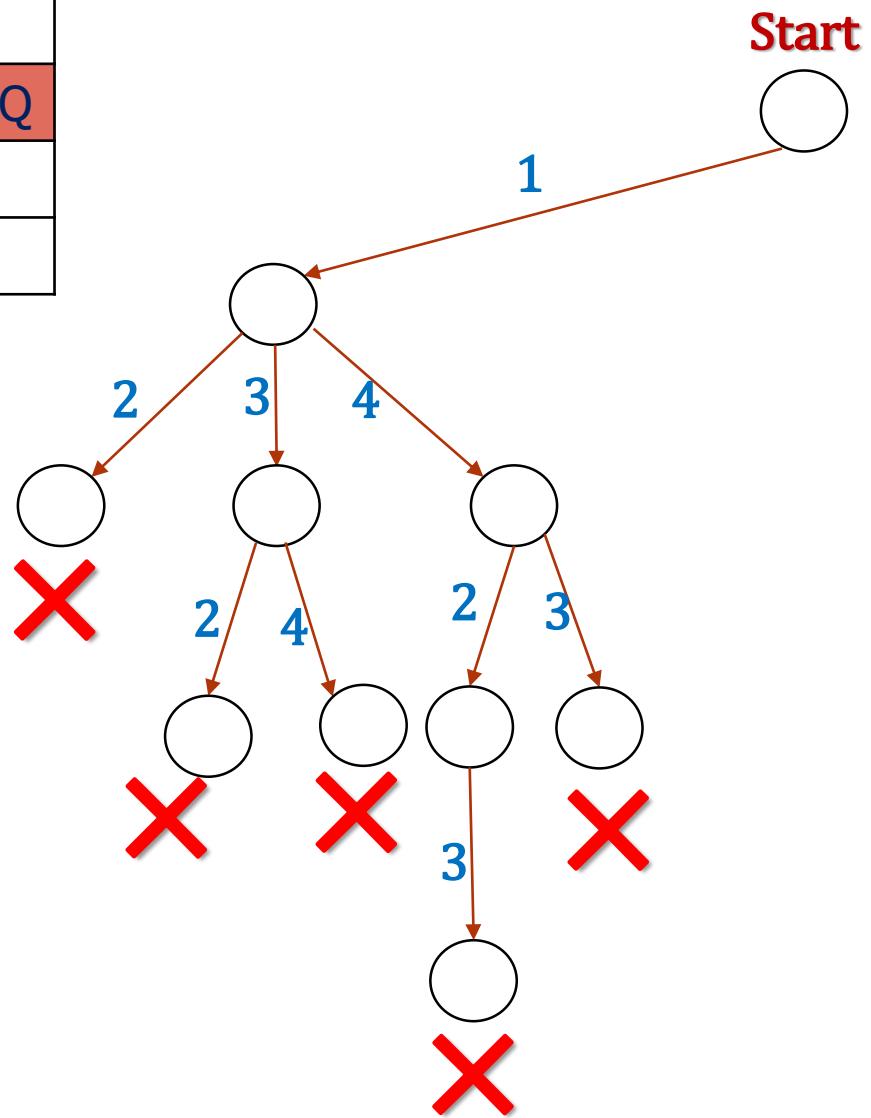
	1	2	3	4
1	Q			
2				Q
3		Q		
4				



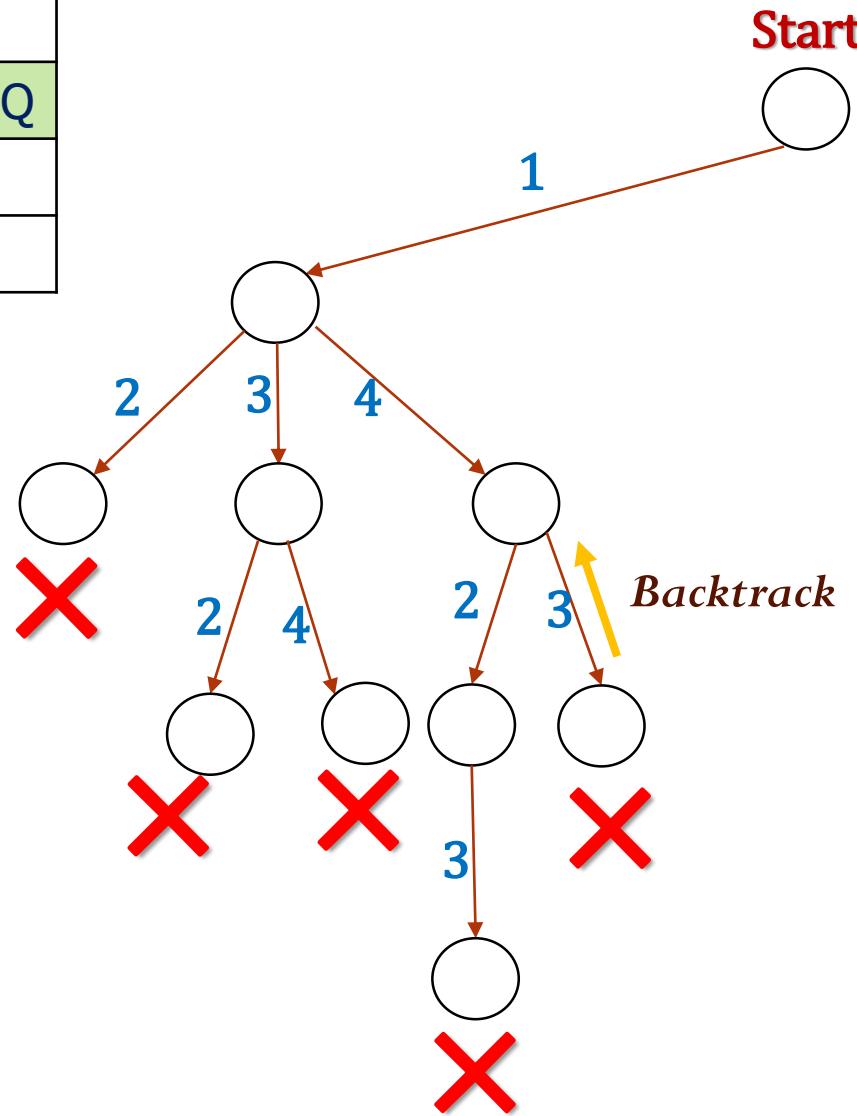
	1	2	3	4
1	Q			
2				Q
3				
4				



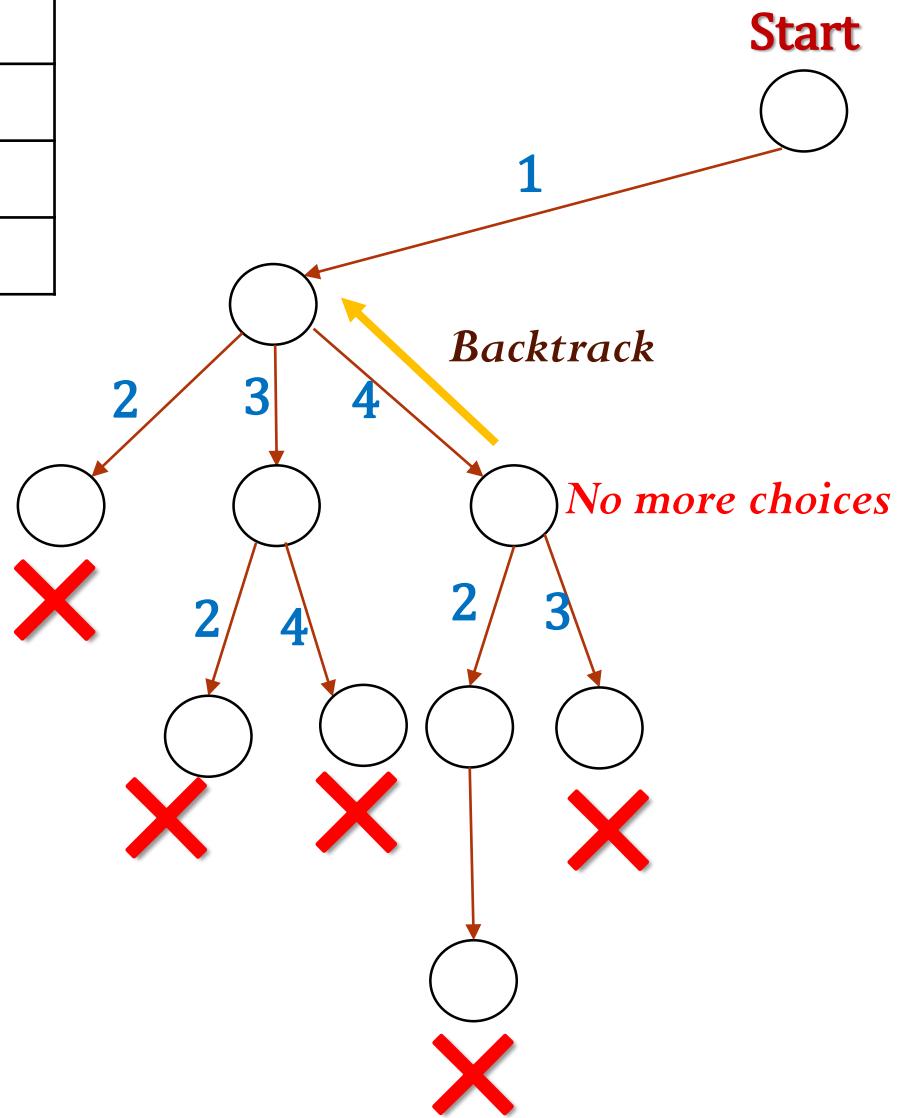
	1	2	3	4
1	Q			
2				Q
3			Q	
4				



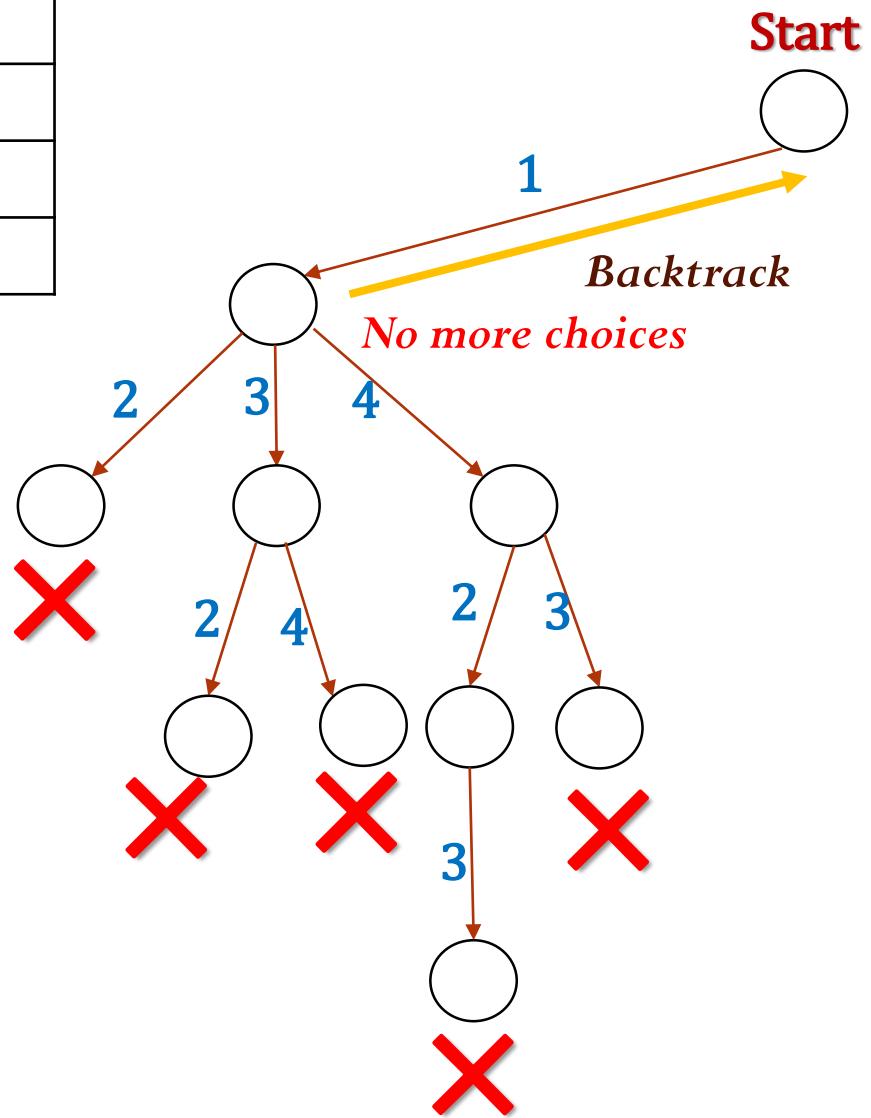
	1	2	3	4
1	Q			
2				Q
3				
4				



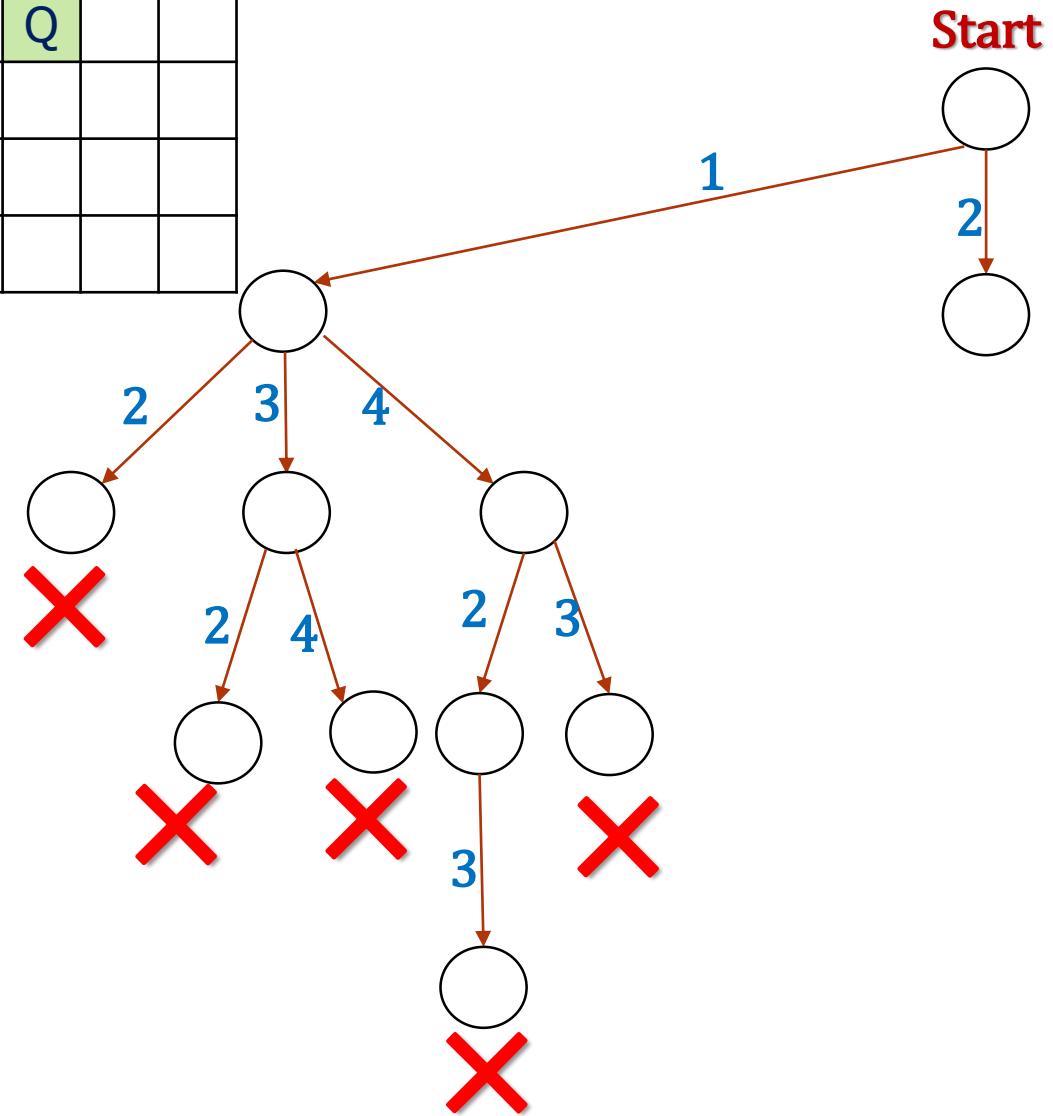
	1	2	3	4
1	Q			
2				
3				
4				



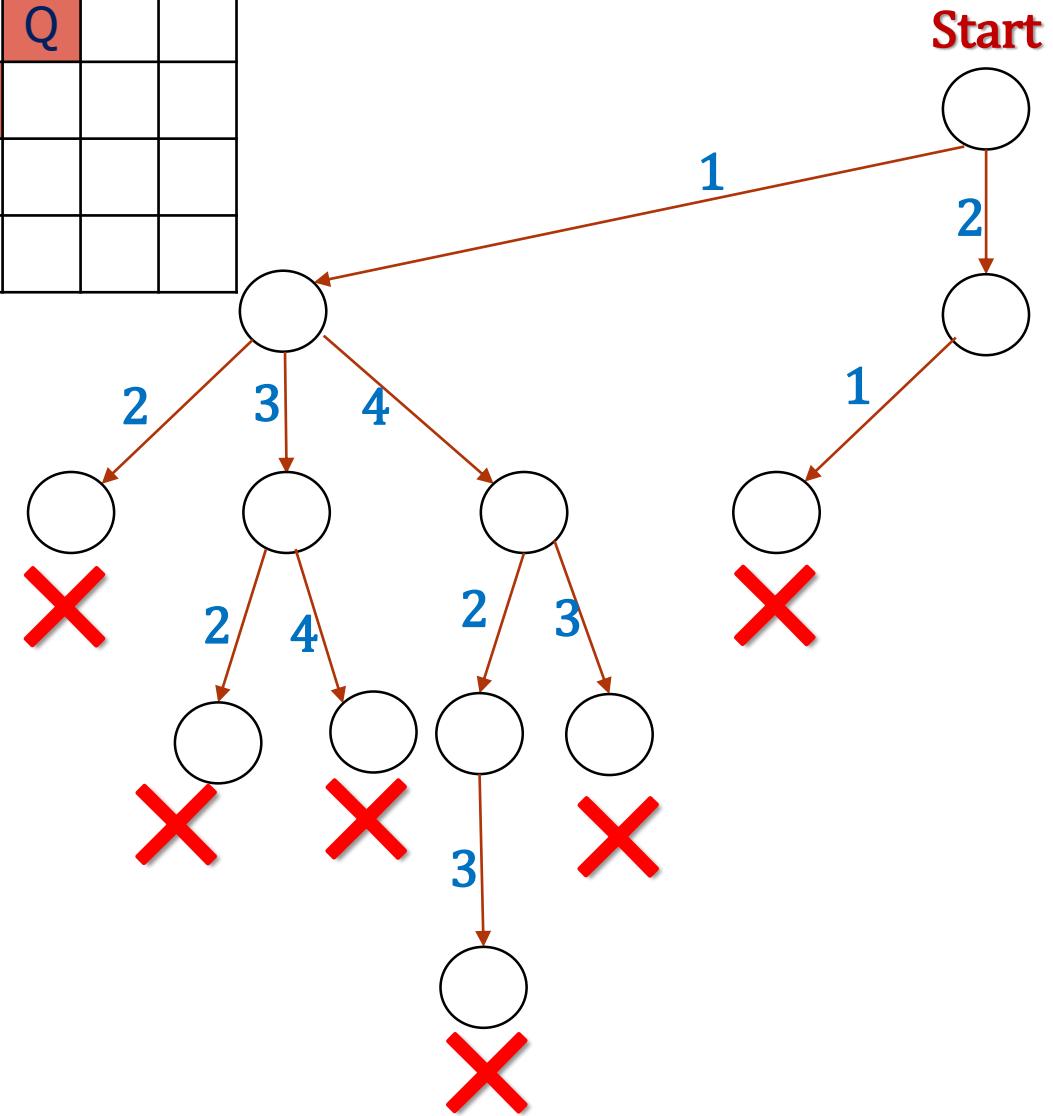
	1	2	3	4
1				
2				
3				
4				



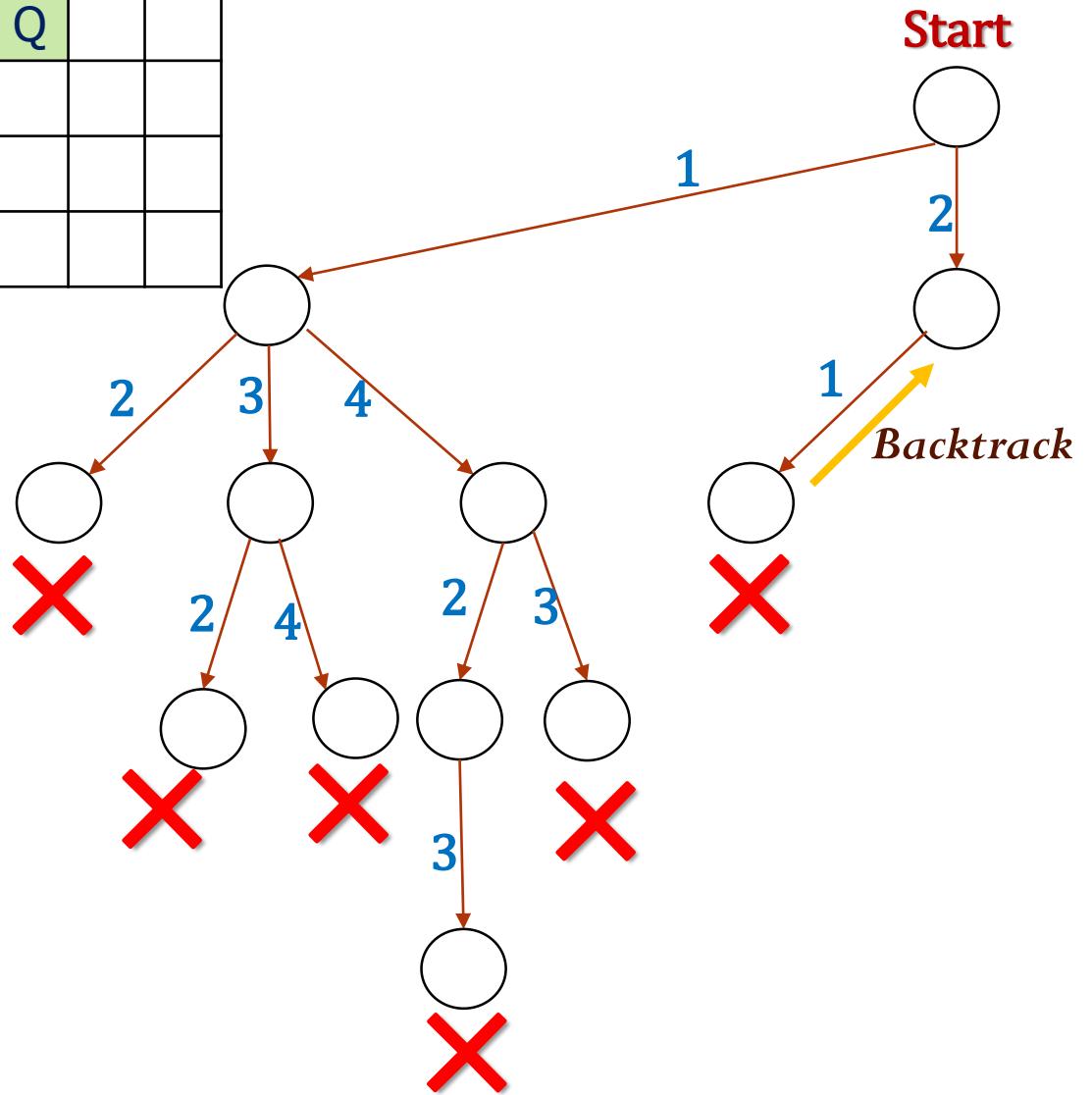
	1	2	3	4
1		Q		
2				
3				
4				



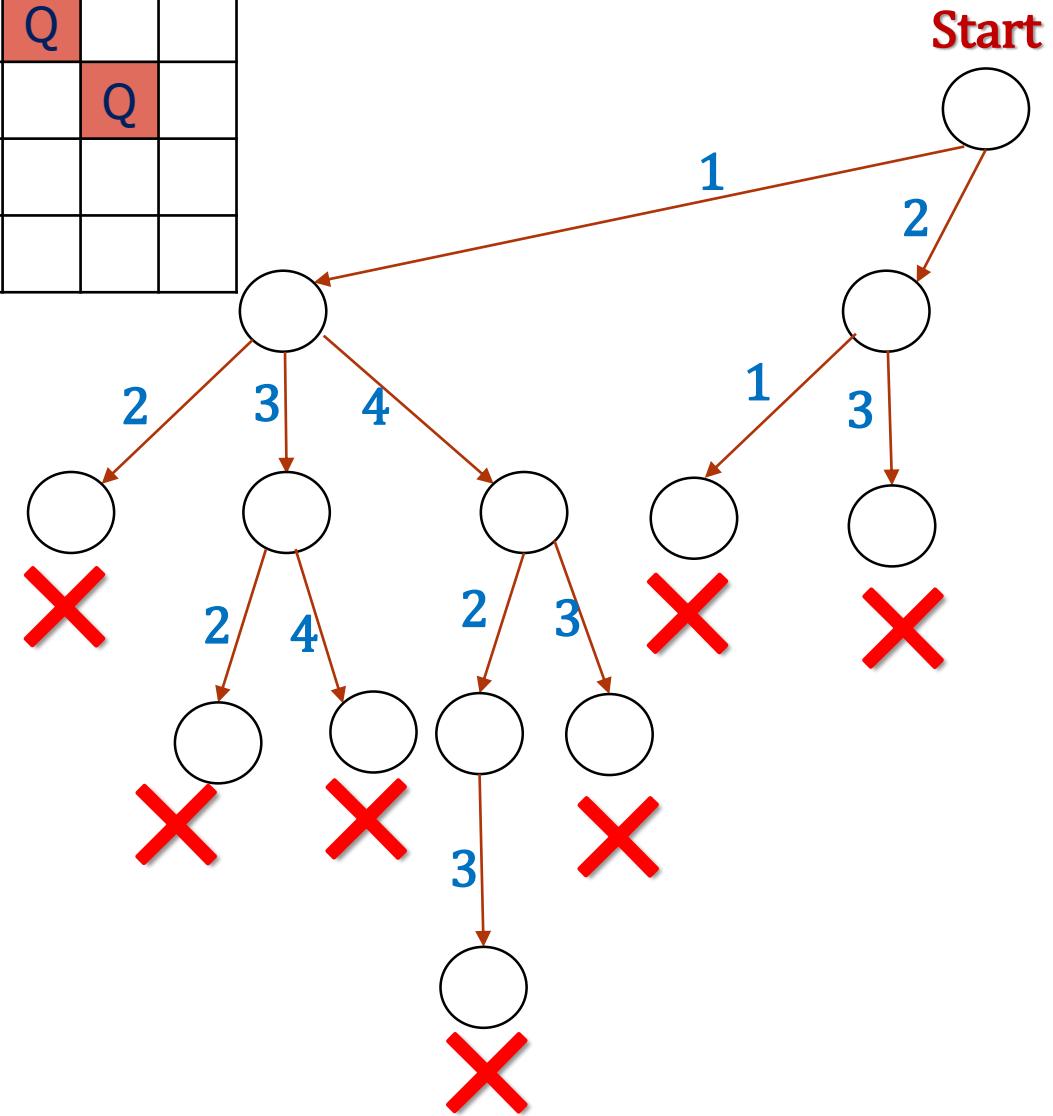
	1	2	3	4
1		Q		
2	Q			
3				
4				



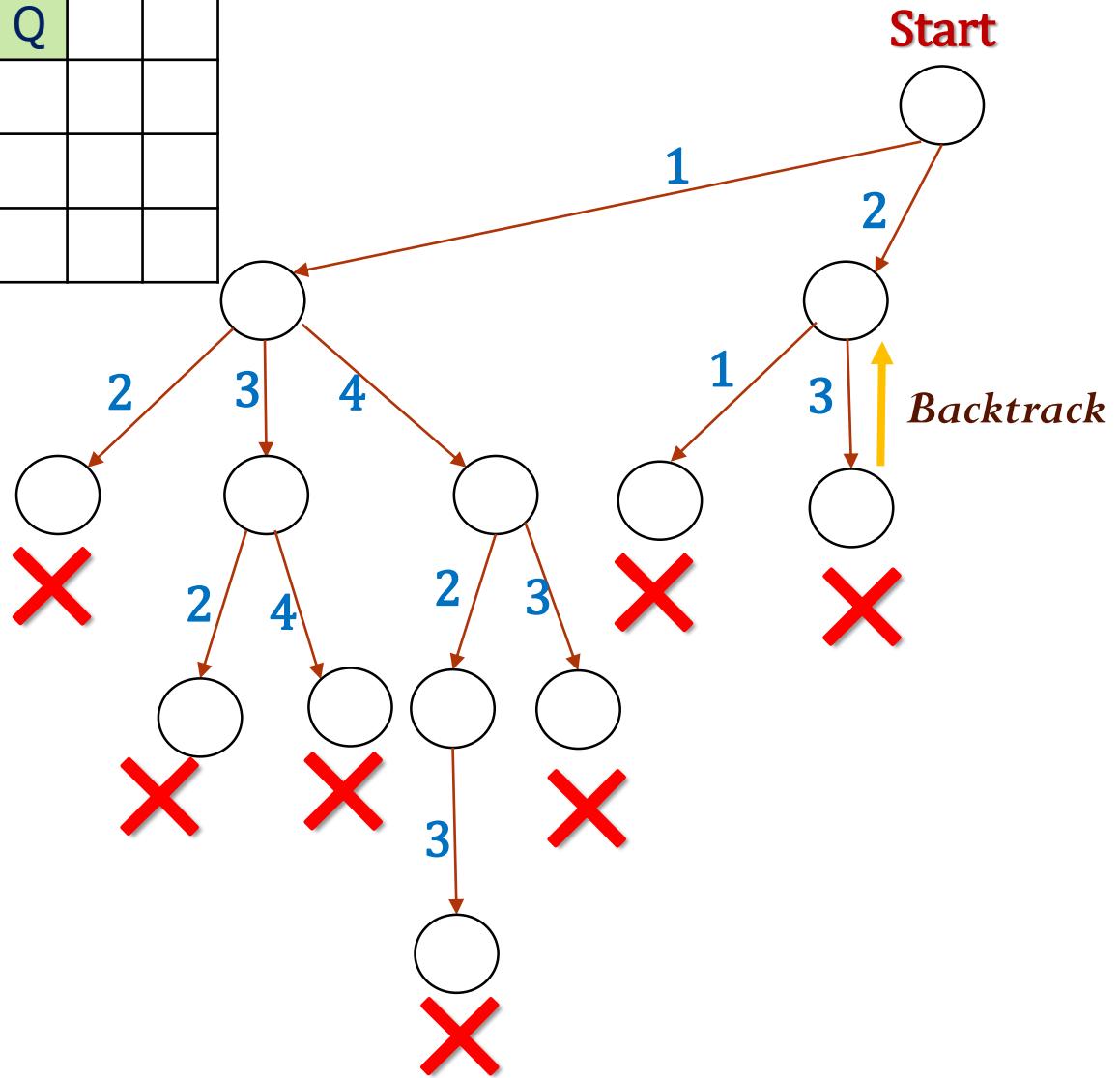
	1	2	3	4
1		Q		
2				
3				
4				



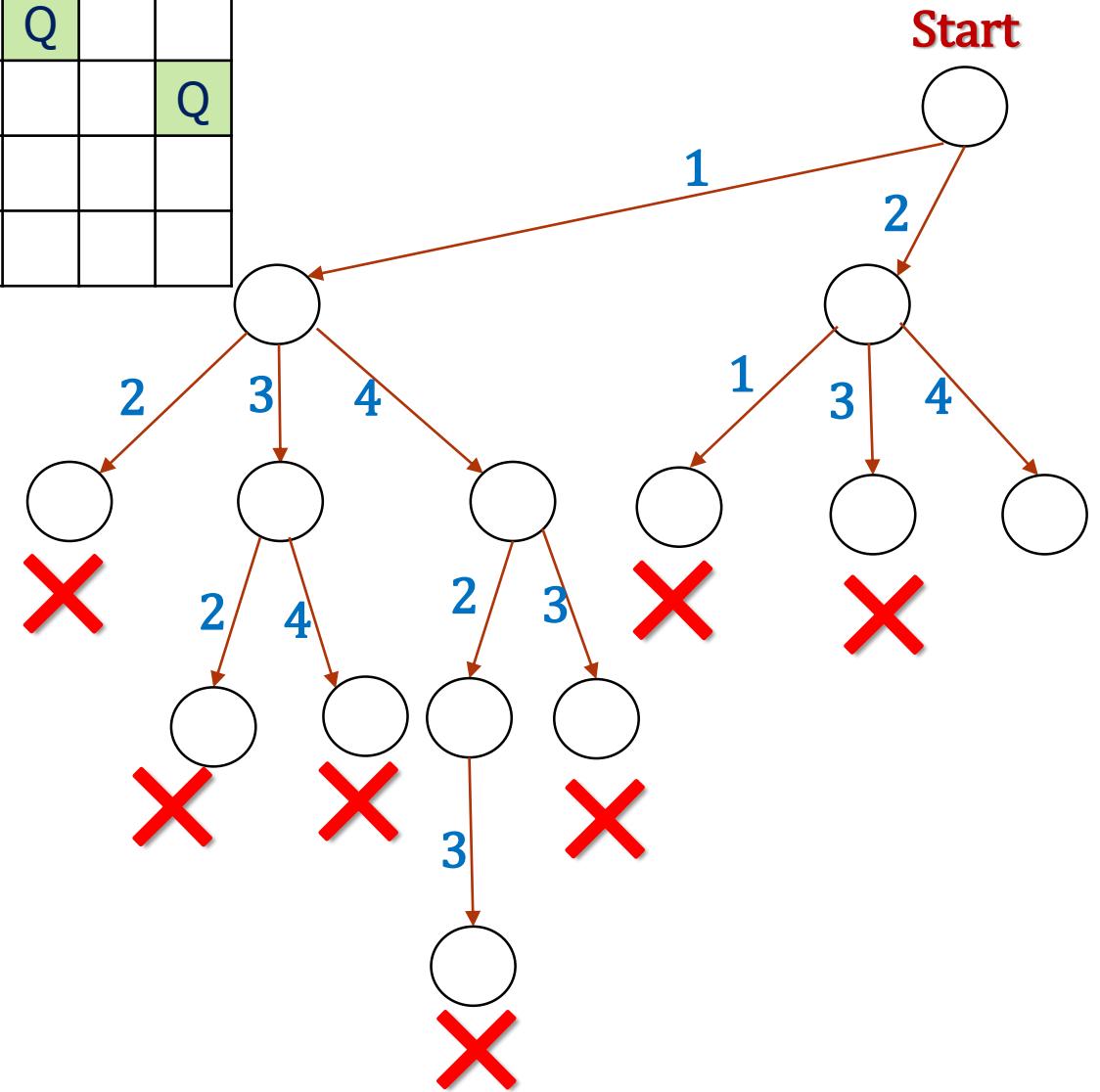
	1	2	3	4
1		Q		
2			Q	
3				
4				



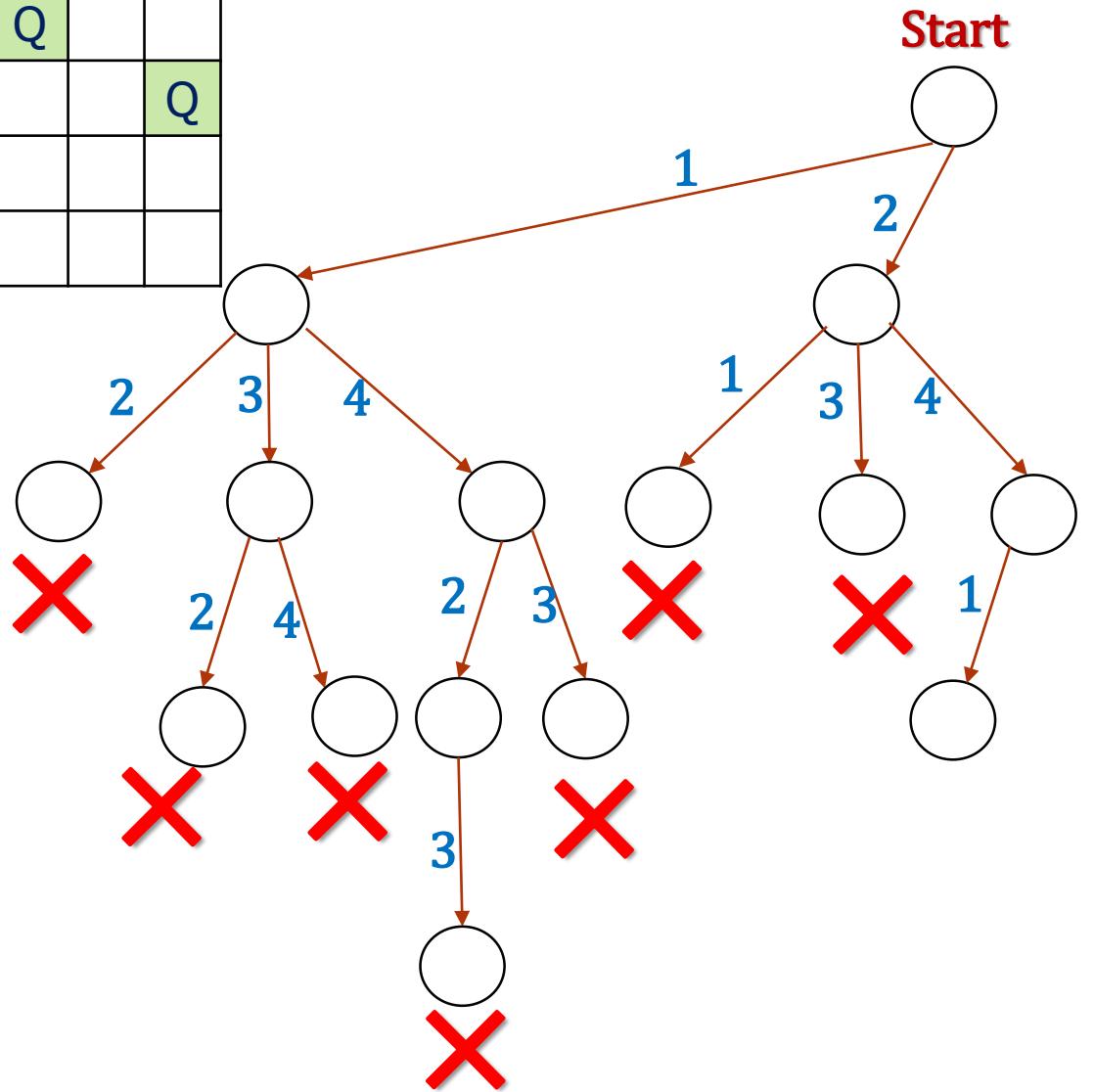
	1	2	3	4
1		Q		
2				
3				
4				



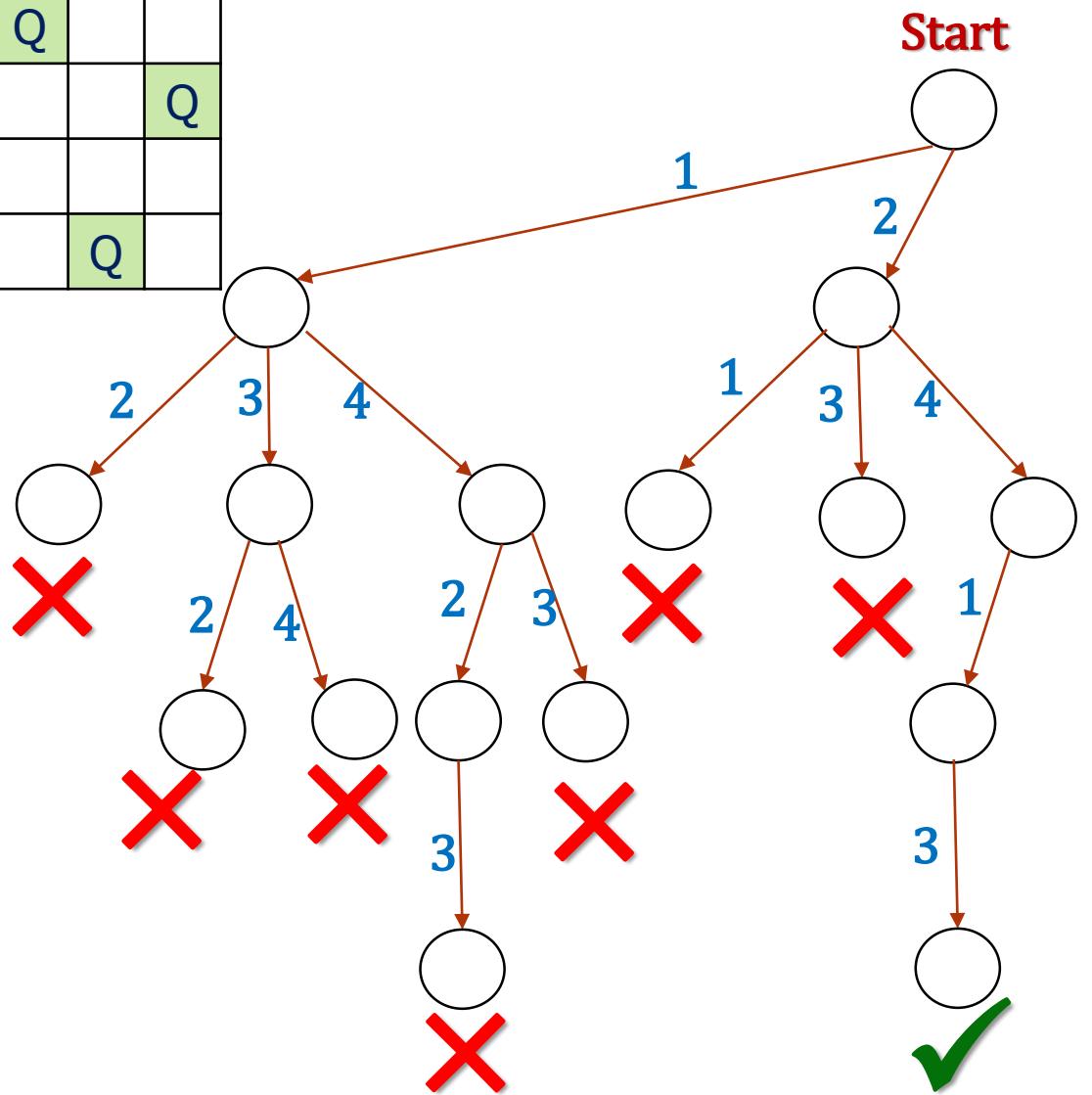
	1	2	3	4
1		Q		
2				Q
3				
4				



	1	2	3	4
1		Q		
2				Q
3	Q			
4				



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

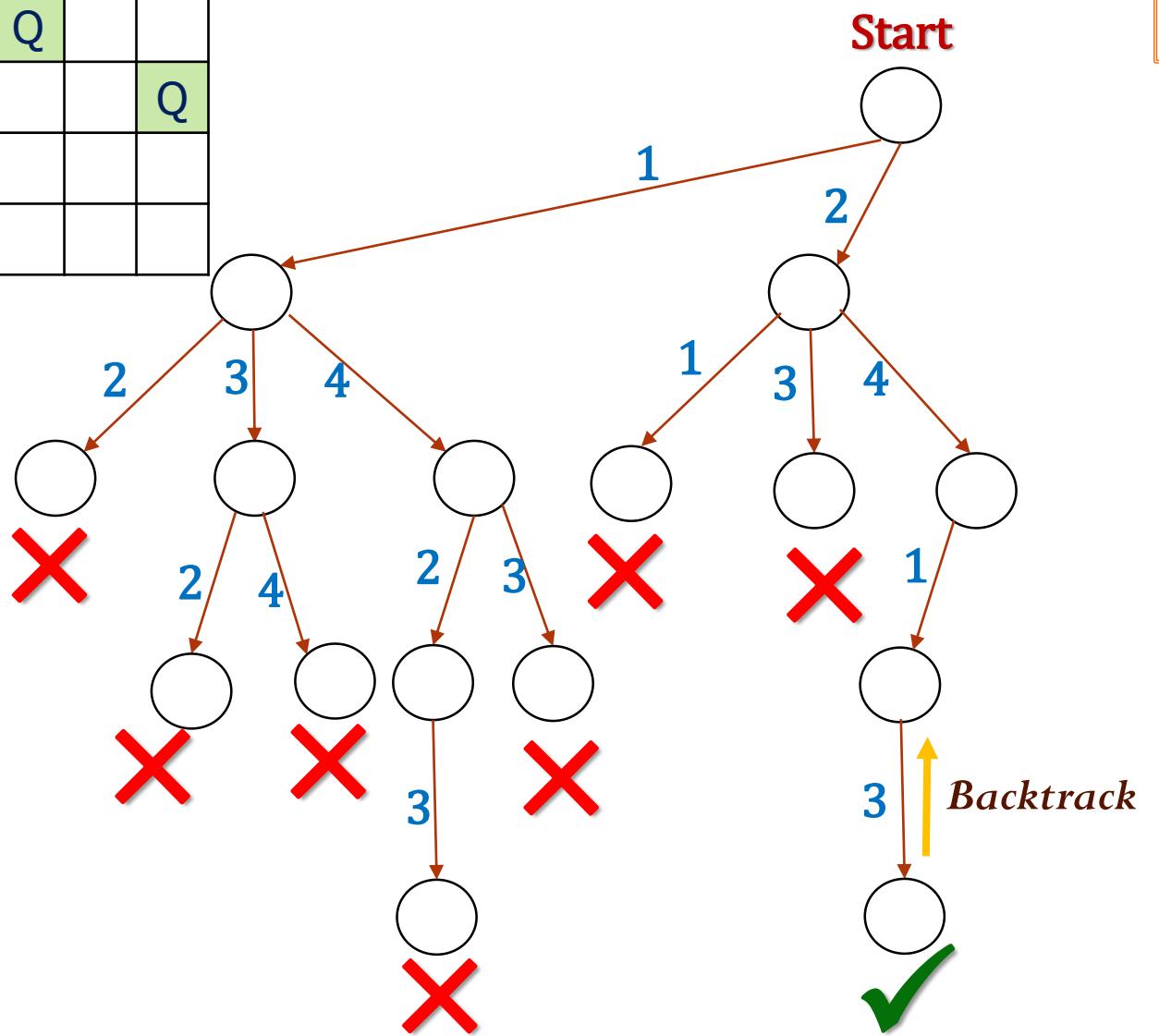


Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				Q
3	Q			
4				

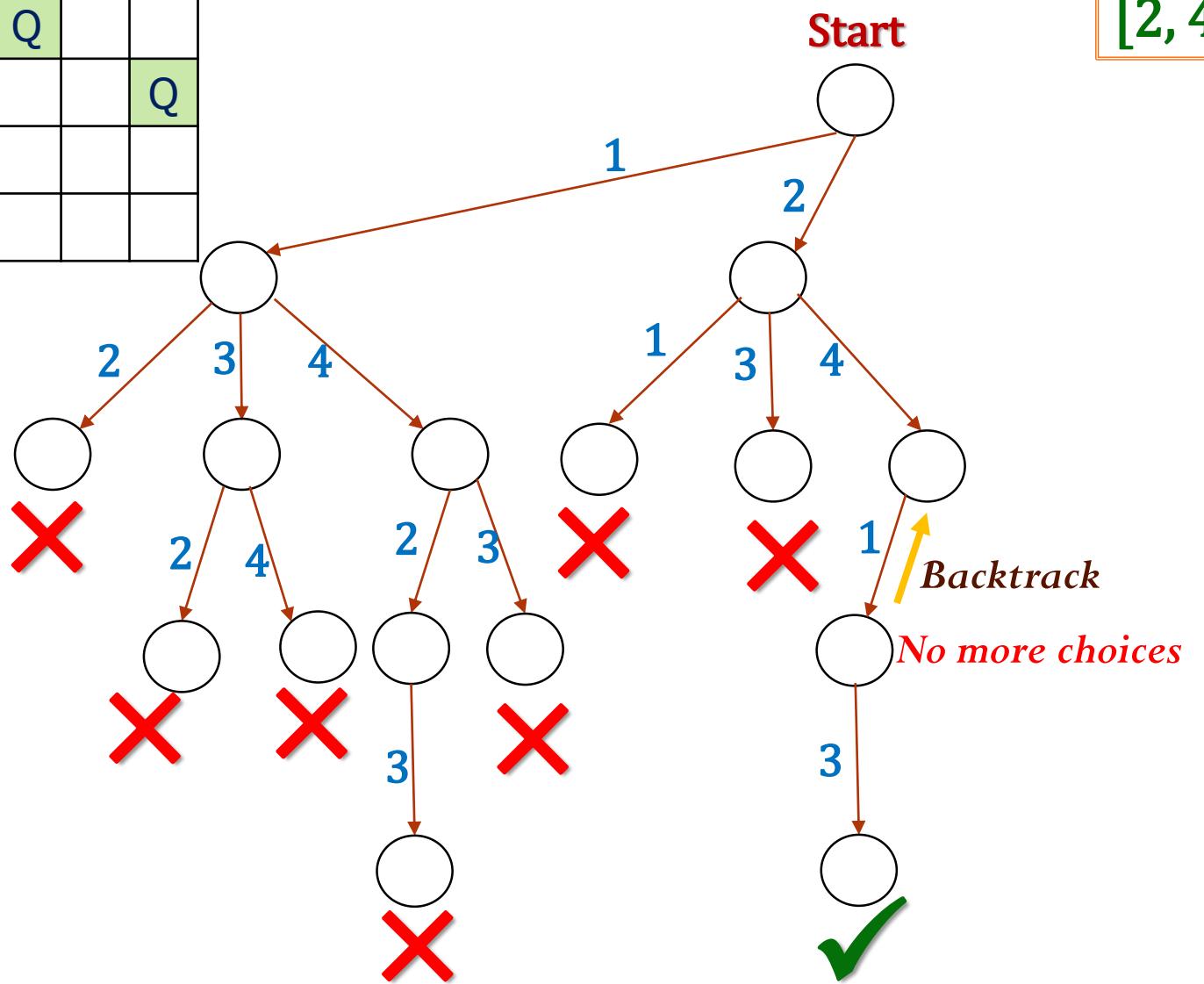


Solutions

[2, 4, 1, 3]



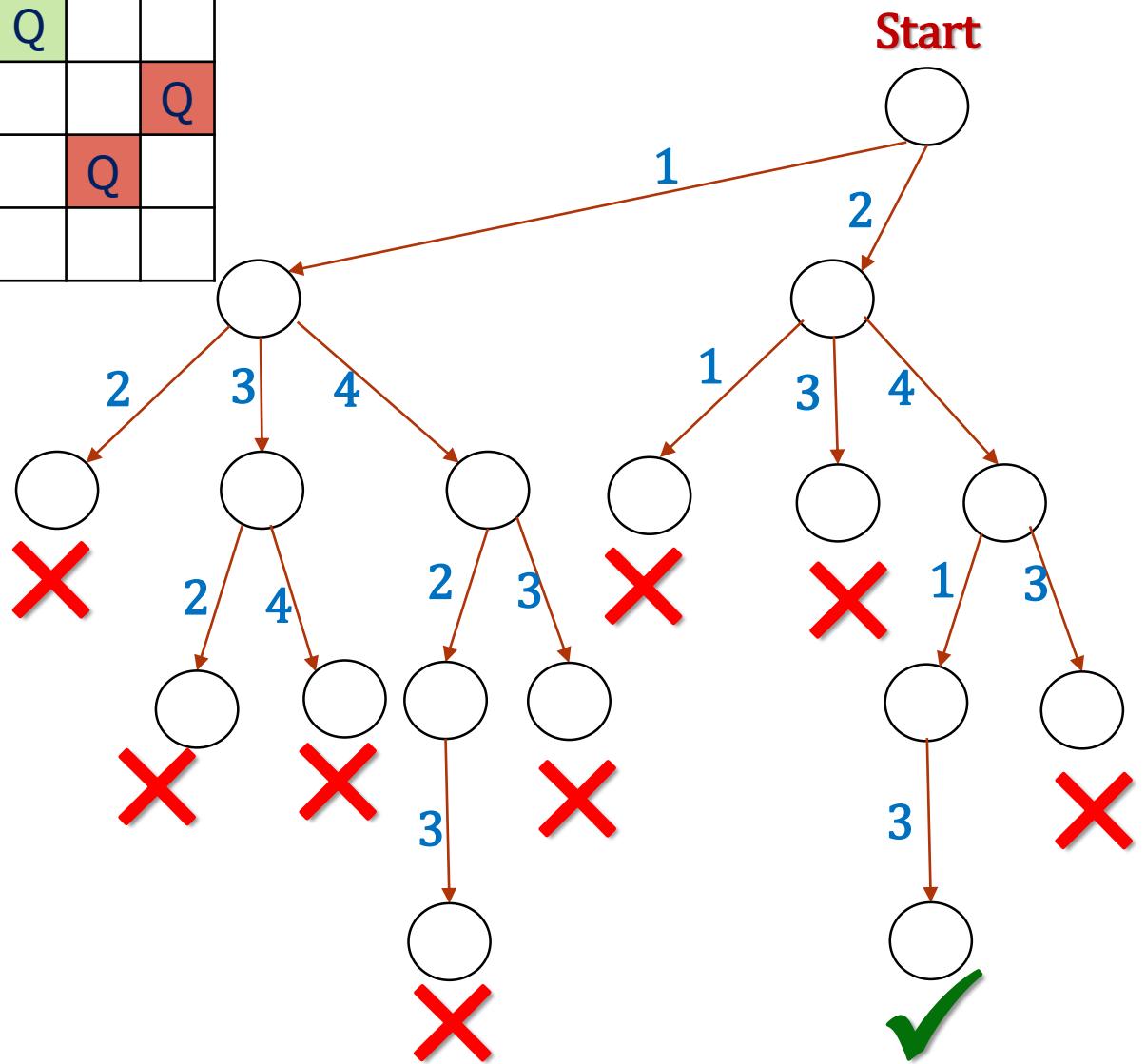
	1	2	3	4
1		Q		
2				Q
3				
4				



Solutions

[2, 4, 1, 3]

	1	2	3	4
1		Q		
2				Q
3			Q	
4				

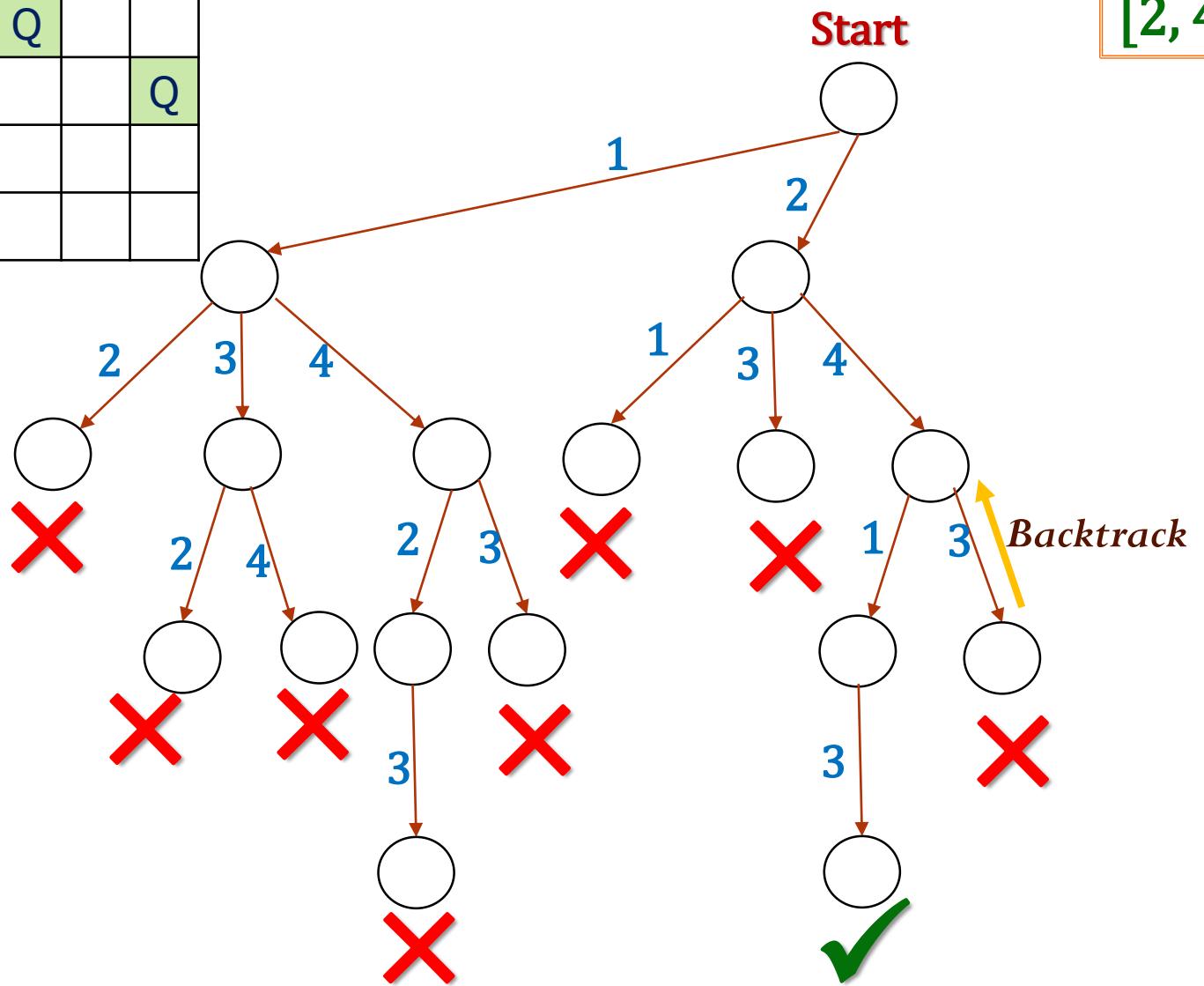


Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				Q
3				
4				

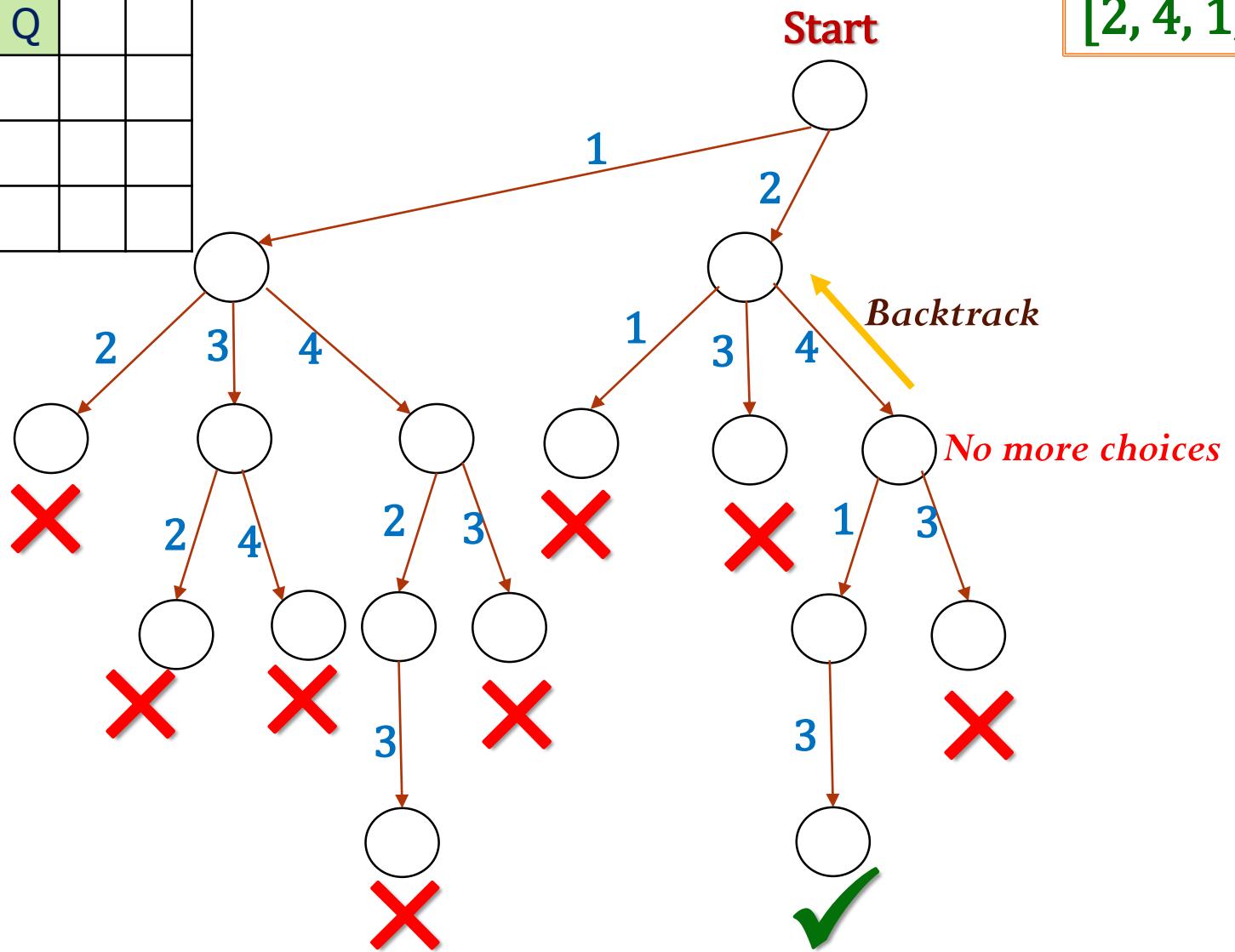


Solutions

[2, 4, 1, 3]



	1	2	3	4
1		Q		
2				
3				
4				

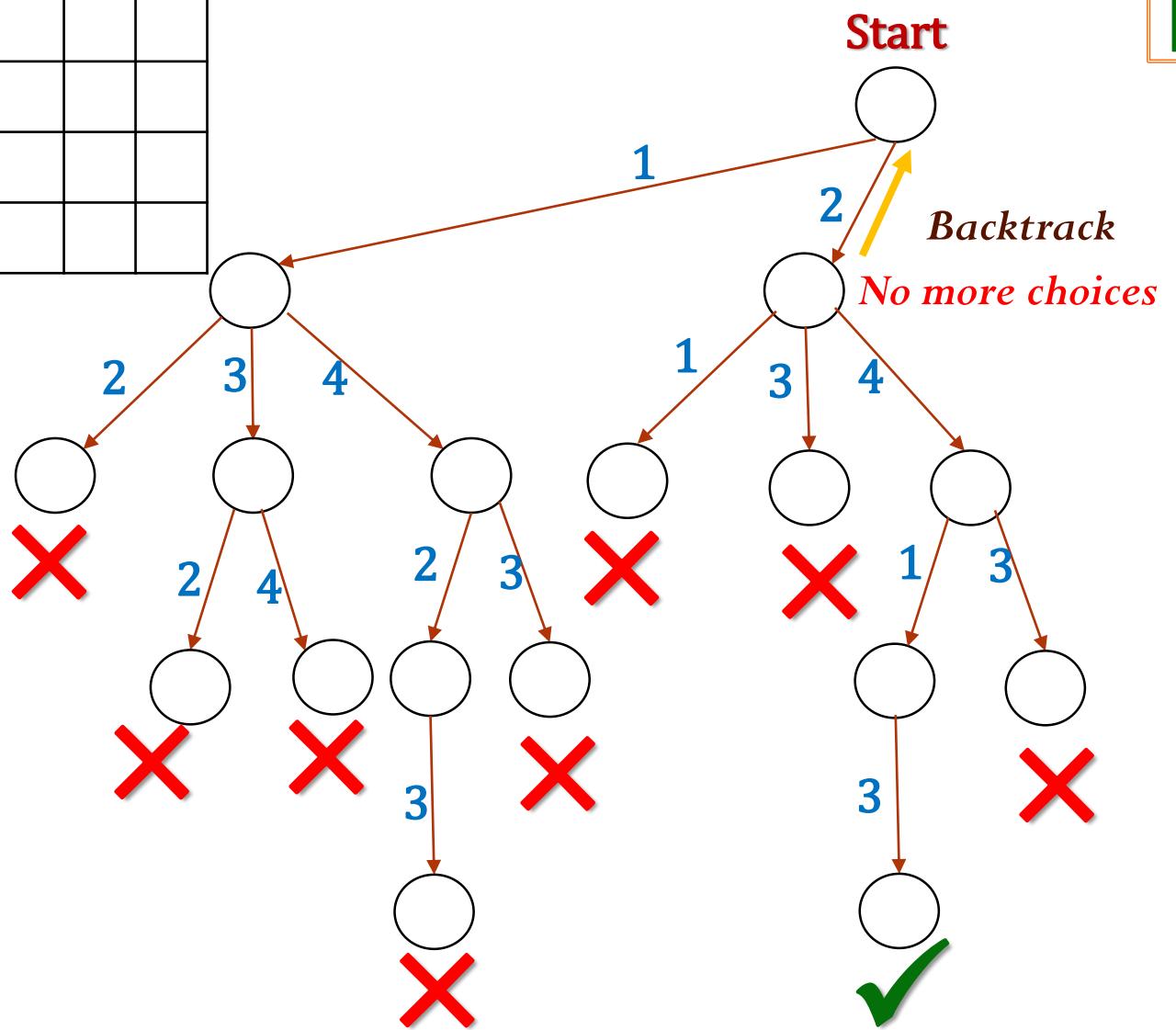


Solutions

[2, 4, 1, 3]



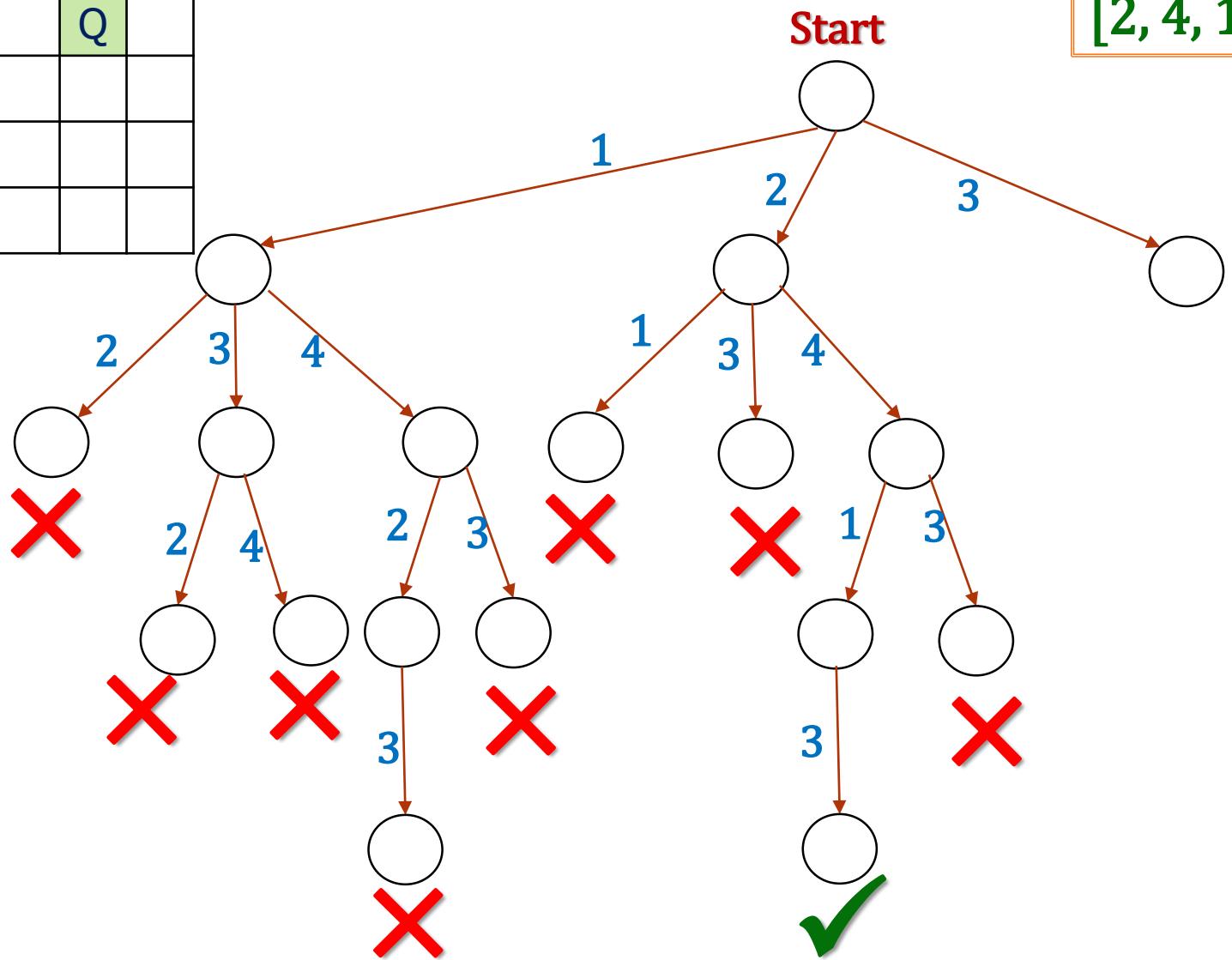
	1	2	3	4
1				
2				
3				
4				



Solutions

[2, 4, 1, 3]

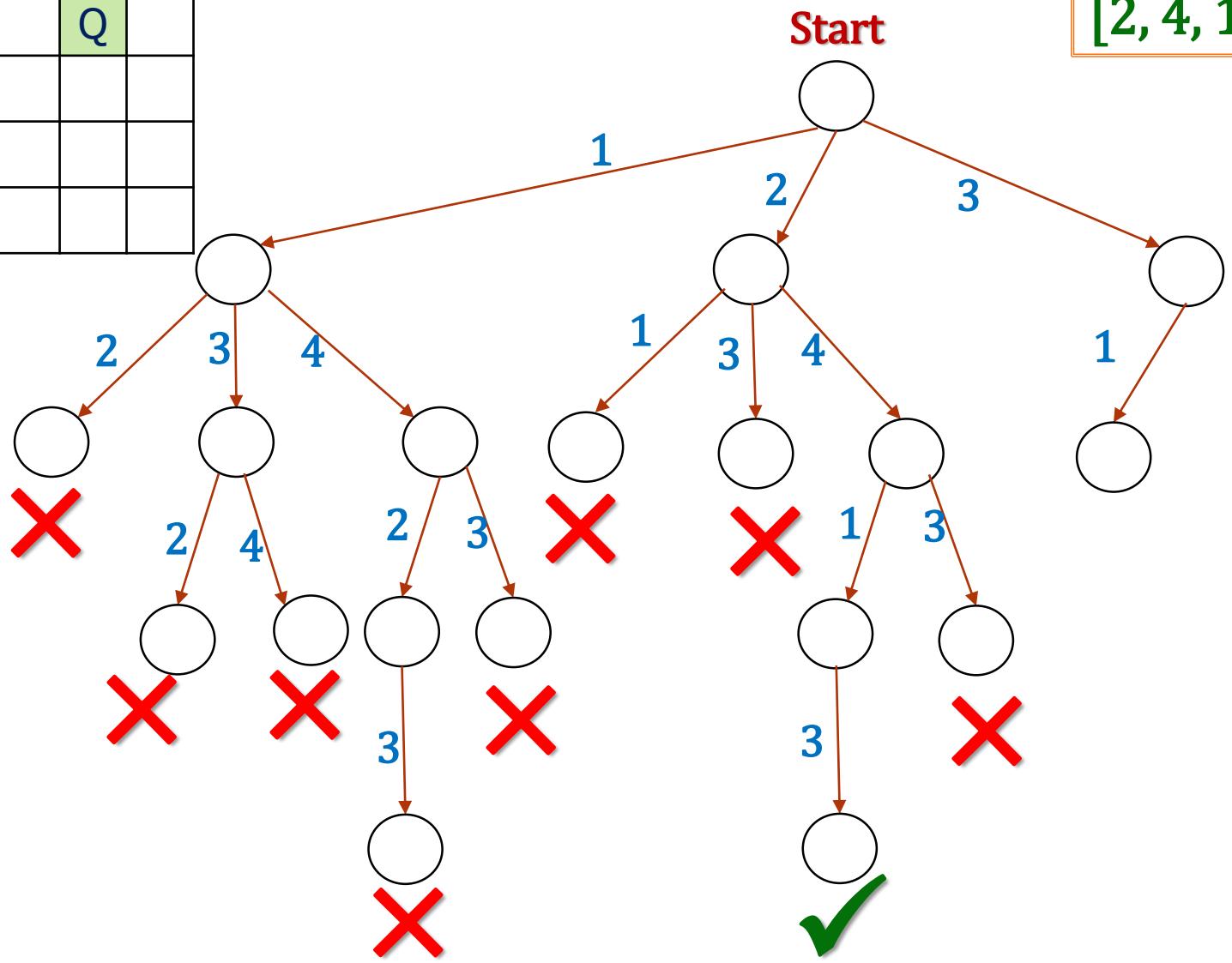
	1	2	3	4
1			Q	
2				
3				
4				



Solutions

[2, 4, 1, 3]

	1	2	3	4
1			Q	
2	Q			
3				
4				



Solutions

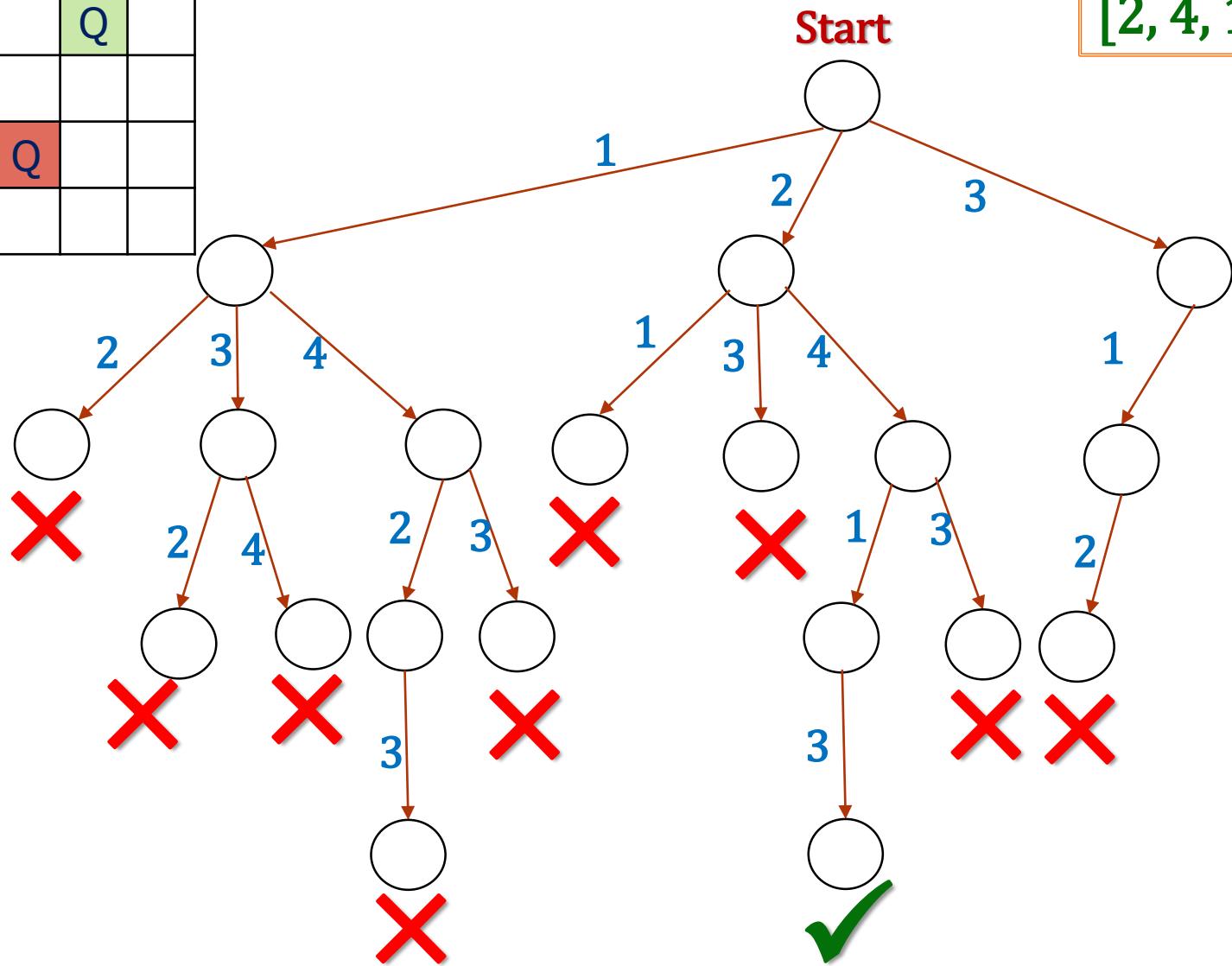
[2, 4, 1, 3]



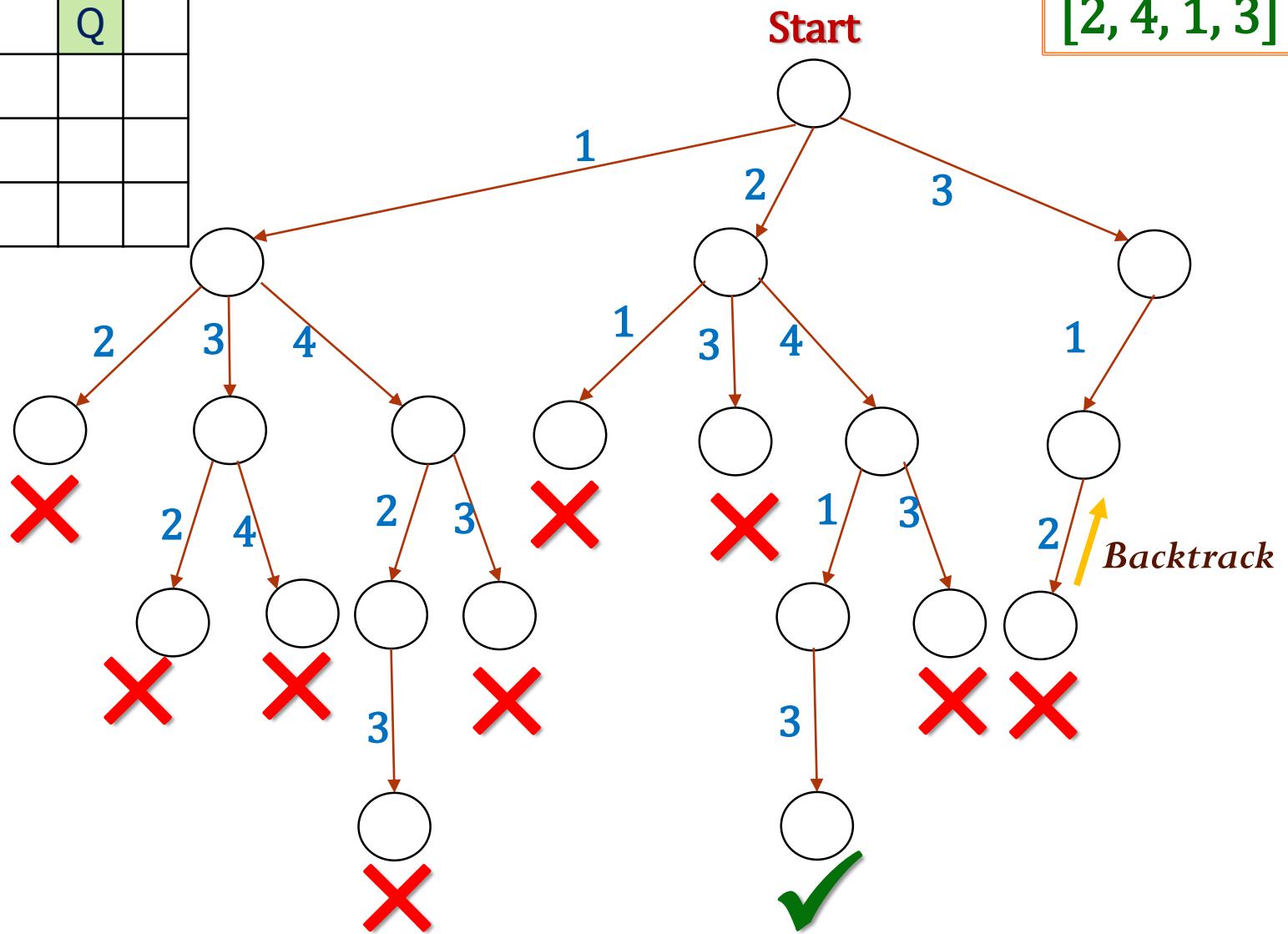
	1	2	3	4
1			Q	
2	Q			
3		Q		
4				

Solutions

[2, 4, 1, 3]



	1	2	3	4
1				Q
2	Q			
3				
4				



Solutions

[2, 4, 1, 3]

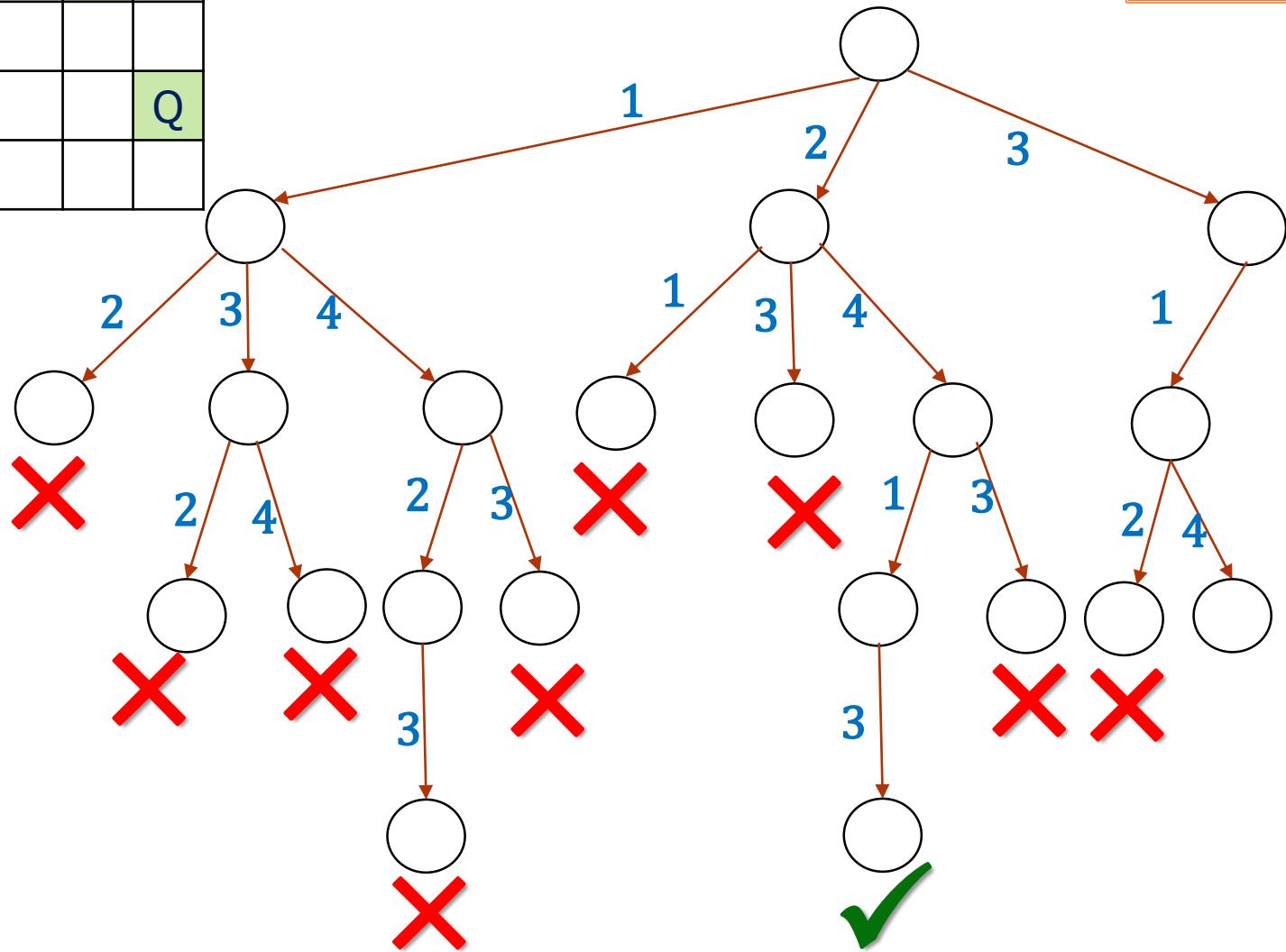


	1	2	3	4
1			Q	
2	Q			
3				Q
4				

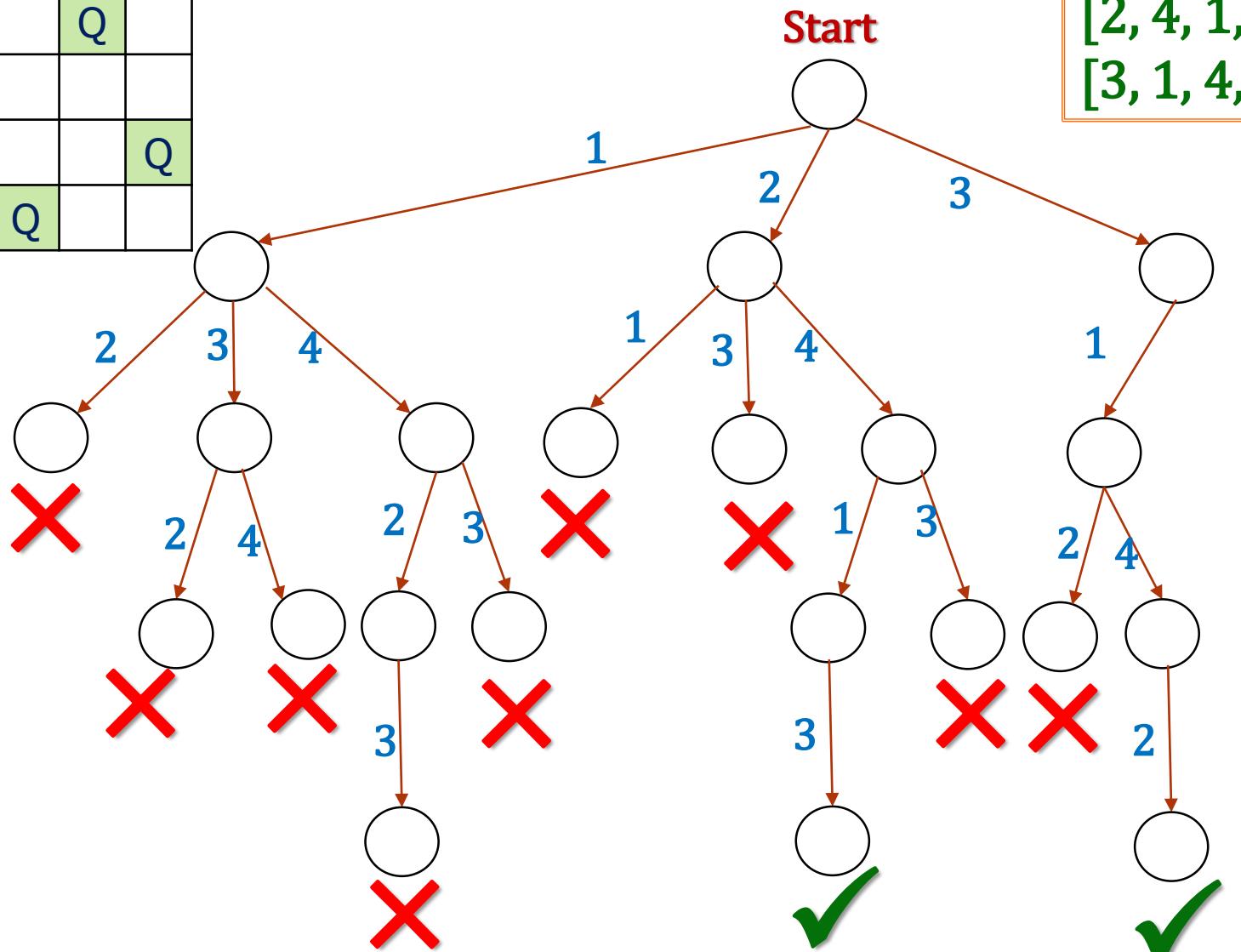
Start

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]

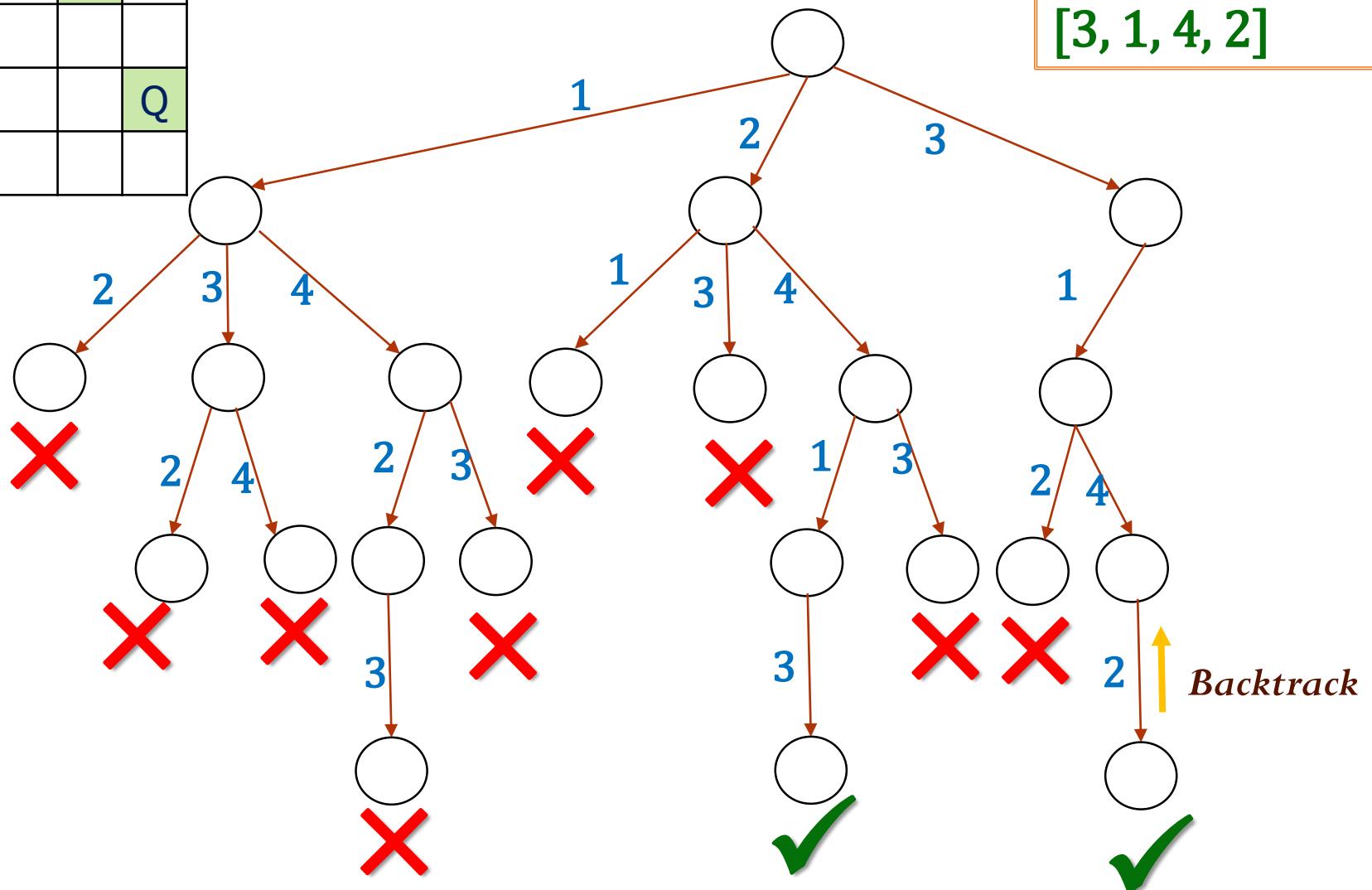


	1	2	3	4
1			Q	
2	Q			
3				Q
4				

Start

Solutions

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

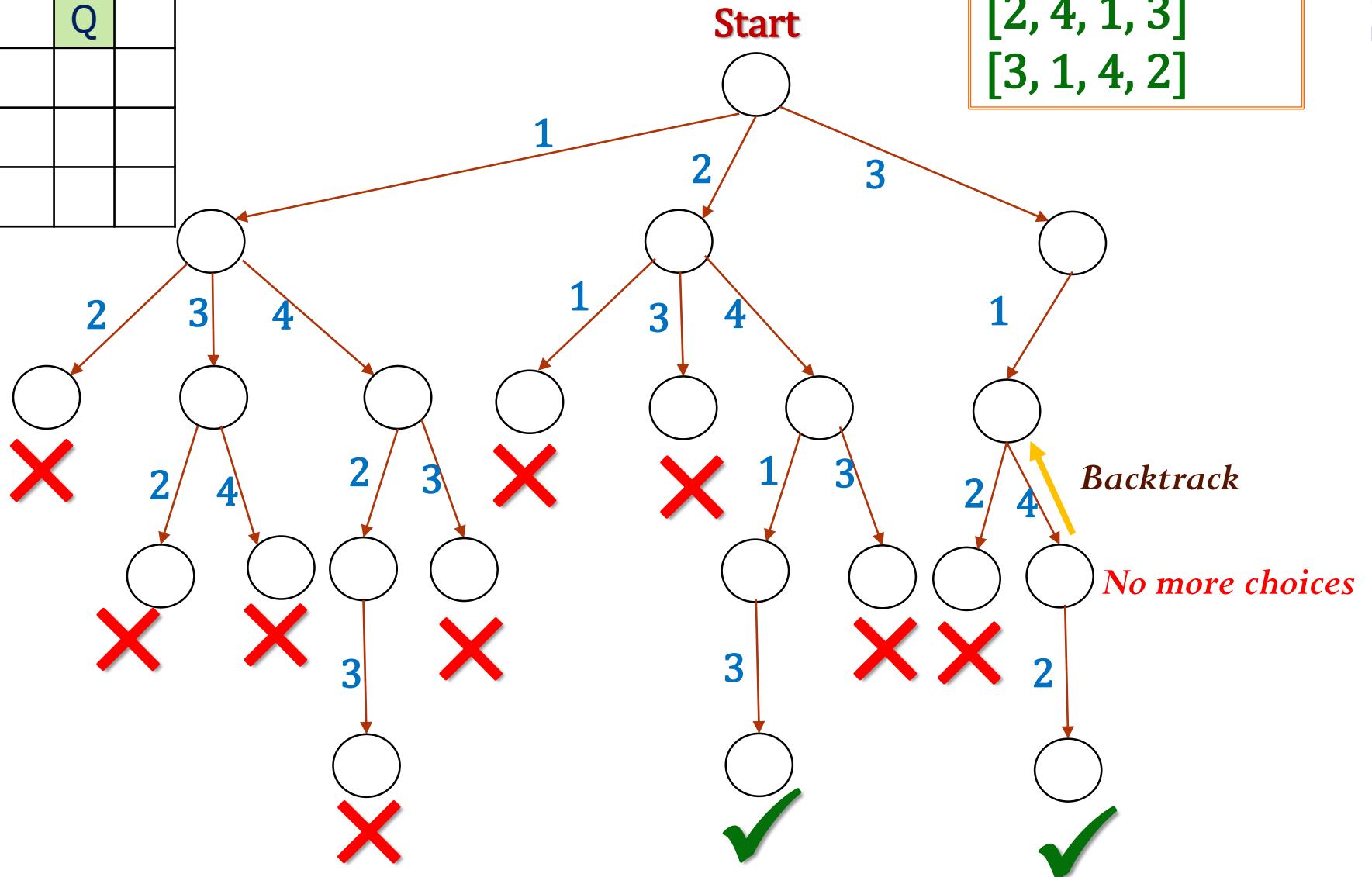


	1	2	3	4
1				Q
2	Q			
3				
4				

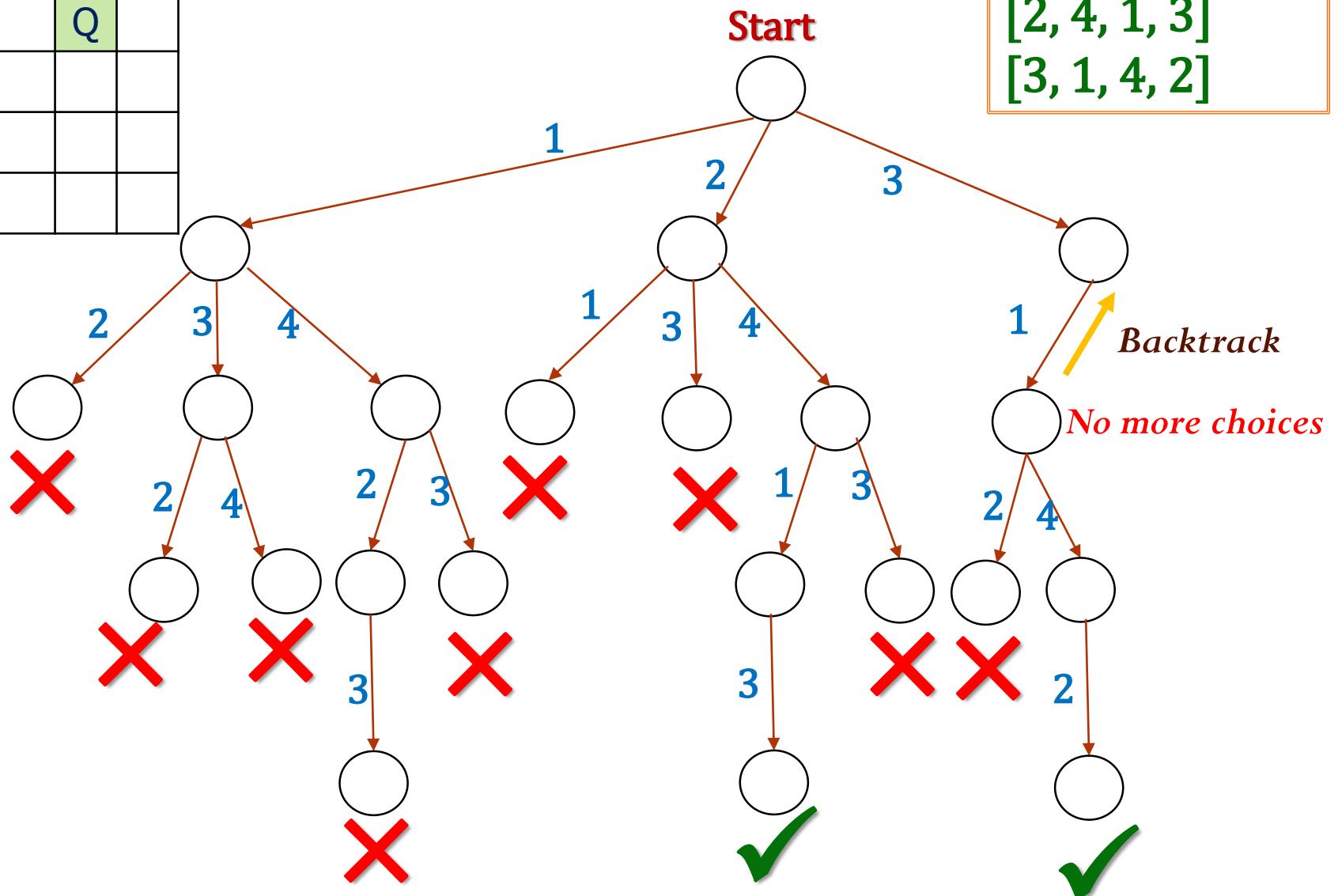
Start

Solutions

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



	1	2	3	4
1			Q	
2				
3				
4				



Solutions

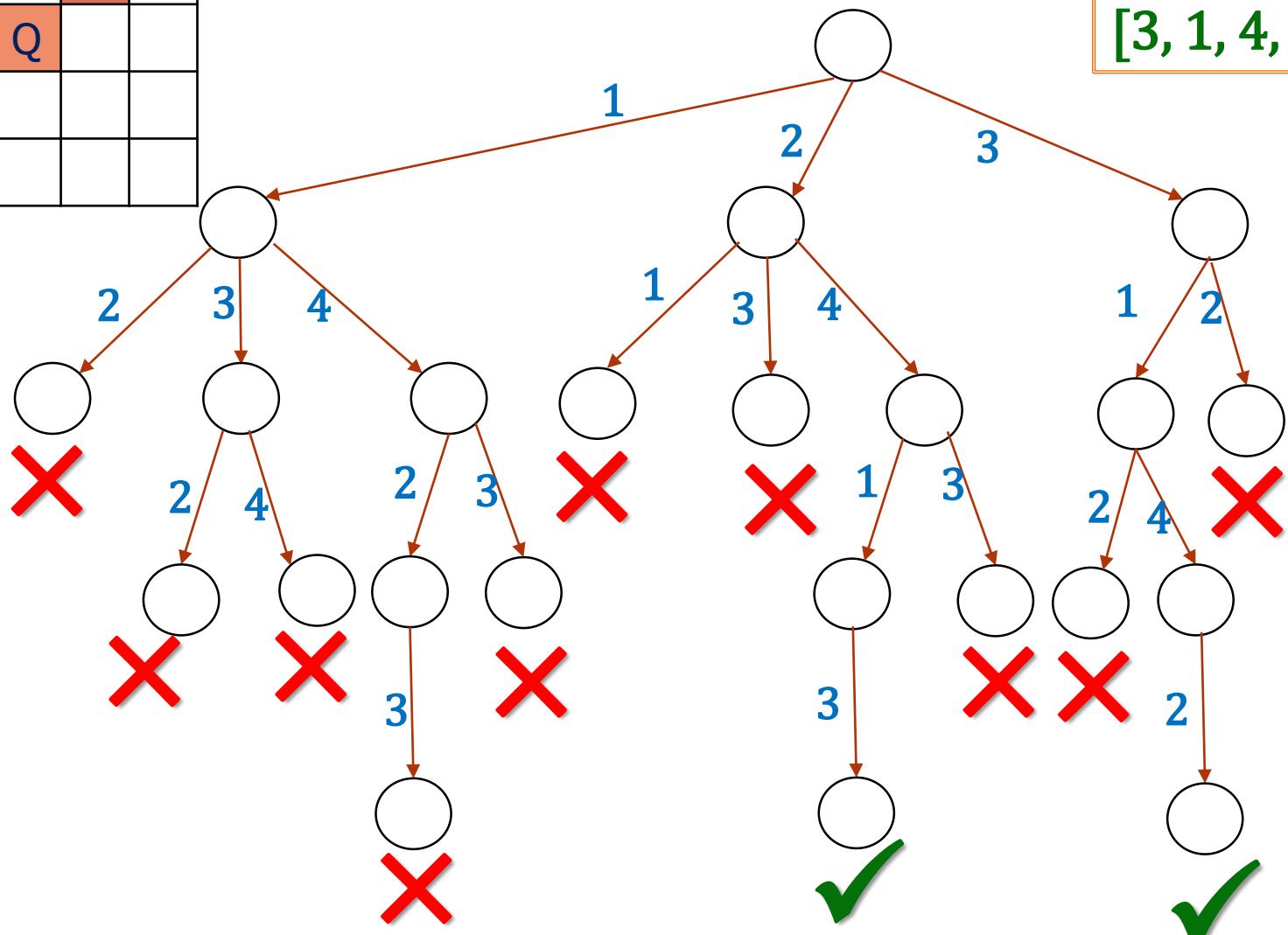
[2, 4, 1, 3]
[3, 1, 4, 2]

	1	2	3	4
1			Q	
2		Q		
3				
4				

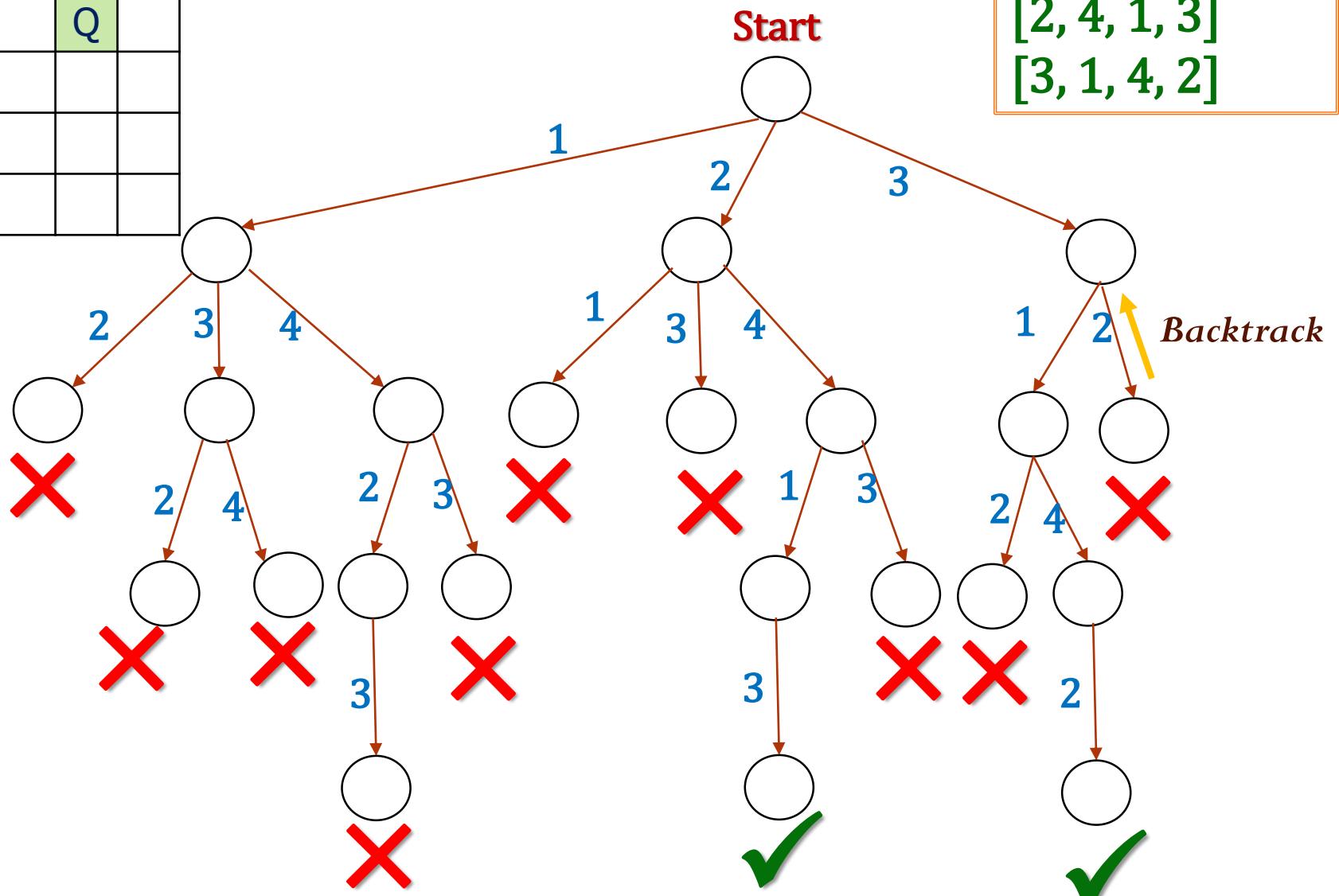
Start

Solutions

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



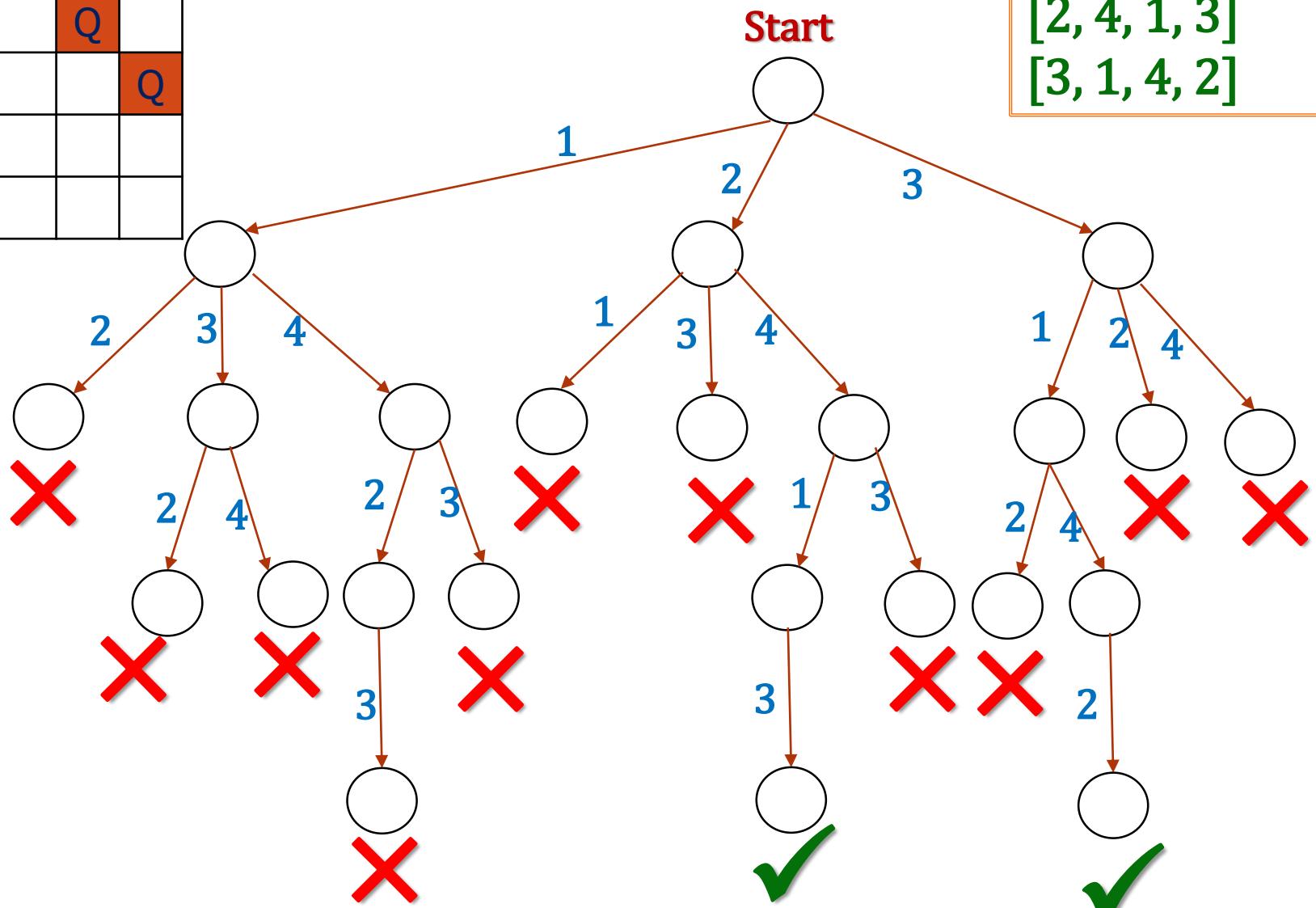
	1	2	3	4
1			Q	
2				
3				
4				



Solutions

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

	1	2	3	4
1			Q	
2				Q
3				
4				

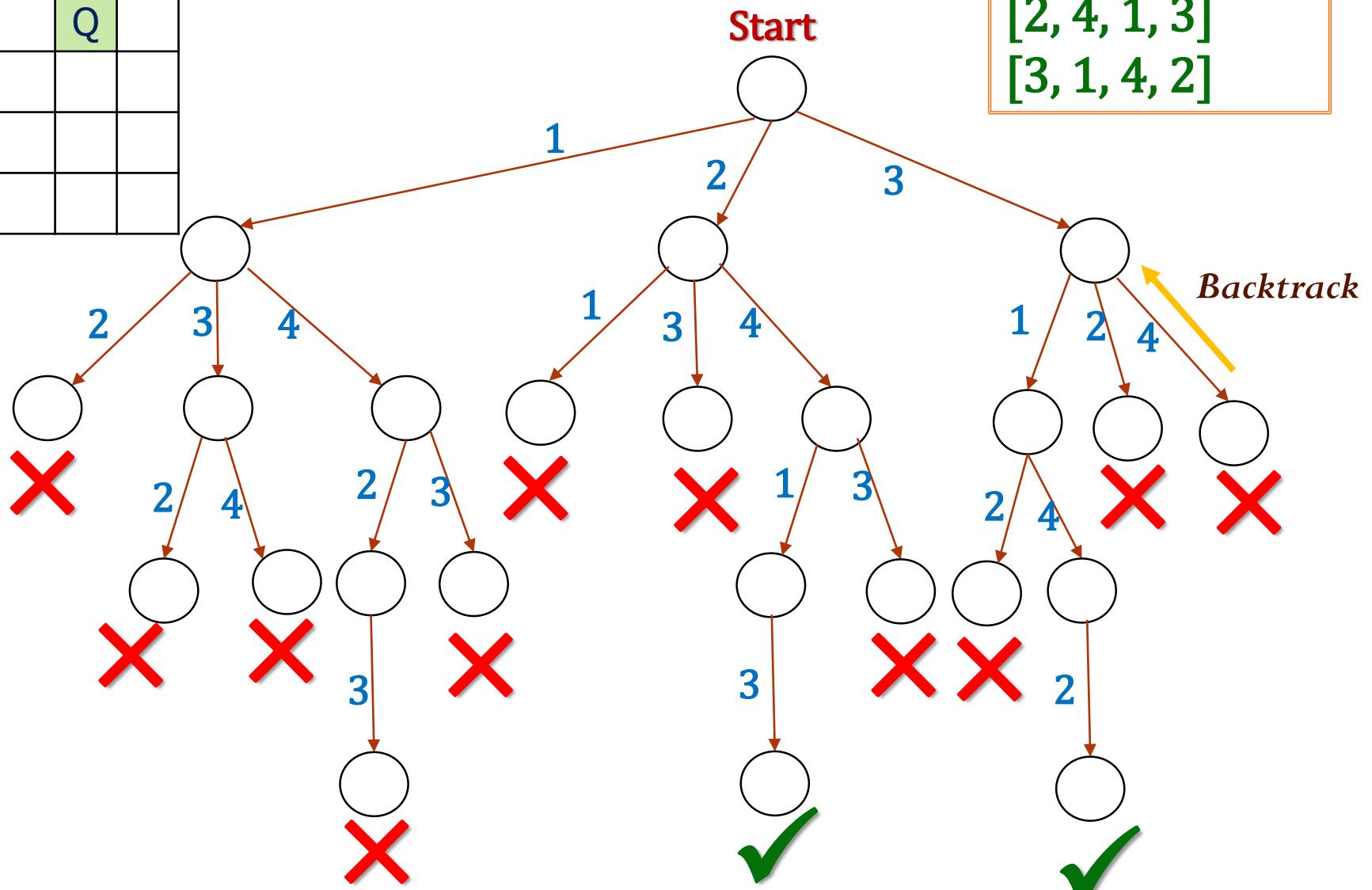


Solutions

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



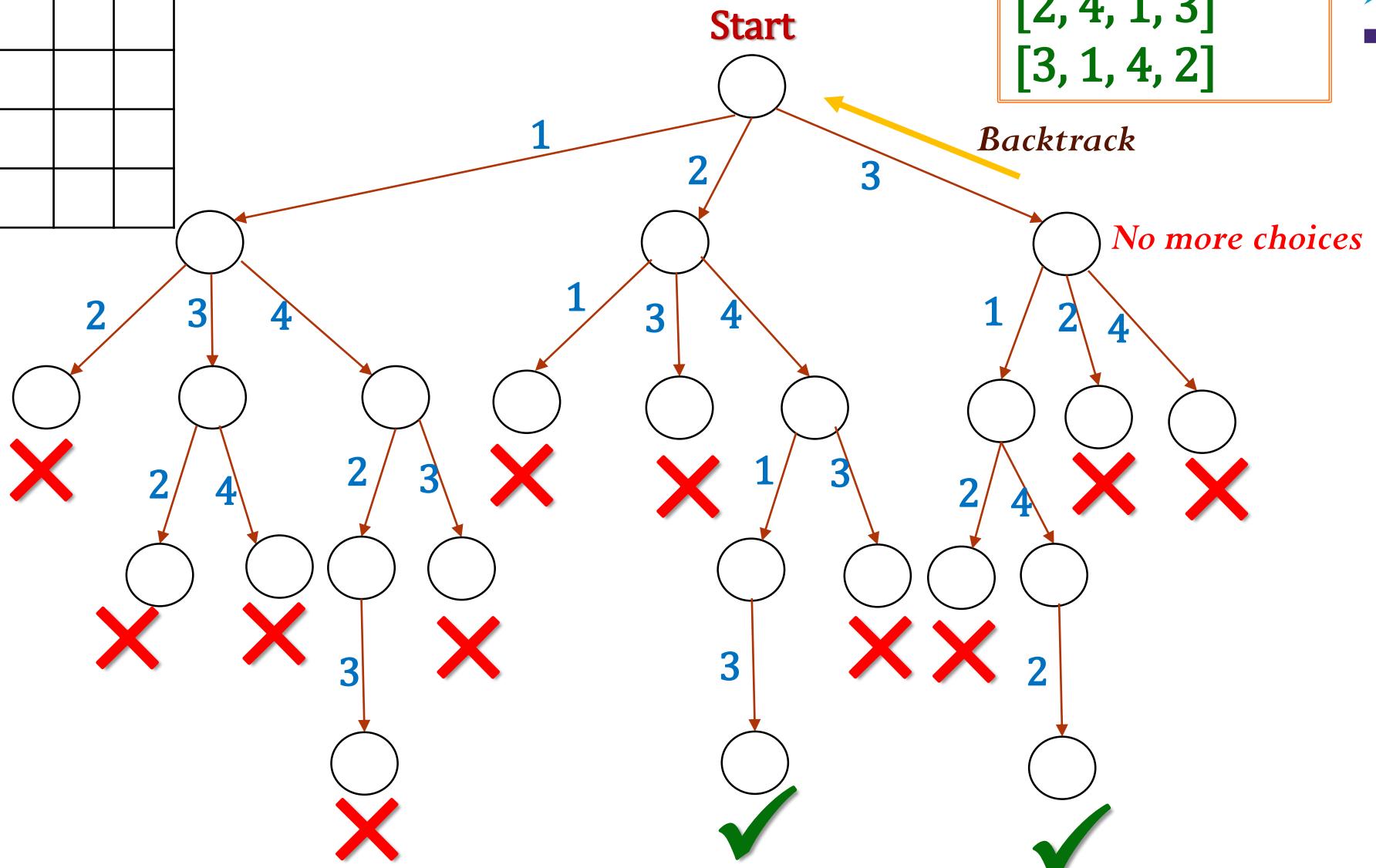
	1	2	3	4
1			Q	
2				
3				
4				



Solutions

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

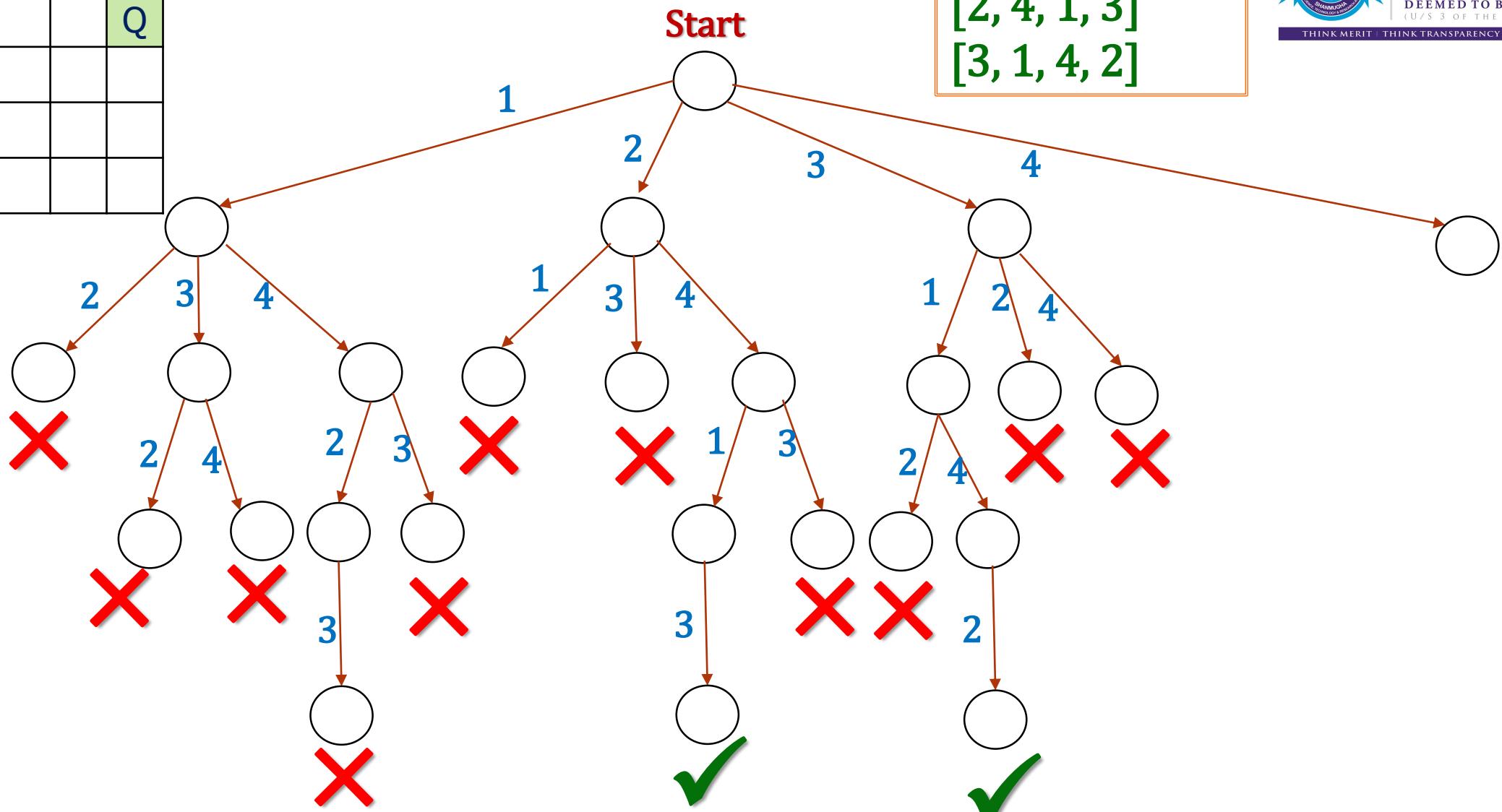
	1	2	3	4
1				
2				
3				
4				



Solutions

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

	1	2	3	4
1				Q
2				
3				
4				

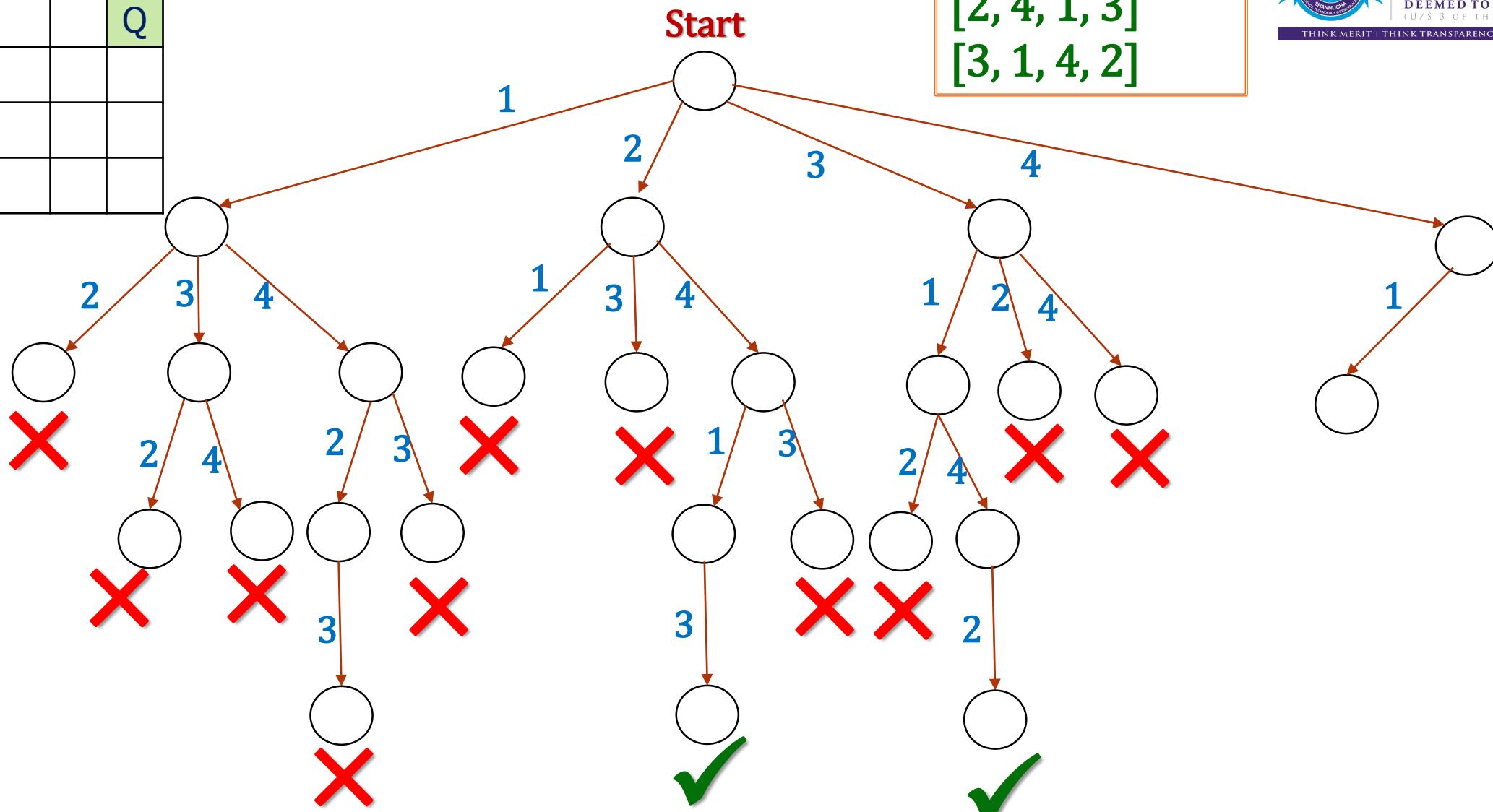


Solutions

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



	1	2	3	4
1				
2	Q			
3				
4				

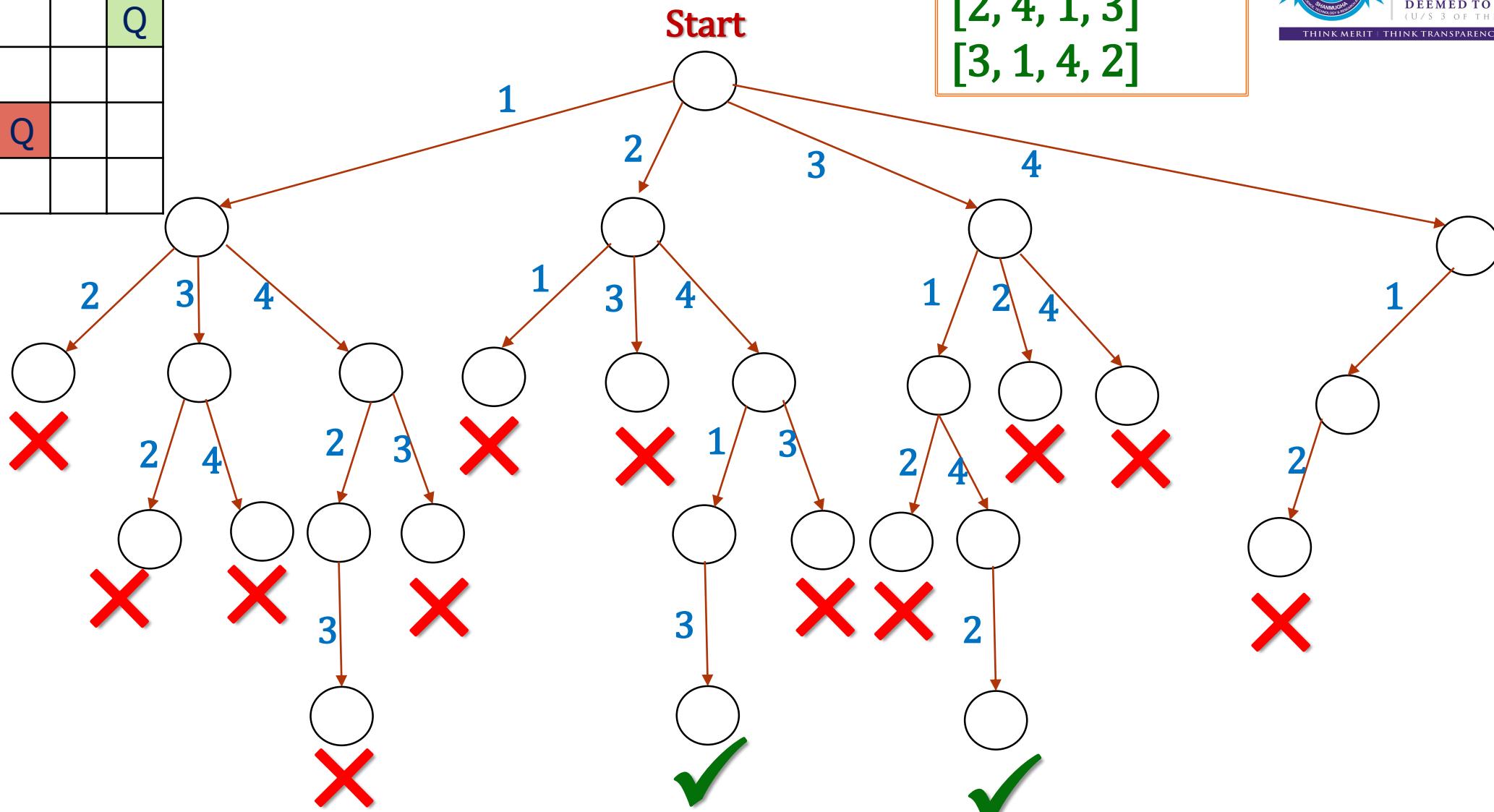


Solutions

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



	1	2	3	4
1				Q
2	Q			
3		Q		
4				

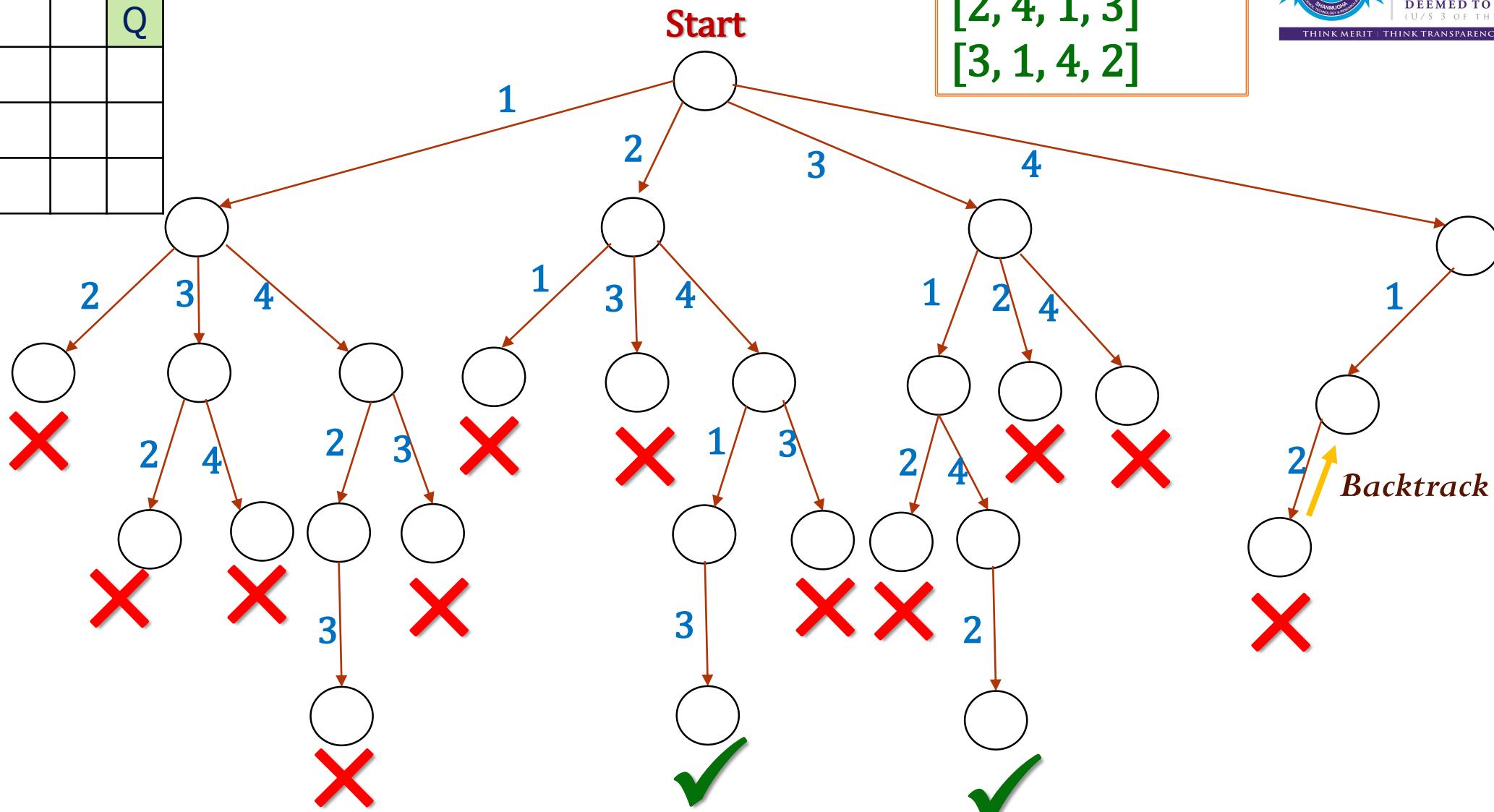


Solutions

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



	1	2	3	4
1				
2	Q			
3				
4				

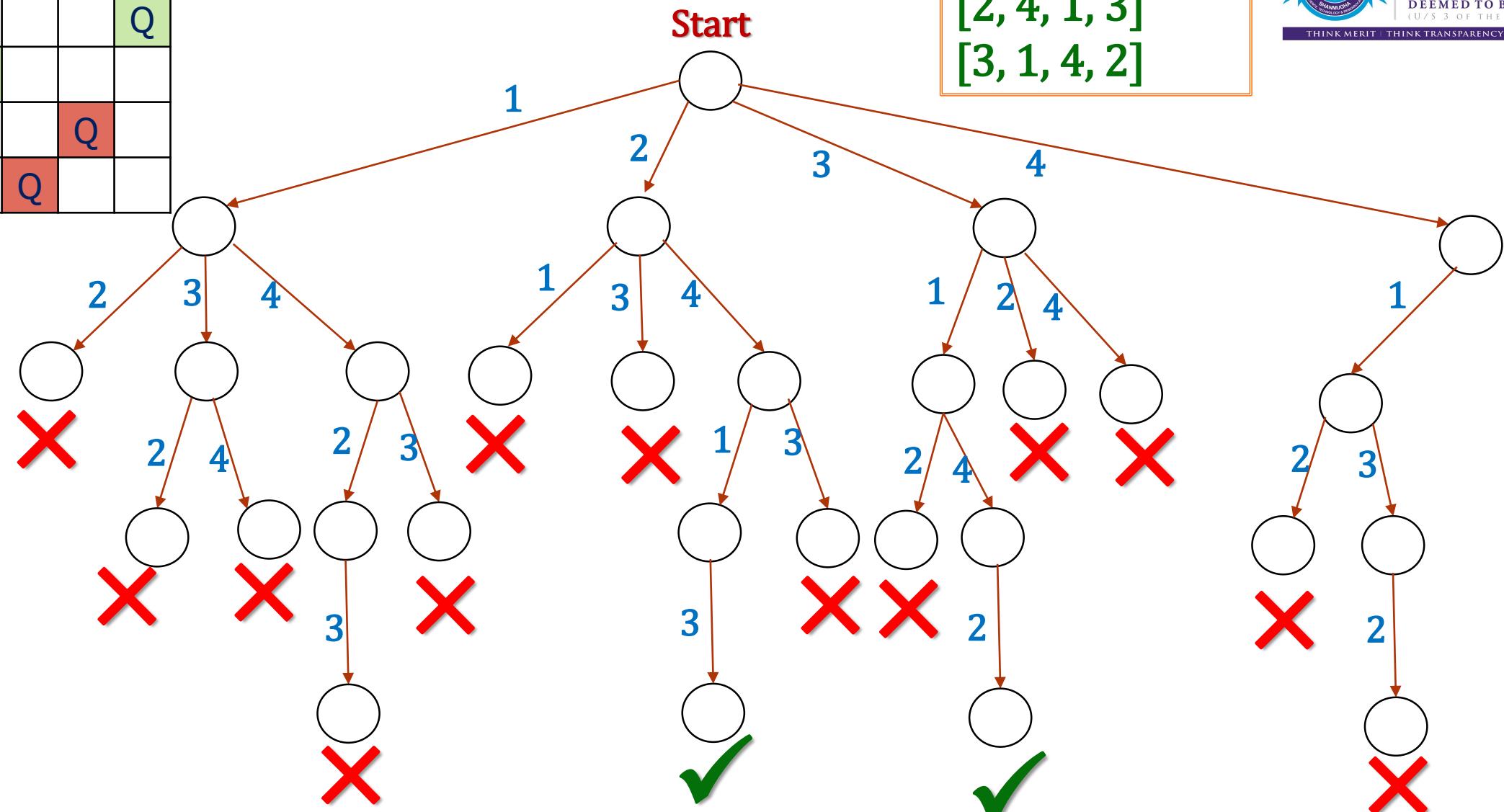


Solutions

[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]

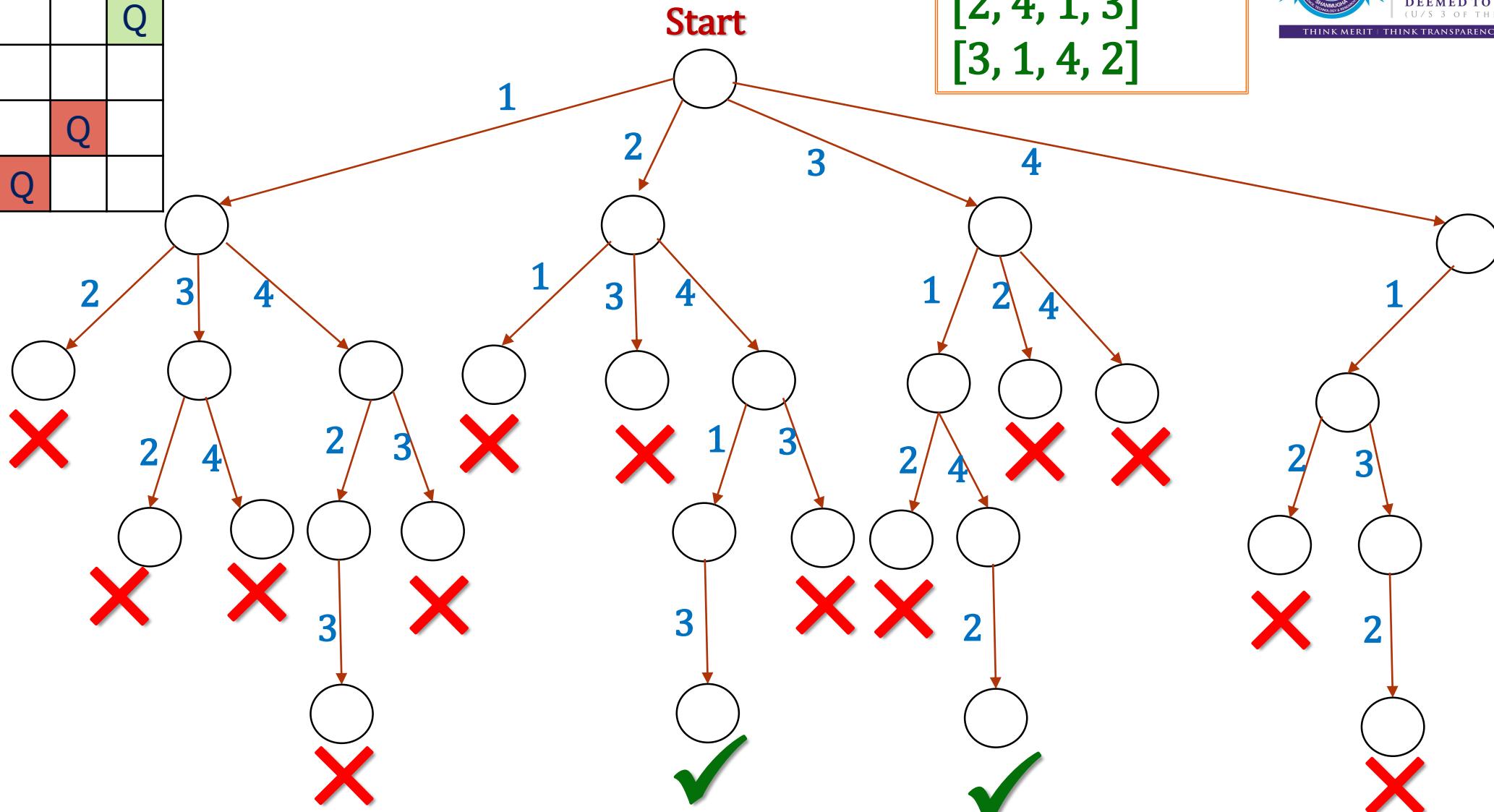


	1	2	3	4
1				Q
2	Q			
3			Q	
4		Q		

Start

Solutions

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

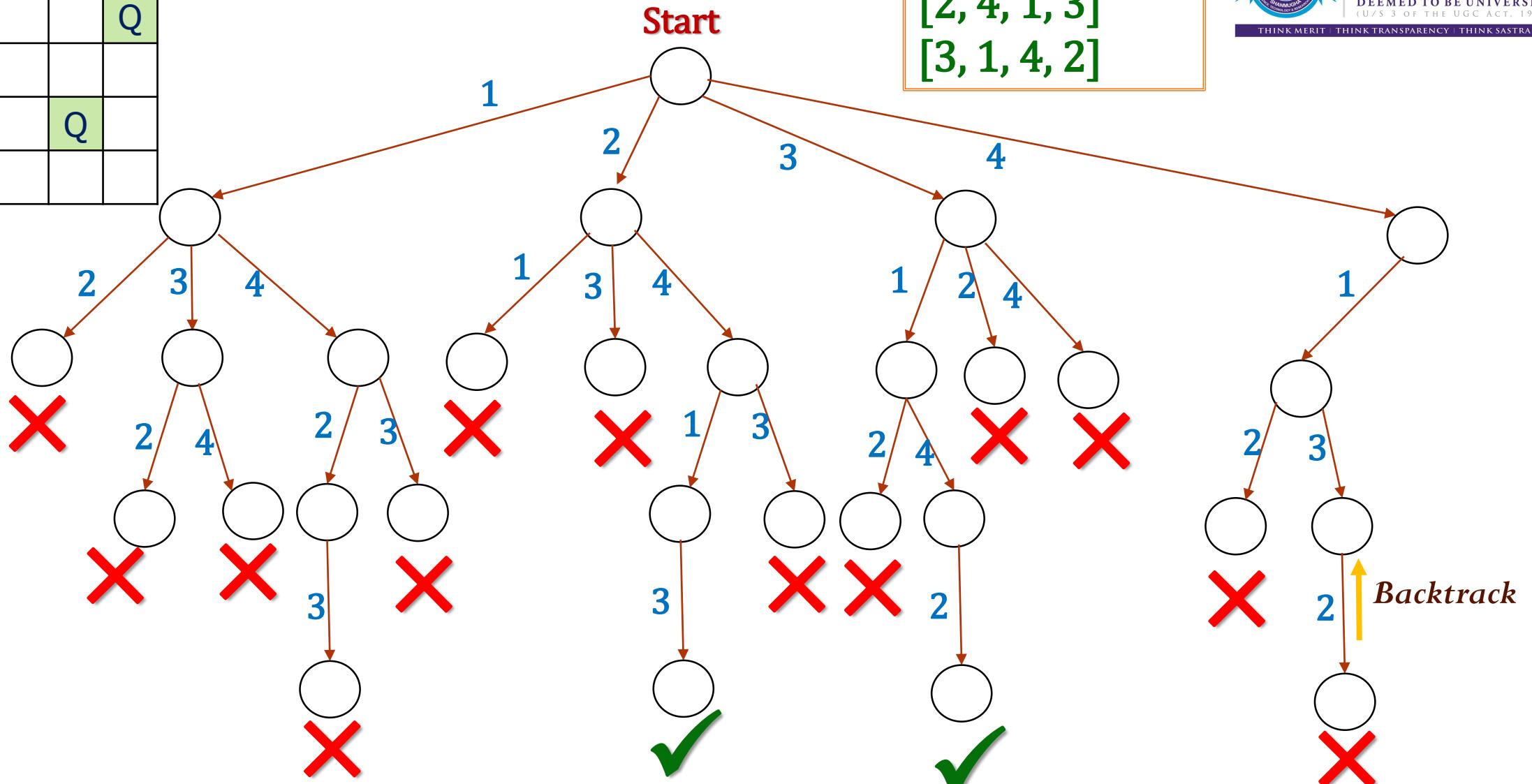


	1	2	3	4
1				Q
2	Q			
3			Q	
4				

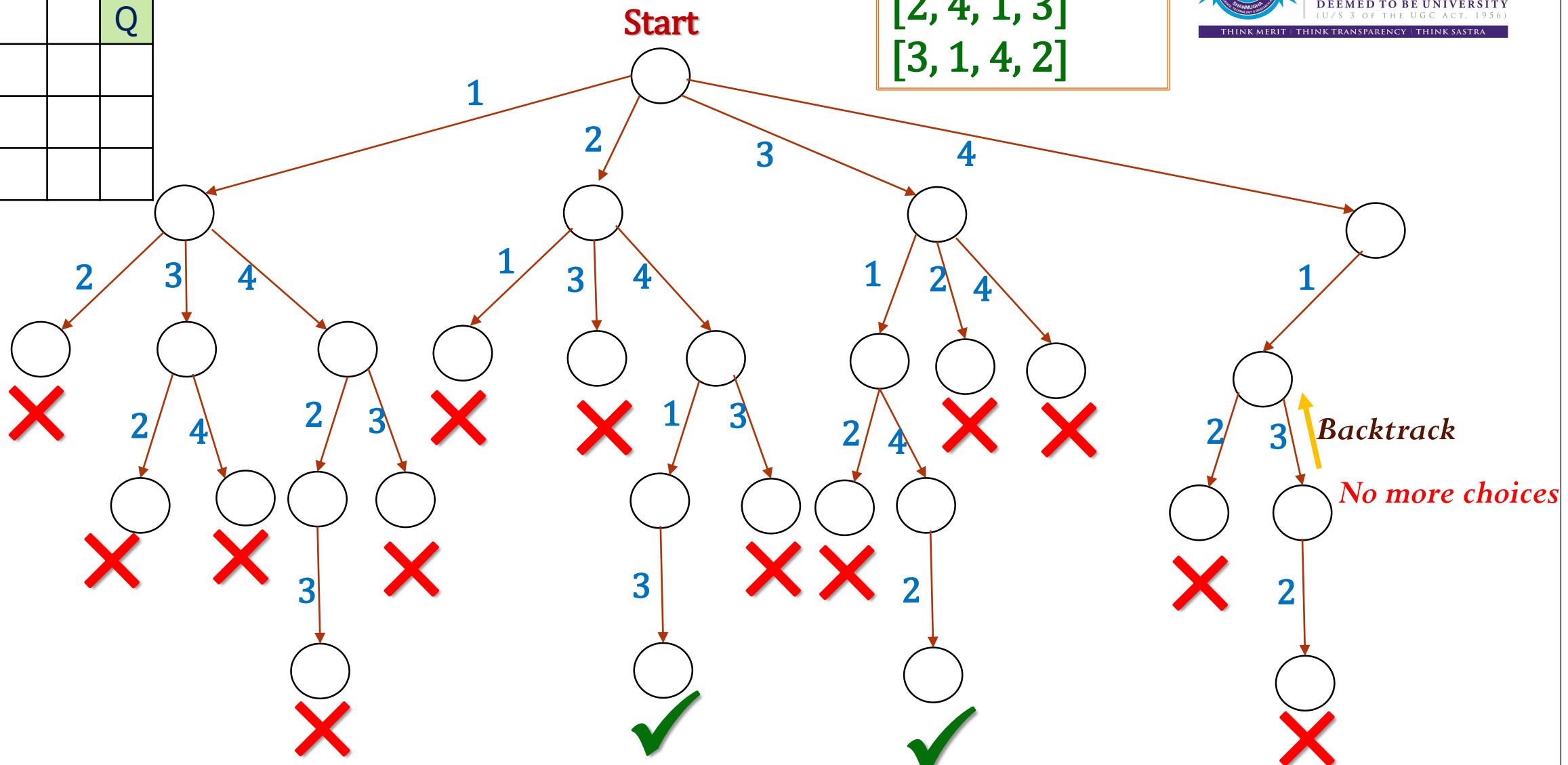
Start

Solutions

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



	1	2	3	4
1				
2	Q			
3				
4				

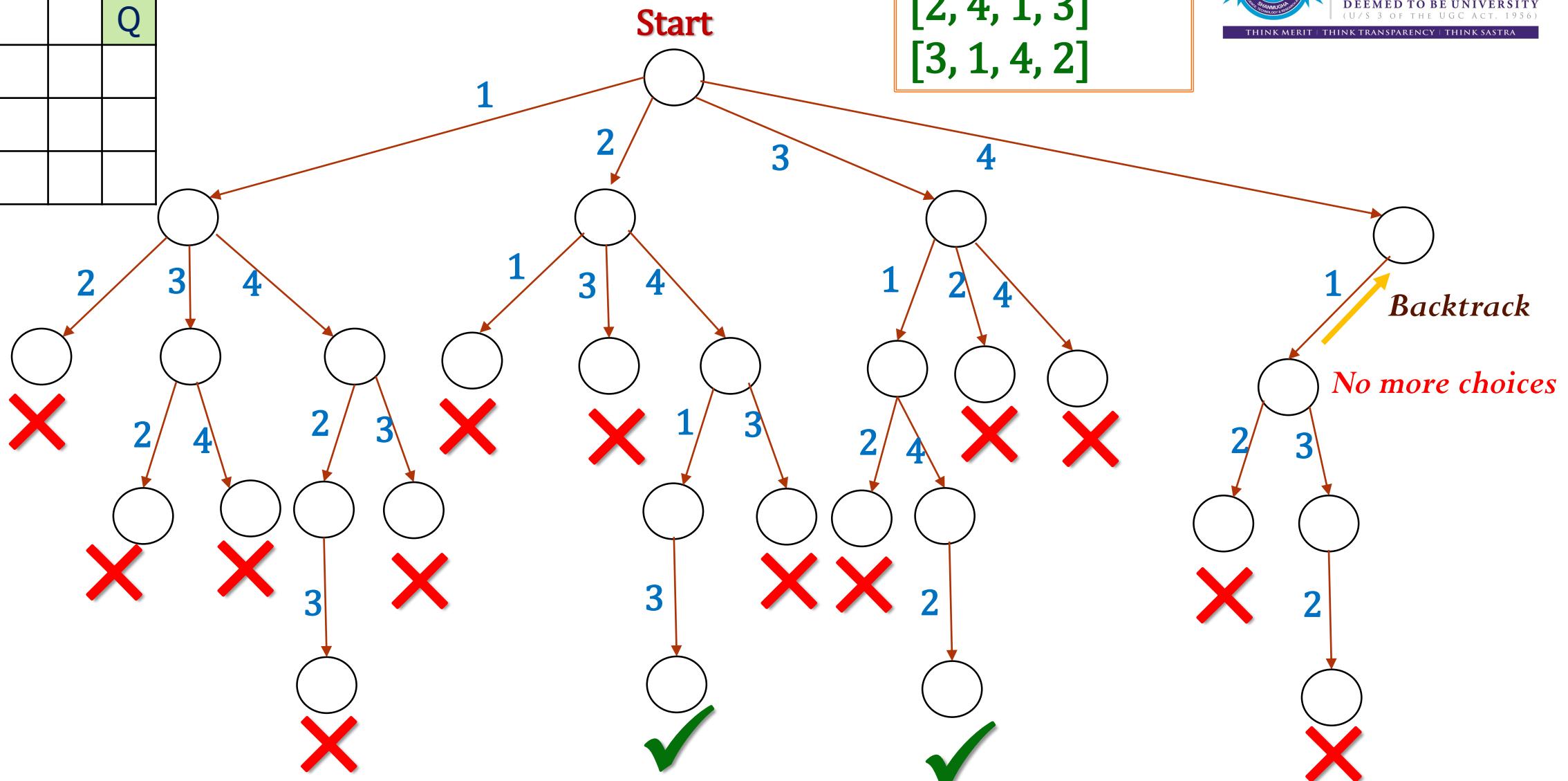


Solutions

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



	1	2	3	4
1				Q
2				
3				
4				

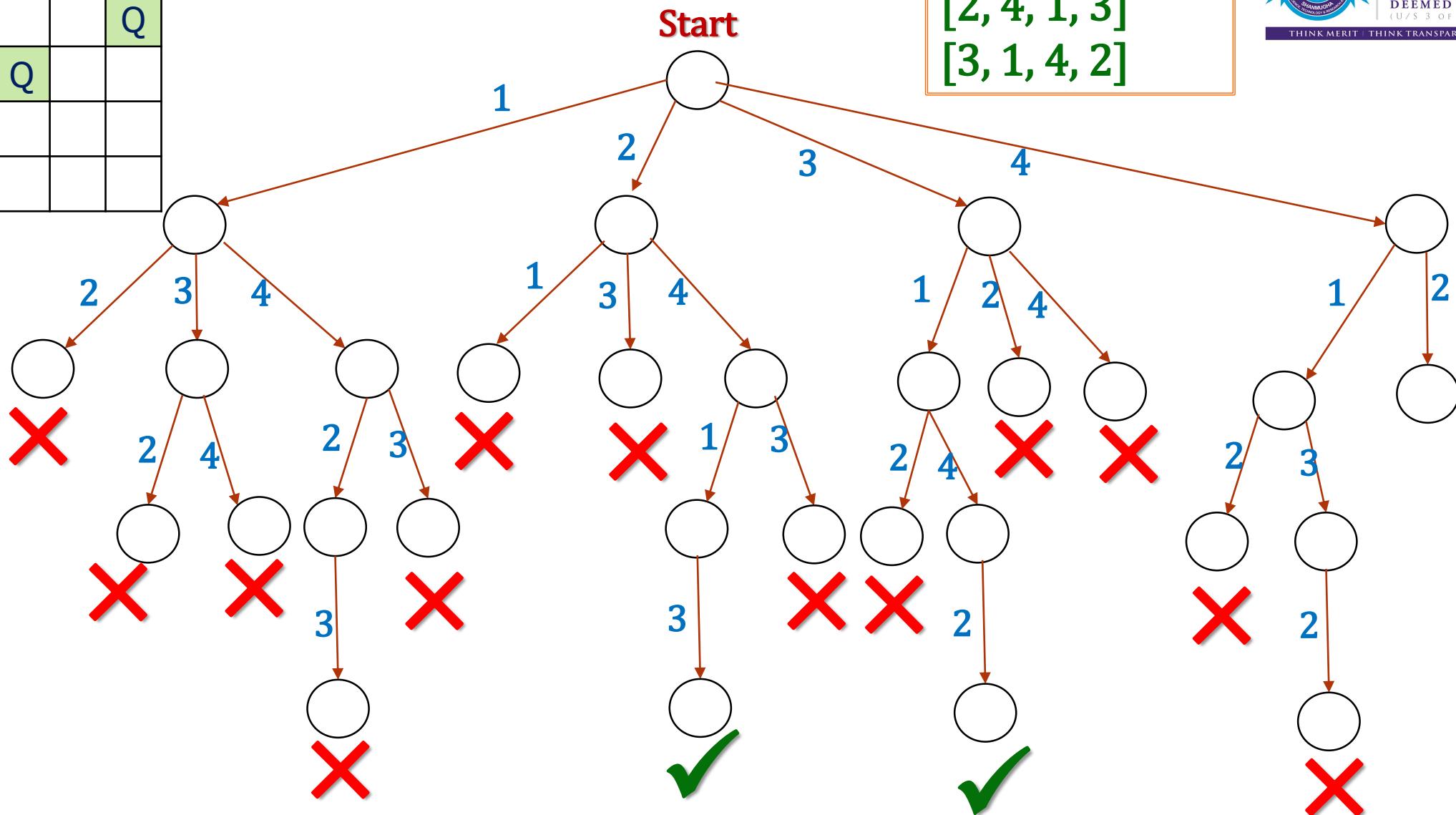


Solutions

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



	1	2	3	4
1				Q
2		Q		
3				
4				

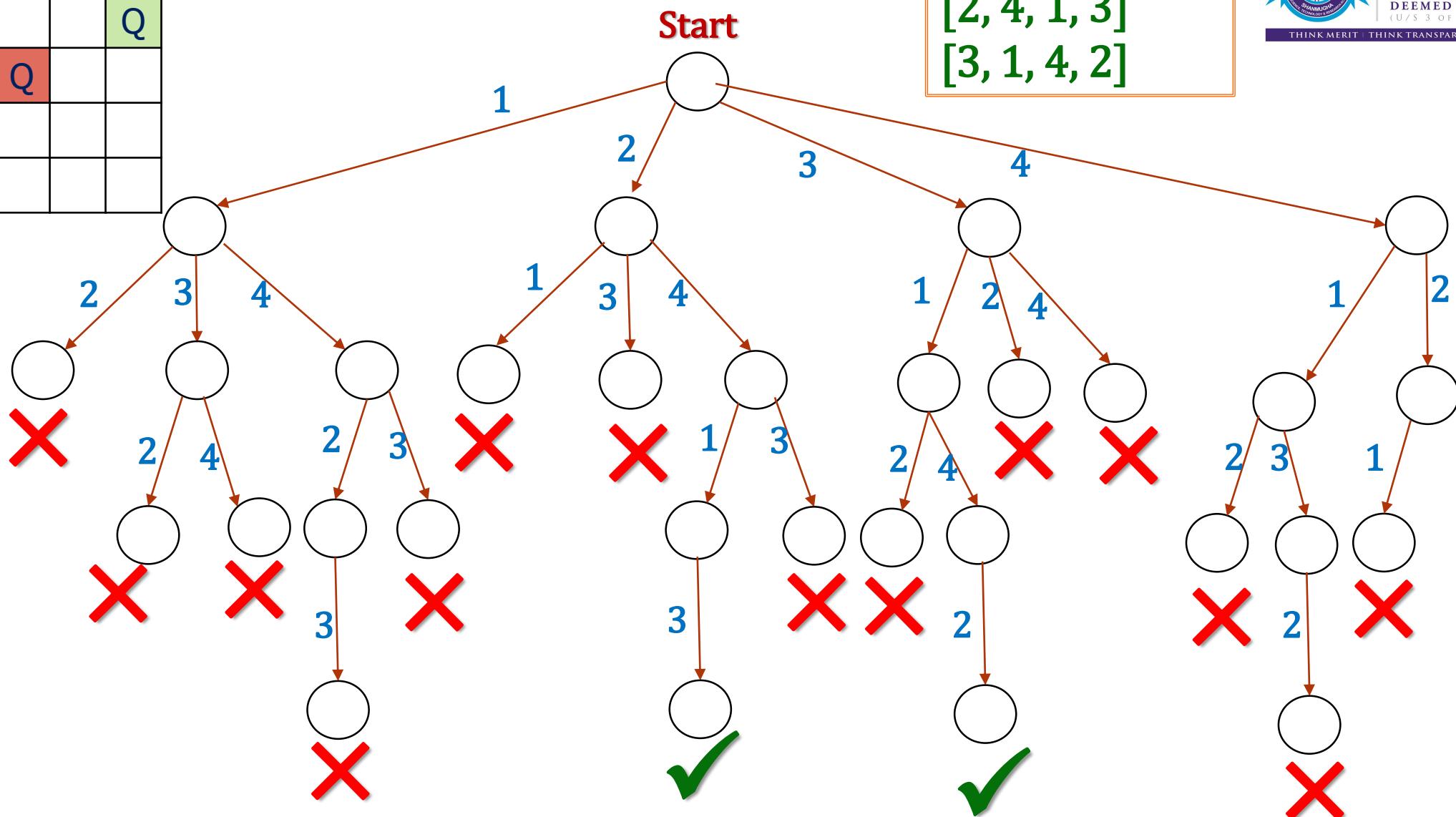


Solutions

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



	1	2	3	4
1				Q
2		Q		
3	Q			
4				

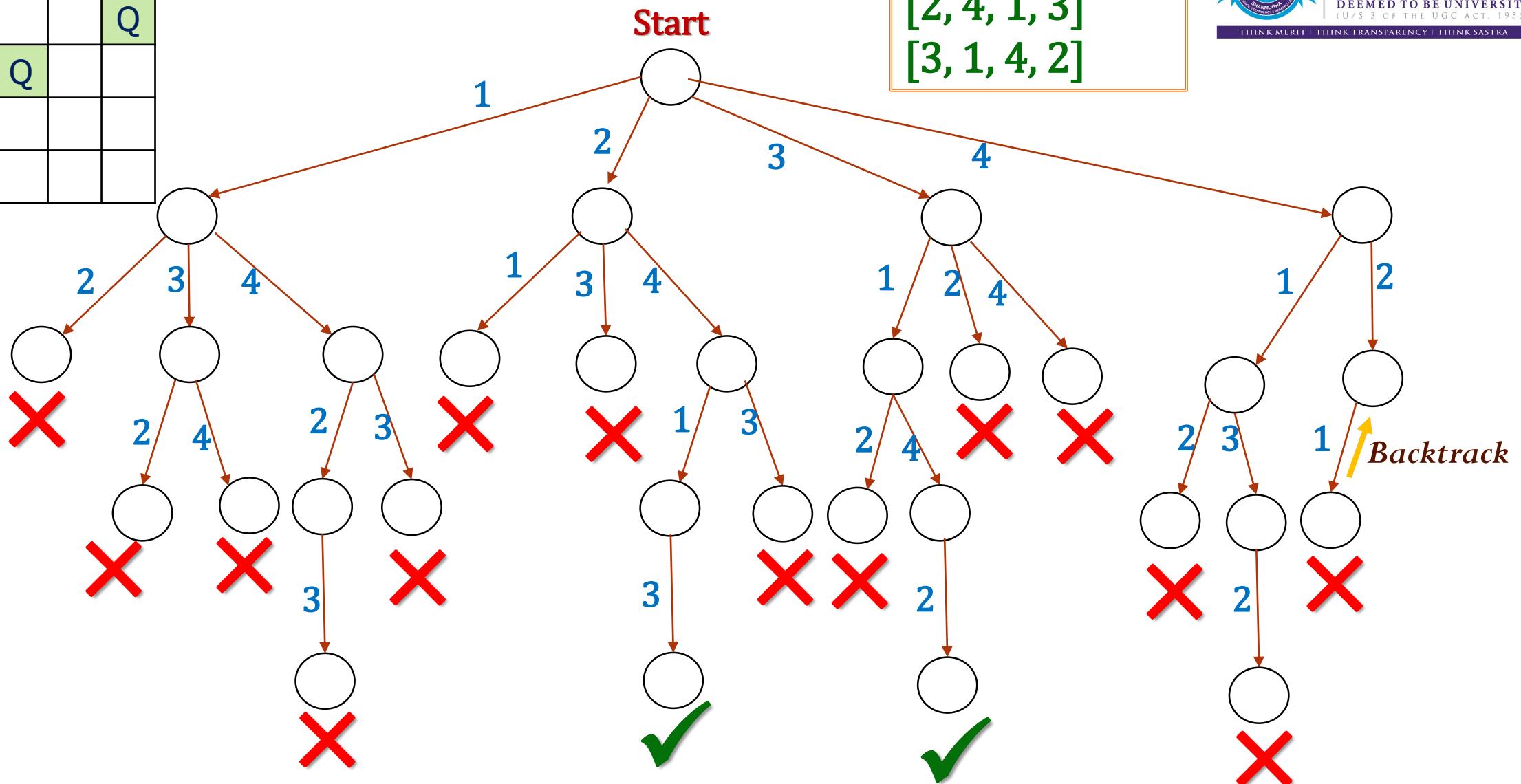


Solutions

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



	1	2	3	4
1				Q
2		Q		
3				
4				

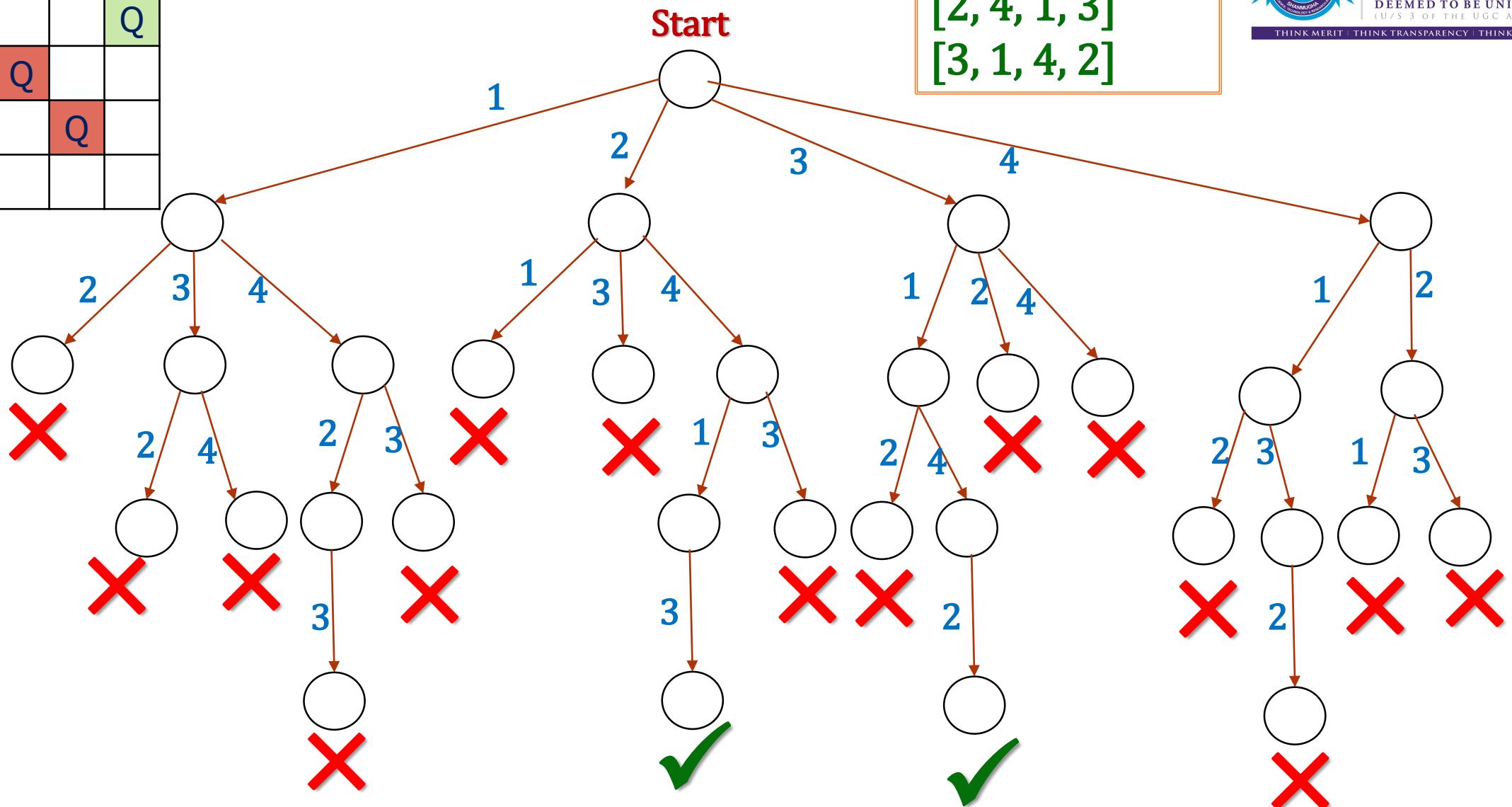


Solutions

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



	1	2	3	4
1				Q
2		Q		
3			Q	
4				

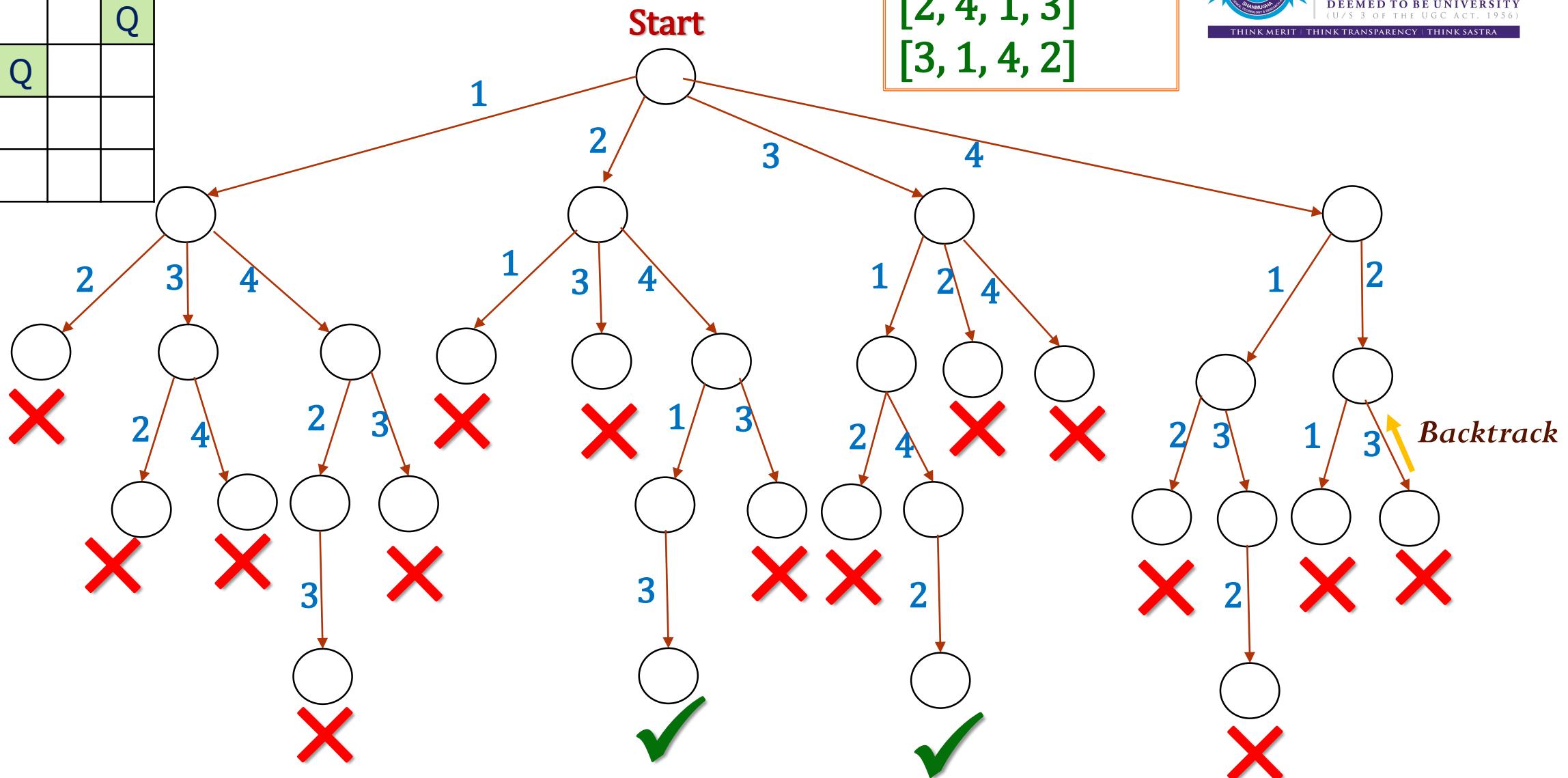


Solutions

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



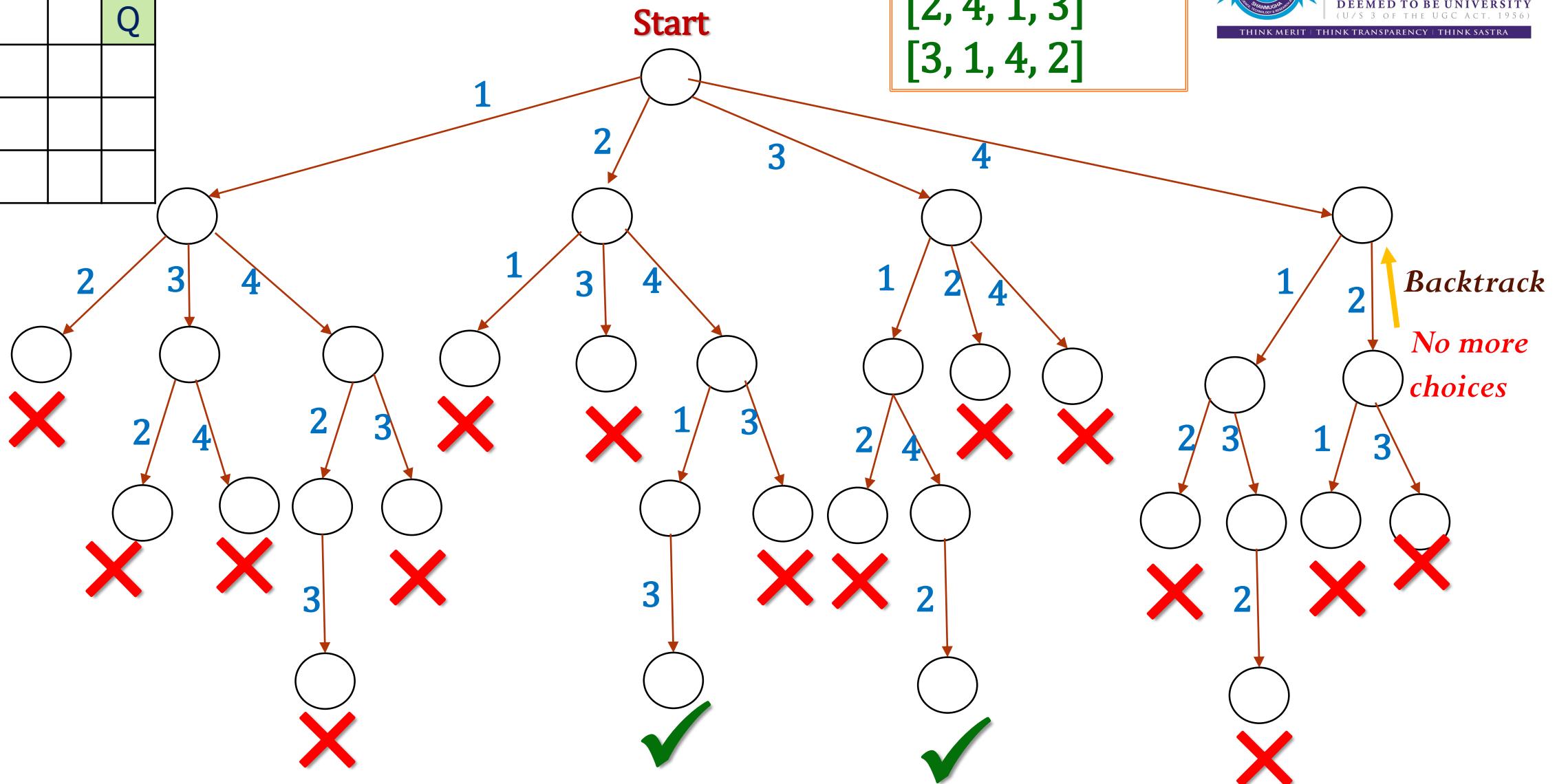
	1	2	3	4
1				Q
2		Q		
3				
4				



Solutions

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

	1	2	3	4
1				Q
2				
3				
4				

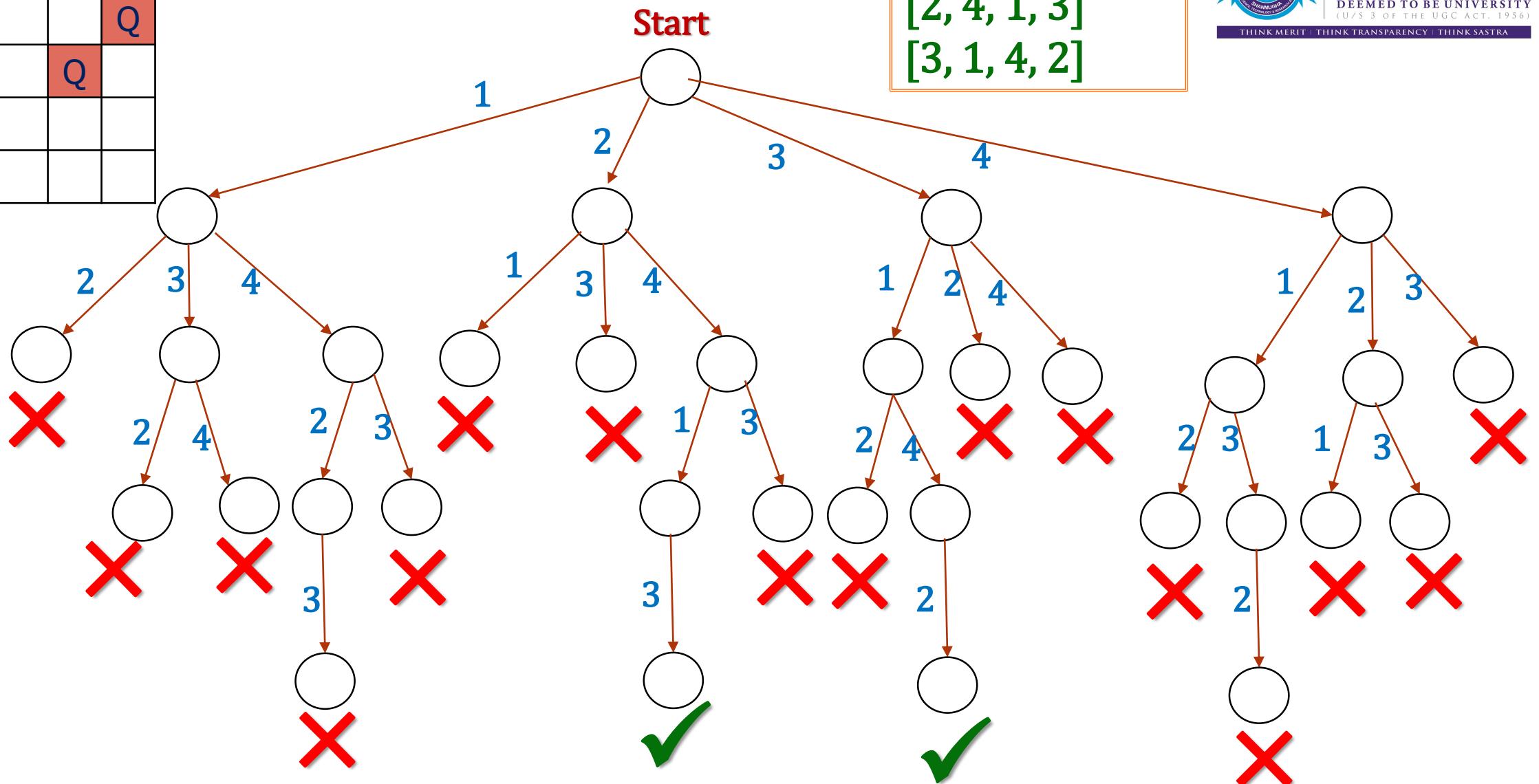


Solutions

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



	1	2	3	4
1				Q
2			Q	
3				
4				

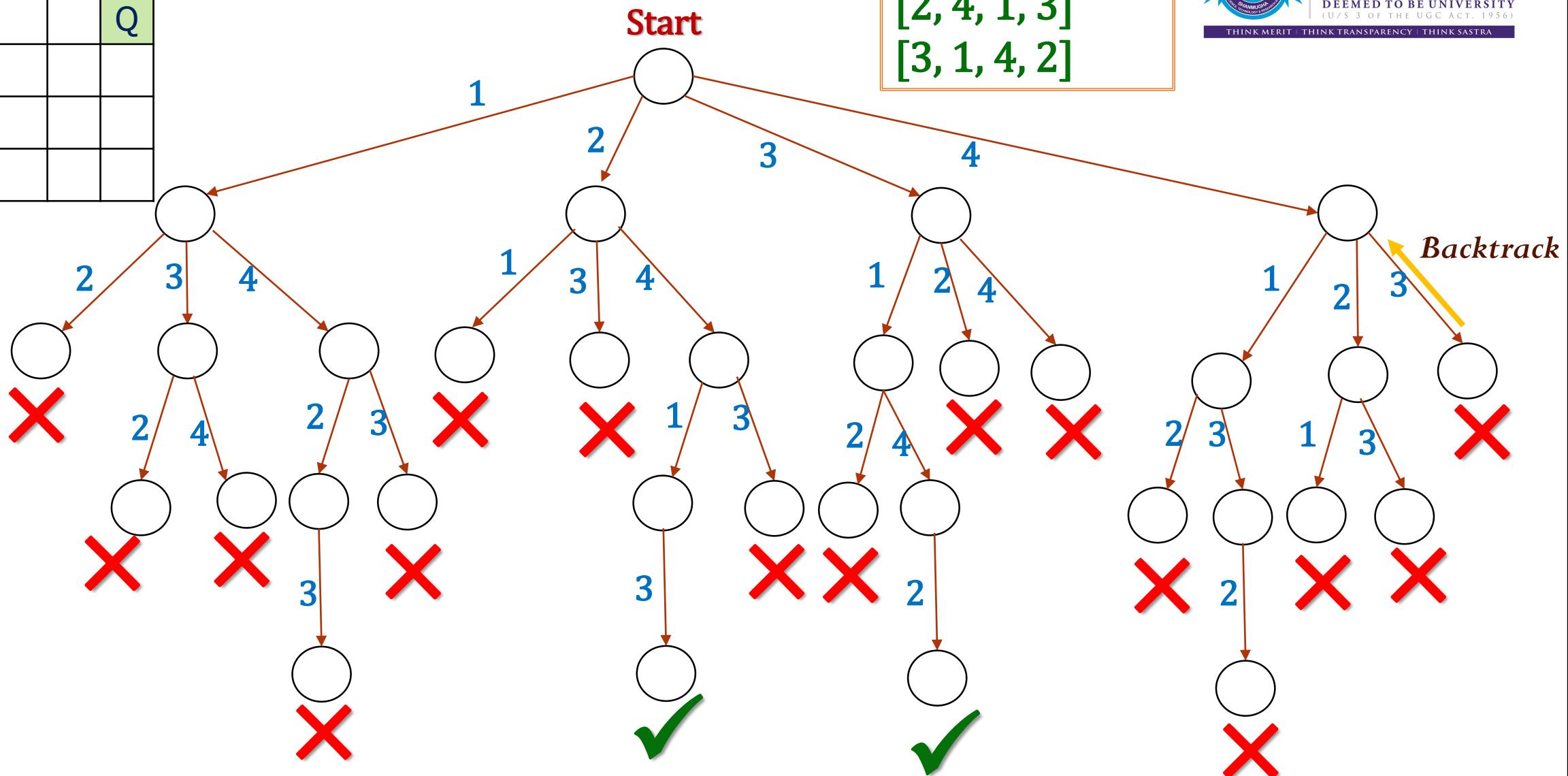


Solutions

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



	1	2	3	4
1				Q
2				
3				
4				

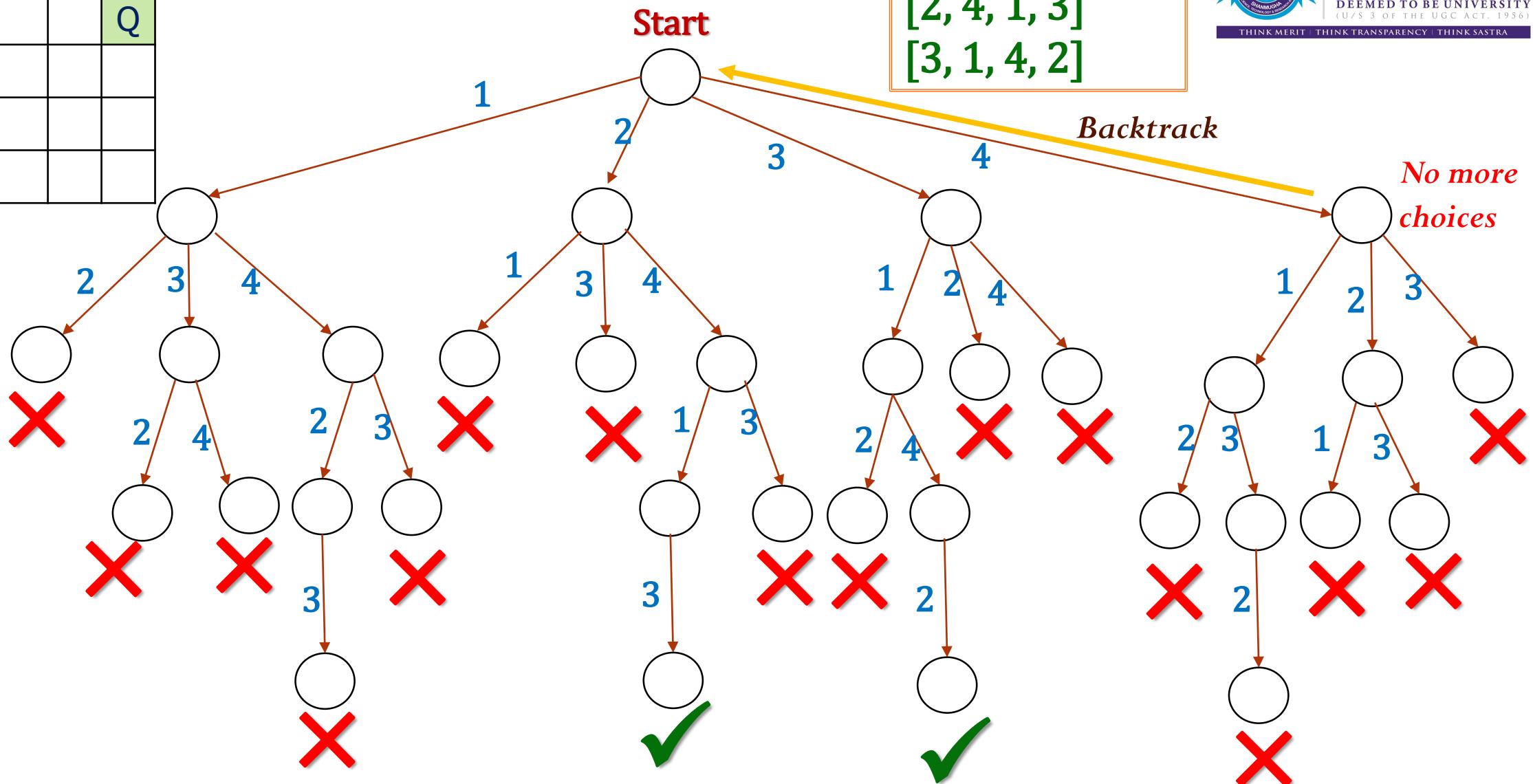


Solutions

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



	1	2	3	4
1				Q
2				
3				
4				



Solutions

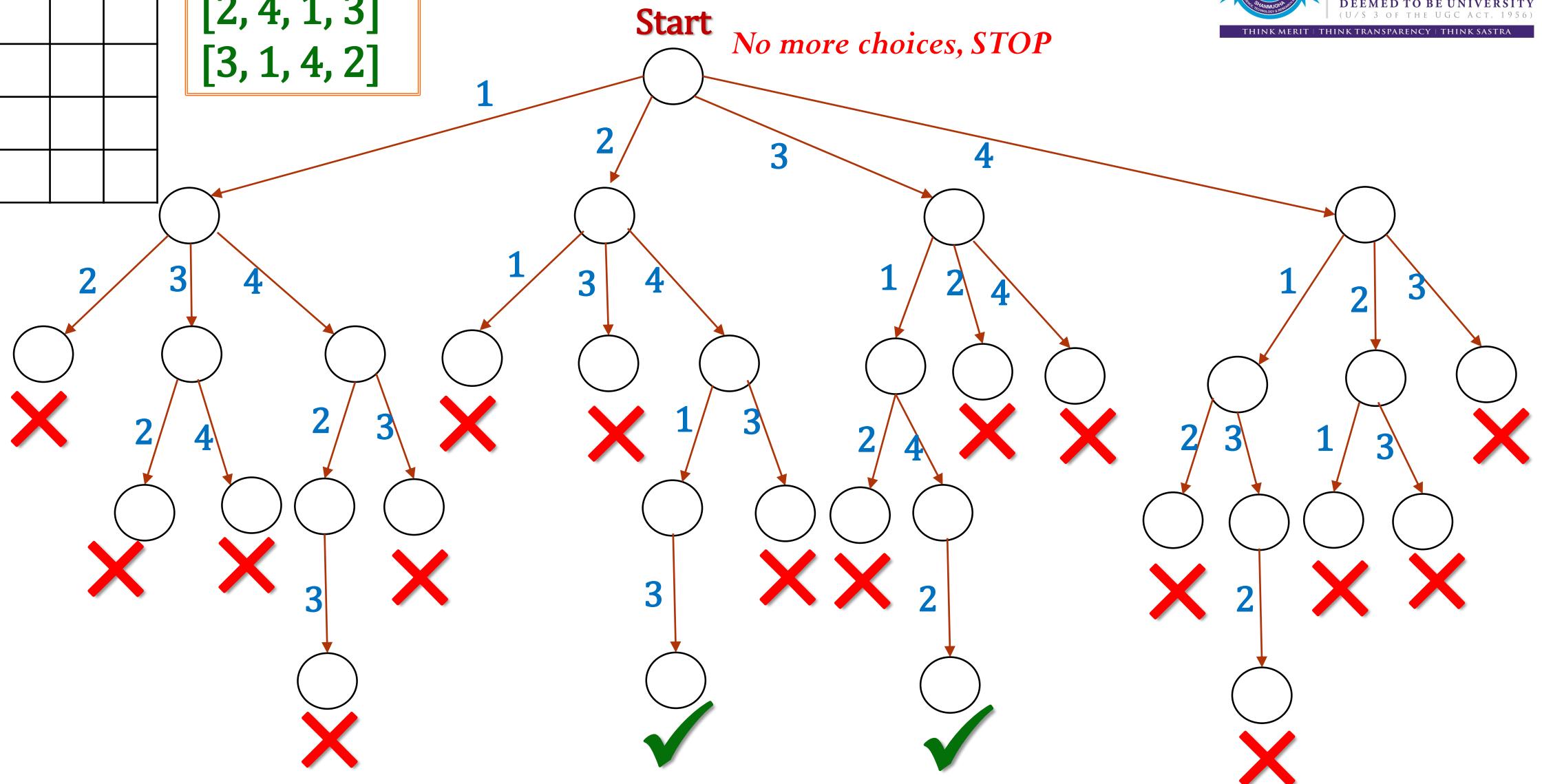
[2, 4, 1, 3]
[3, 1, 4, 2]



	1	2	3	4
1				
2				
3				
4				

Solutions

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



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 0th 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←0 to row-1 **do**

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

Return False;

End If

End For

//Check upper diagonal on left side

i ← Row

j ← Col

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

End If

i ← i – 1

j ← j – 1

End While

//Check upper diagonal on right side

i ← Row

j ← Col

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

End If

i ← i – 1

j ← 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

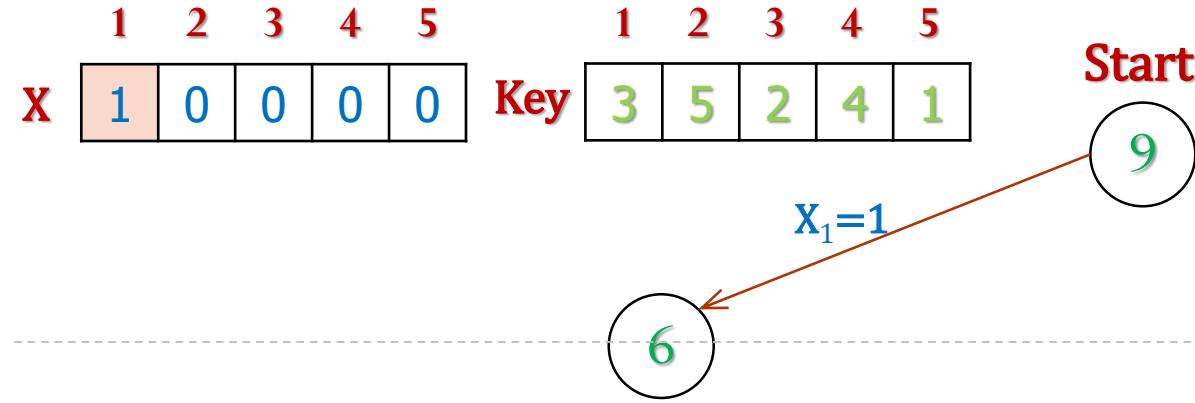
3	5	2	4	1
---	---	---	---	---

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

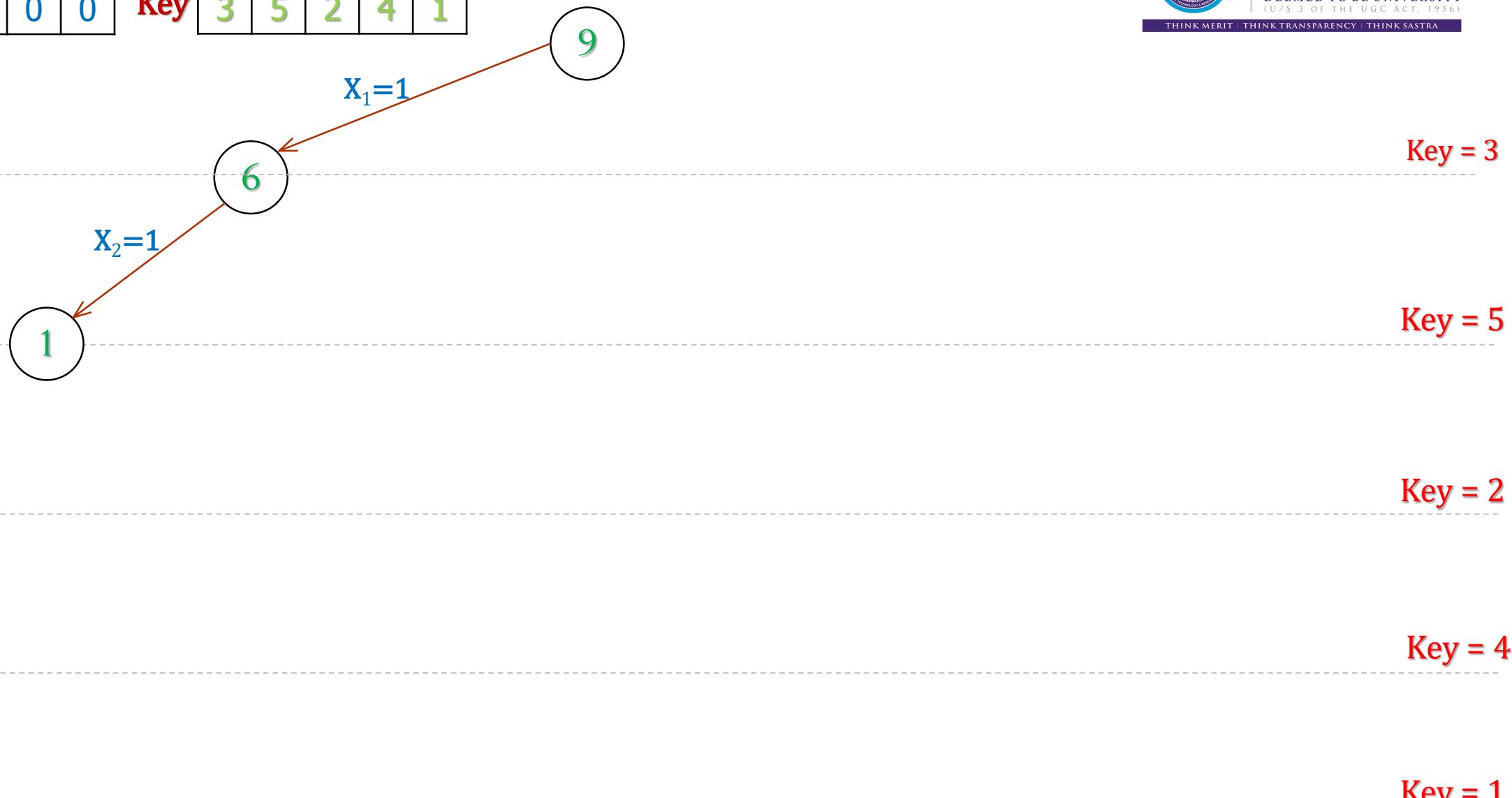


	1	2	3	4	5
X	1	1	0	0	0

Key

	1	2	3	4	5
Key	3	5	2	4	1

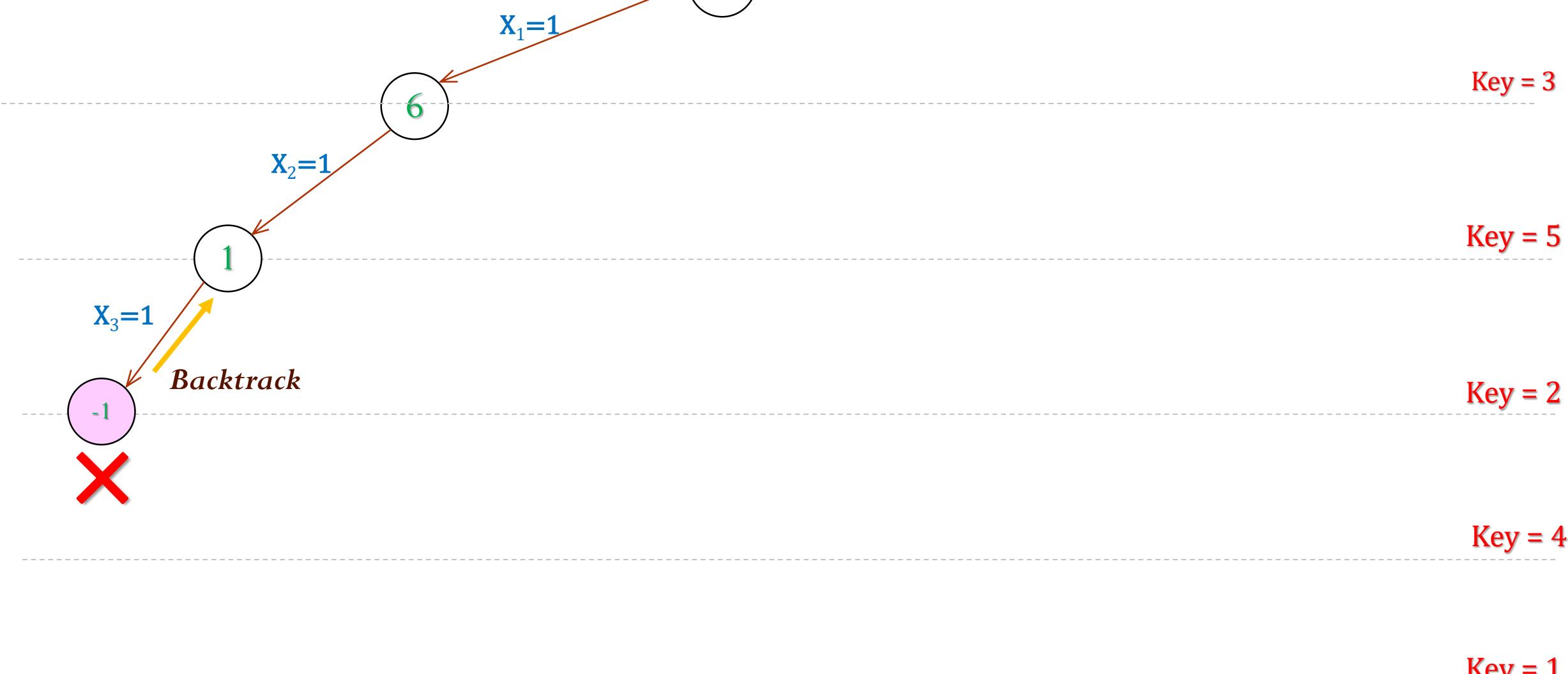
Start



	1	2	3	4	5
X	1	1	1	0	0

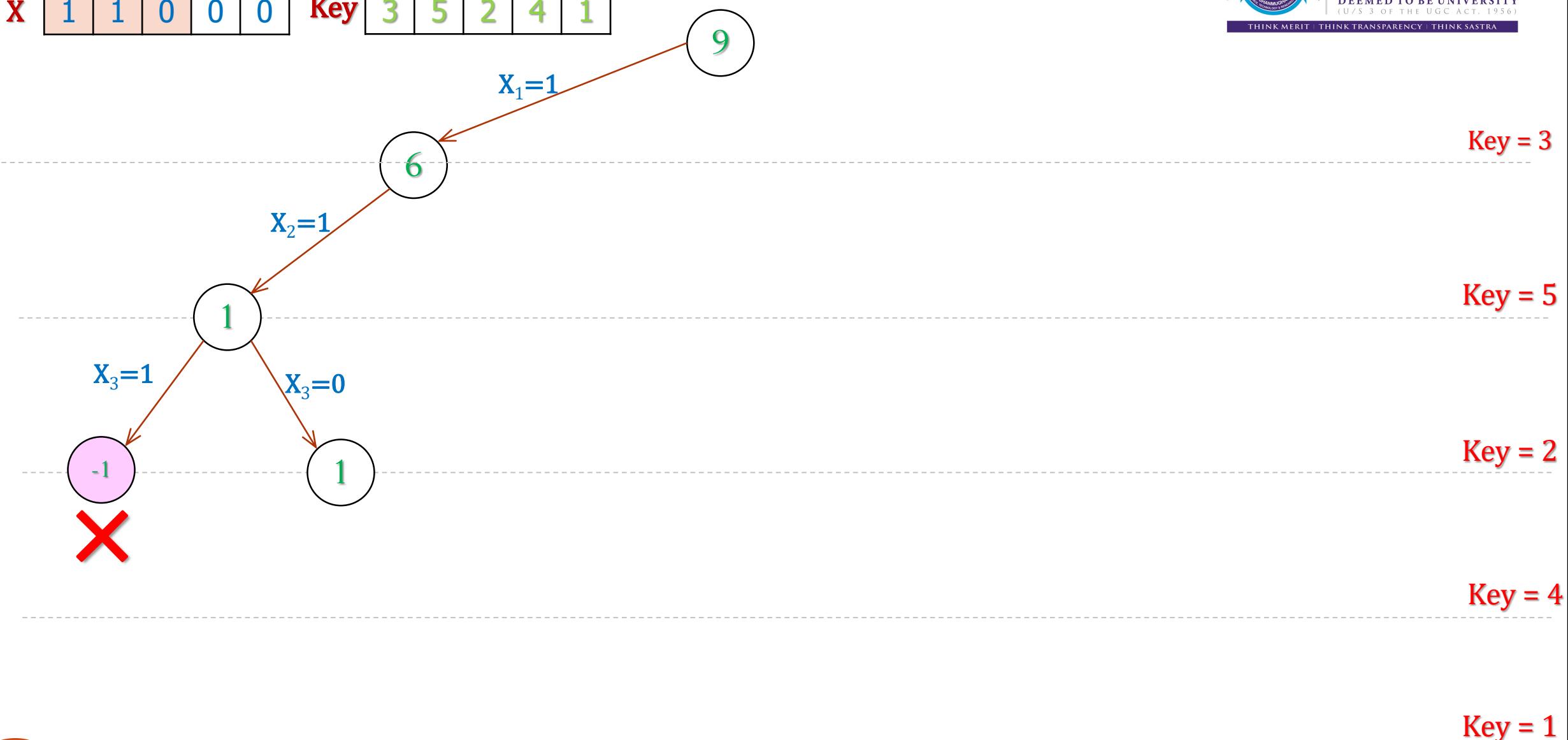
Key [3 5 2 4 1]

Start



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

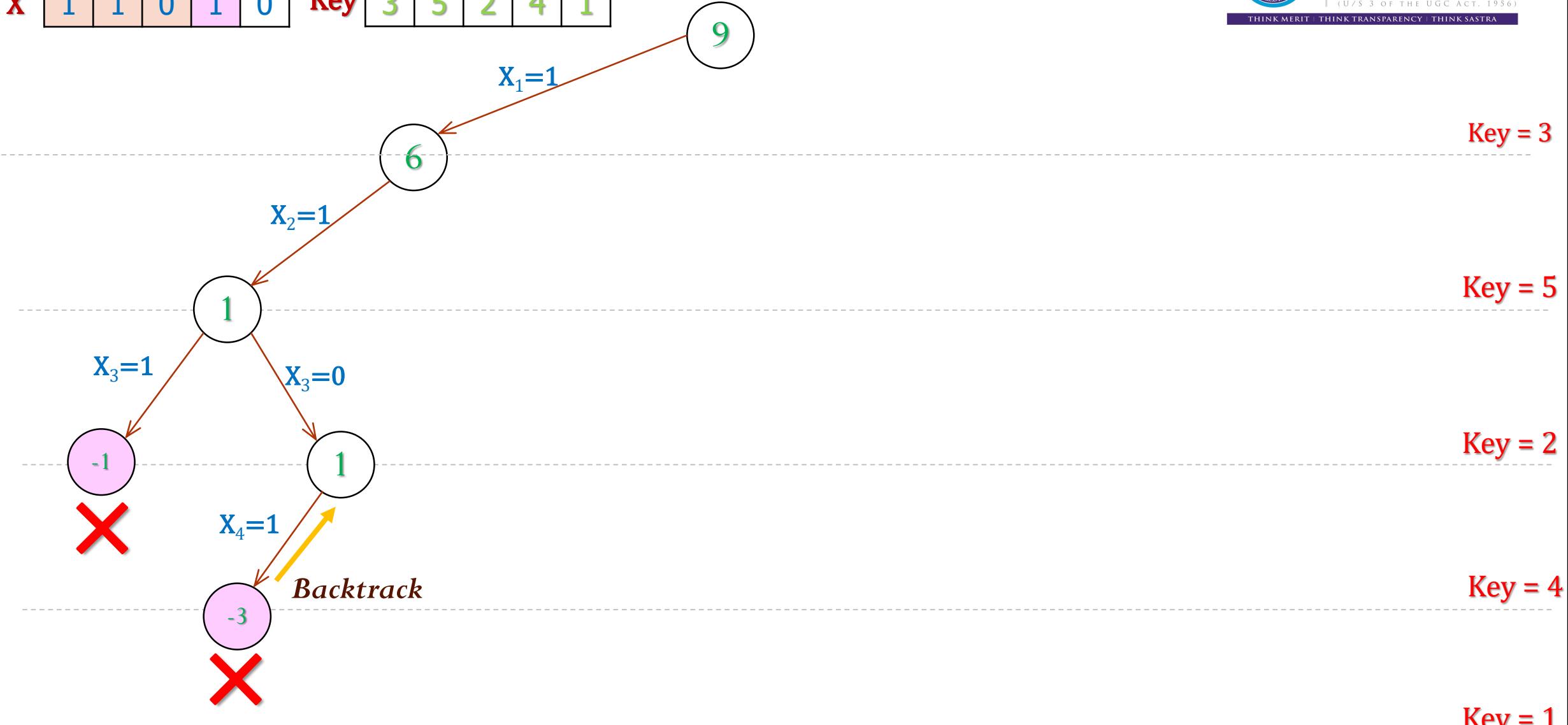
Start



	1	2	3	4	5
X	1	1	0	1	0

Key

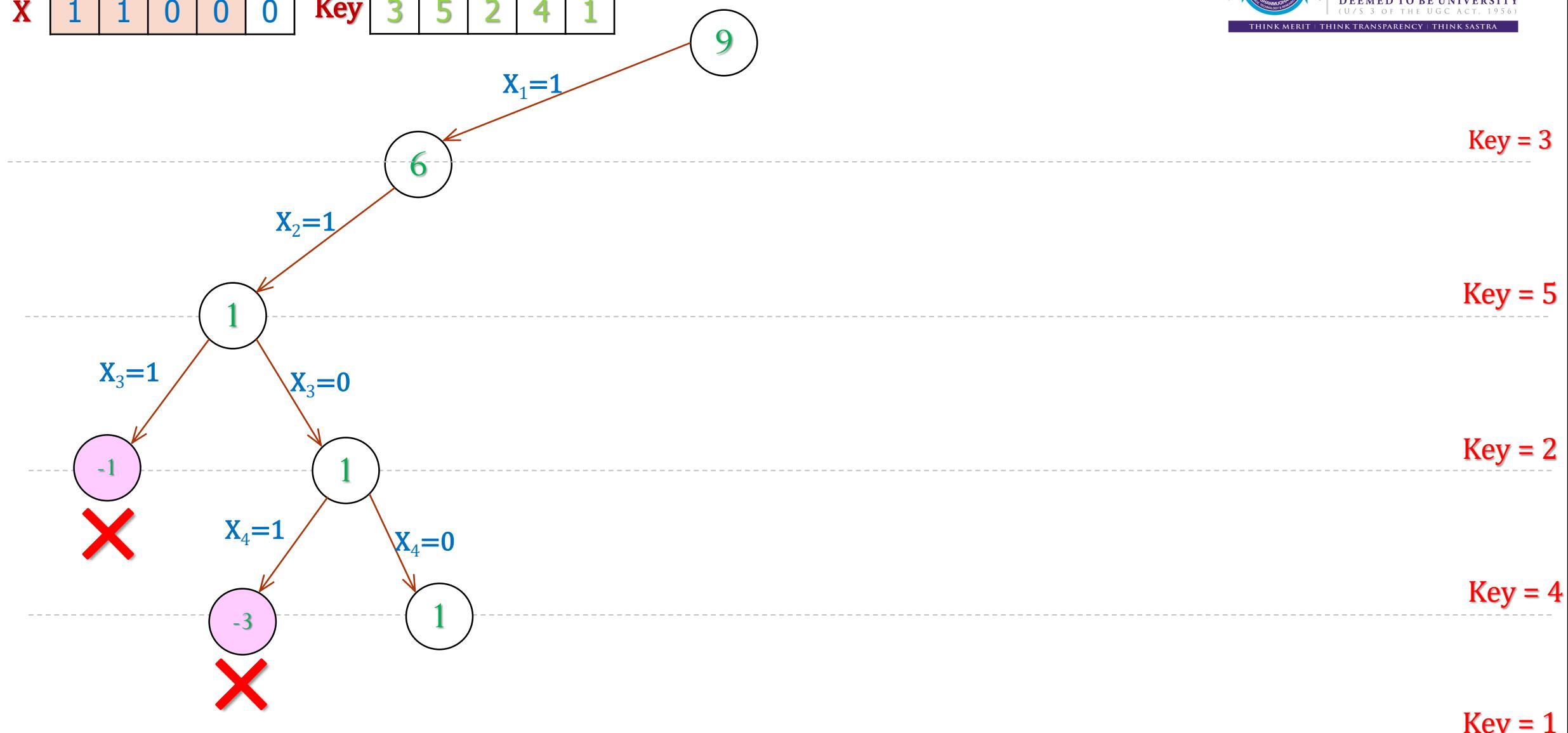
	1	2	3	4	5
Key	3	5	2	4	1



	1	2	3	4	5
X	1	1	0	0	0

Key [3 5 2 4 1]

Start



X

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

Key

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

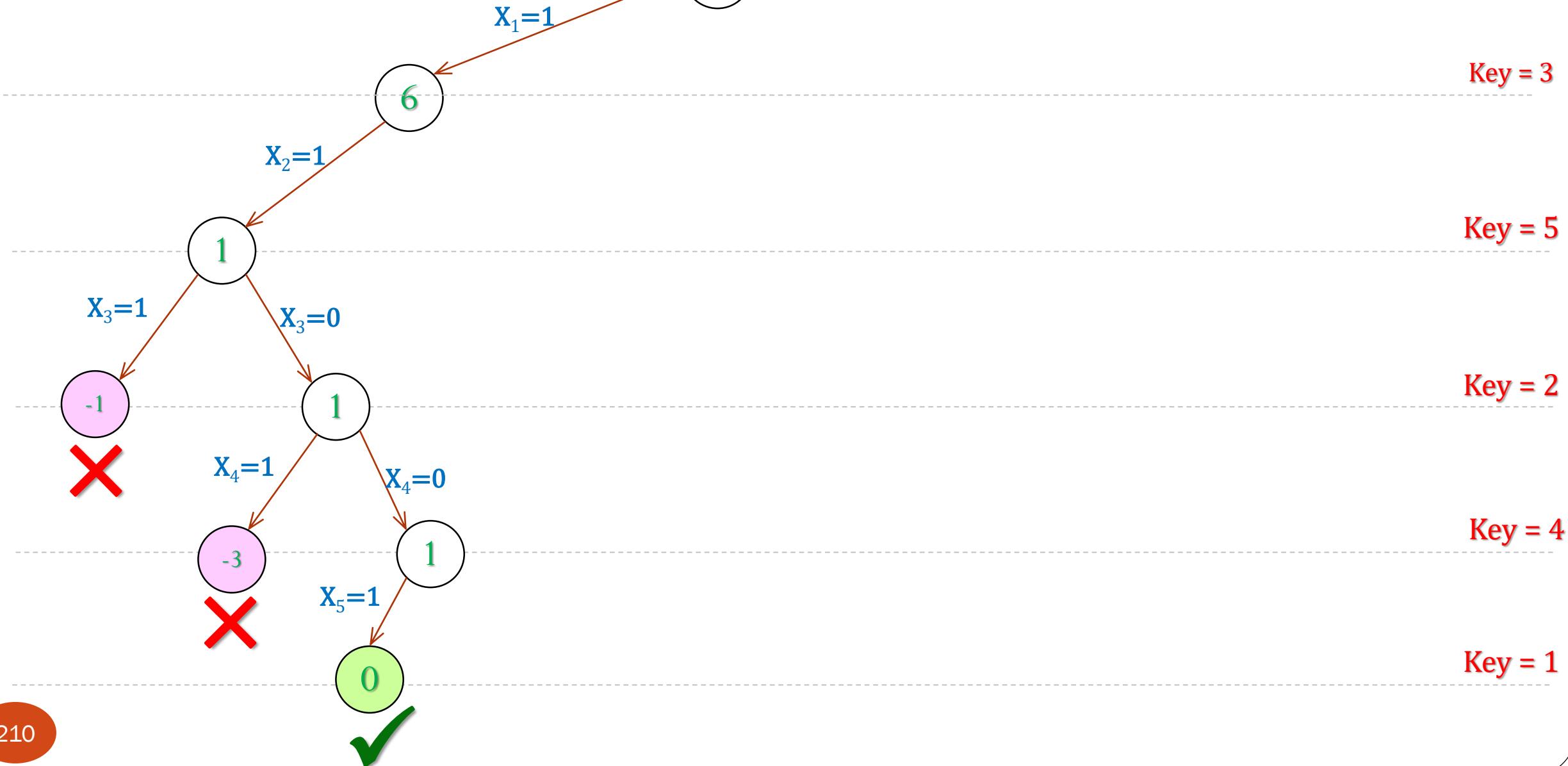
Start

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

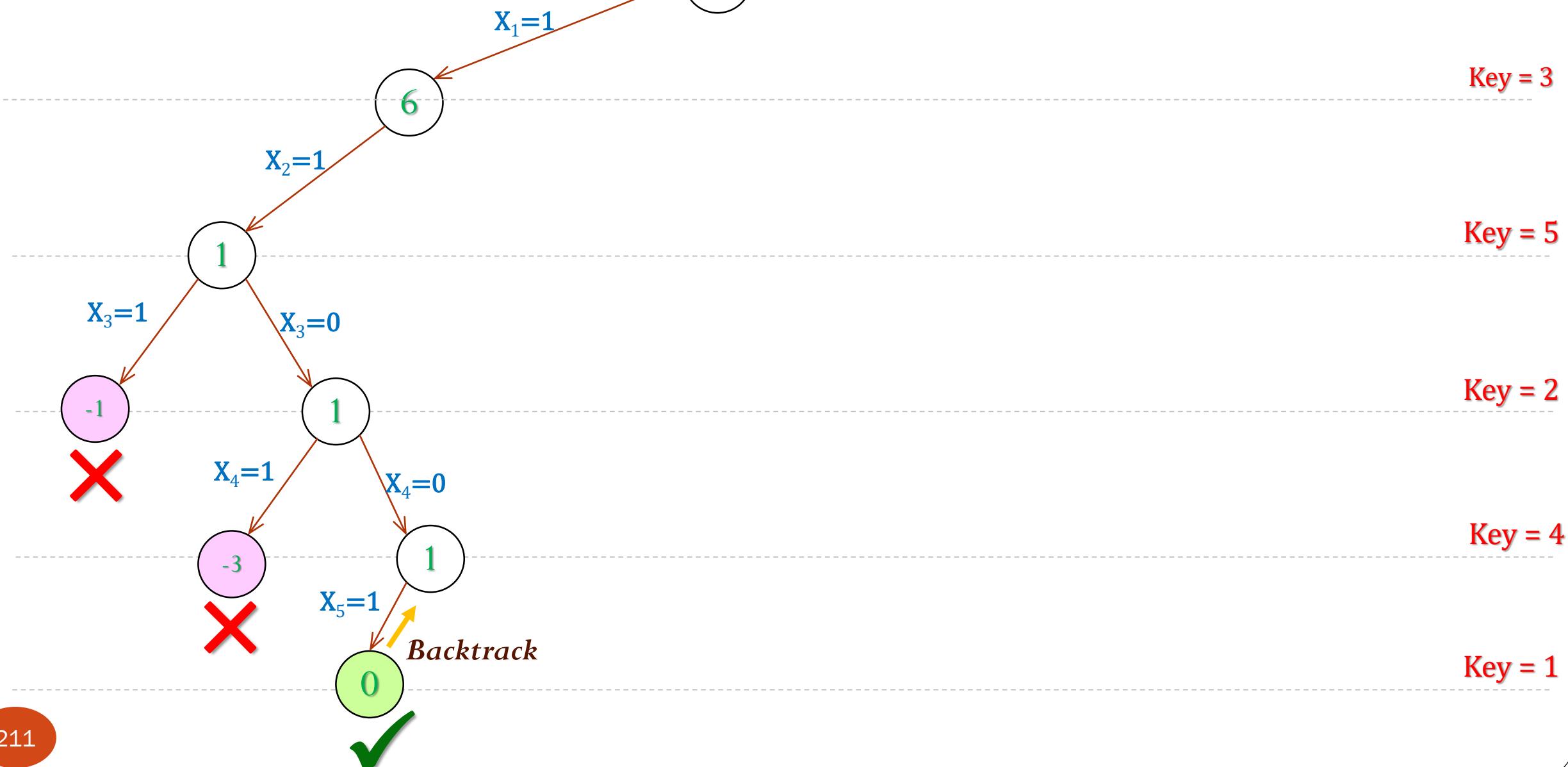
Start

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



	1	2	3	4	5
X	1	1	0	0	0

	1	2	3	4	5
Key	3	5	2	4	1

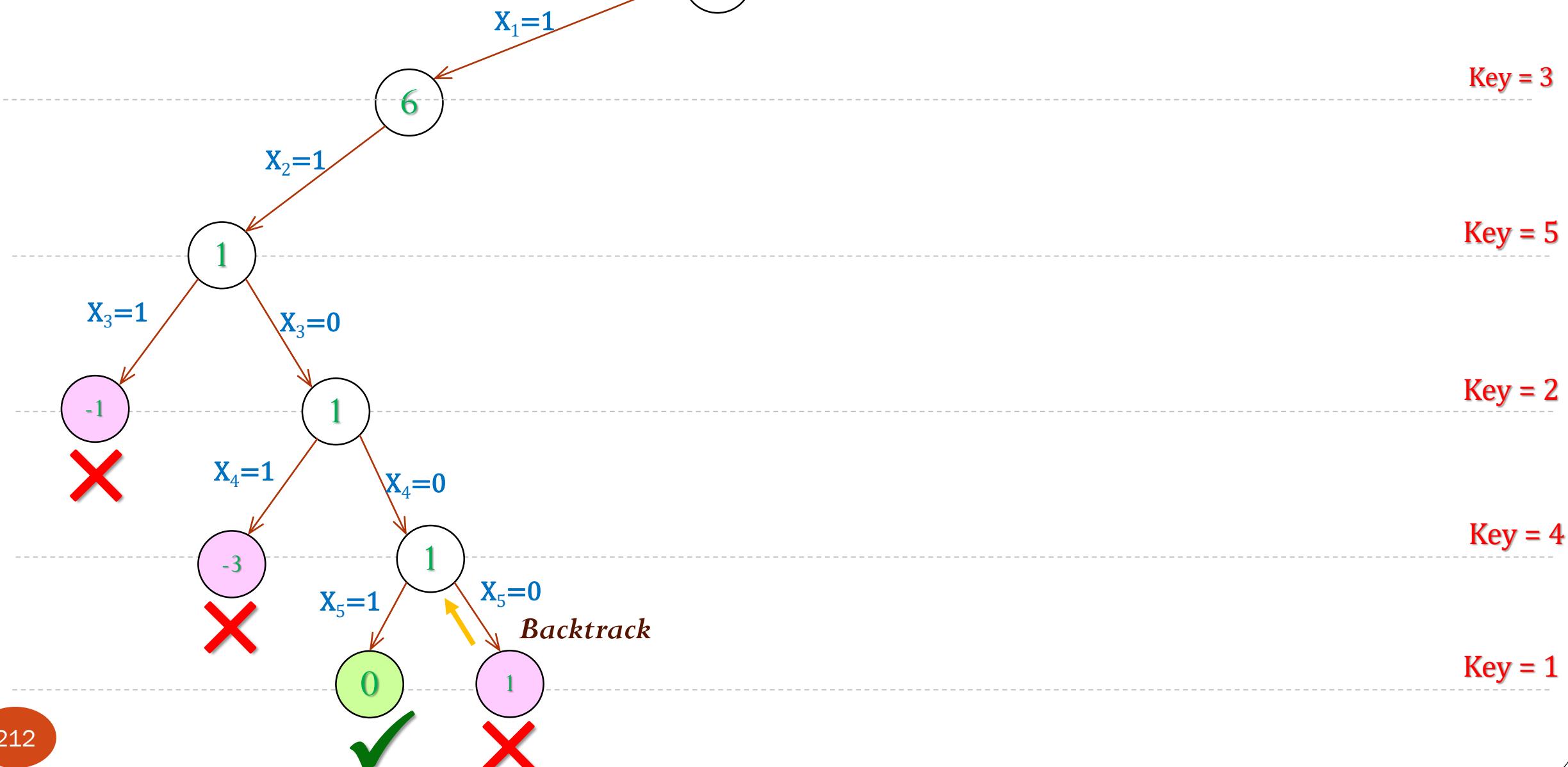
Start

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

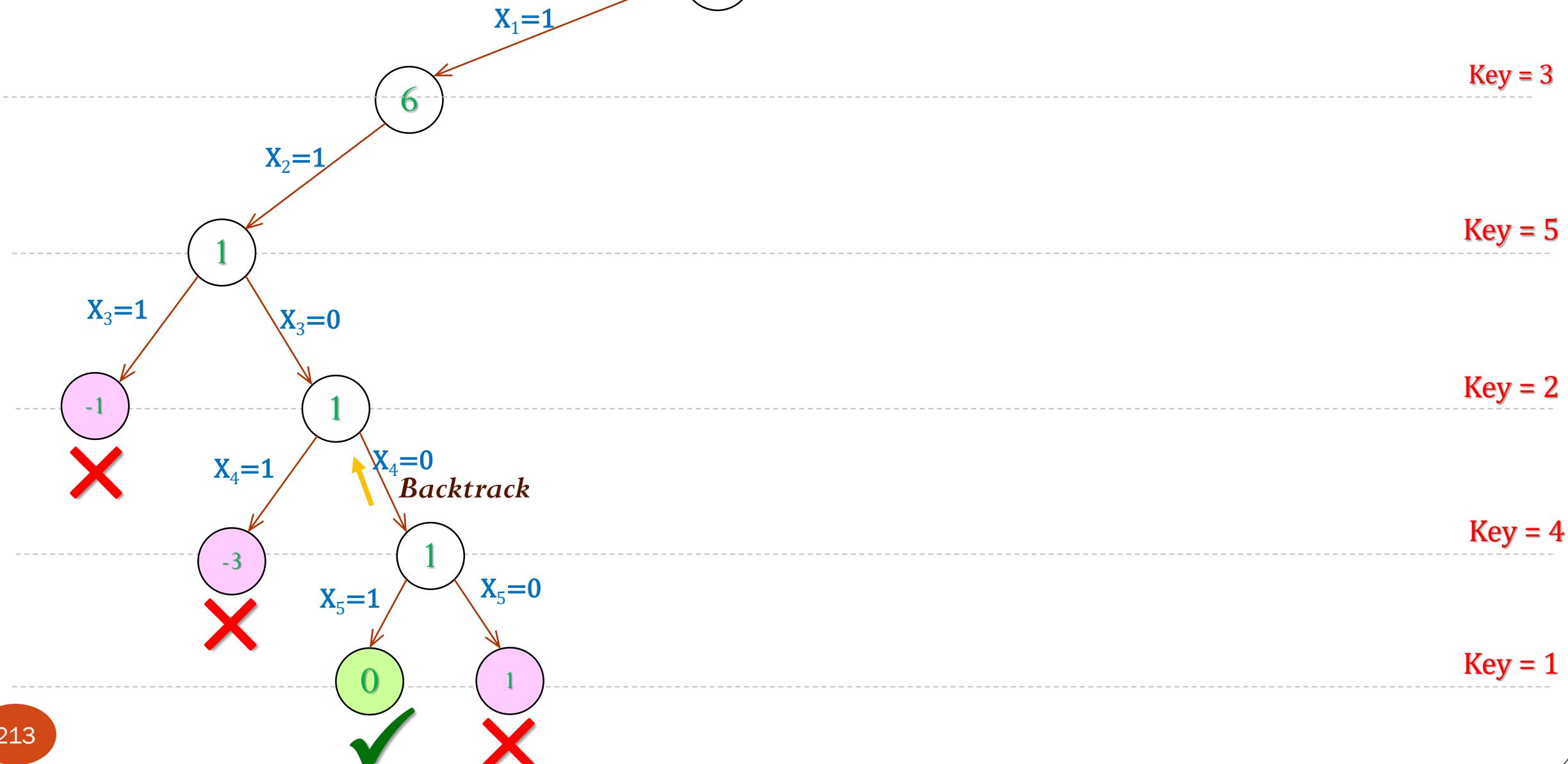
Start

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



	1	2	3	4	5
X	1	1	0	0	0

	1	2	3	4	5
Key	3	5	2	4	1

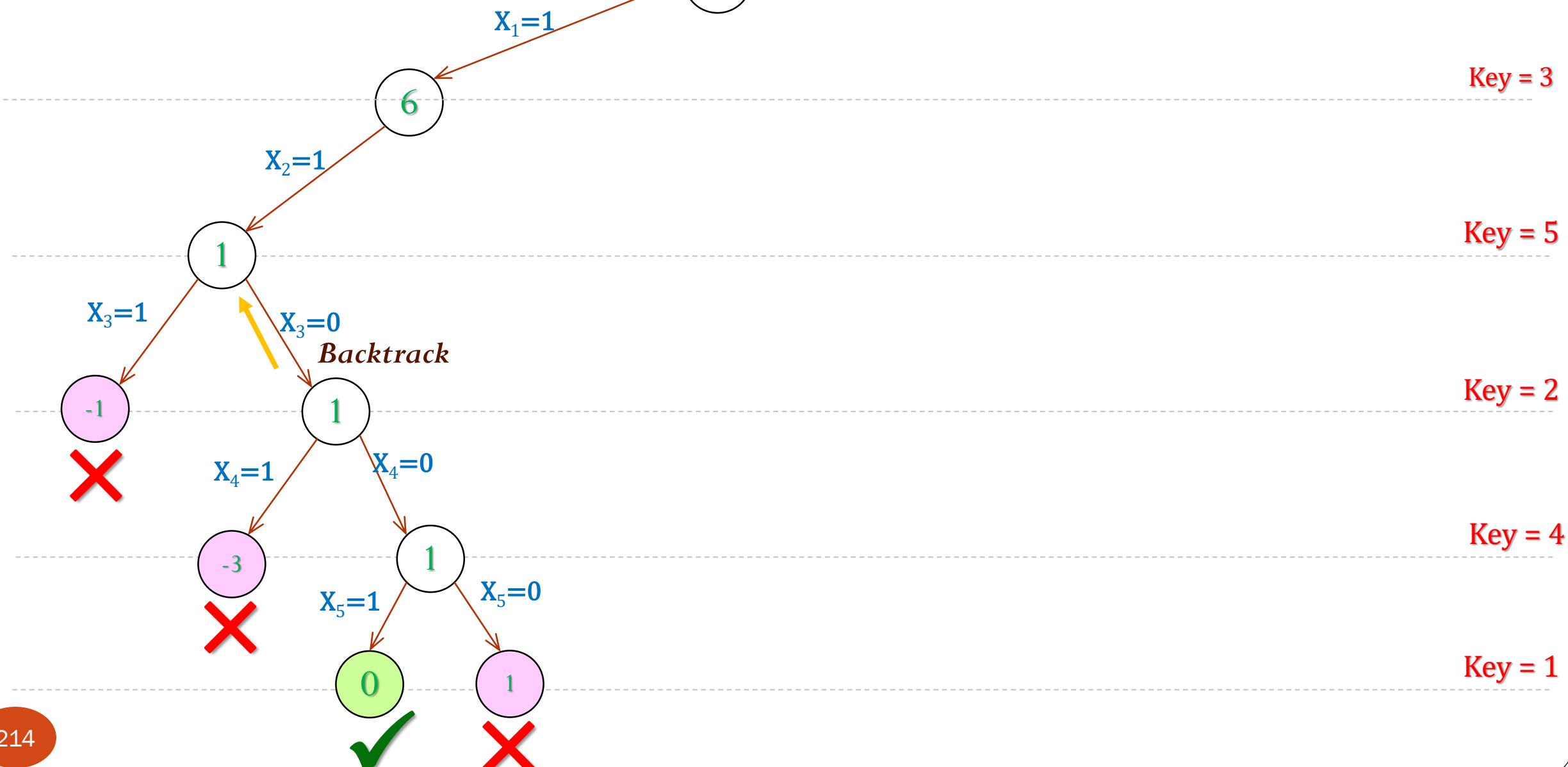
Start

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	0	0	0	0

	1	2	3	4	5
Key	3	5	2	4	1

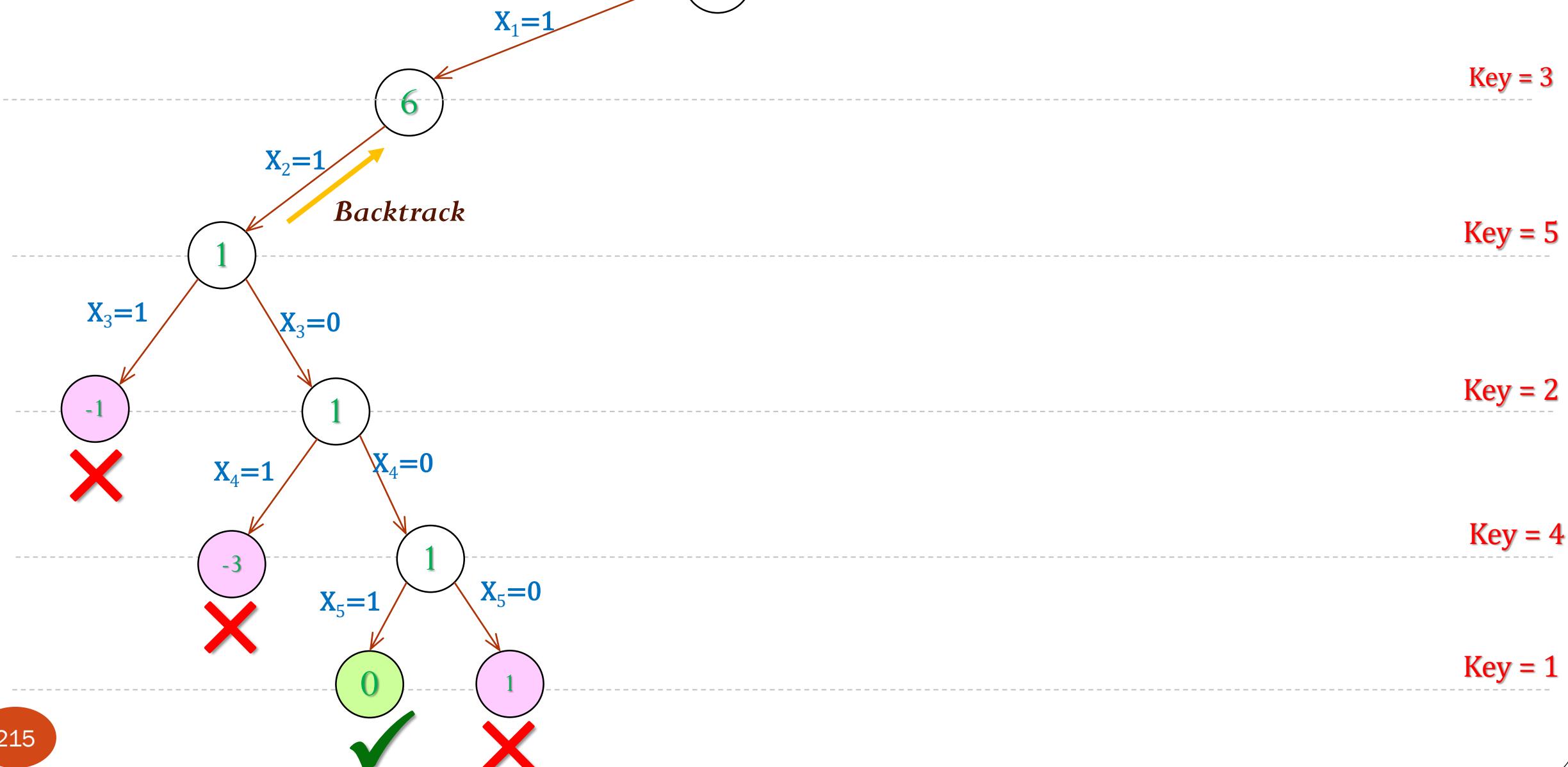
Start

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
---	---	---	---	---

Key

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

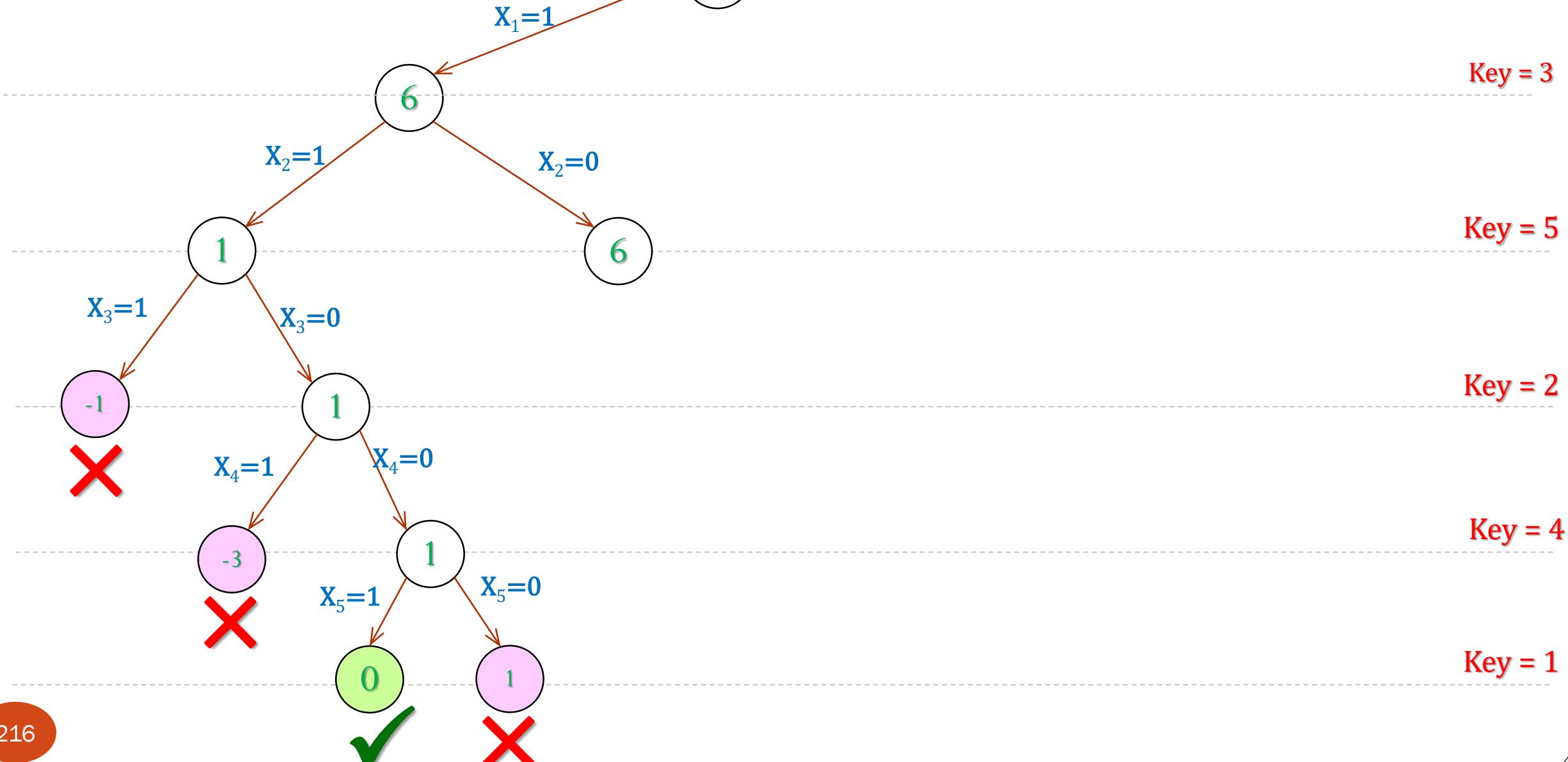
Start

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
---	---	---	---	---

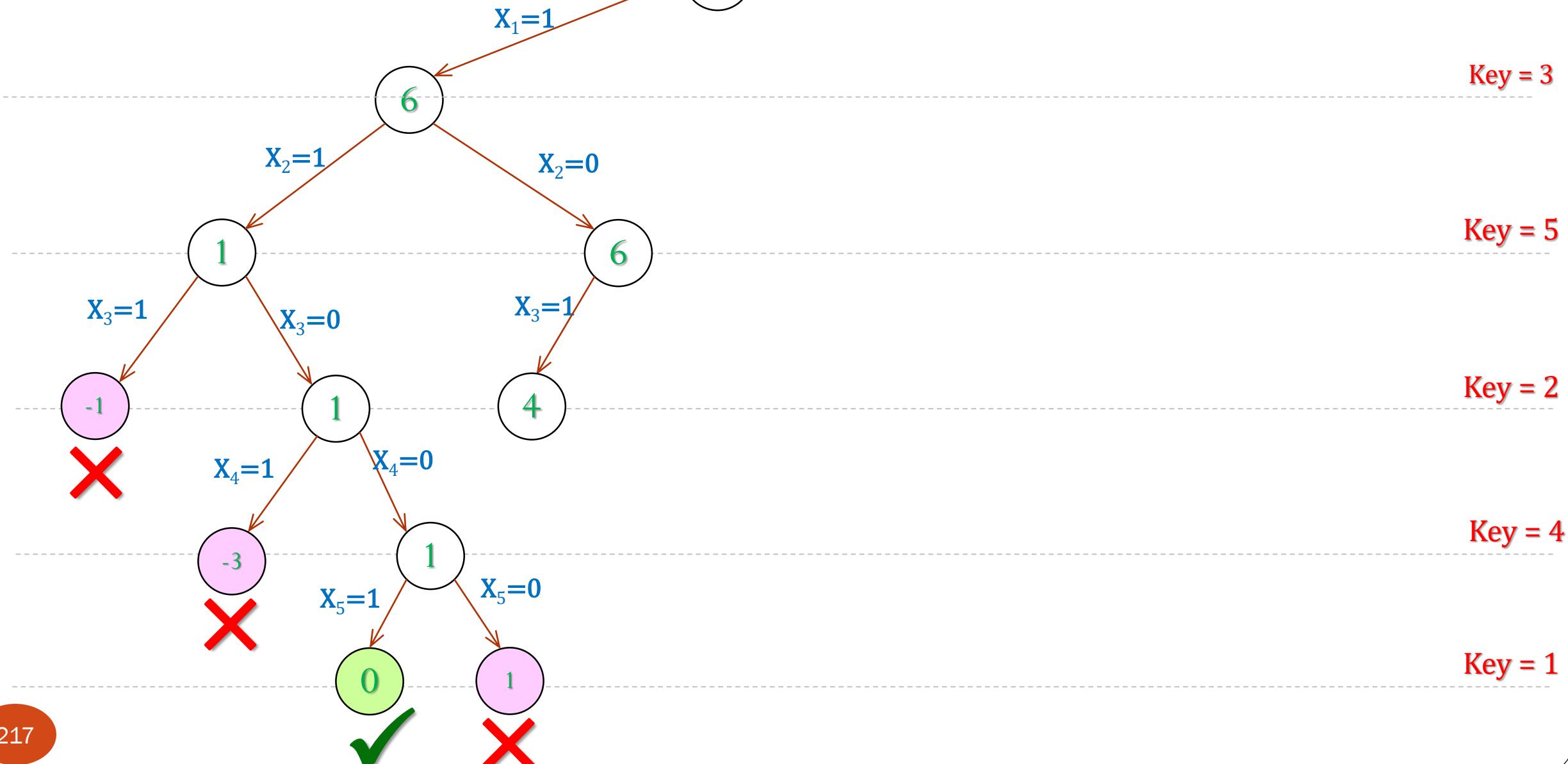
Start

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	1

Start

9

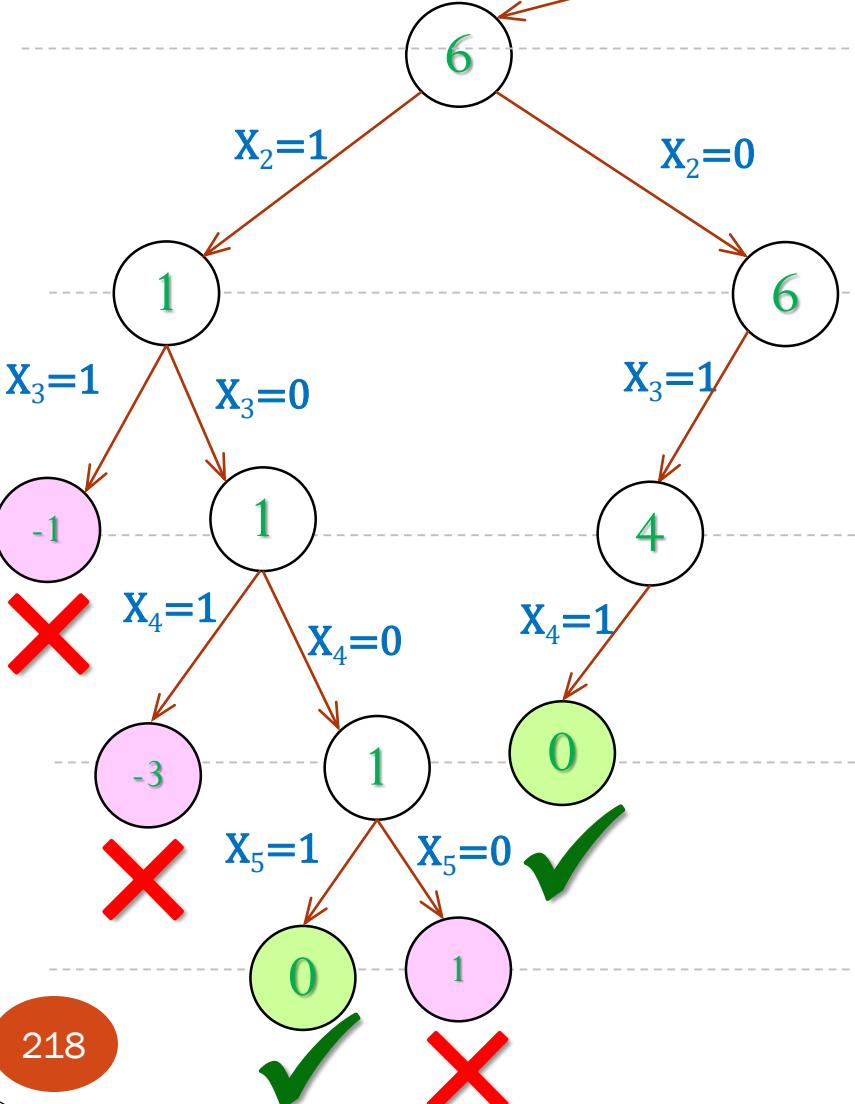
Solutions

1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

Key = 3



	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



$X_1 = 1$

Key = 3

$X_2 = 1$

Key = 5

$X_2 = 0$

$X_3 = 1$

Key = 2

$X_3 = 0$

$X_3 = 1$

Key = 4

$X_4 = 1$

Key = 1

$X_4 = 0$

$X_4 = 1$

Backtrack

$X_5 = 1$

$X_5 = 1$

$X_5 = 0$

	1	2	3	4	5
X	1	0	1	0	0

	1	2	3	4	5
Key	3	5	2	4	1

Start

Solutions

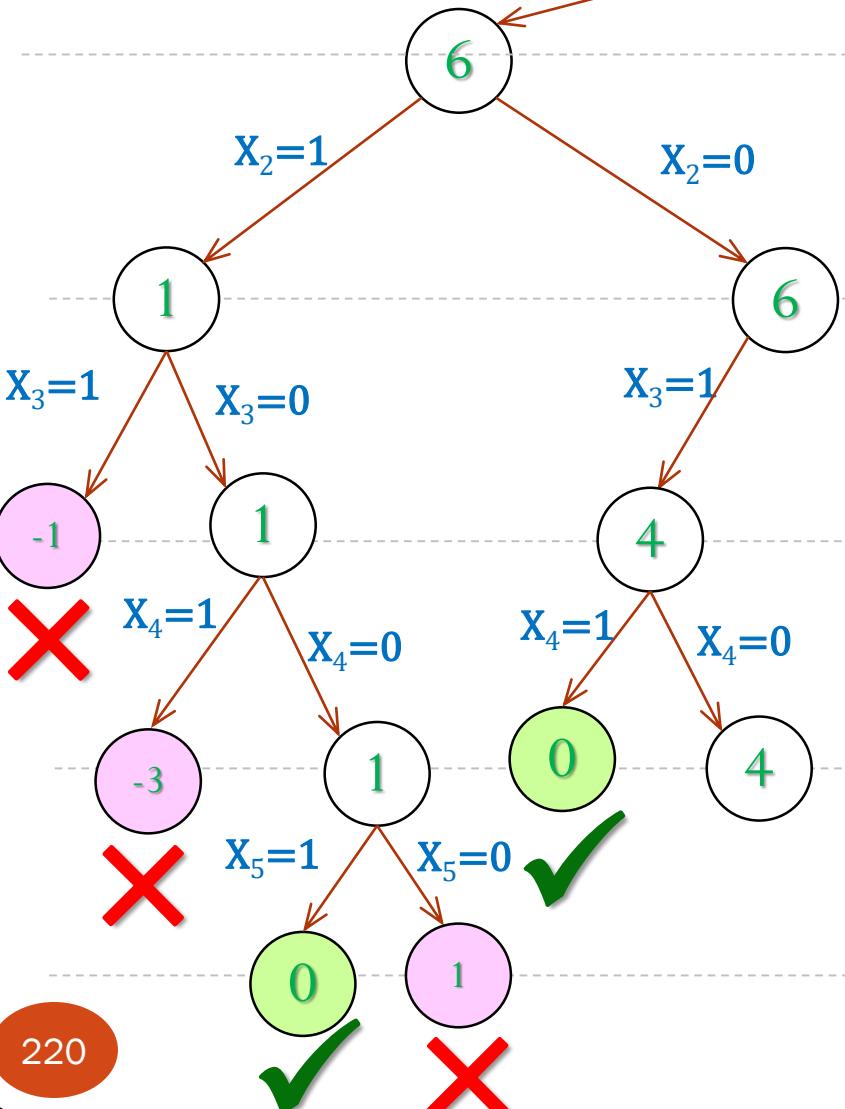
1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3



Key = 5

Key = 2

Key = 4

Key = 1

X	1	0	1	0	1
1	2	3	4	5	

Key	3	5	2	4	1
1	2	3	4	5	

Start

Solutions

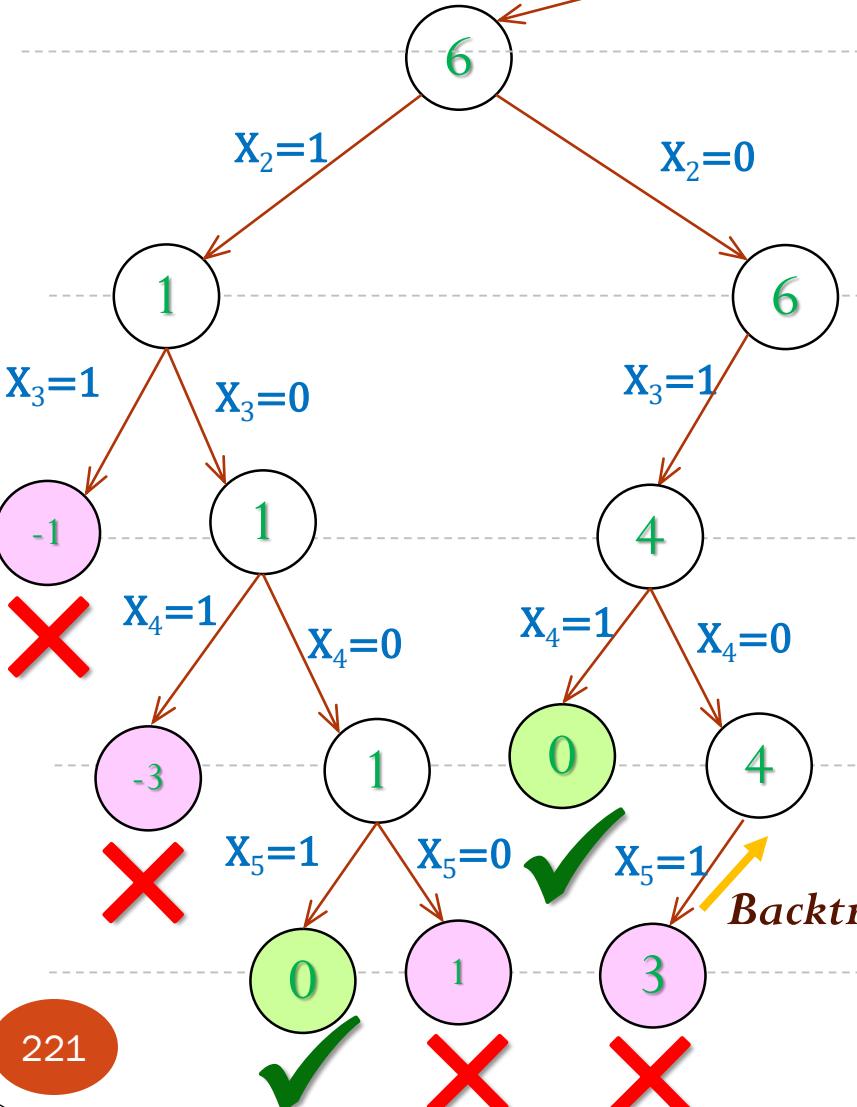
1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3



X	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

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

$X_3 = 1$
 $X_3 = 0$

$X_3 = 1$

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	1	0	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

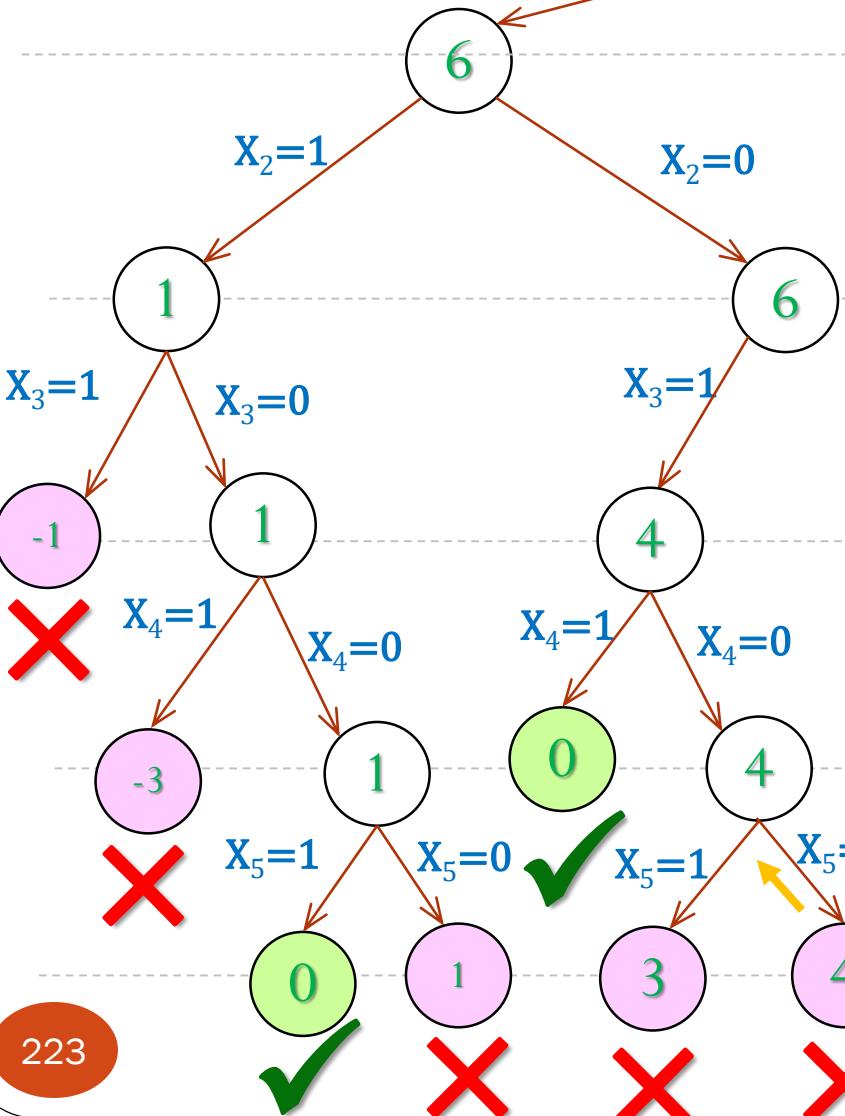
1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3



X	1	0	1	0	0
	1	2	3	4	5

Key	3	5	2	4	1
	1	2	3	4	5

Start

Solutions

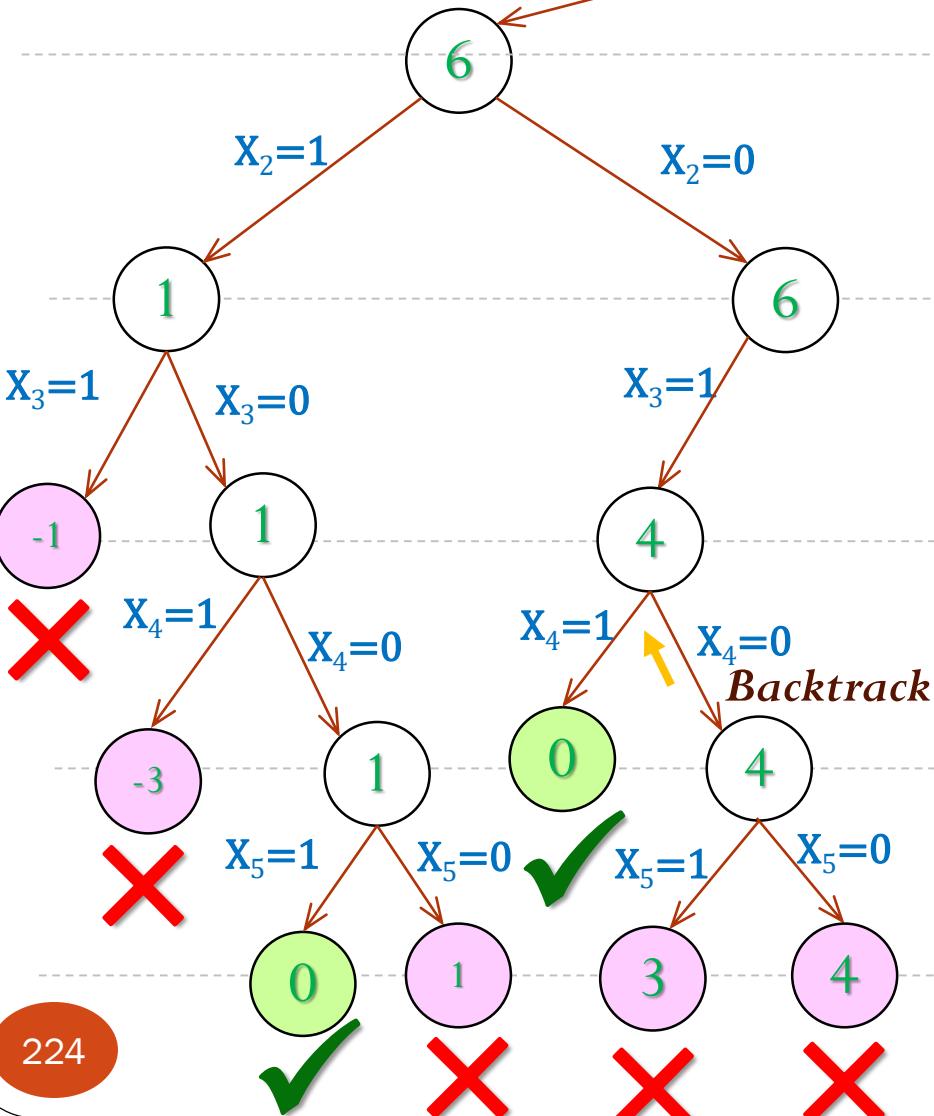
1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3



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

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

Start

Solutions

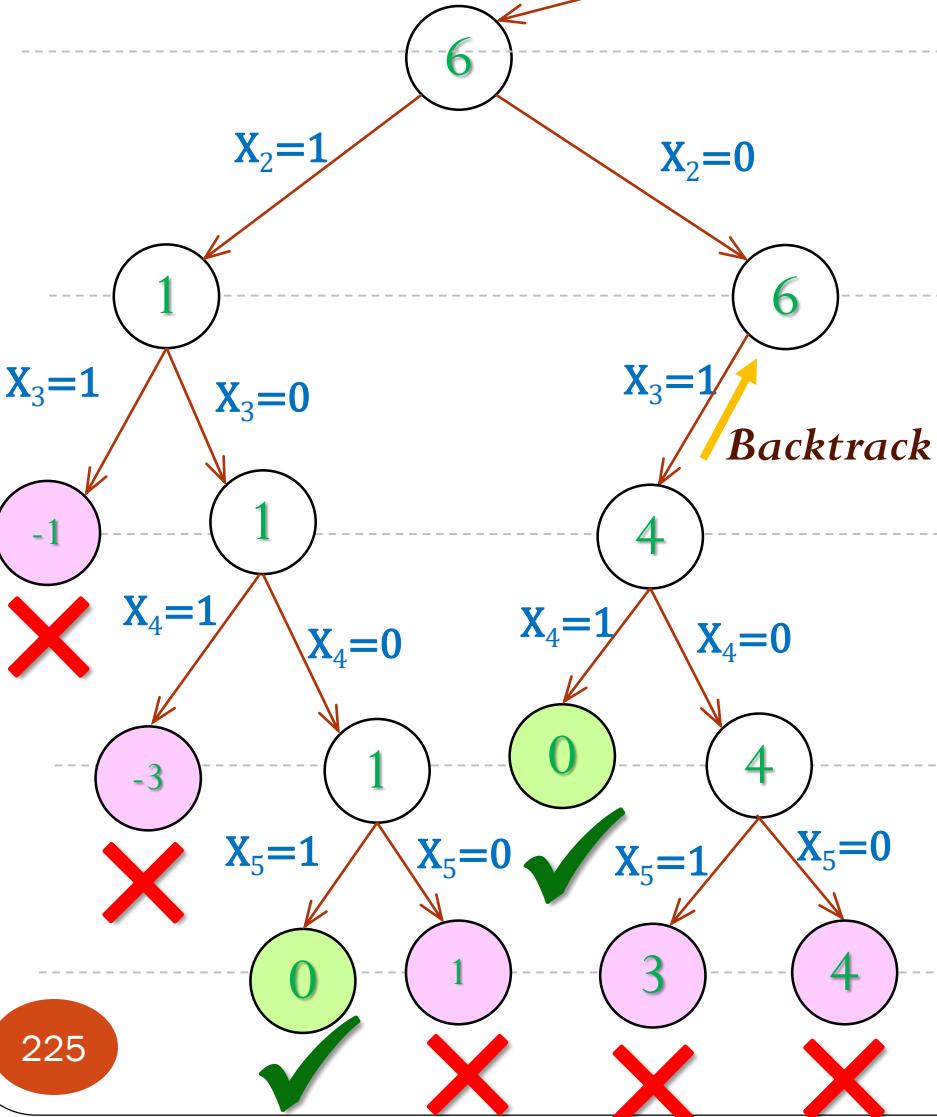
1	1	0	0	1
1	0	1	1	0



$x_1 = 1$

9

Key = 3



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

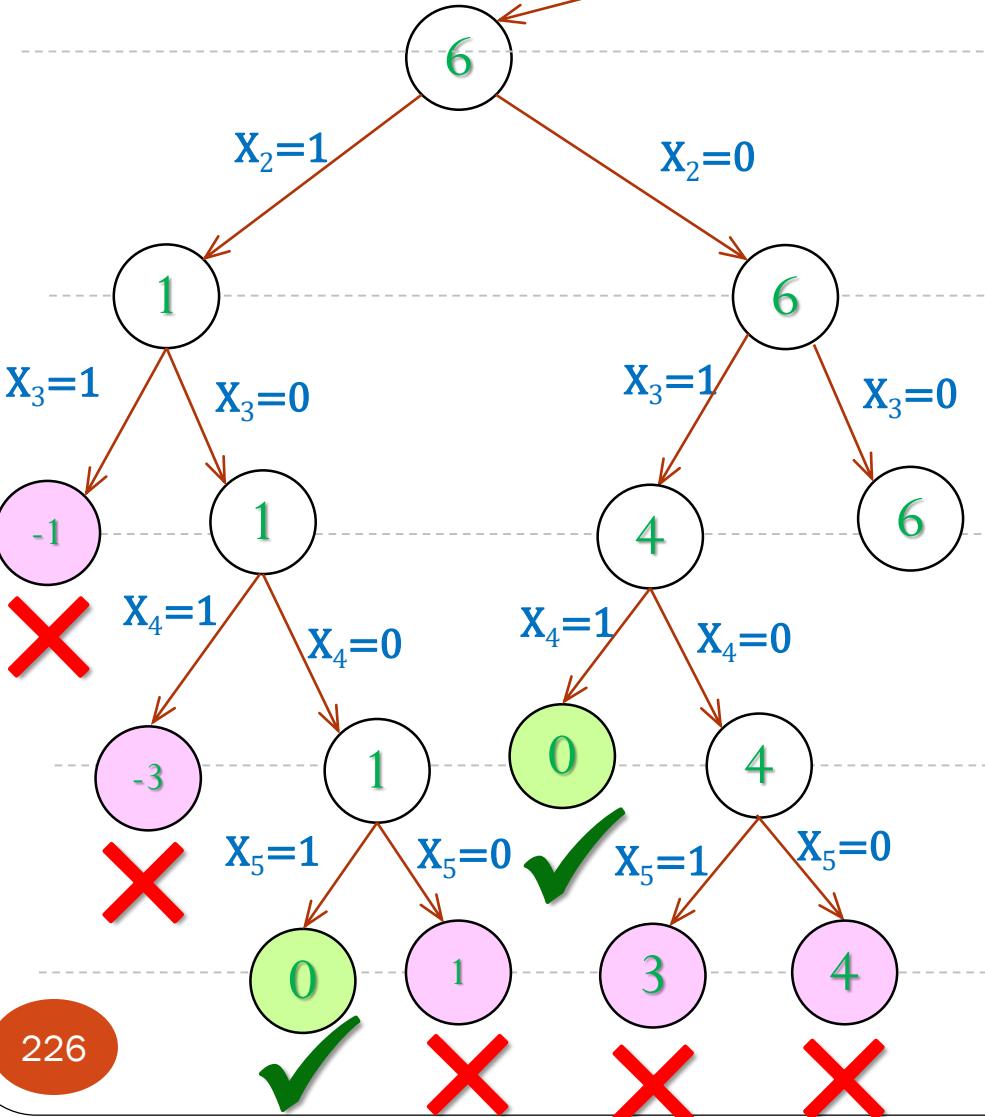
1	1	0	0	1
1	0	1	1	0



$X_1 = 1$

9

Key = 3



Key = 5

Key = 2

Key = 4

Key = 1

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

Key = 2

$X_4 = 1$

X

$X_4 = 0$

-1

$X_4 = 1$

X

$X_4 = 0$

-3

$X_4 = 1$

X

$X_4 = 0$

1

Key = 4

$X_5 = 1$

✓

$X_5 = 0$

0

$X_5 = 1$

✓

$X_5 = 0$

1

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



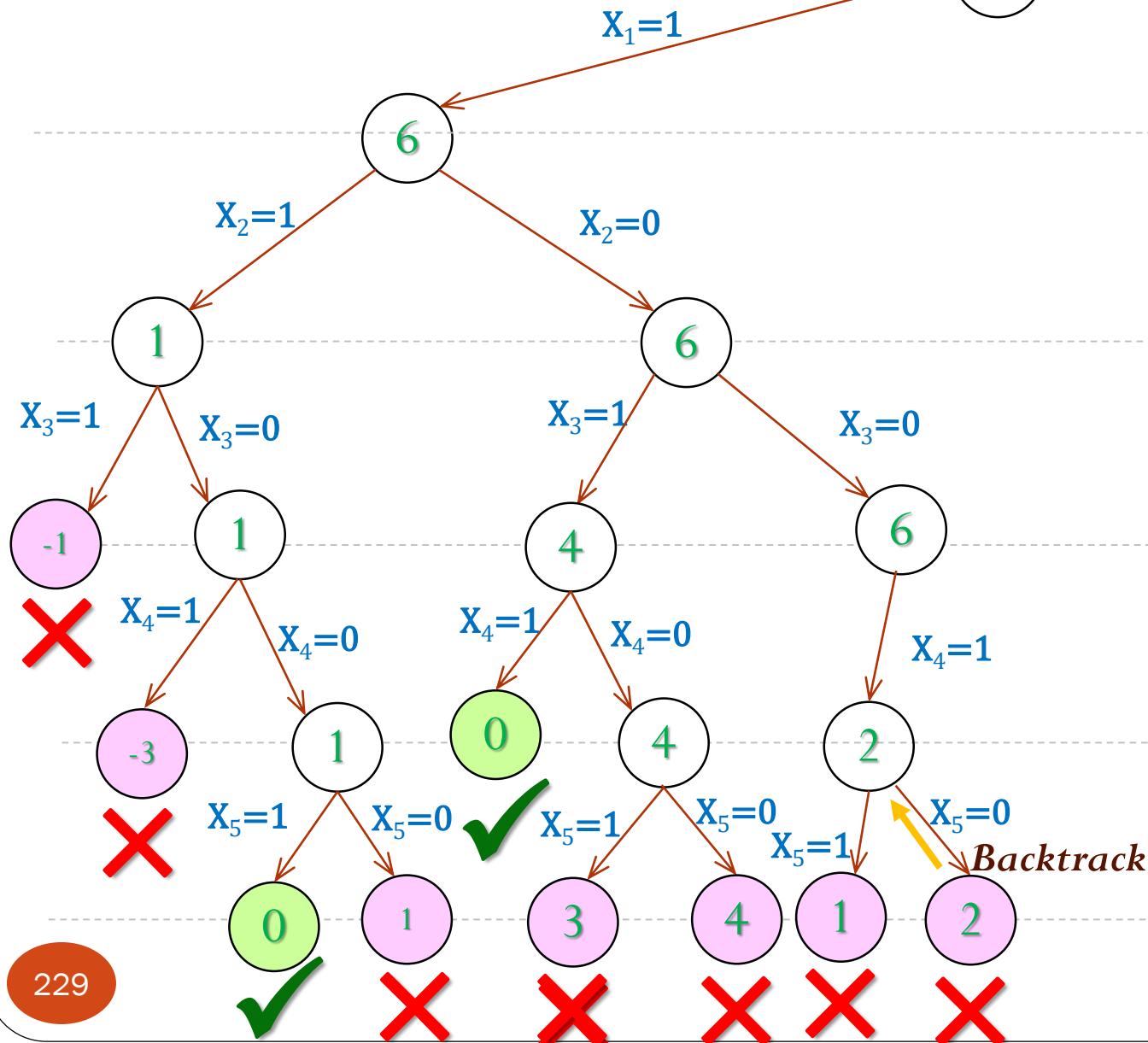
Key = 3

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

231

✓ ✗ ✗ ✗ ✗

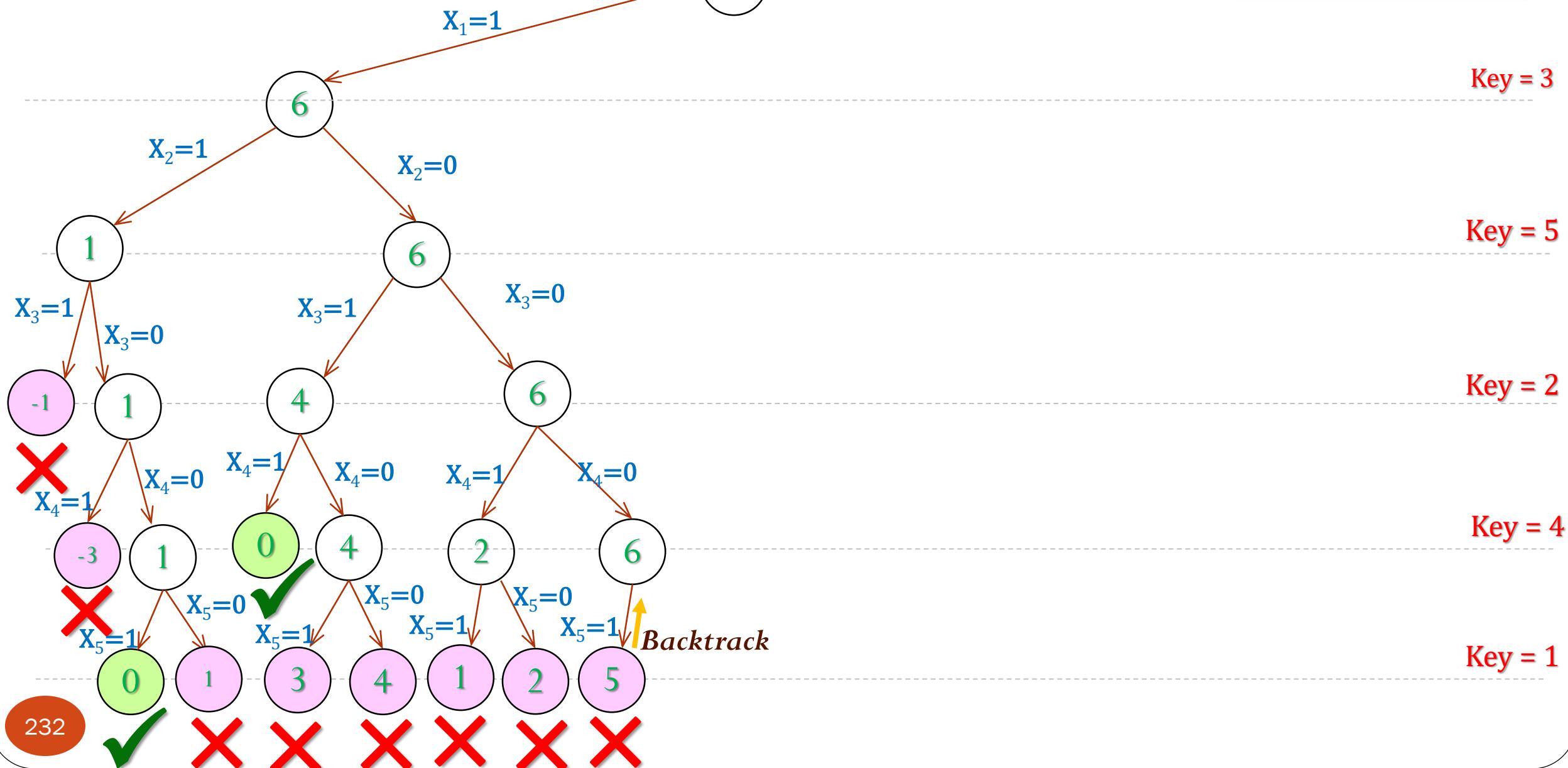
X	1	2	3	4	5
	1	0	0	0	1

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



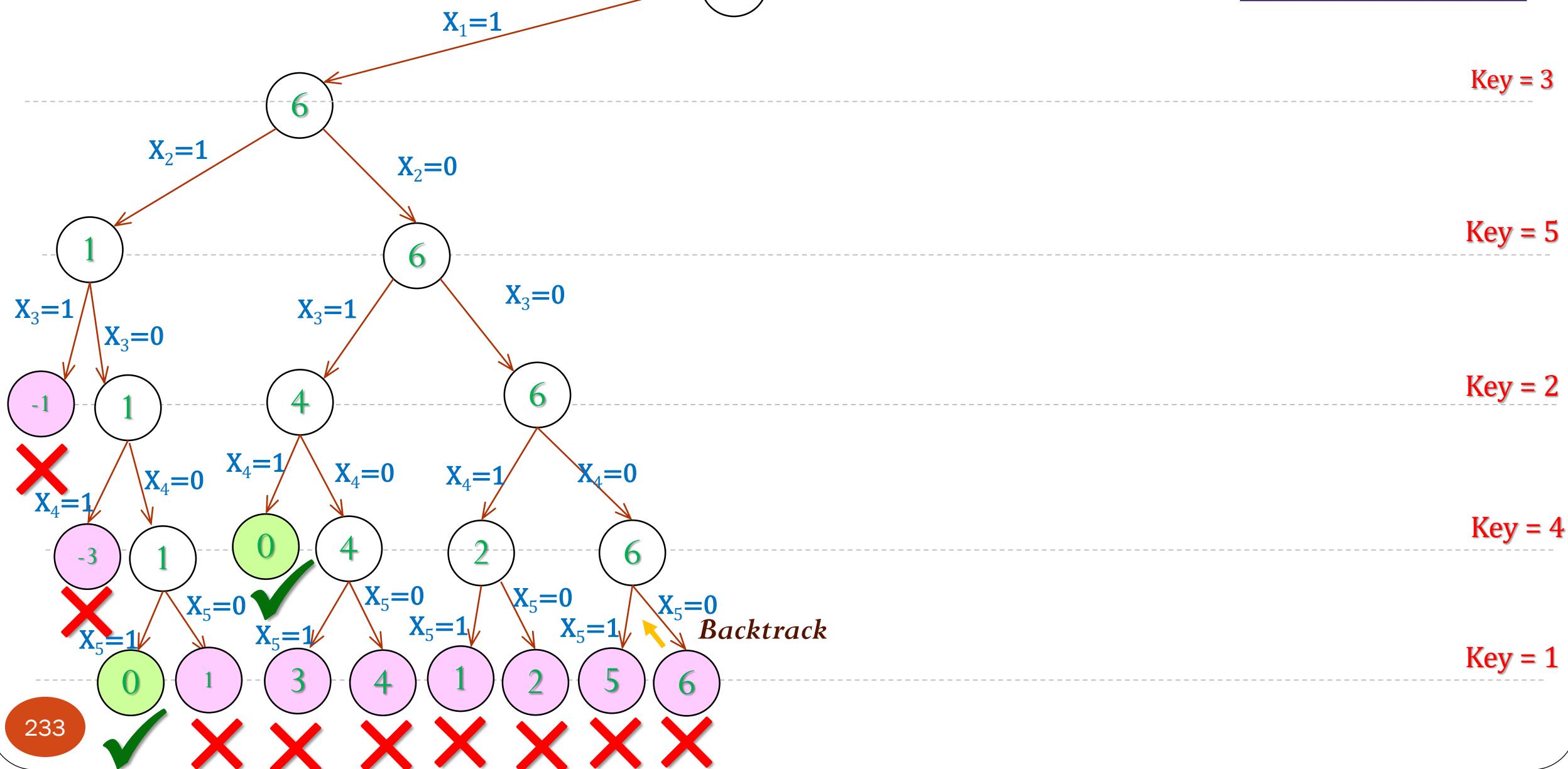
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



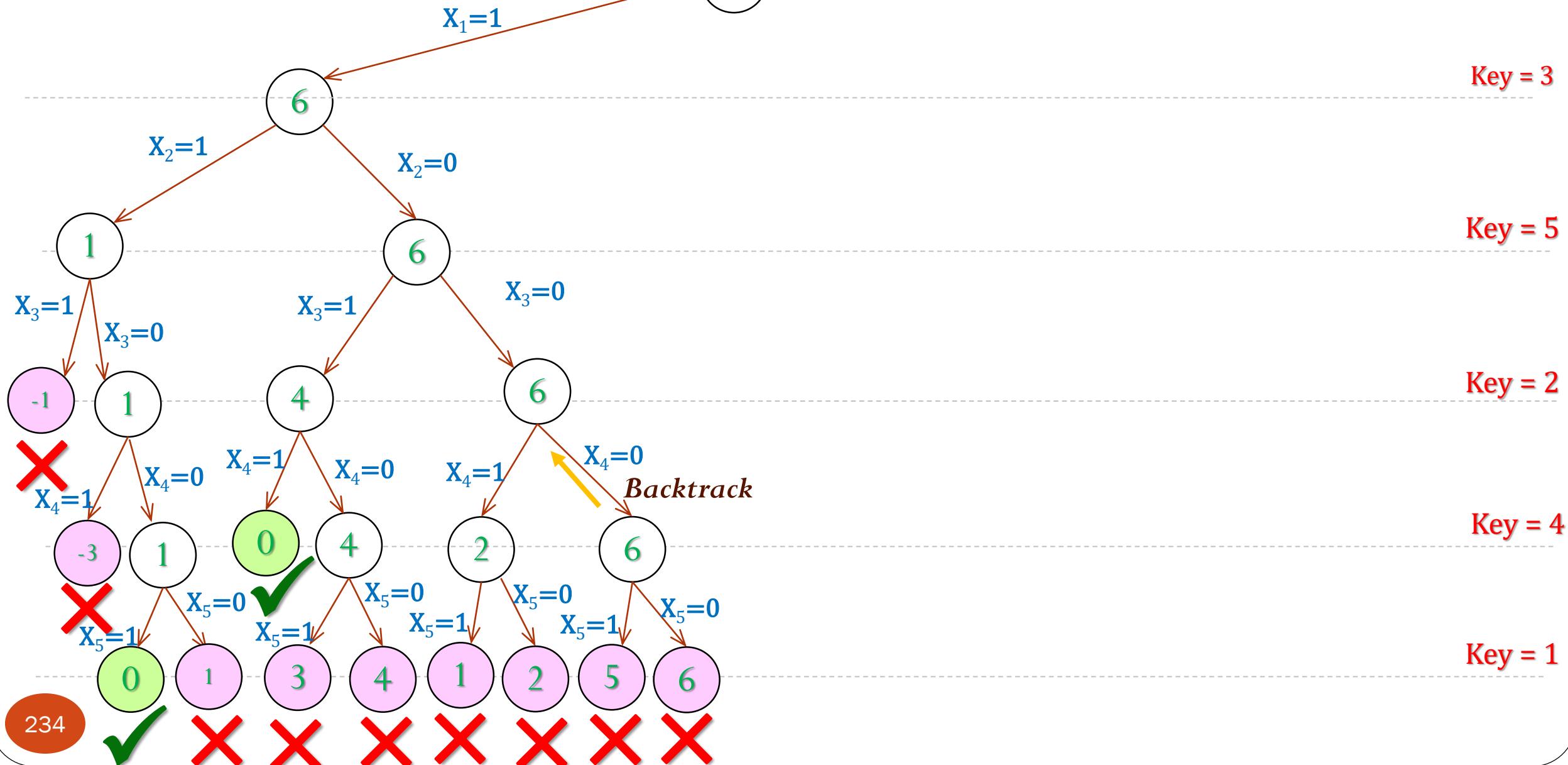
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



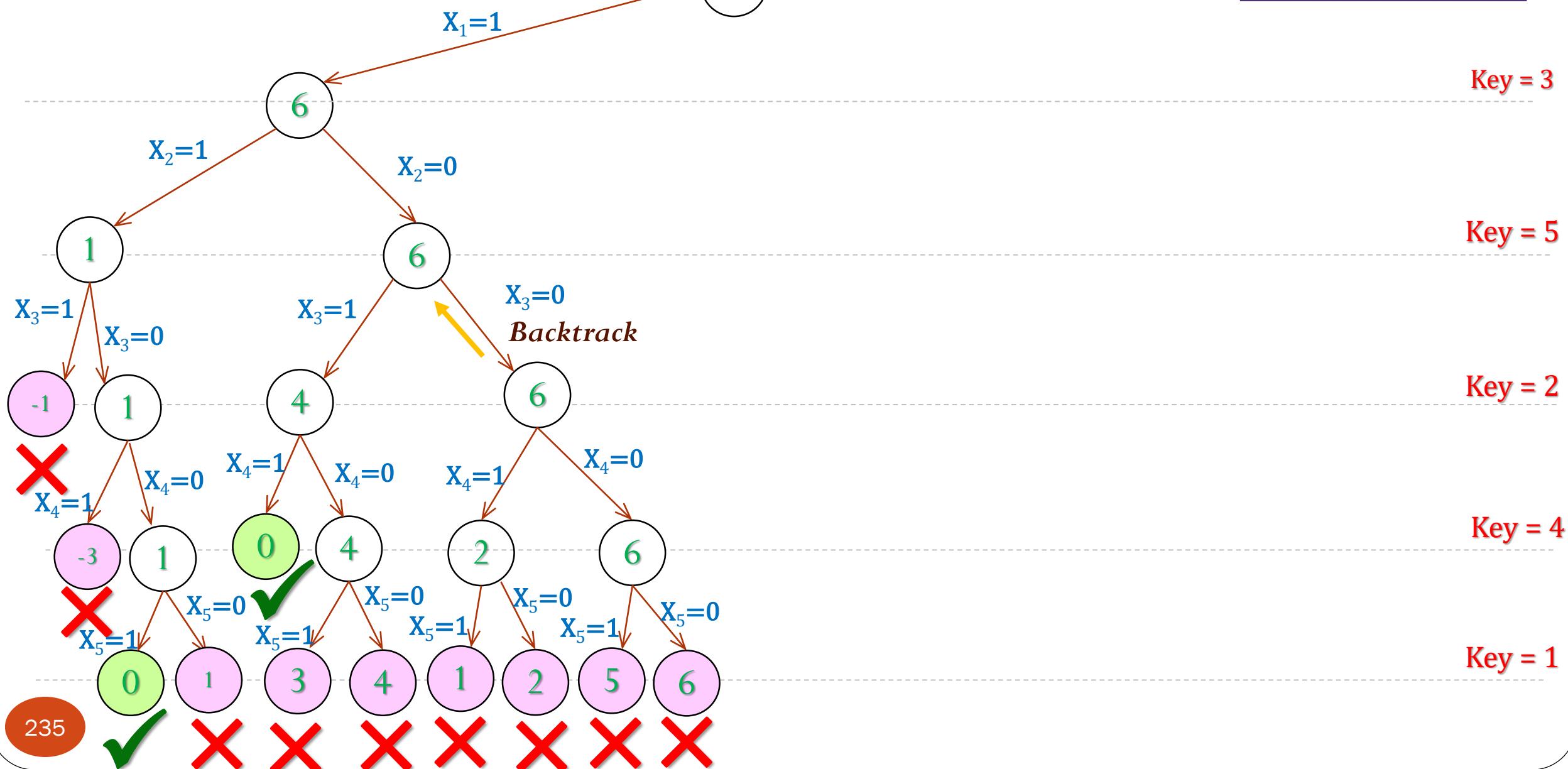
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



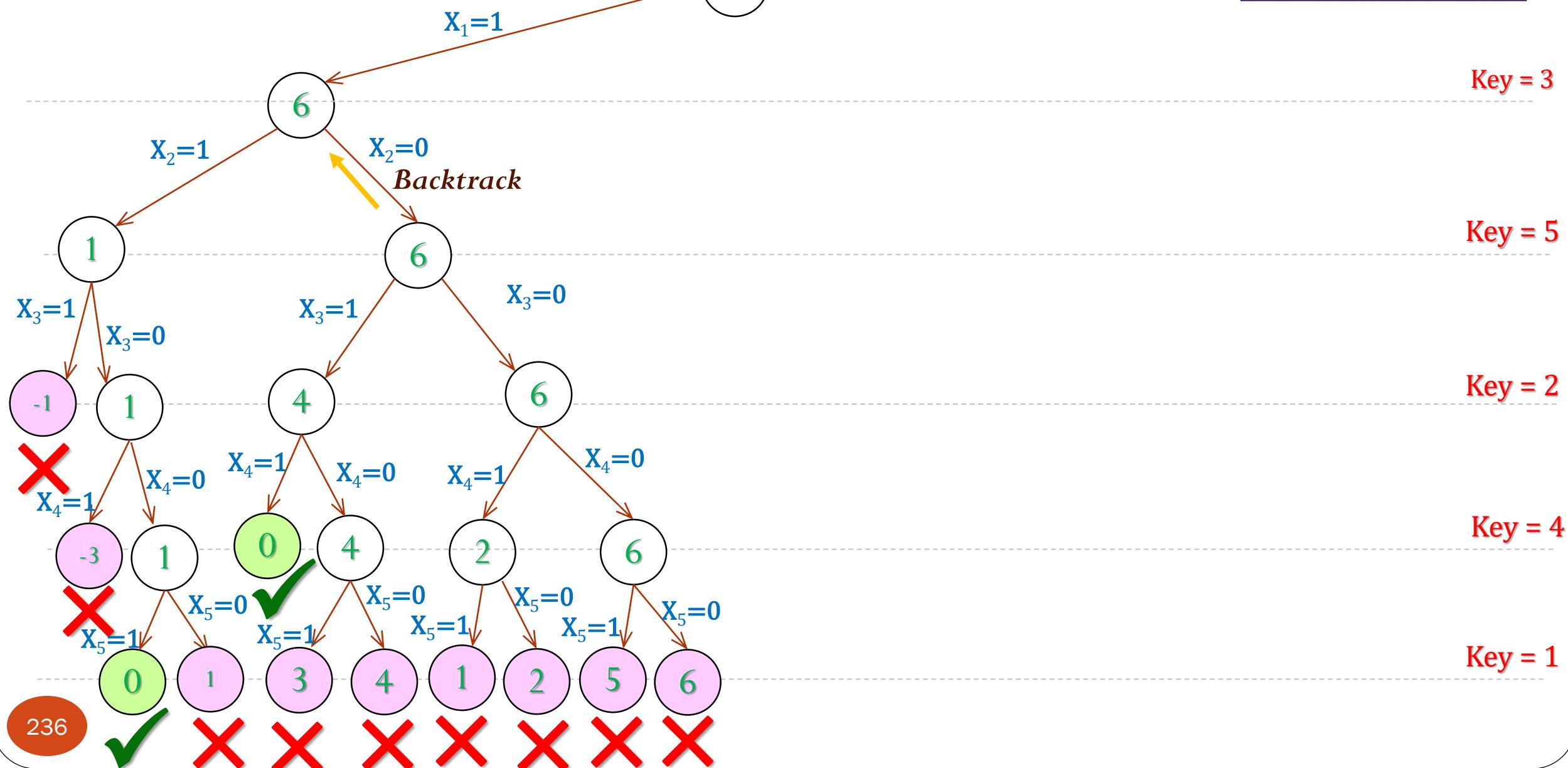
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



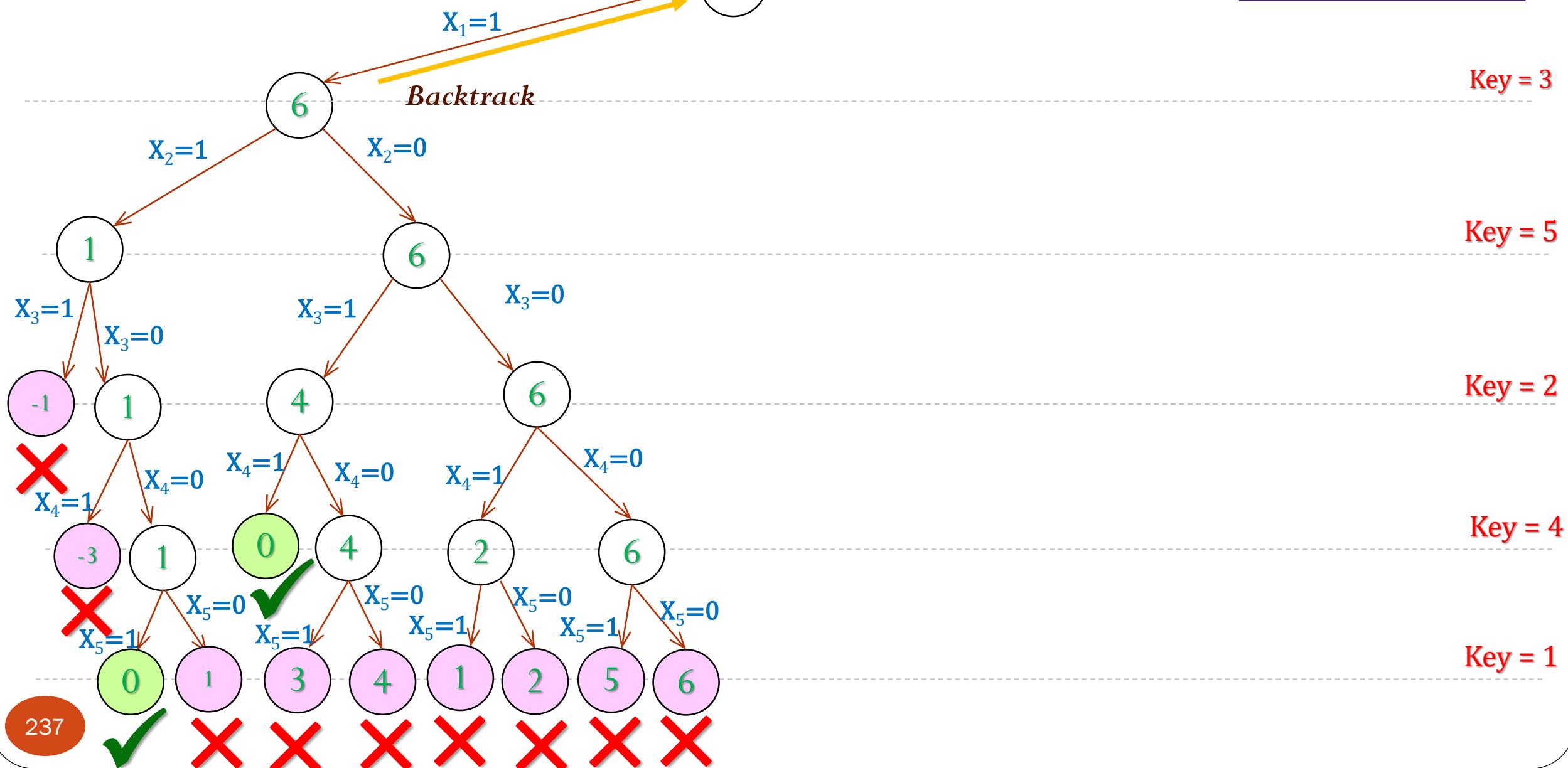
X	1	2	3	4	5
	1	0	0	0	0

Key	1	2	3	4	5
	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



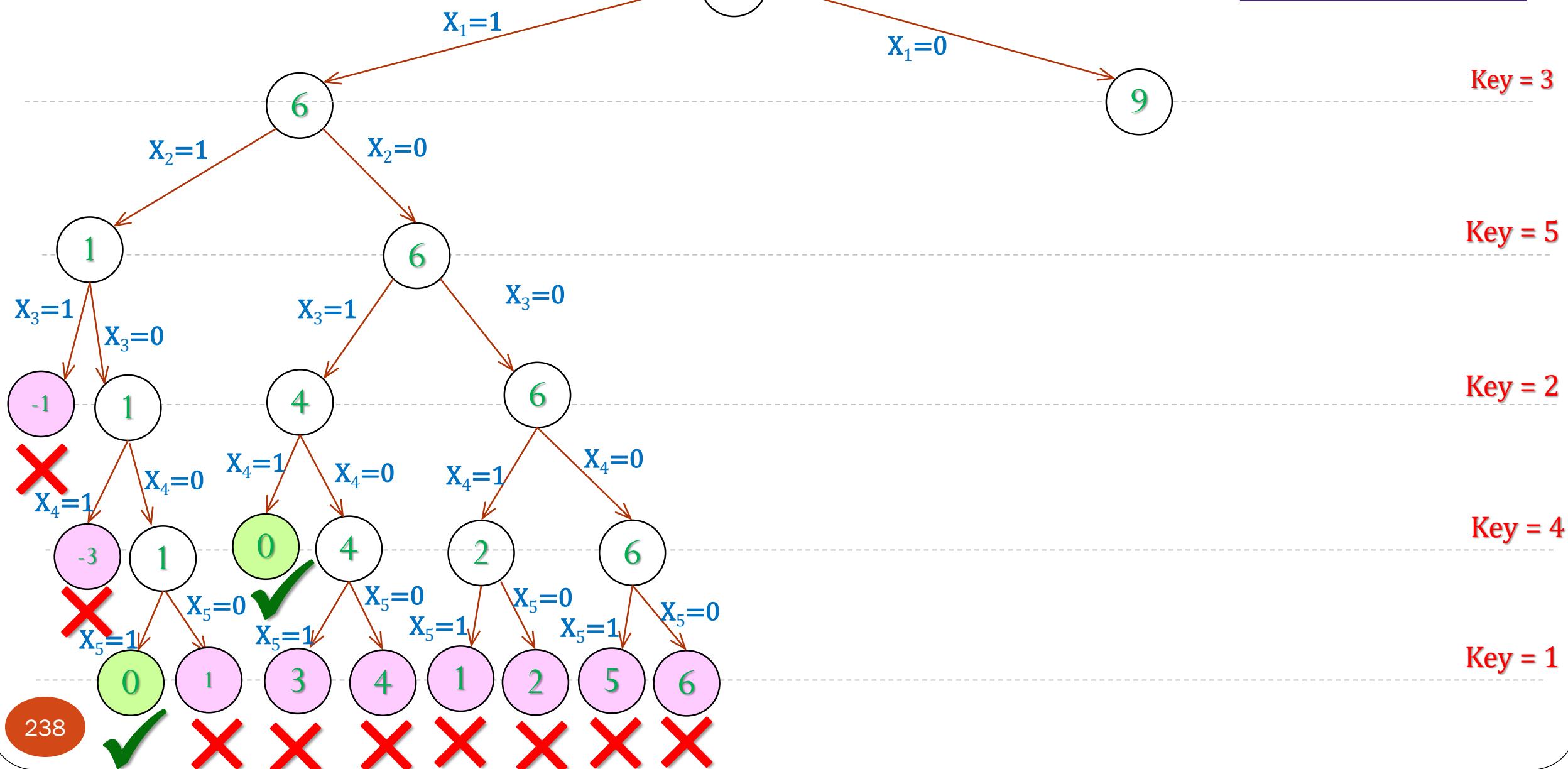
1	2	3	4	5
X	0	0	0	0

1	2	3	4	5
Key	3	5	2	4

Start

Solutions

1	1	0	0	1
1	0	1	1	0



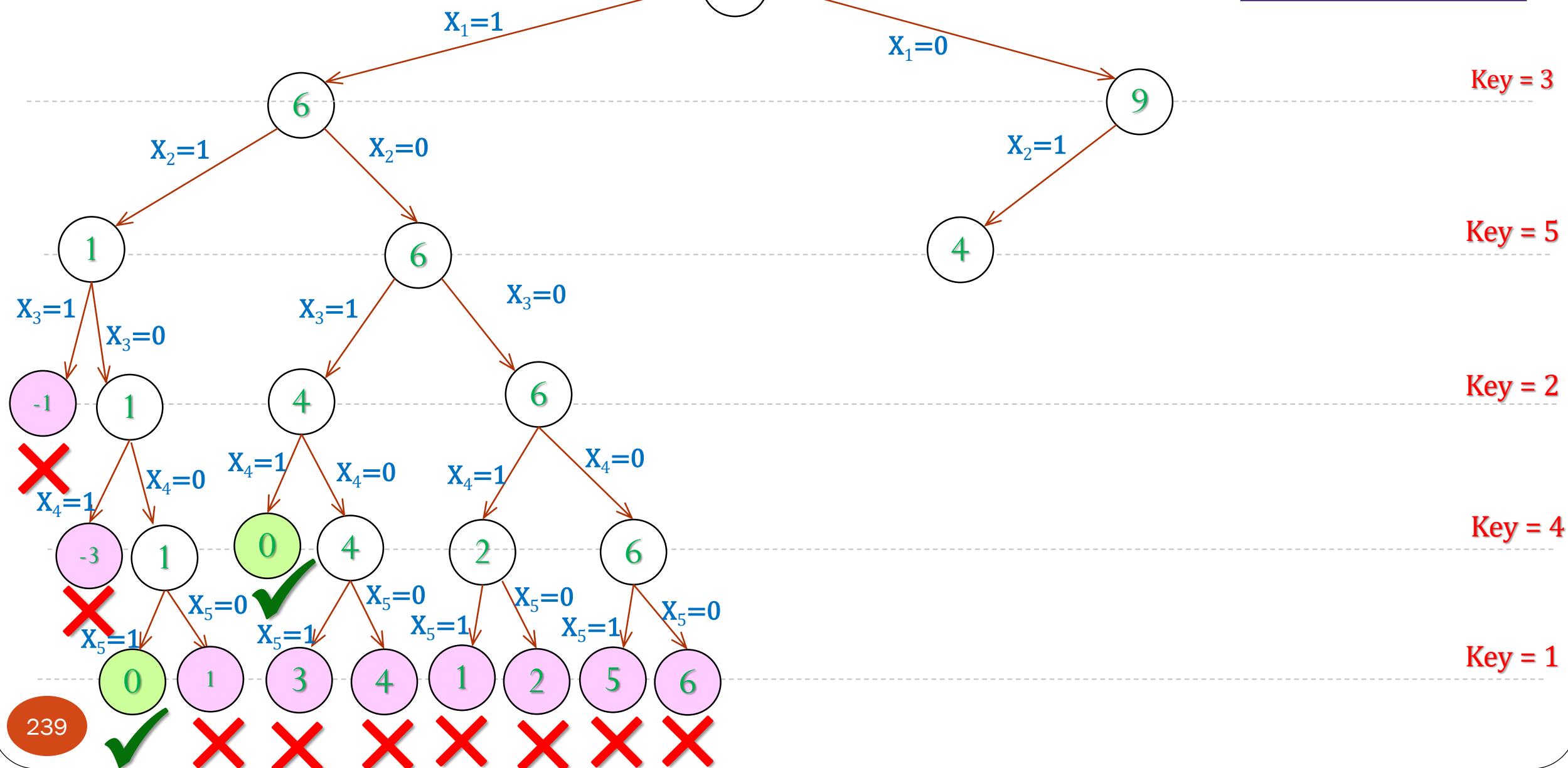
1	2	3	4	5
0	1	0	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



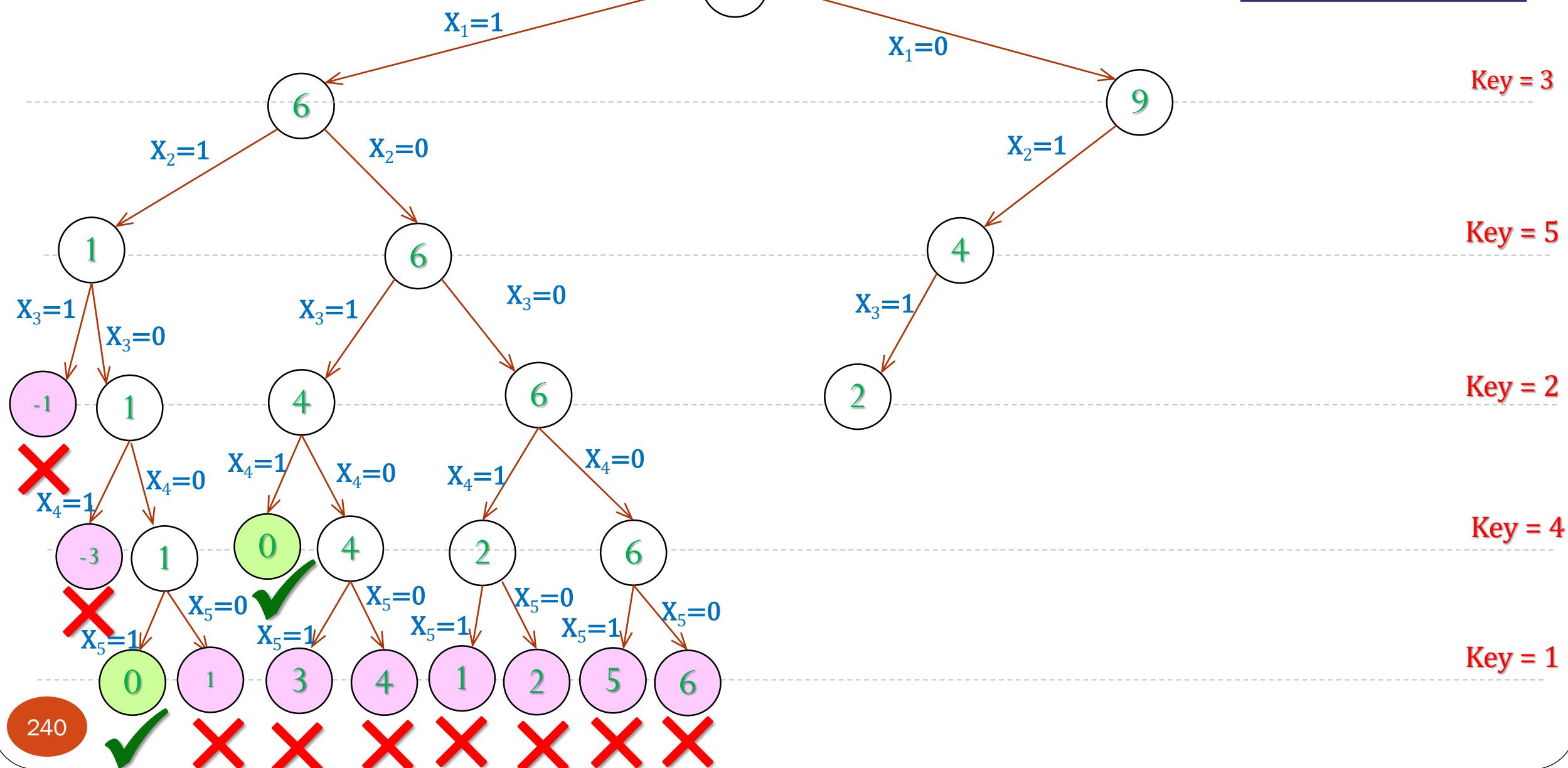
X	1	2	3	4	5
0	0	1	1	0	0

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

Start

Solutions

1	1	0	0	1
1	0	1	1	0



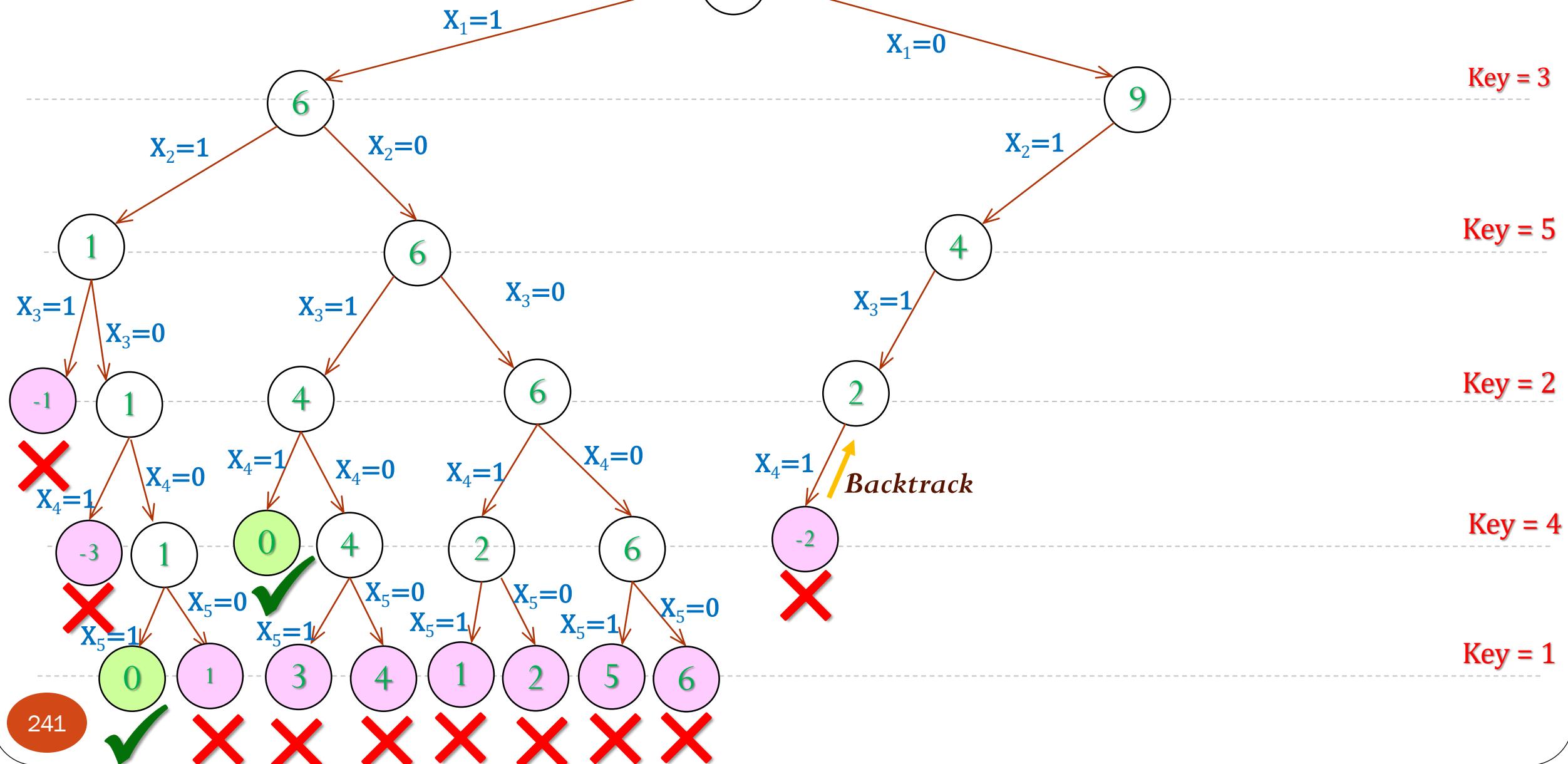
	1	2	3	4	5
X	1	1	1	1	0

	1	2	3	4	5
Key	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



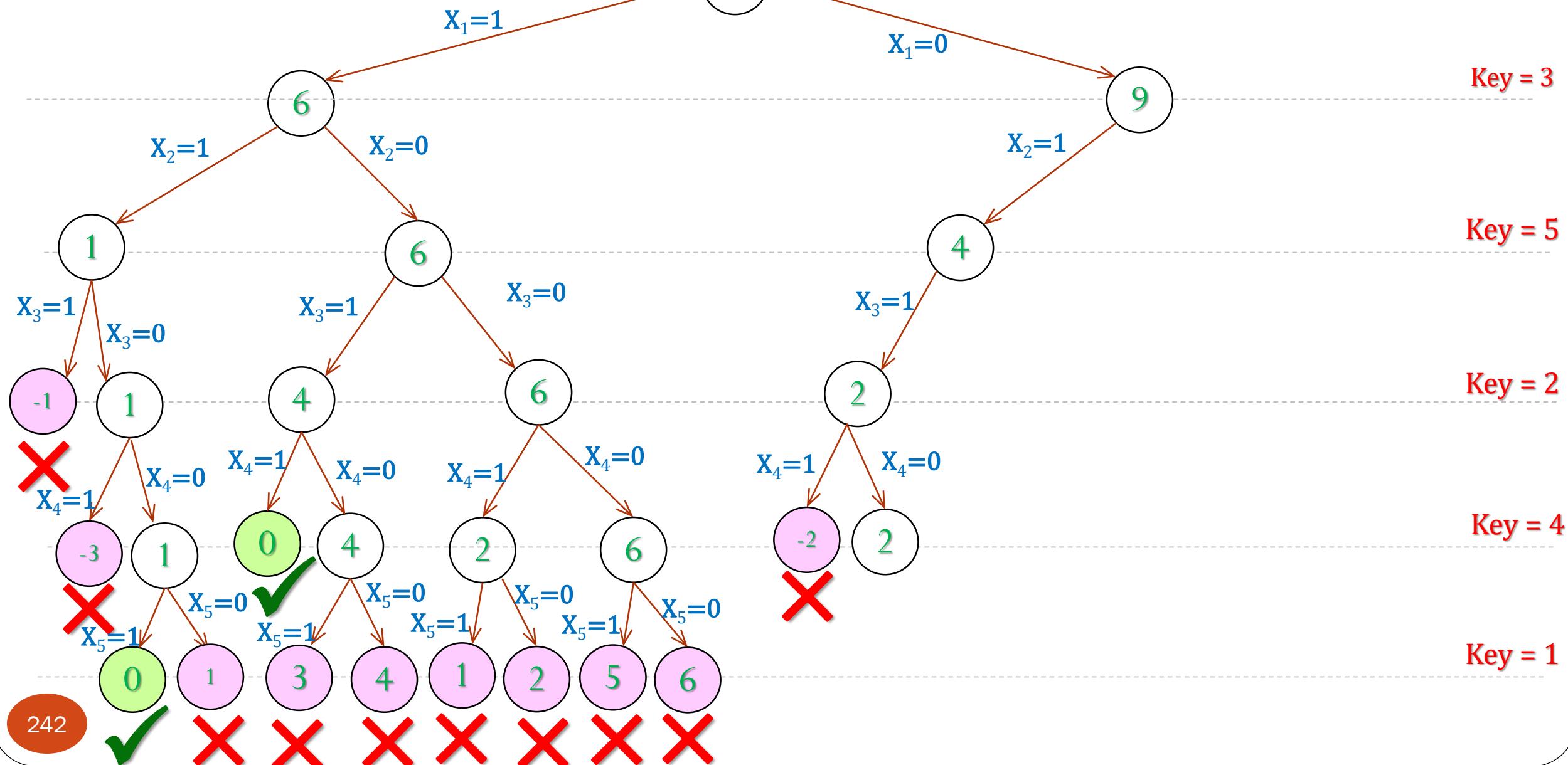
1	2	3	4	5
0	1	1	0	0

1	2	3	4	5
3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



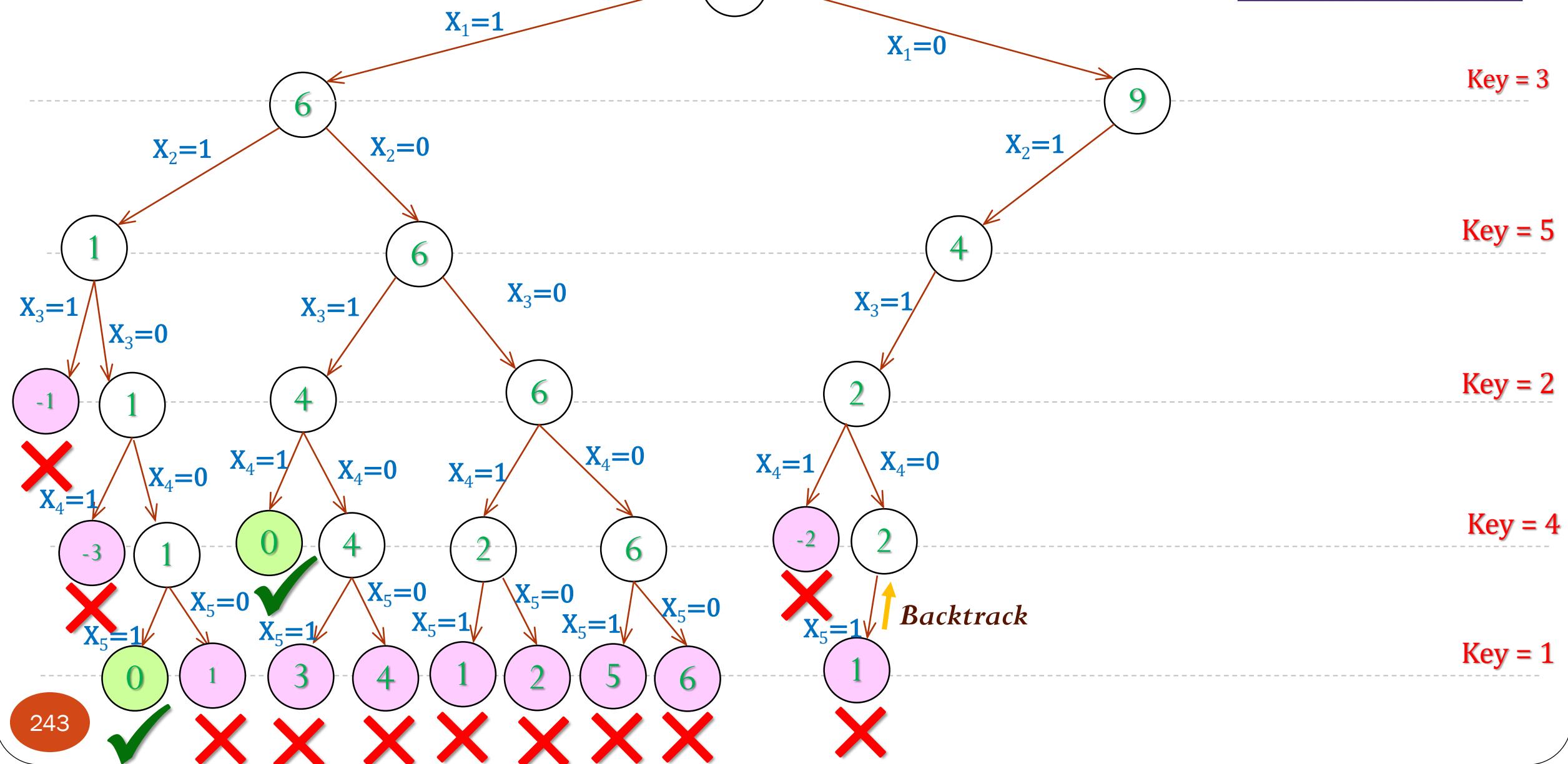
X	1	2	3	4	5
0	1	1	0	1	

Key	1	2	3	4	5
3	5	2	4	1	

Start

Solutions

1	1	0	0	1
1	0	1	1	0



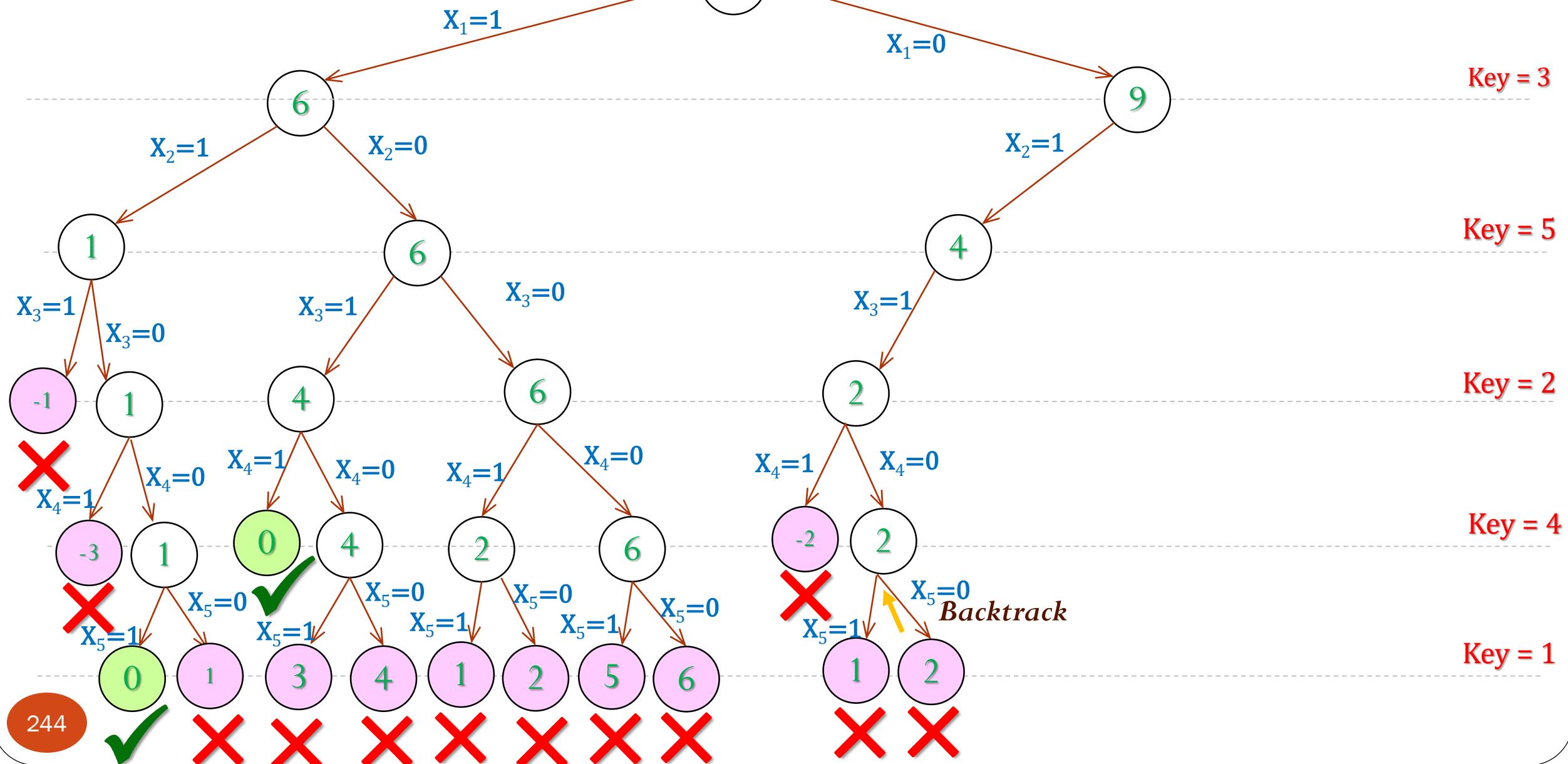
	1	2	3	4	5
X	0	1	1	0	0

	1	2	3	4	5
Key	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



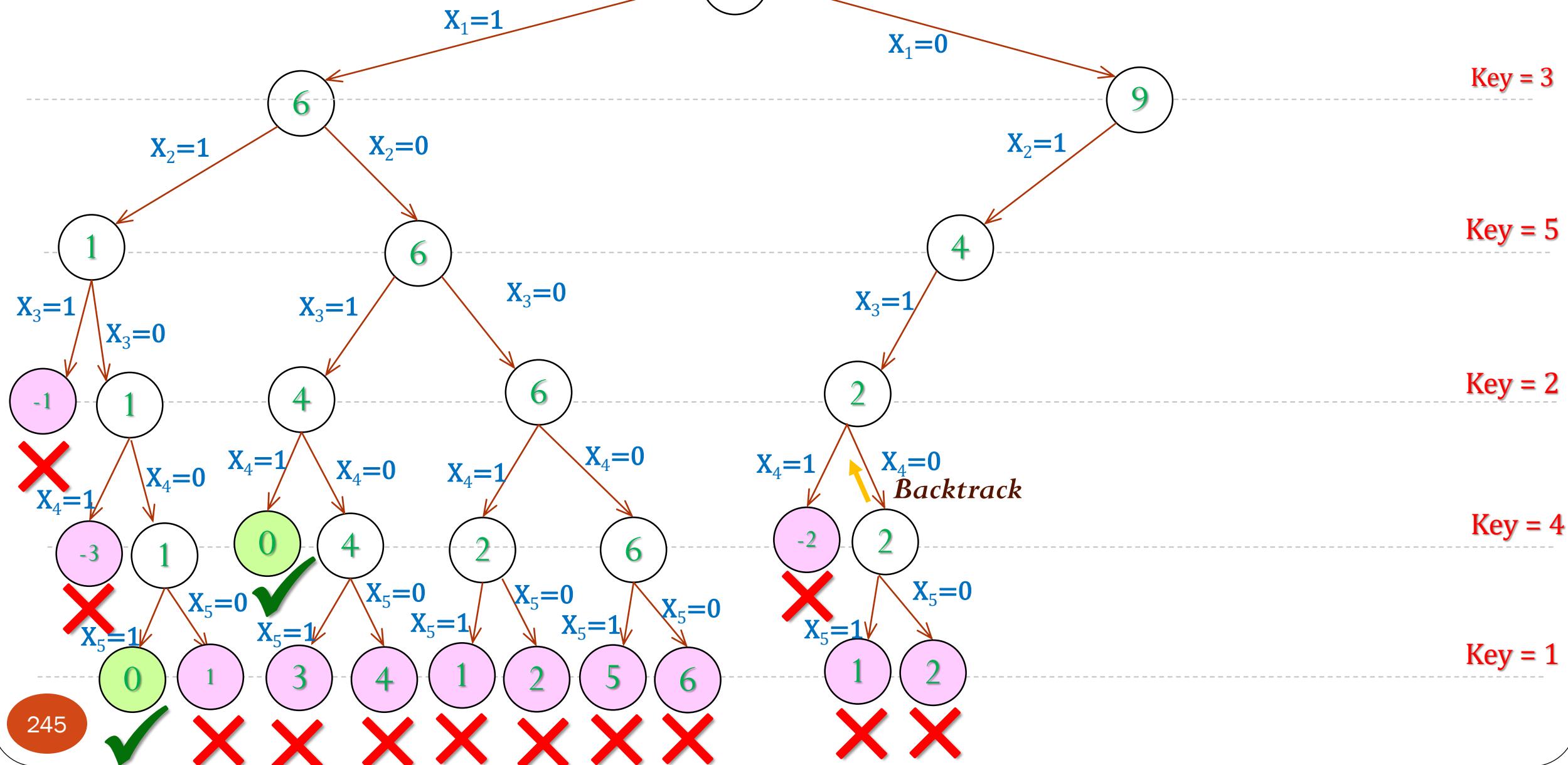
1	2	3	4	5	
X	1	0	1	0	0

1	2	3	4	5	
Key	3	5	2	4	1

Start

Solutions

1	1	0	0	1
1	0	1	1	0



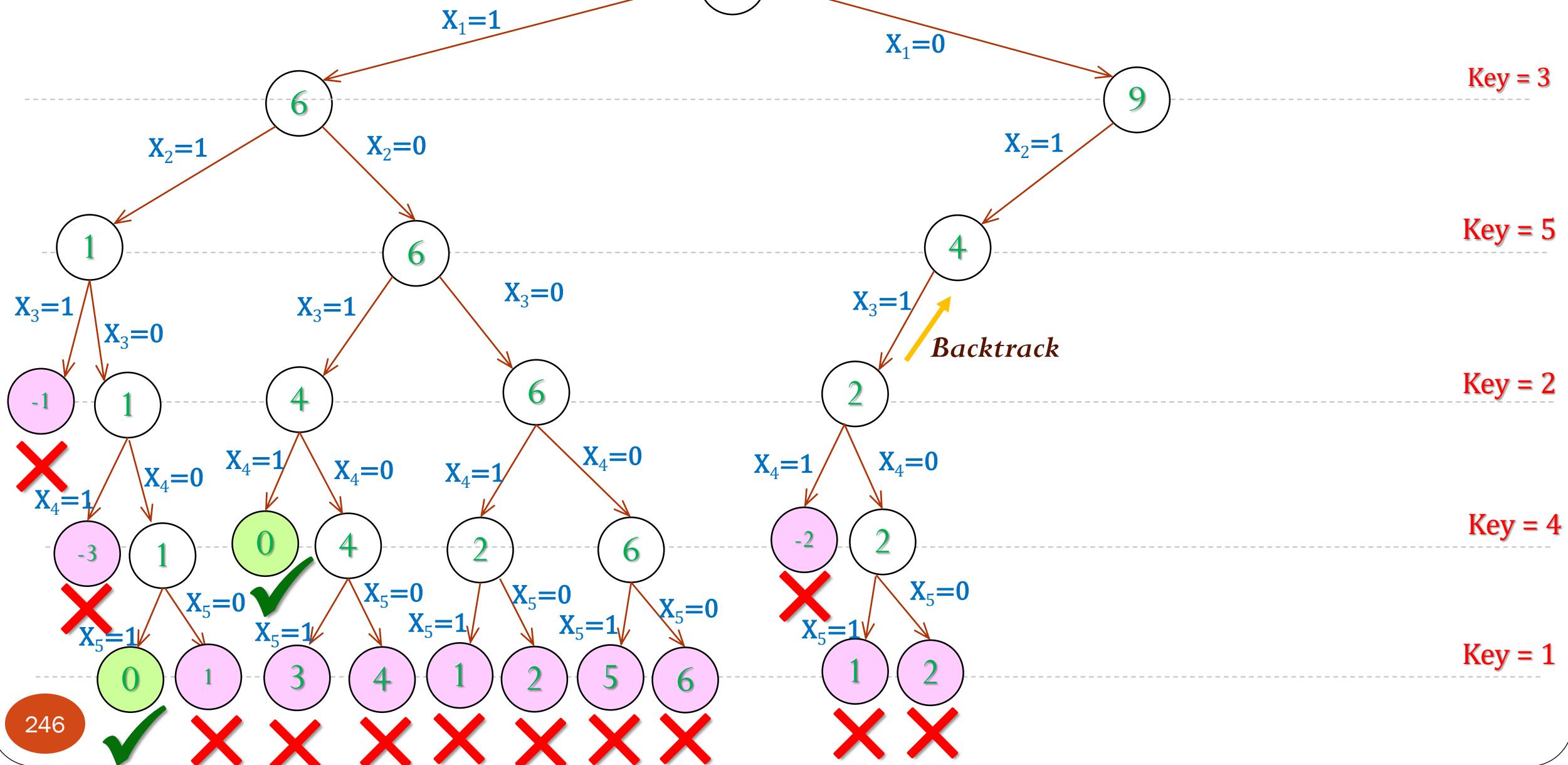
1	2	3	4	5
0	1	1	0	0

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

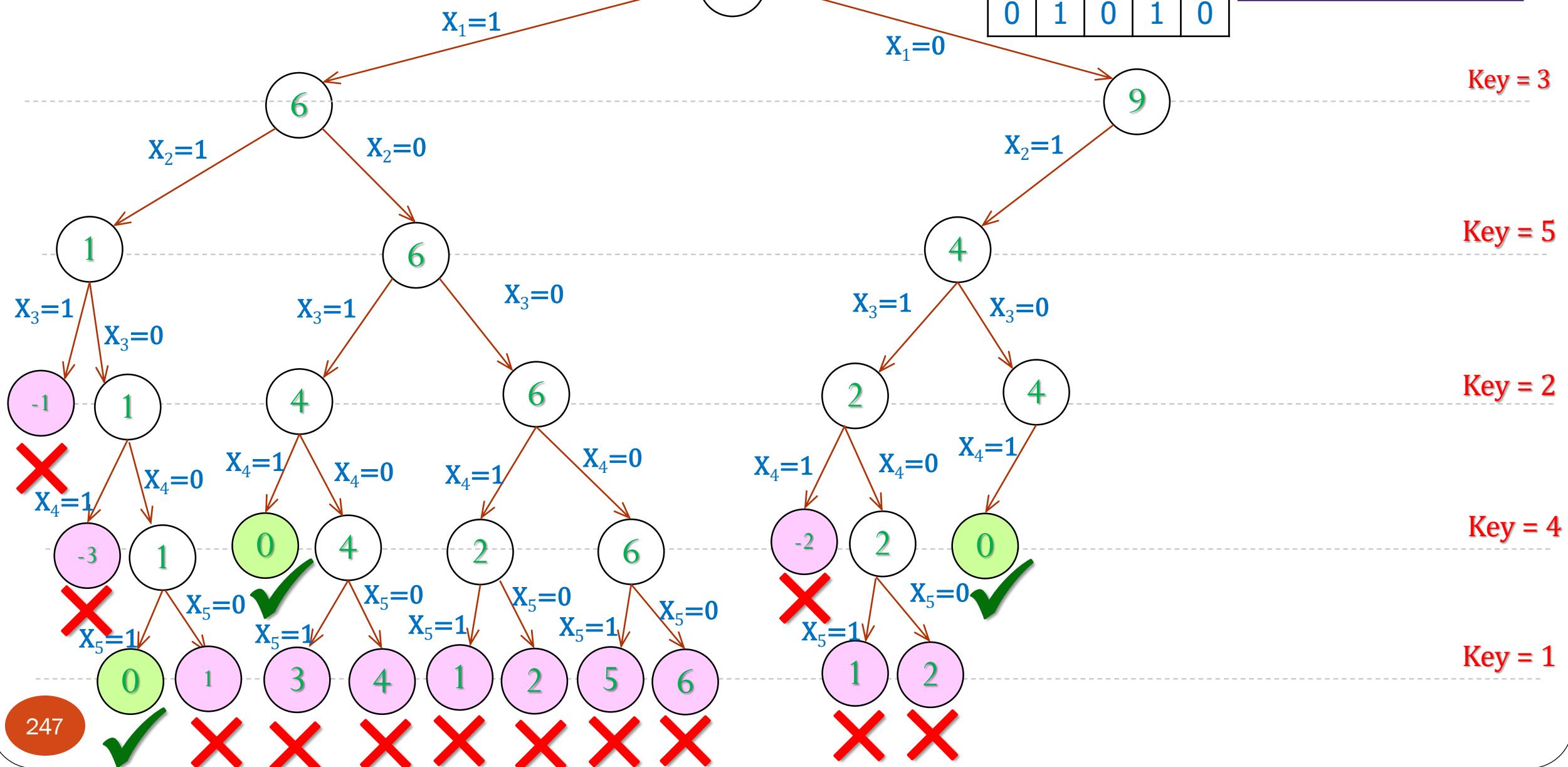
Key

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



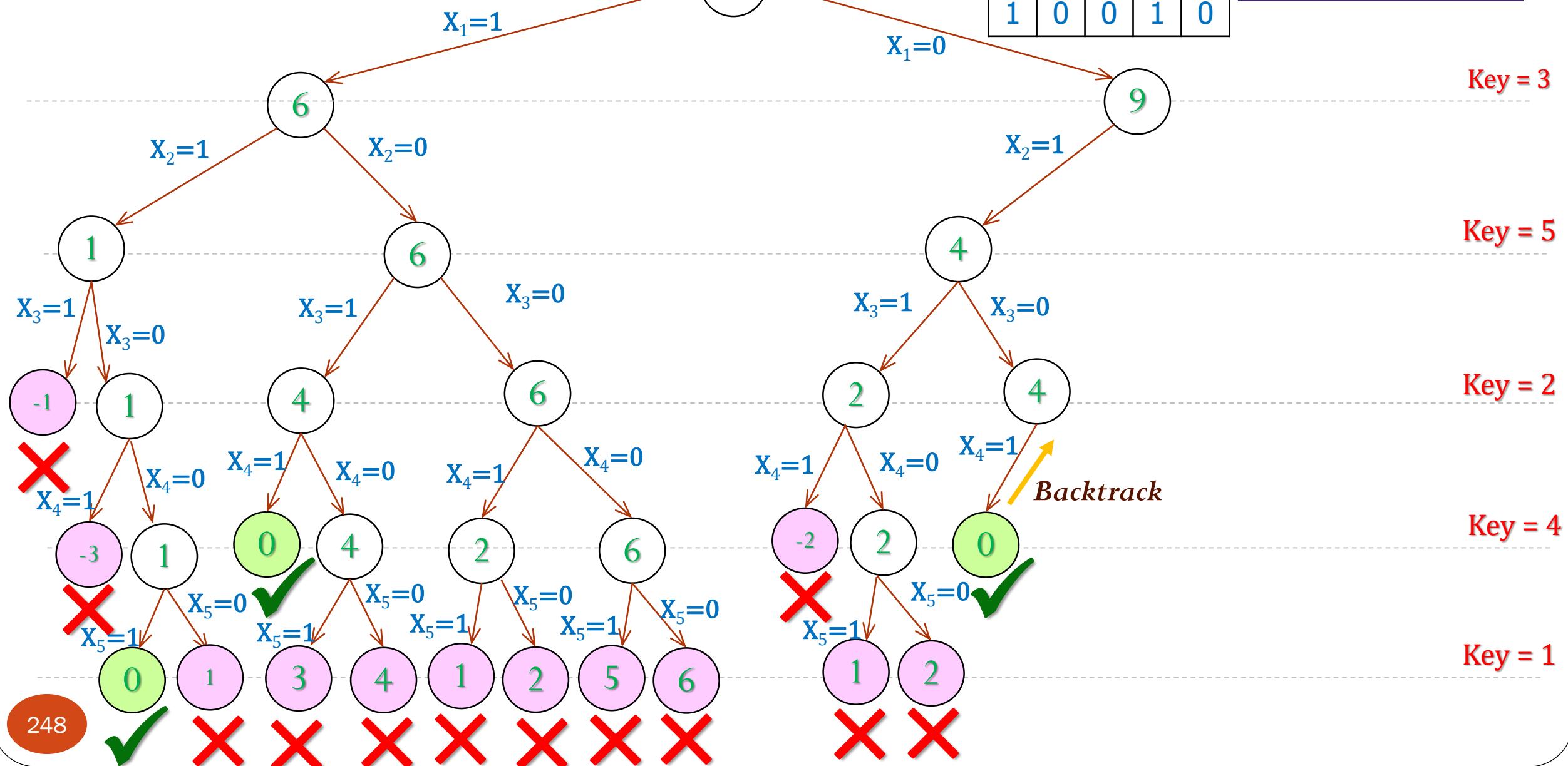
1	2	3	4	5
0	1	0	0	0

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



X

1	2	3	4	5
0	1	0	0	0

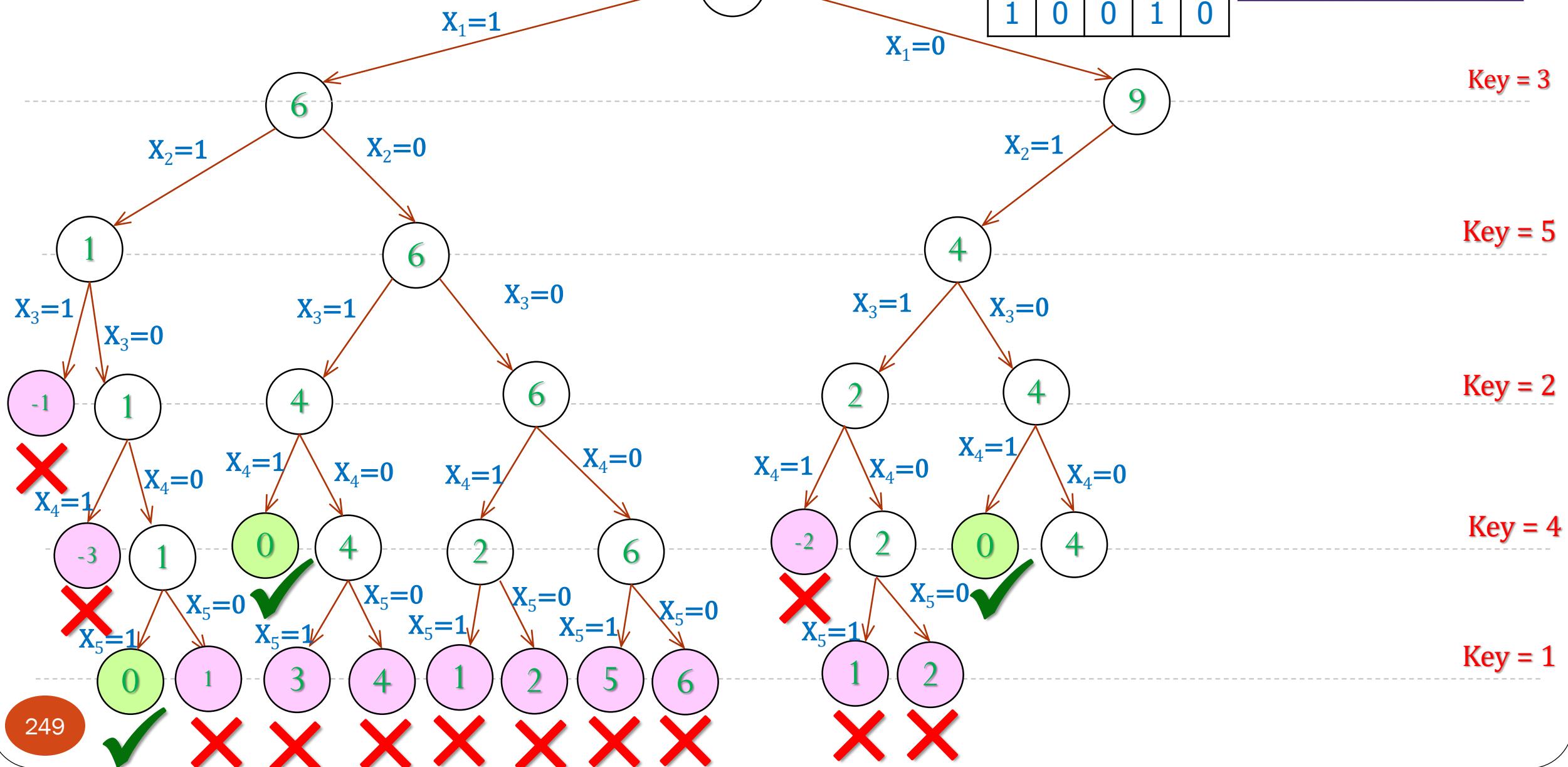
Key

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



X

1	2	3	4	5
0	1	0	0	1

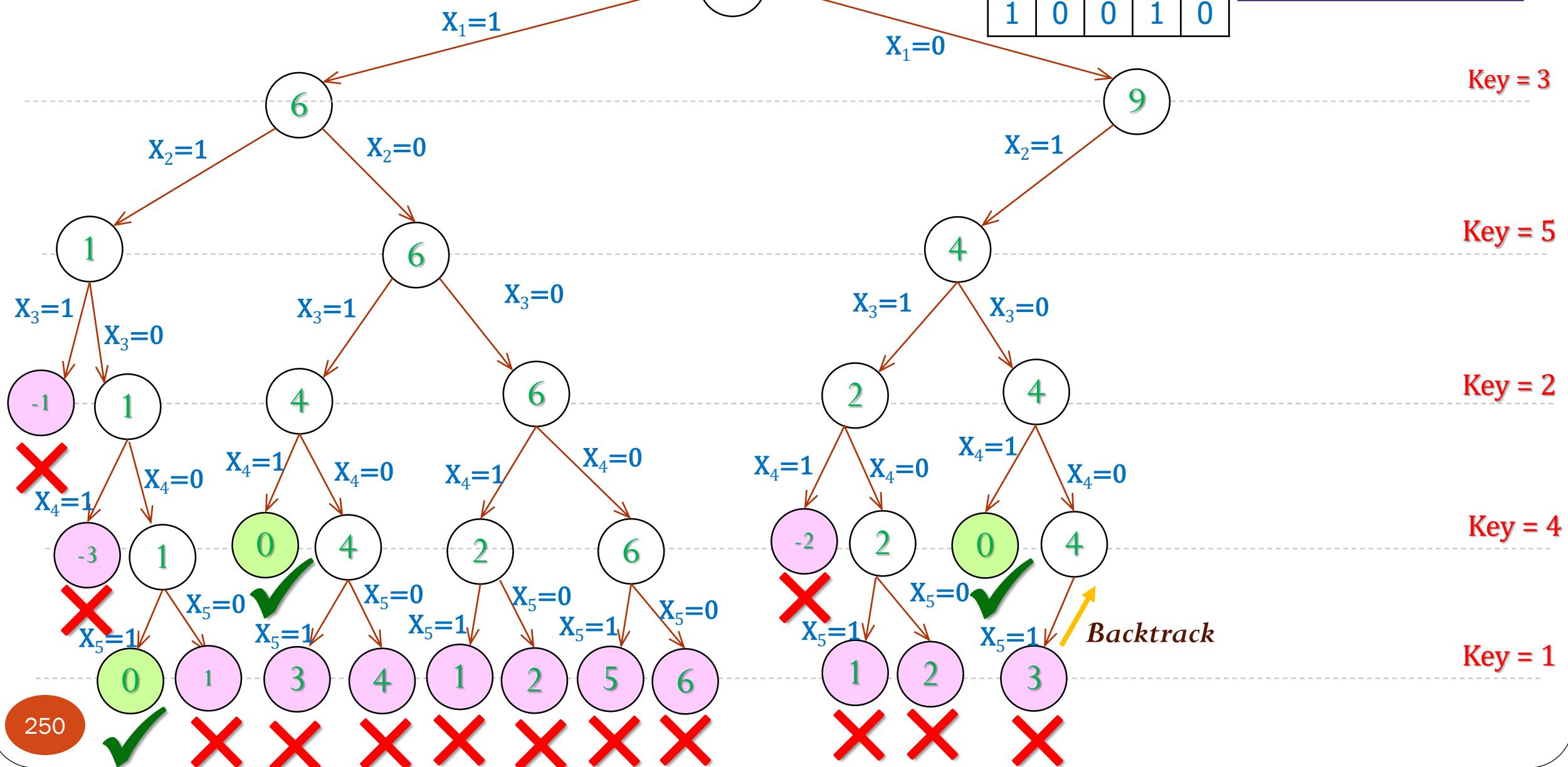
Key

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



X 1 2 3 4 5

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

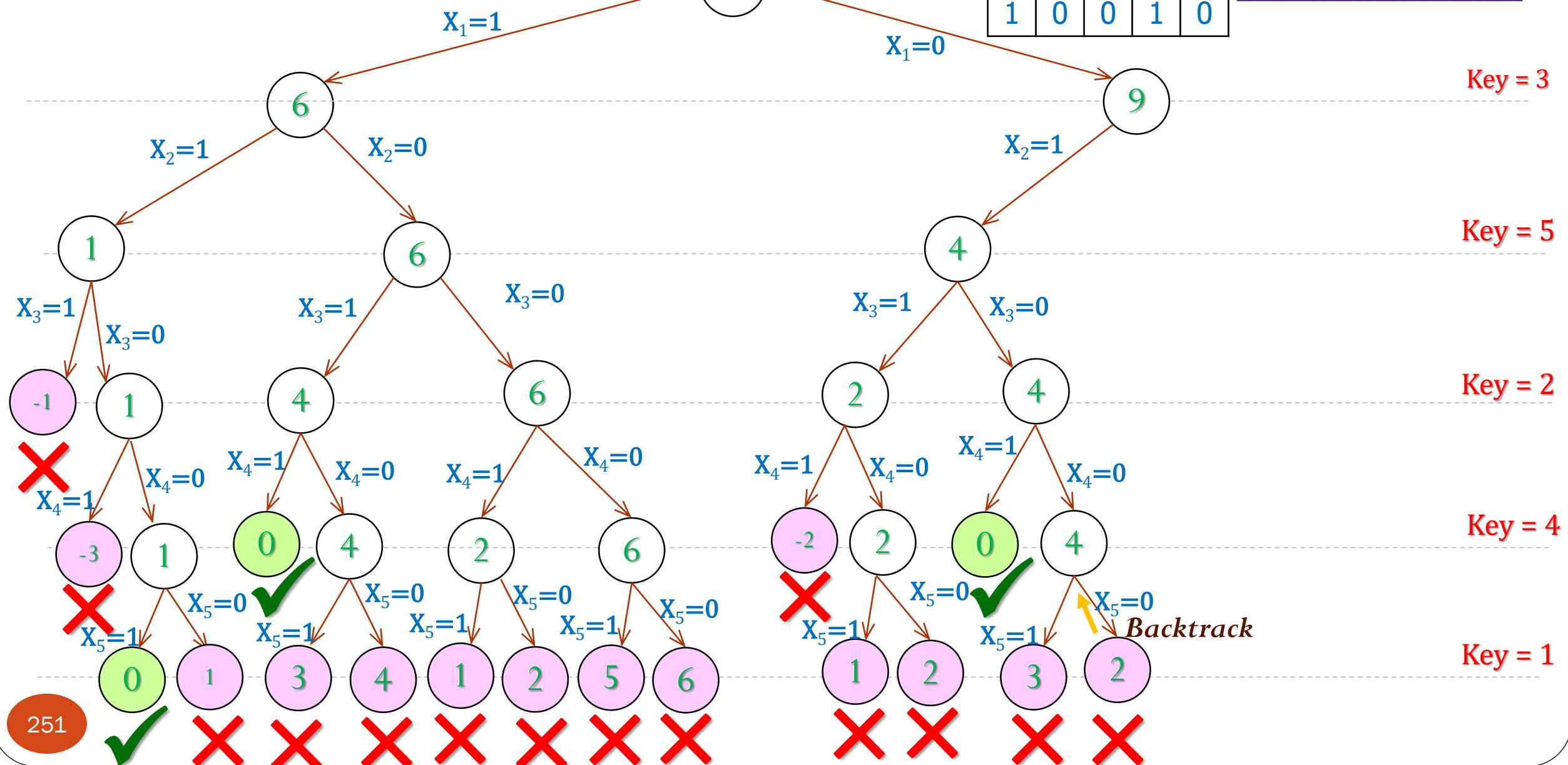
Key 1 2 3 4 5

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

Start

Solutions

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



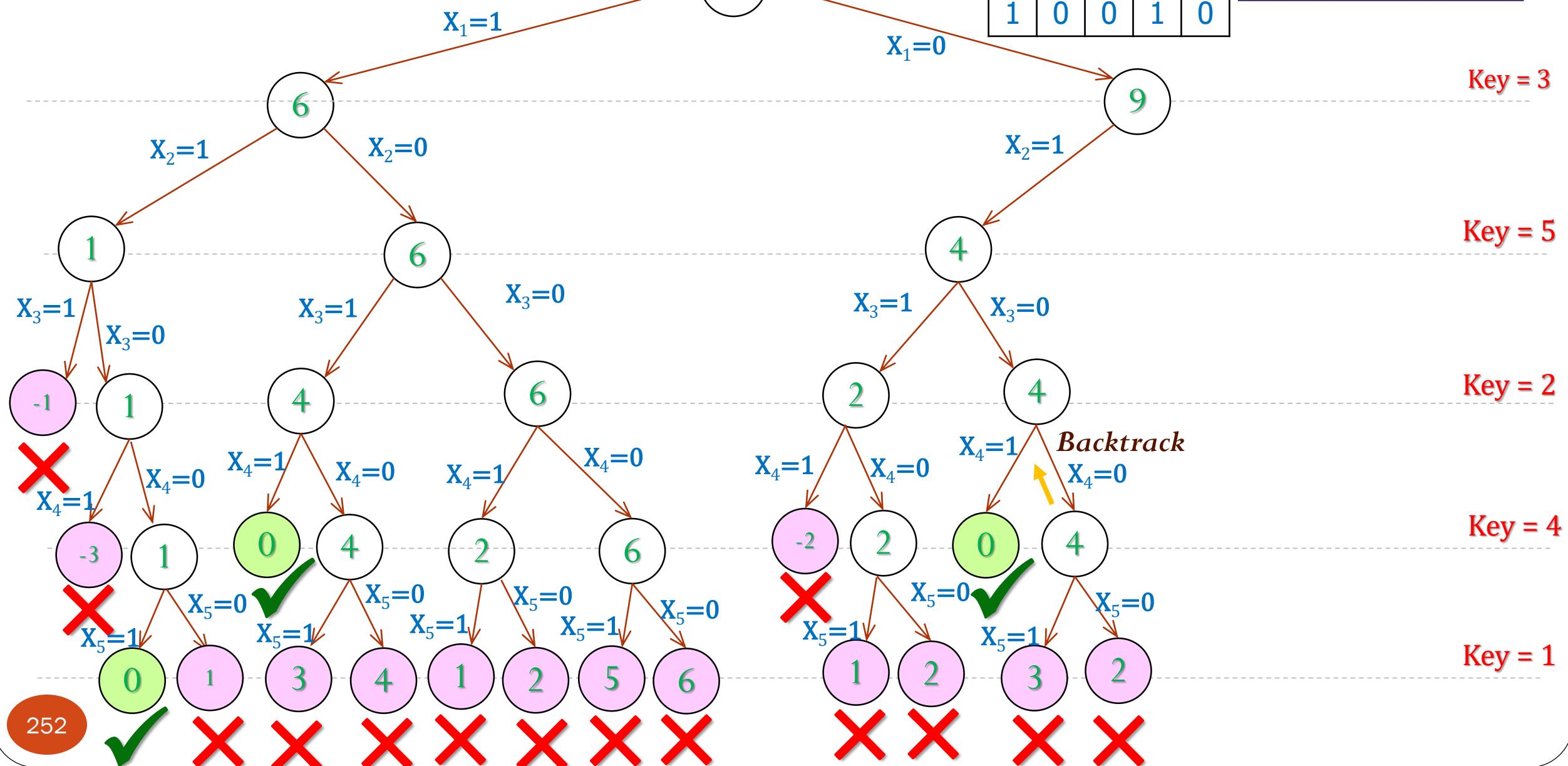
X	1	2	3	4	5
0	0	1	0	0	0

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



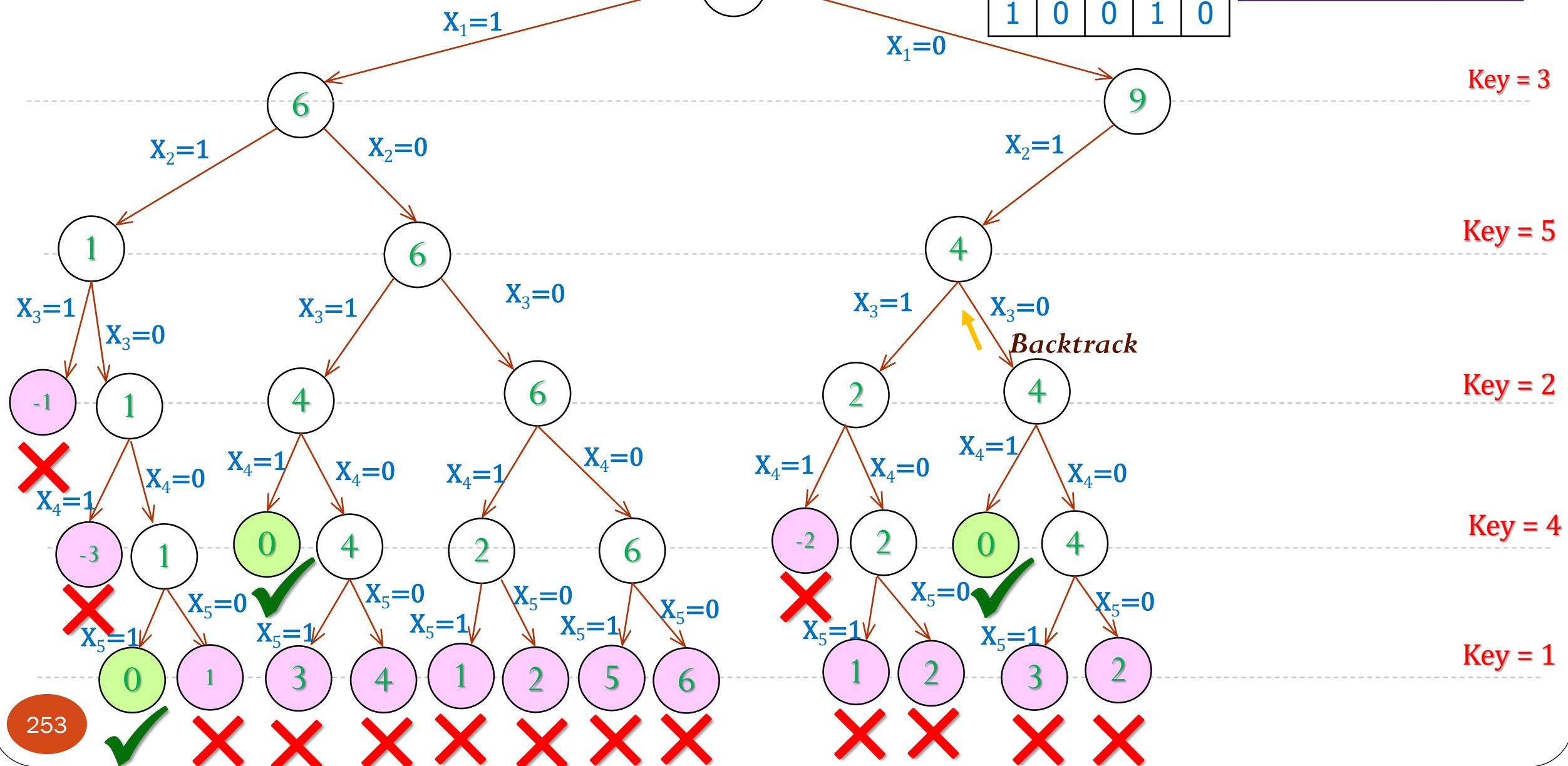
1	2	3	4	5
0	1	0	0	0

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



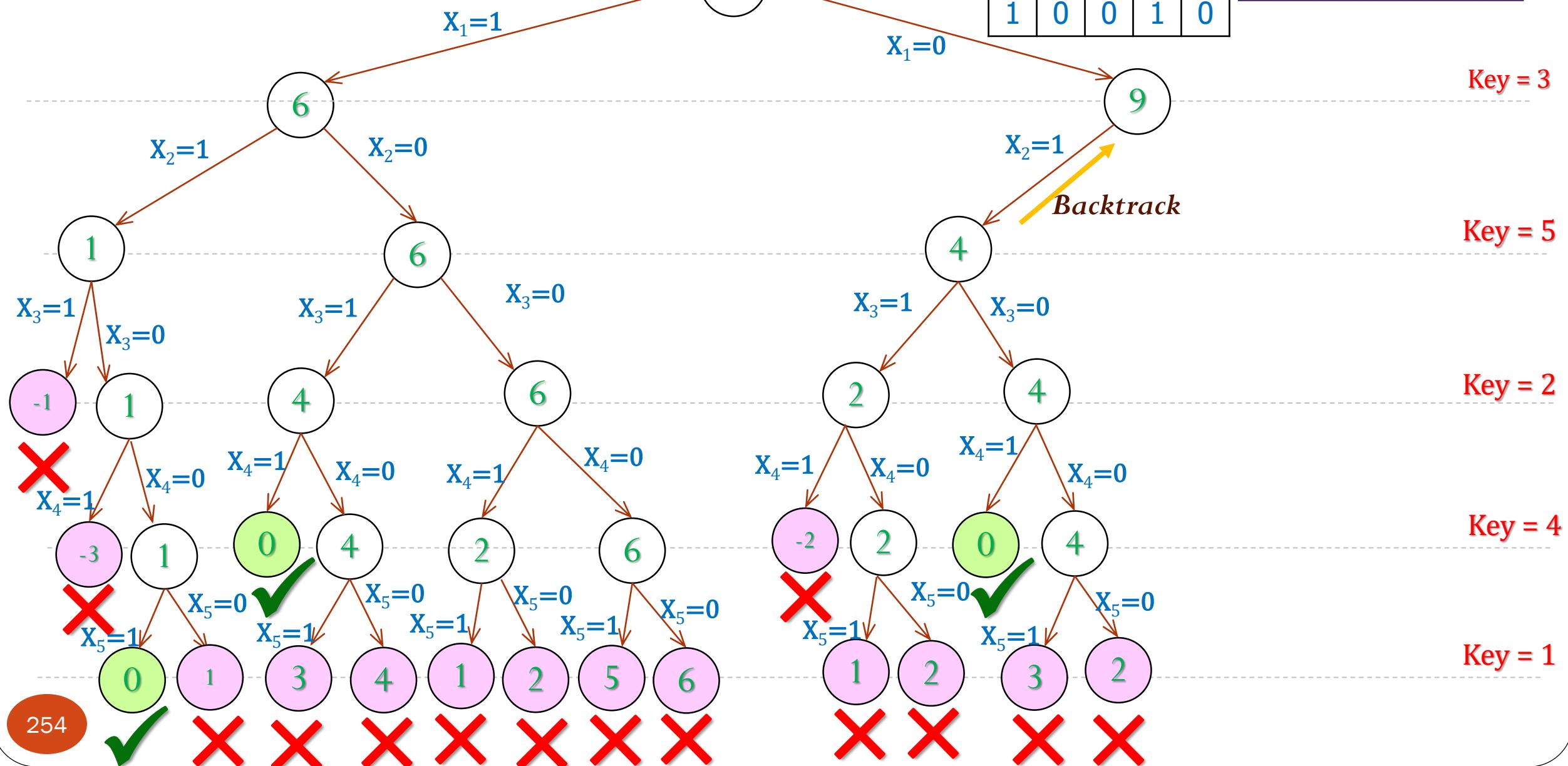
1	2	3	4	5
0	1	0	0	0

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



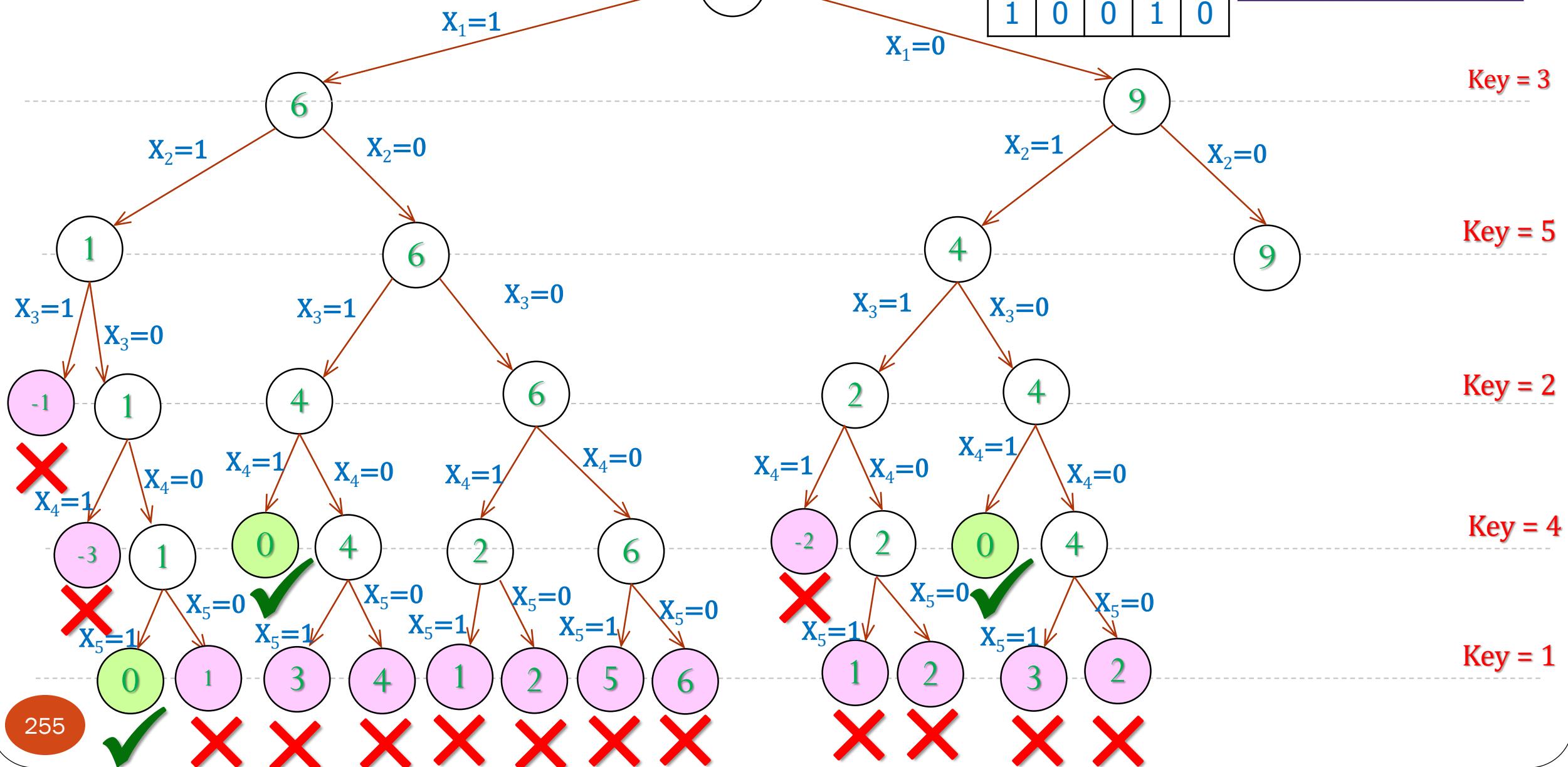
1	2	3	4	5
0	0	0	0	0

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 1 0 0

Key 1 2 3 4 5
 3 5 2 4 1

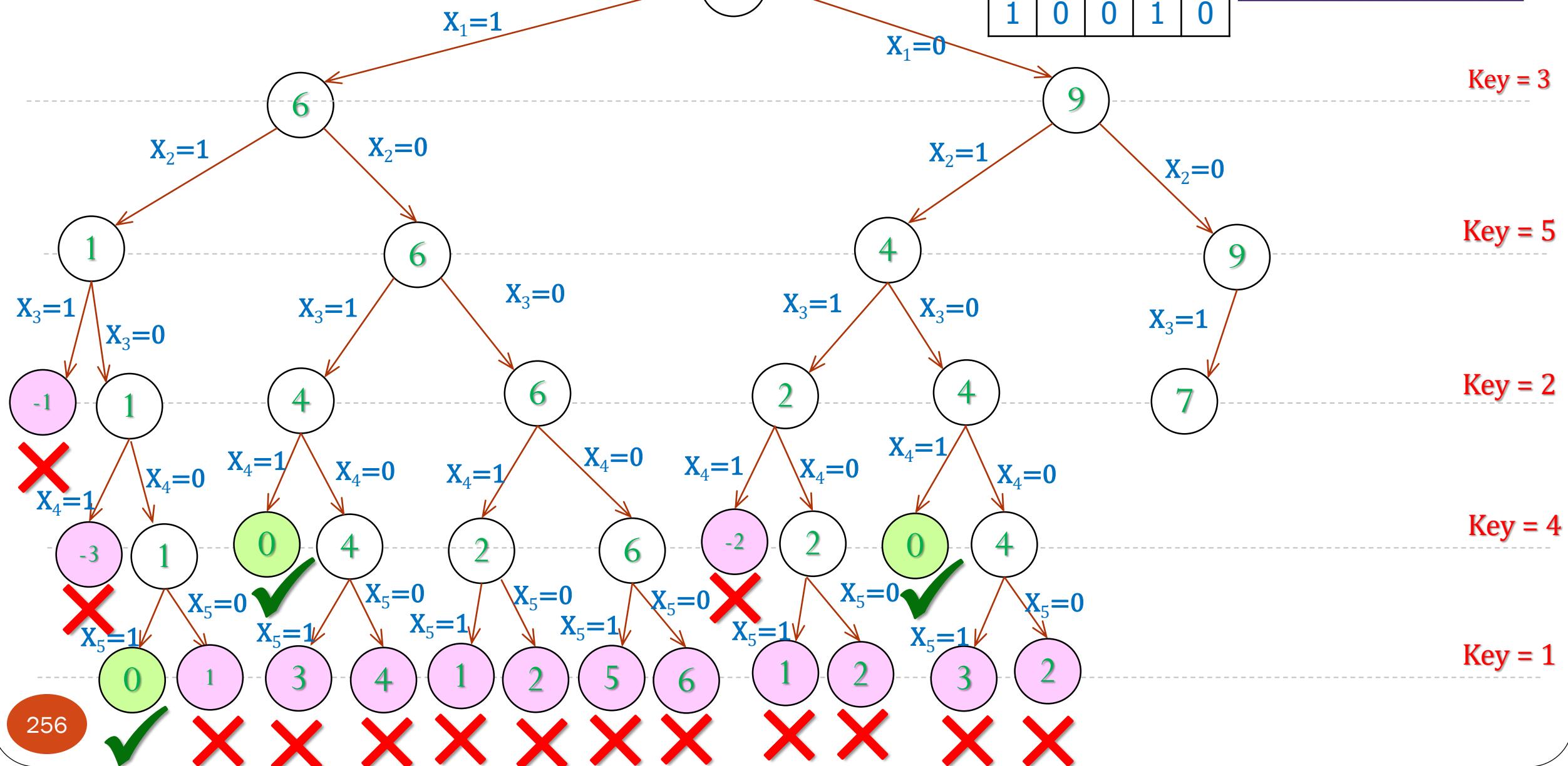
Start

Solutions

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



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
0	0	1	1	0

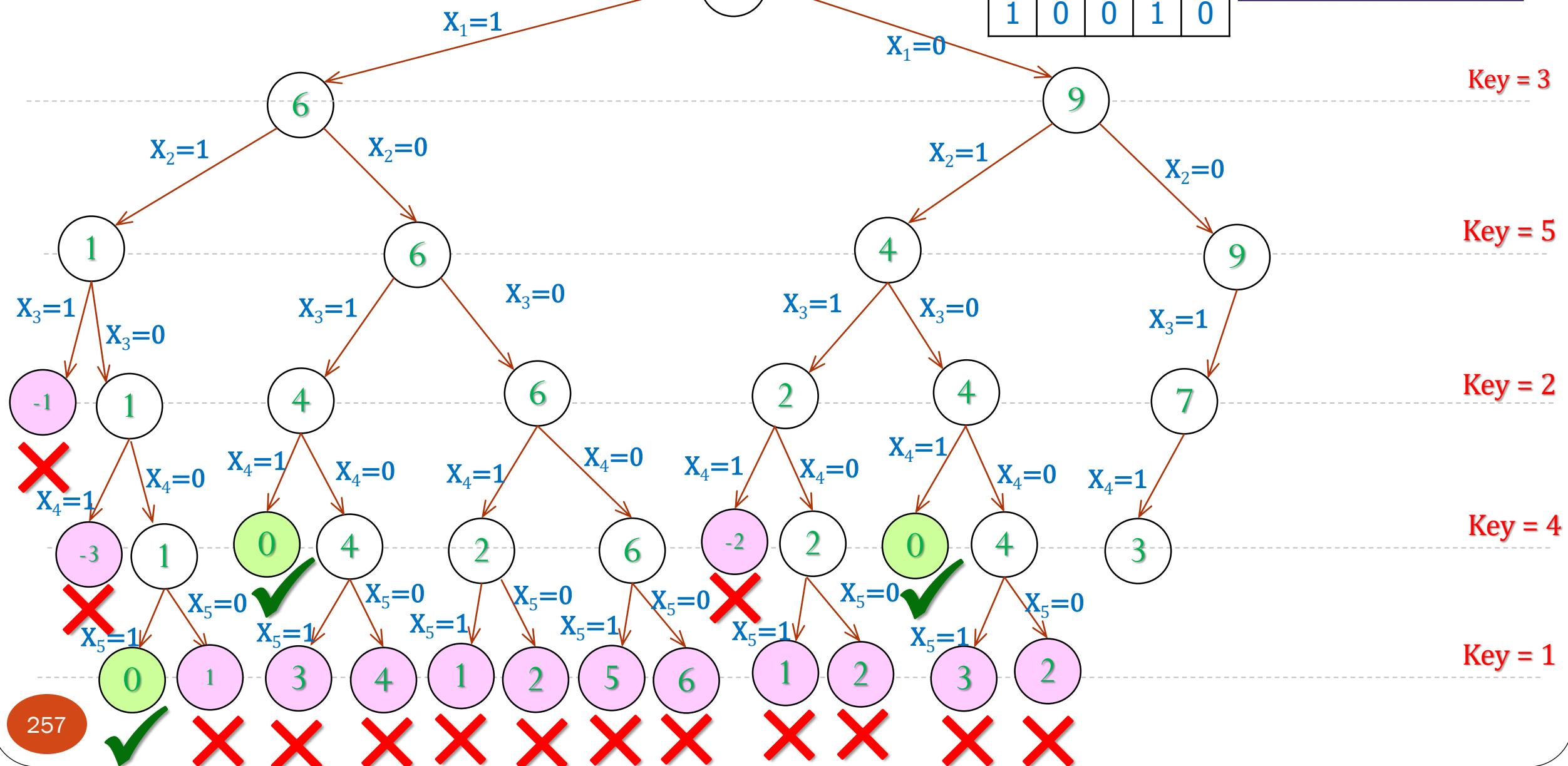
Key

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

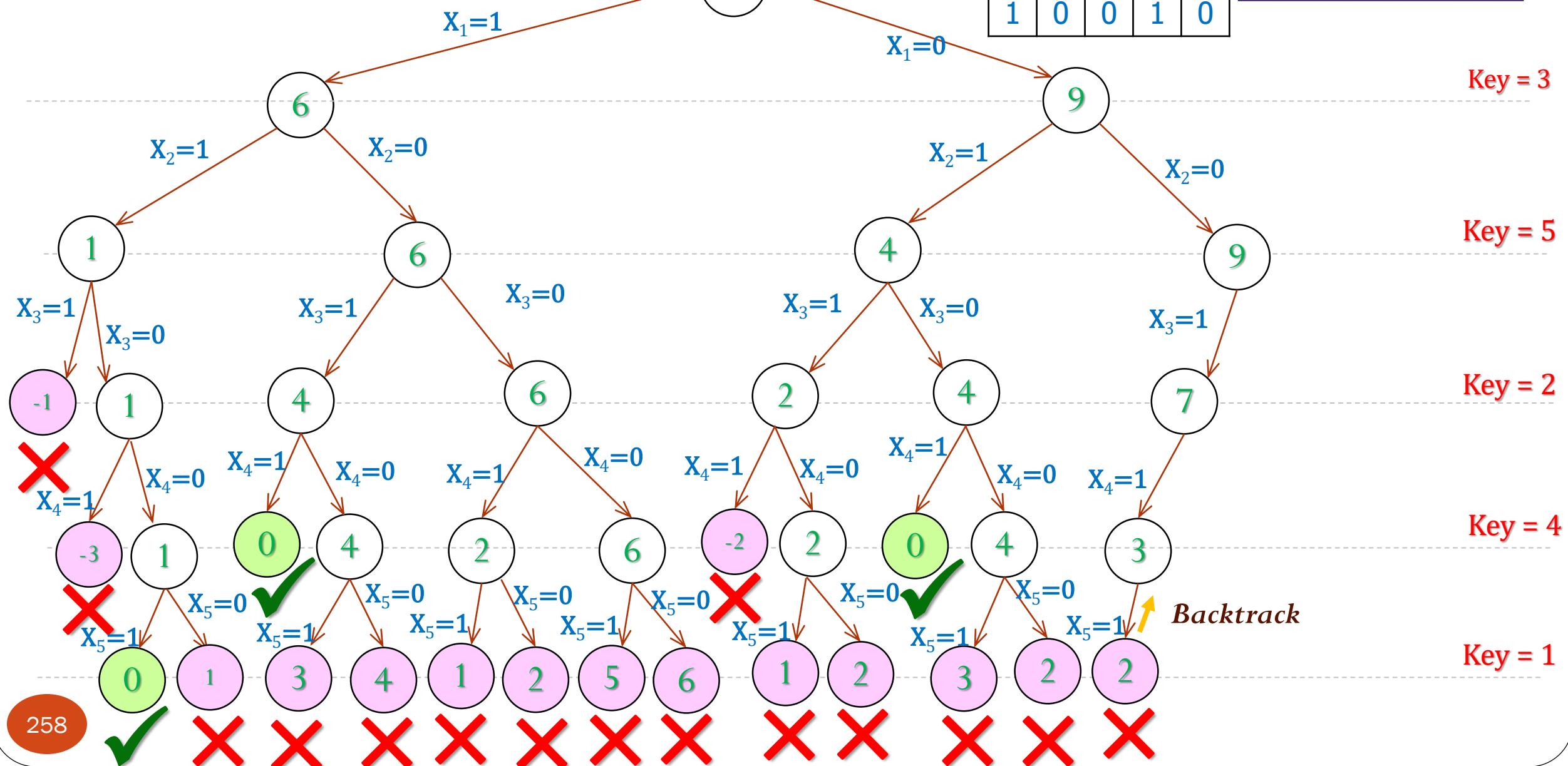
Key

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

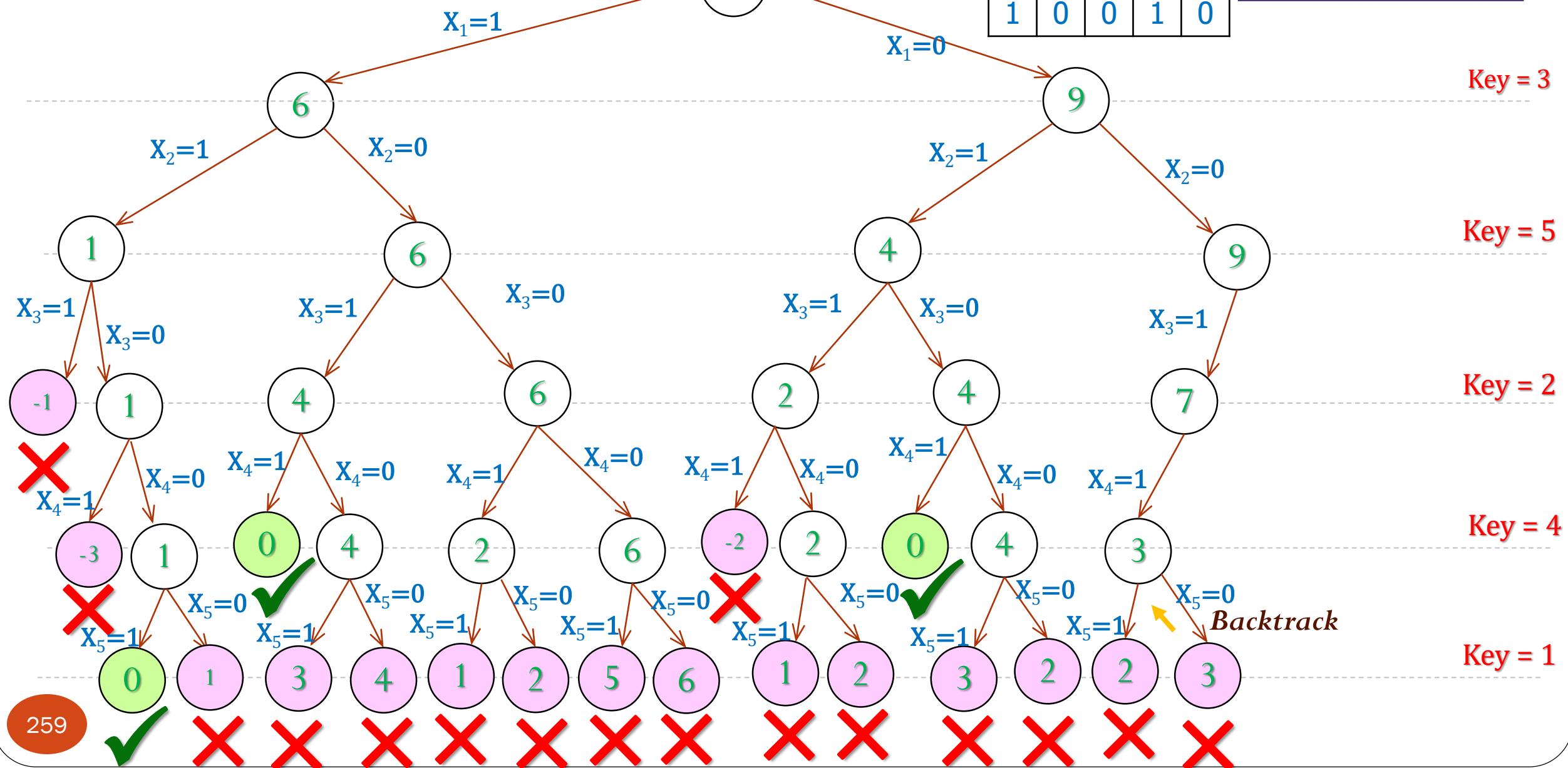
Key

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



X

1	2	3	4	5
0	0	1	1	0

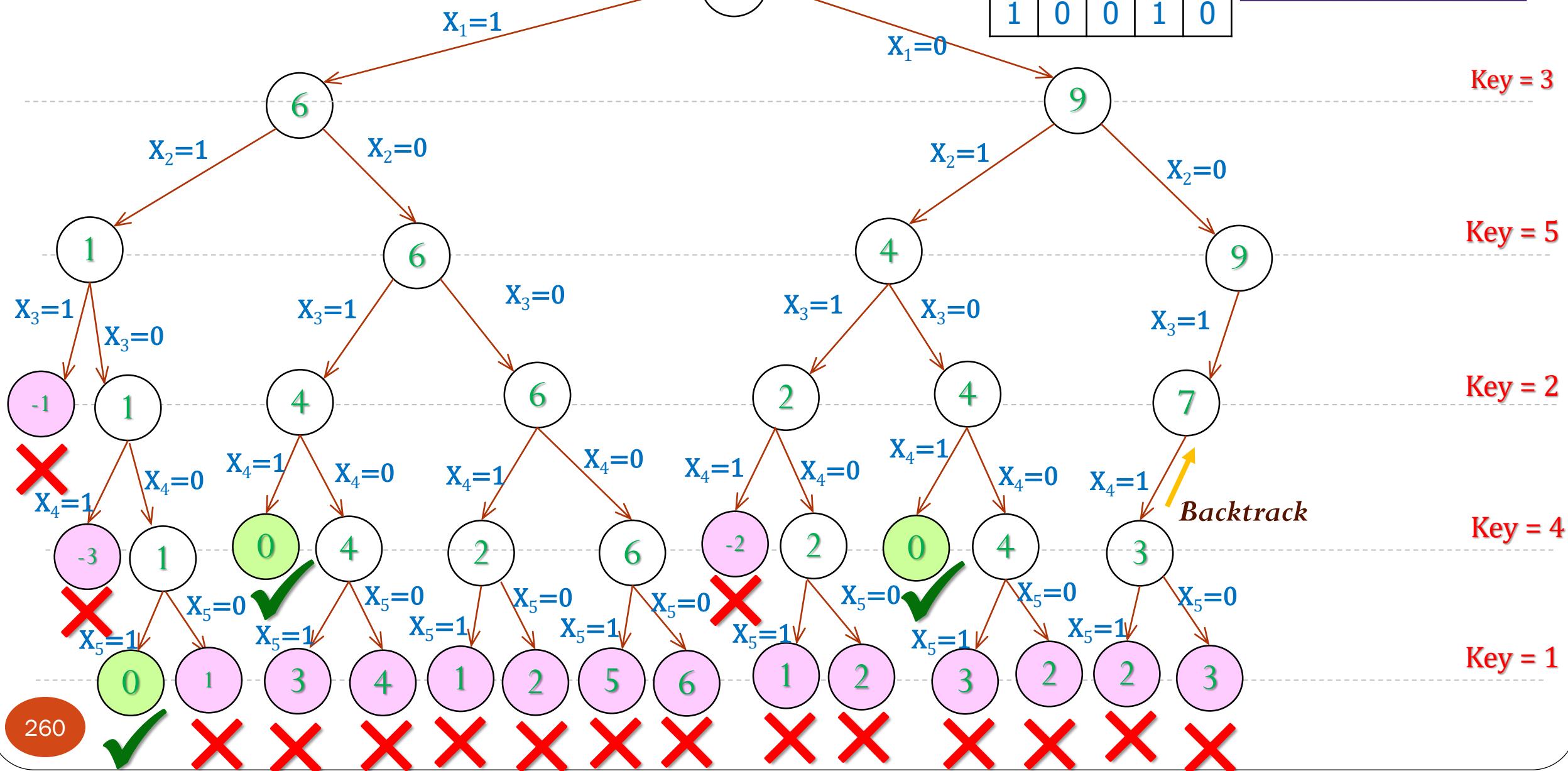
Key

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



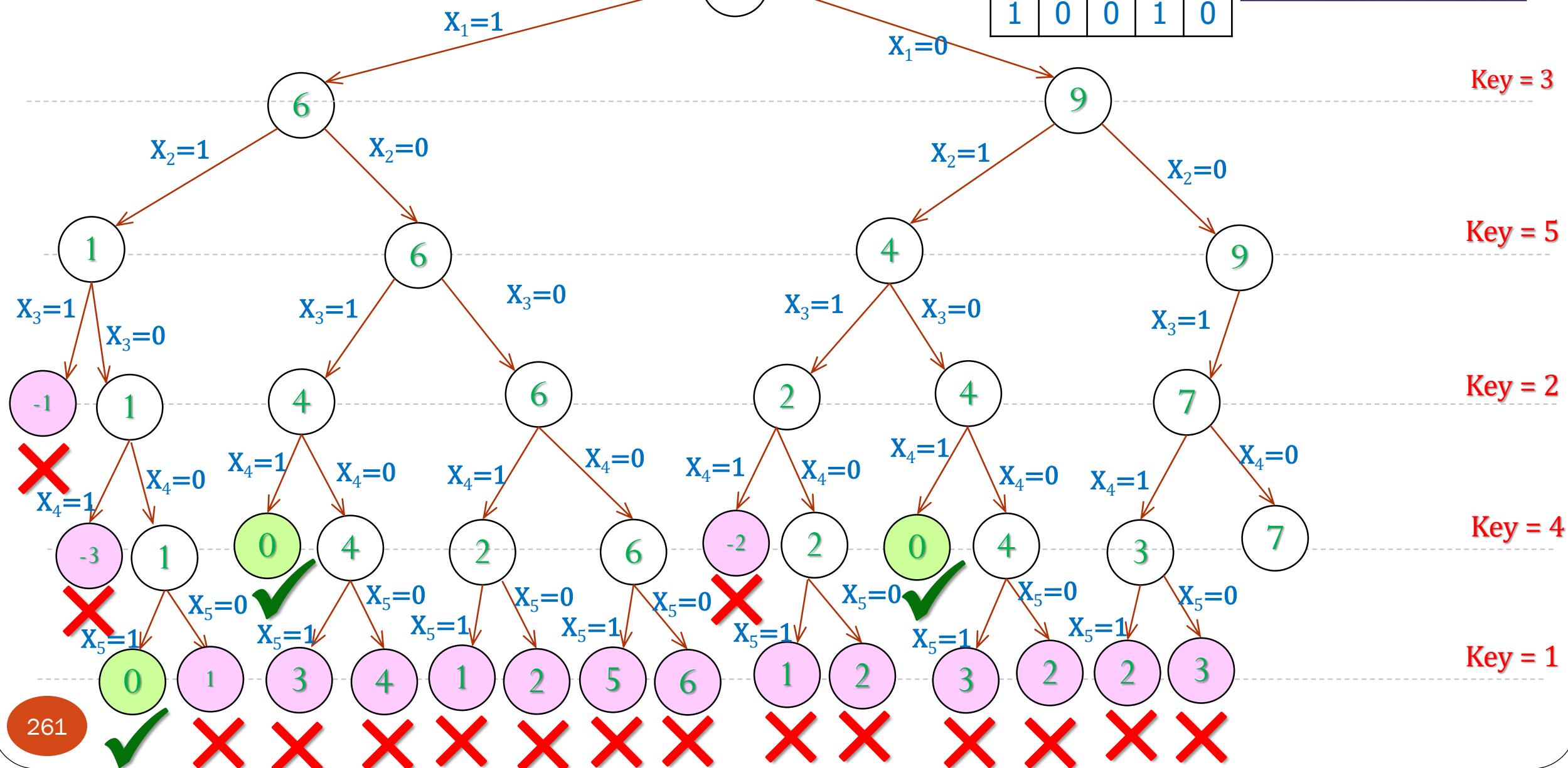
1	2	3	4	5	
X	0	0	1	0	0

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



X 1 2 3 4 5
 0 0 1 0 1

Key 1 2 3 4 5
 3 5 2 4 1

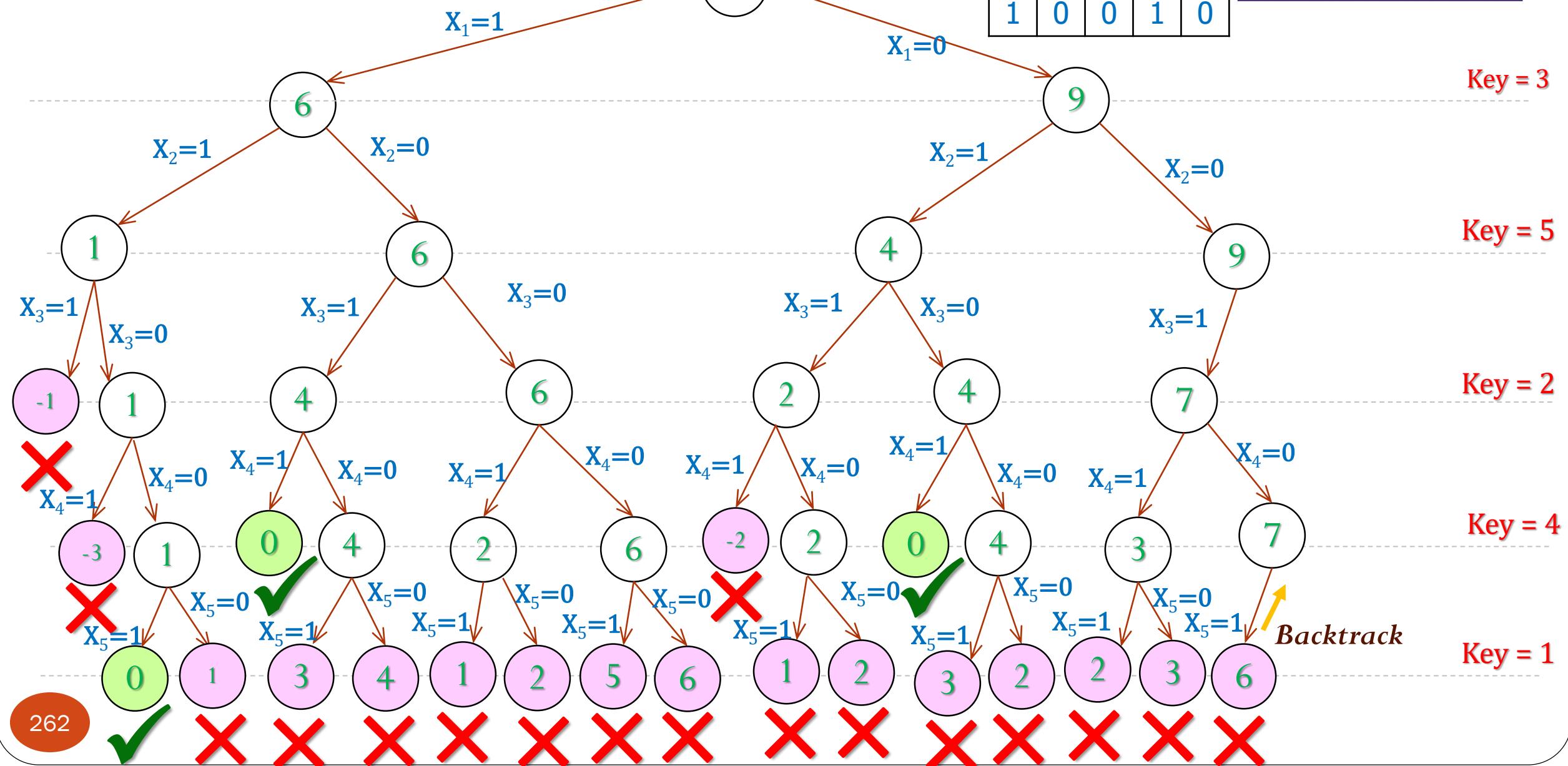
Start

Solutions

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



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
0	0	1	0	0

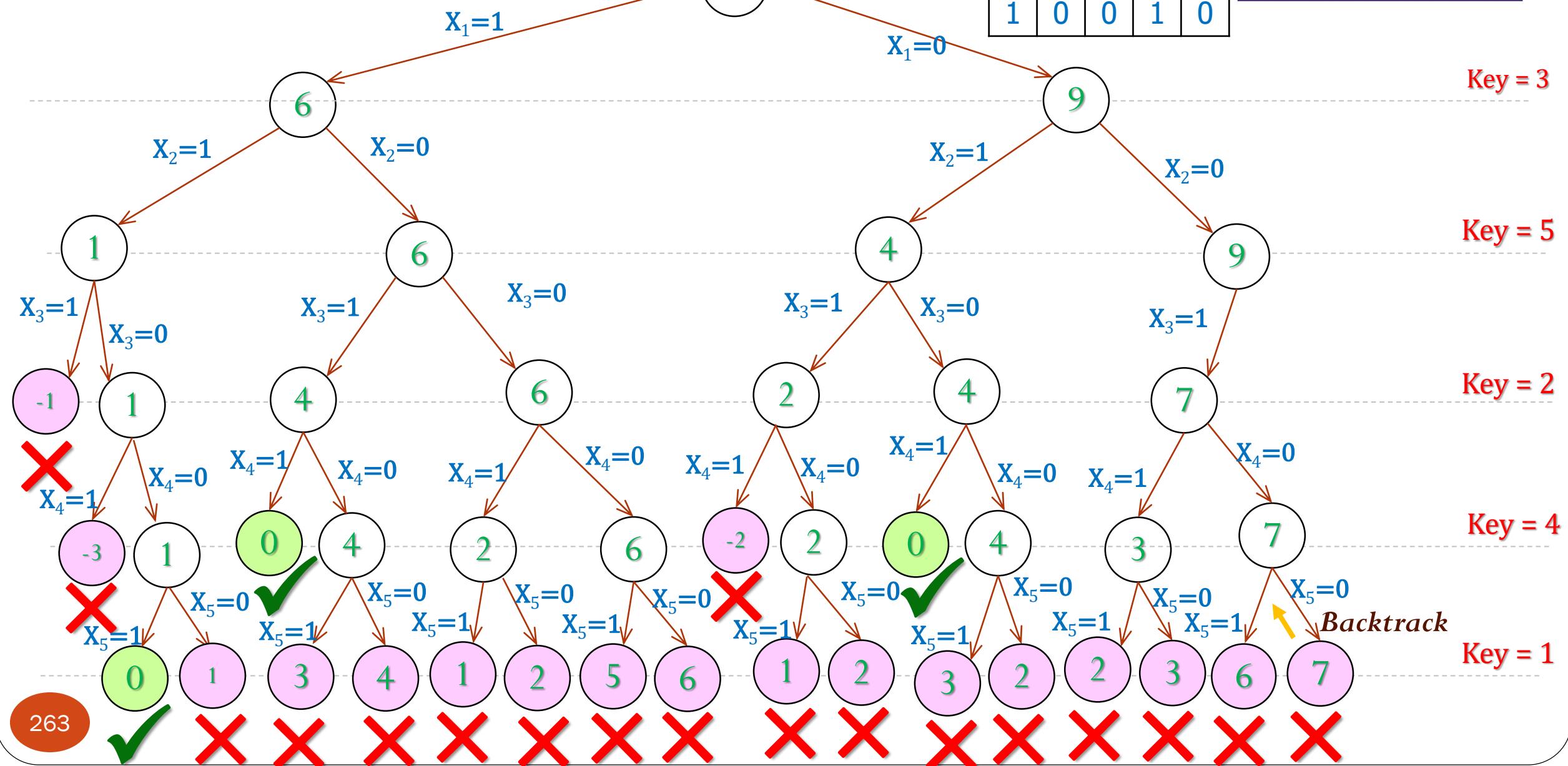
Key

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



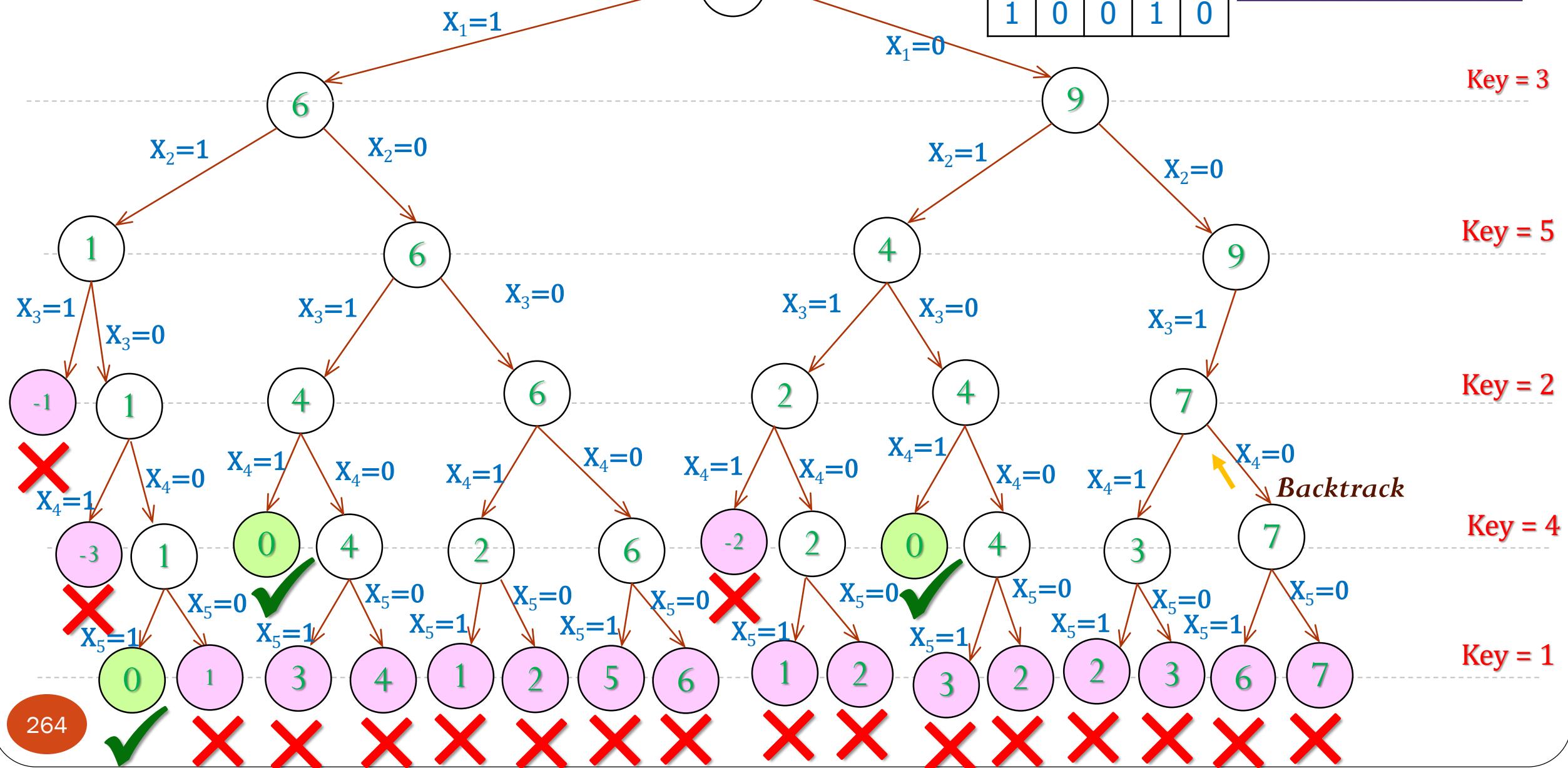
1	2	3	4	5
0	0	1	0	0

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



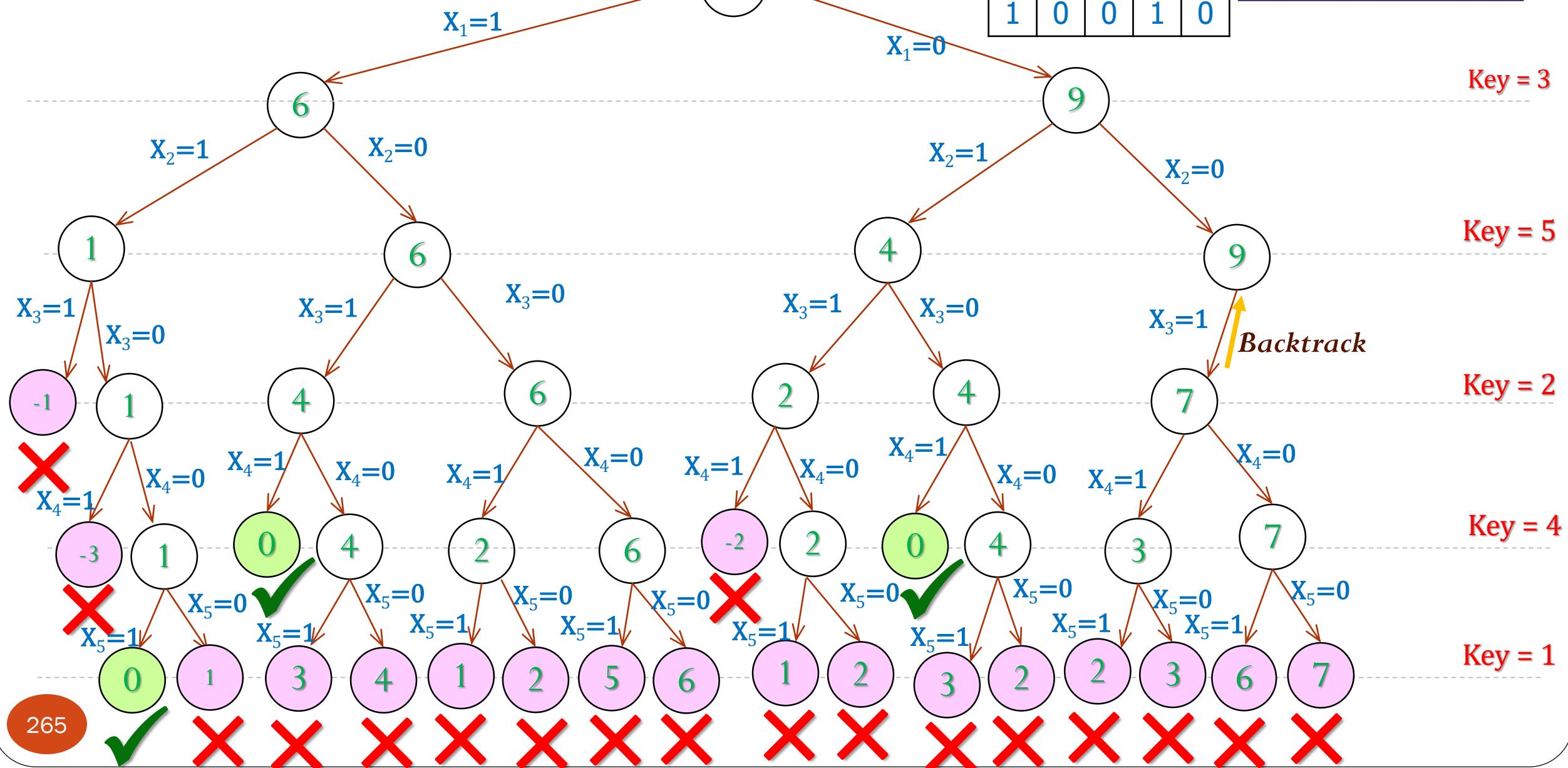
1	2	3	4	5
0	0	1	0	0

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



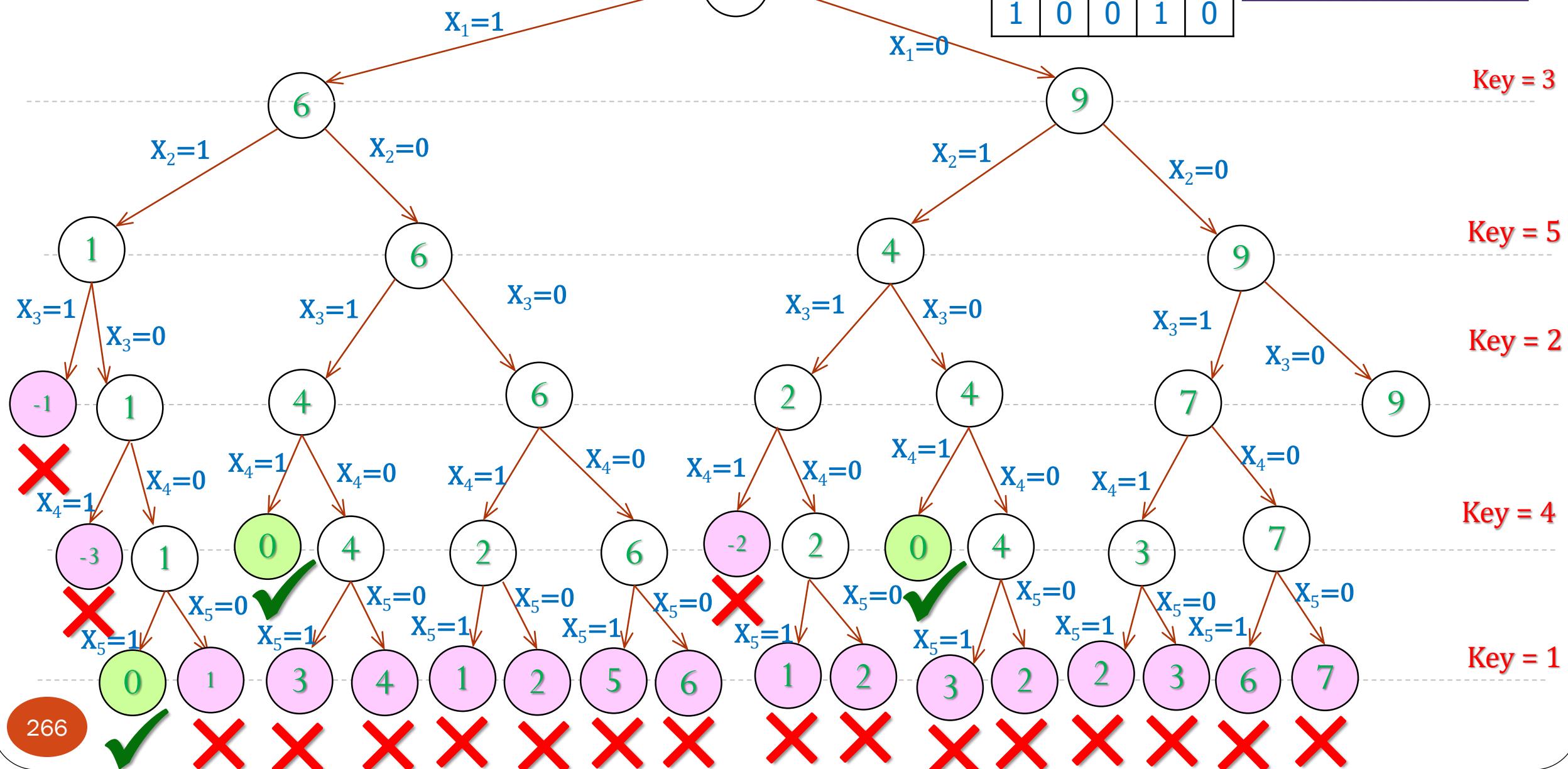
1	2	3	4	5
0	0	0	0	0

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	1	0

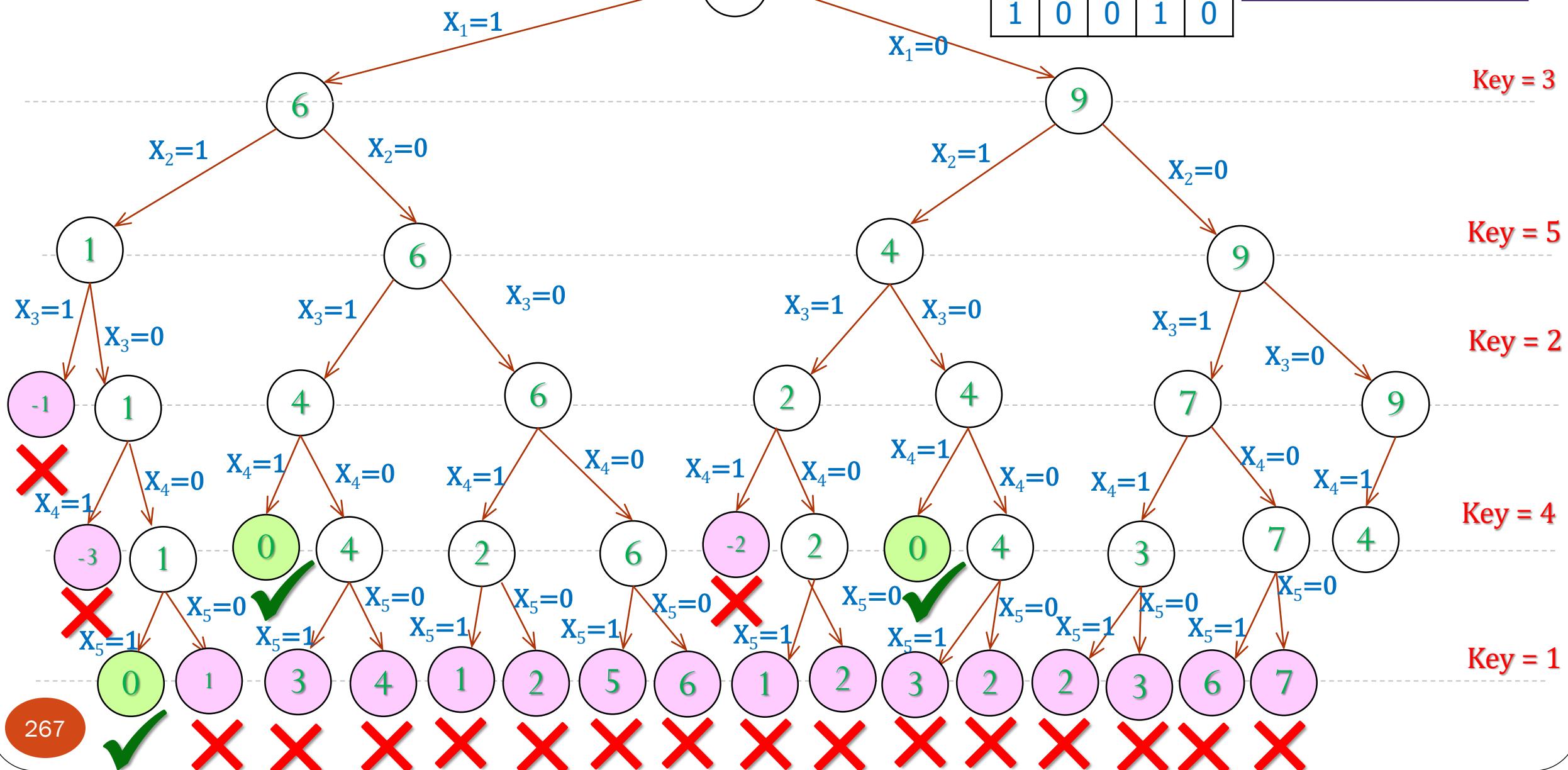
Key

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	0	1	1

Key

1	2	3	4	5
3	5	2	4	1

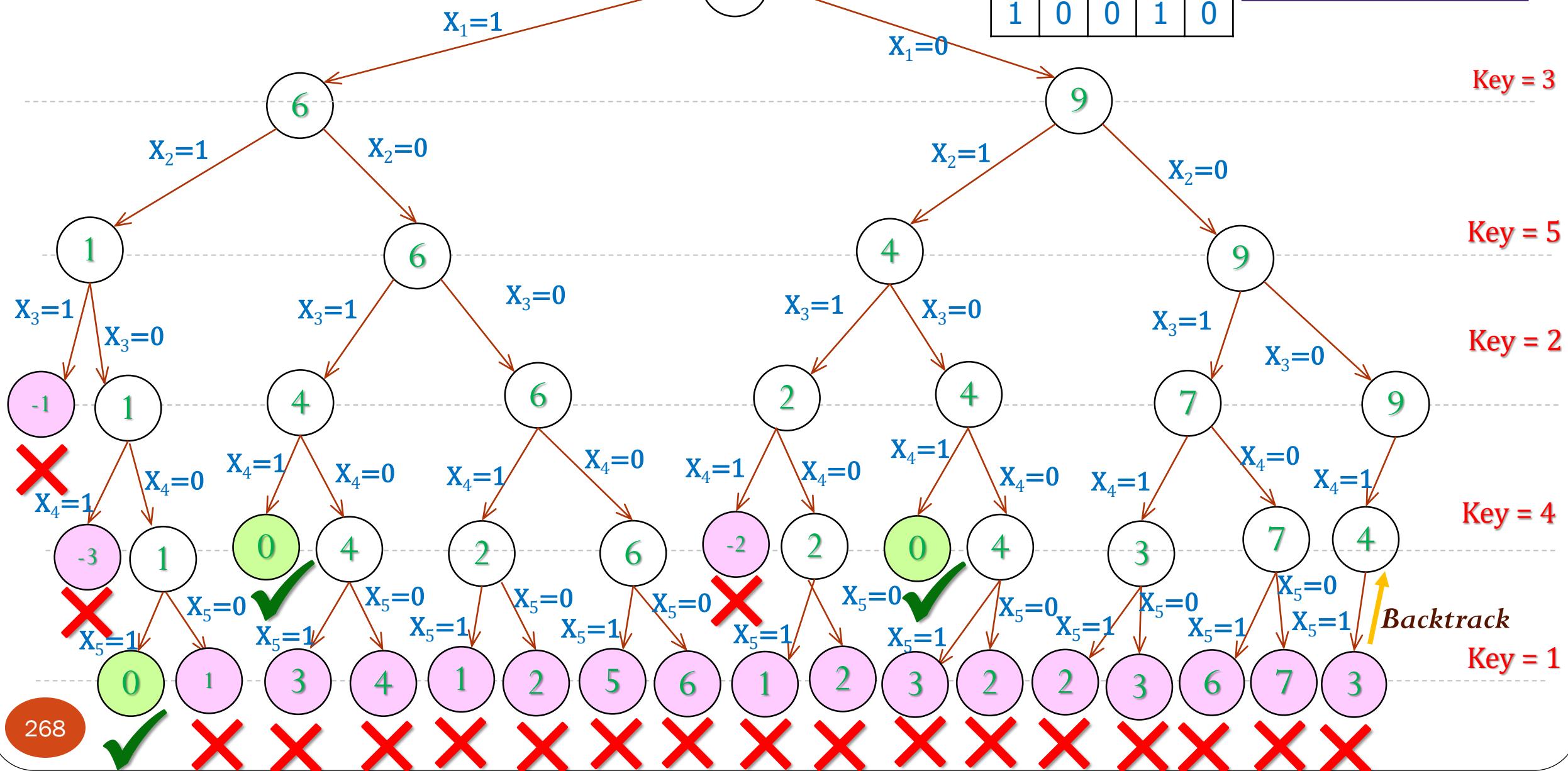
Start

Solutions

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



Key = 3



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



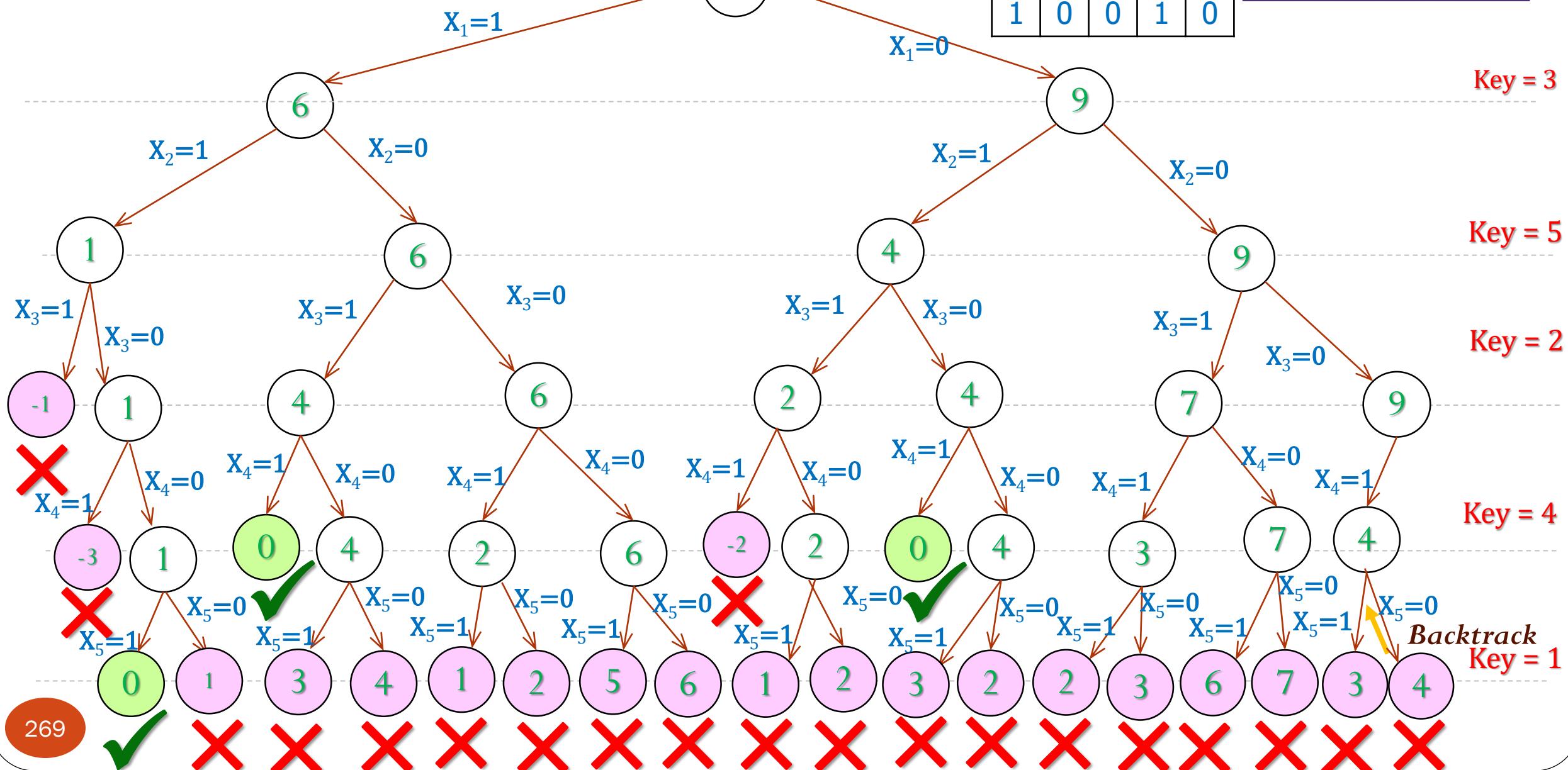
Key = 3

Key = 5

Key = 2

Key = 4

**Backtrack
Key = 1**



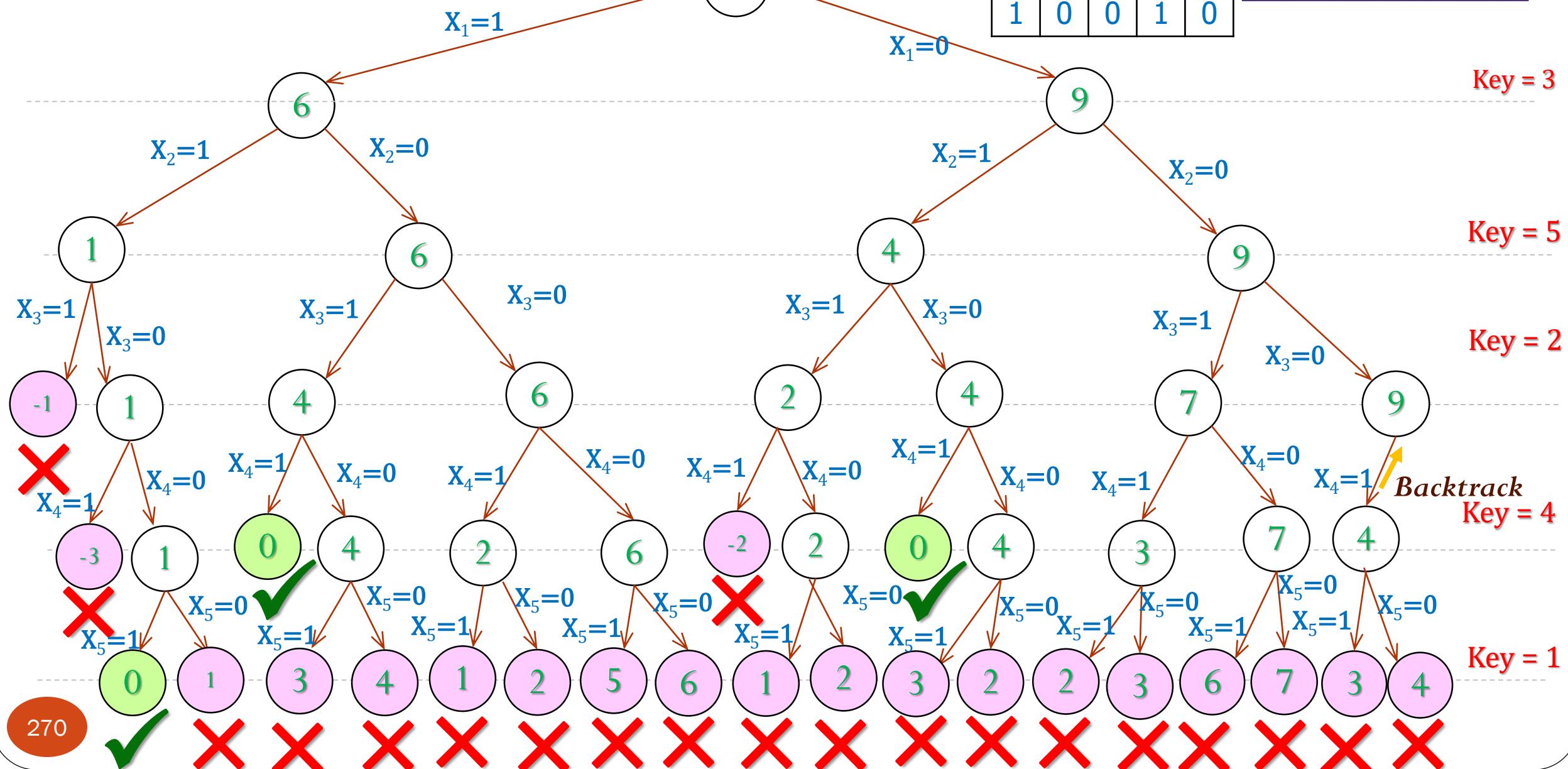
1	2	3	4	5
0	0	0	1	0

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	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



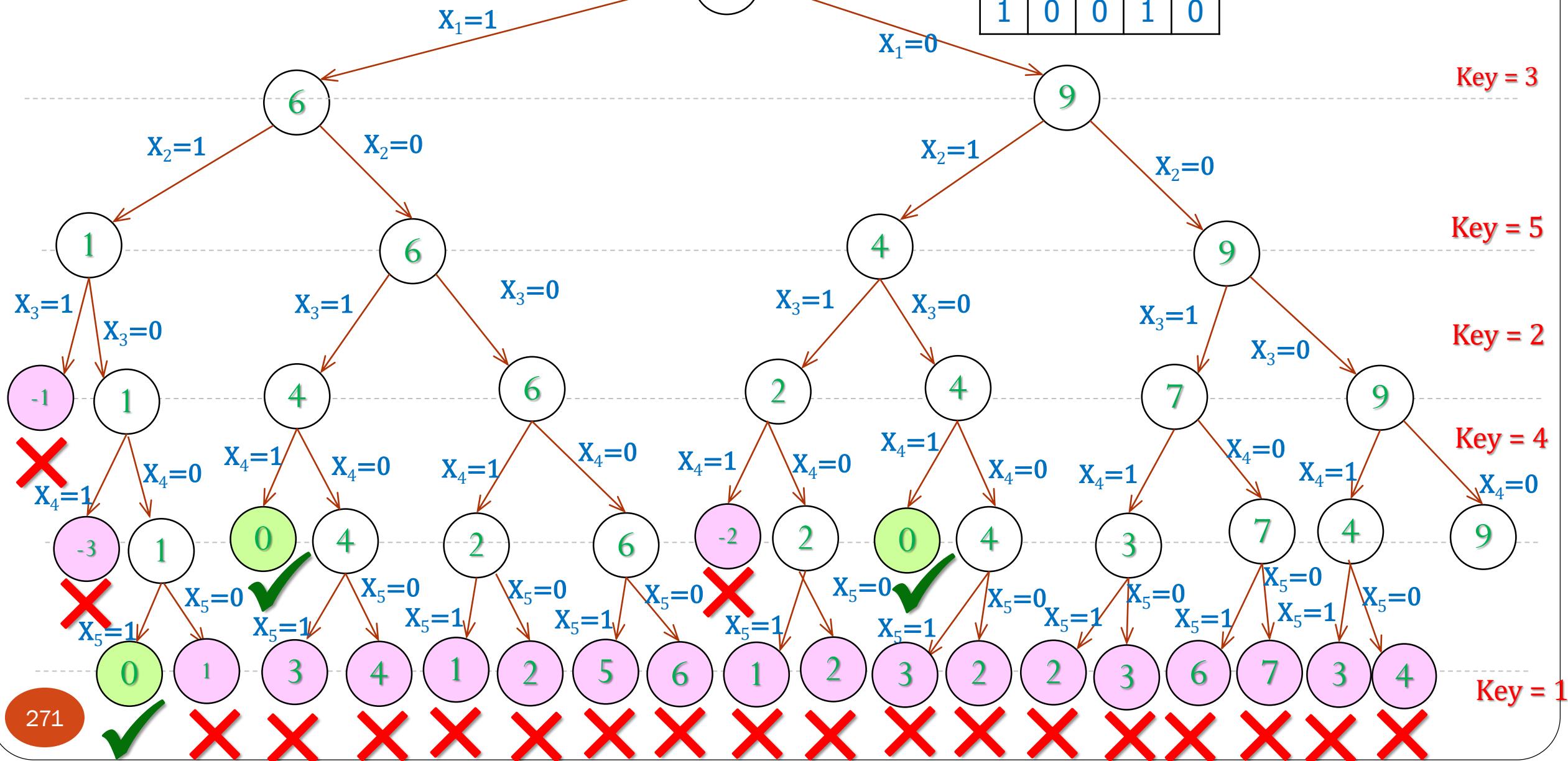
Key = 3

Key = 5

Key = 2

Key = 4

Key = 1



X

1	2	3	4	5
0	0	0	0	1

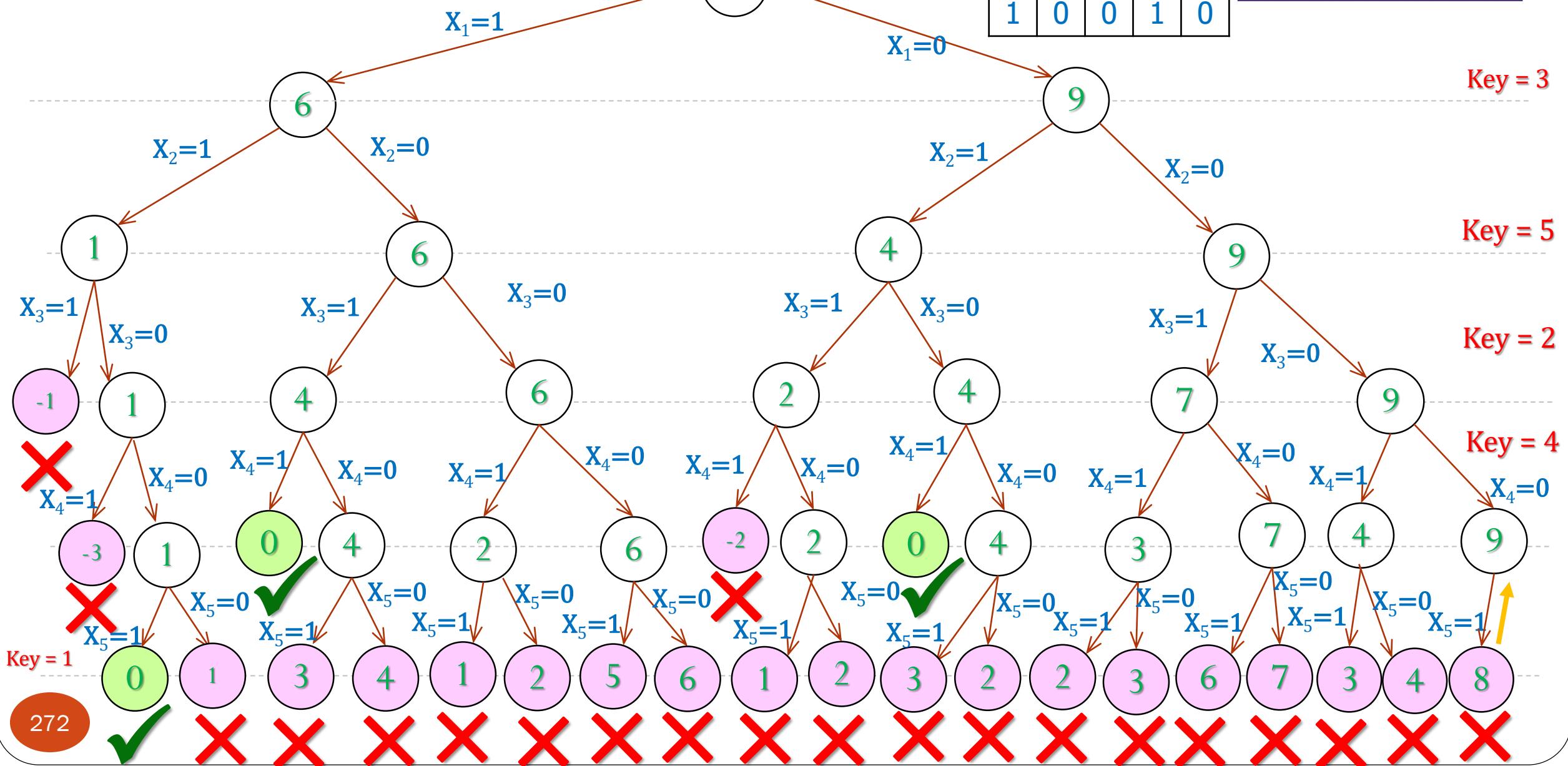
Key

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



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

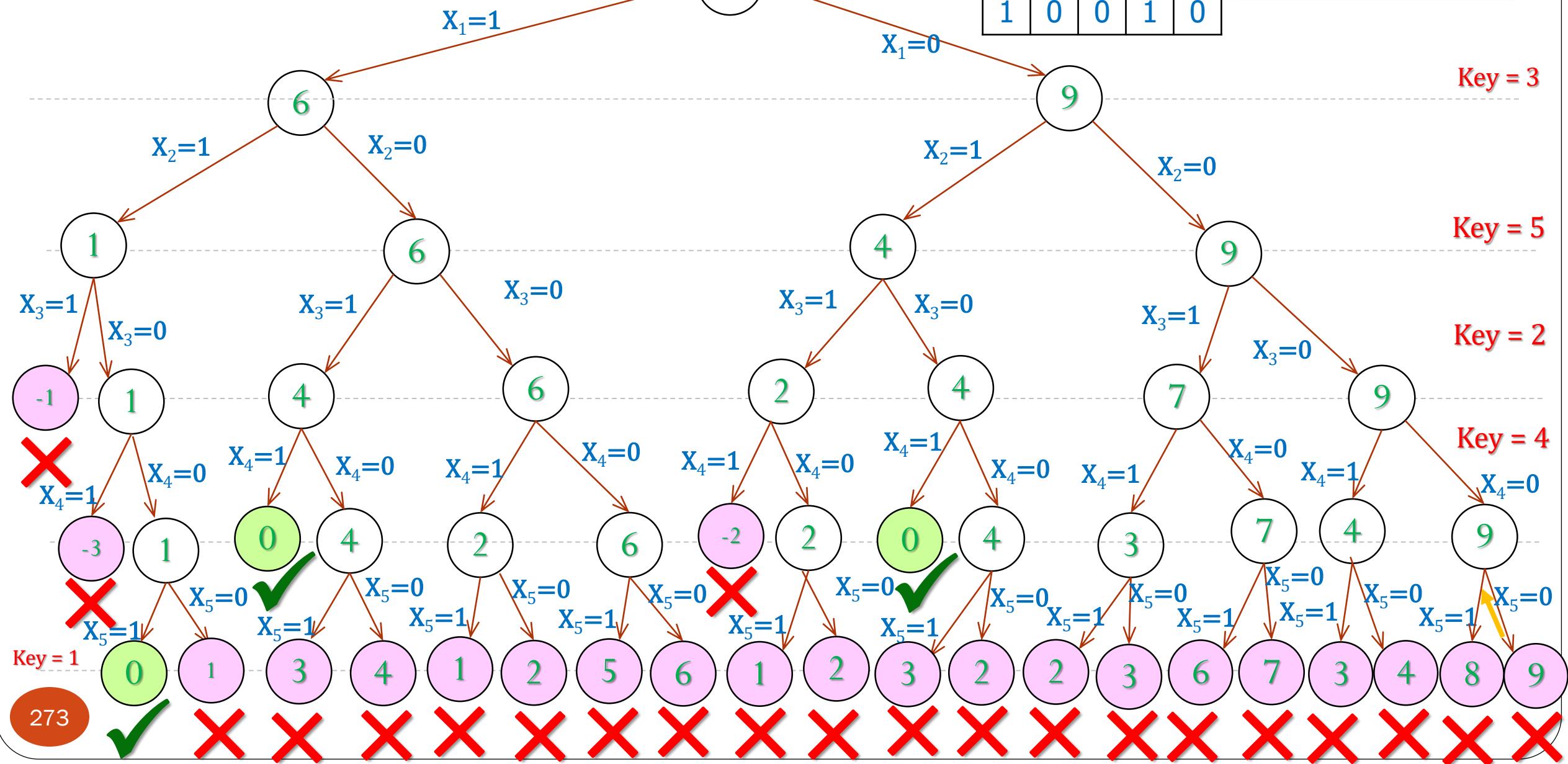


Key = 3

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

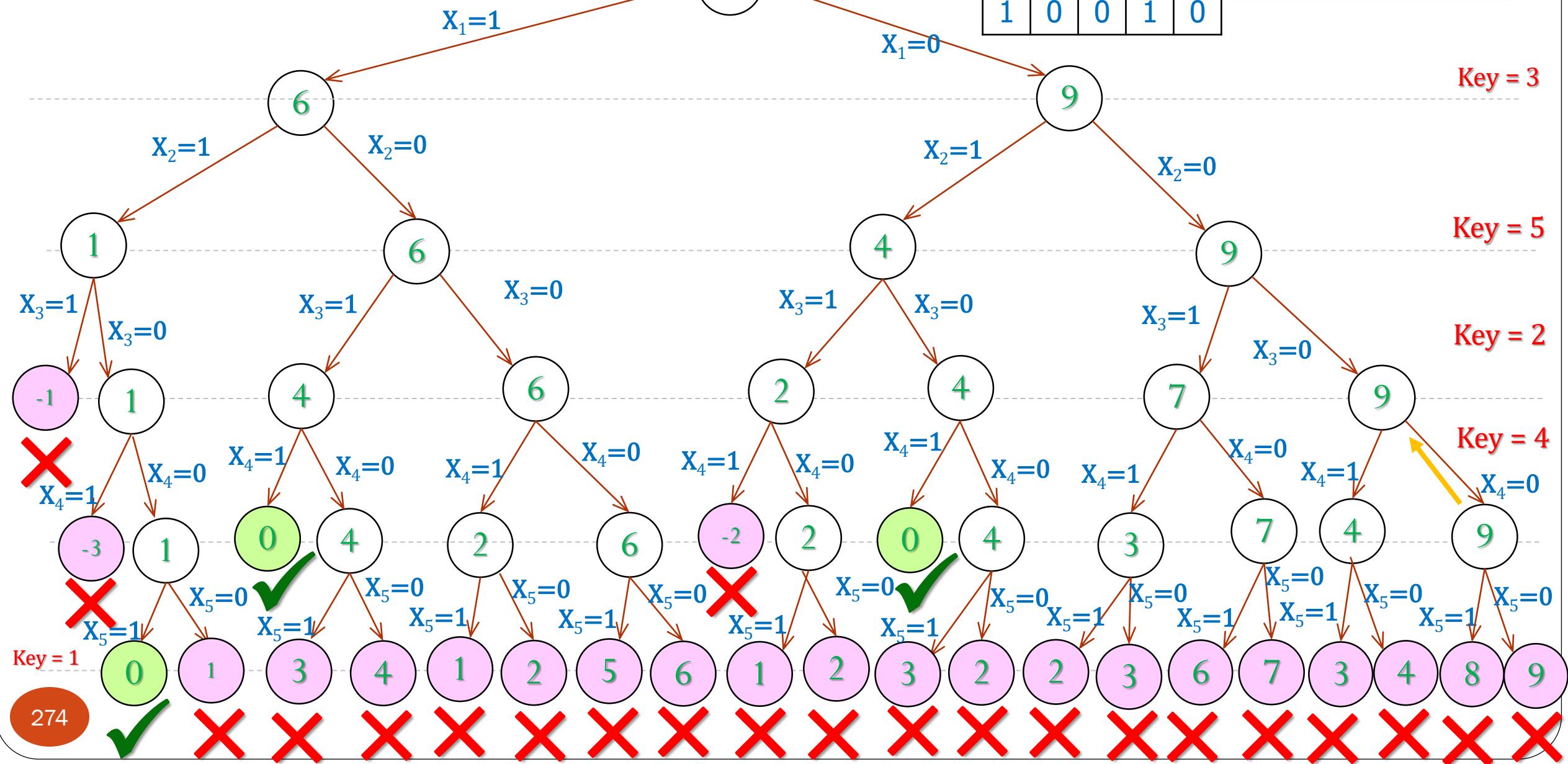


Key = 3

Key = 5

Key = 2

Key = 4



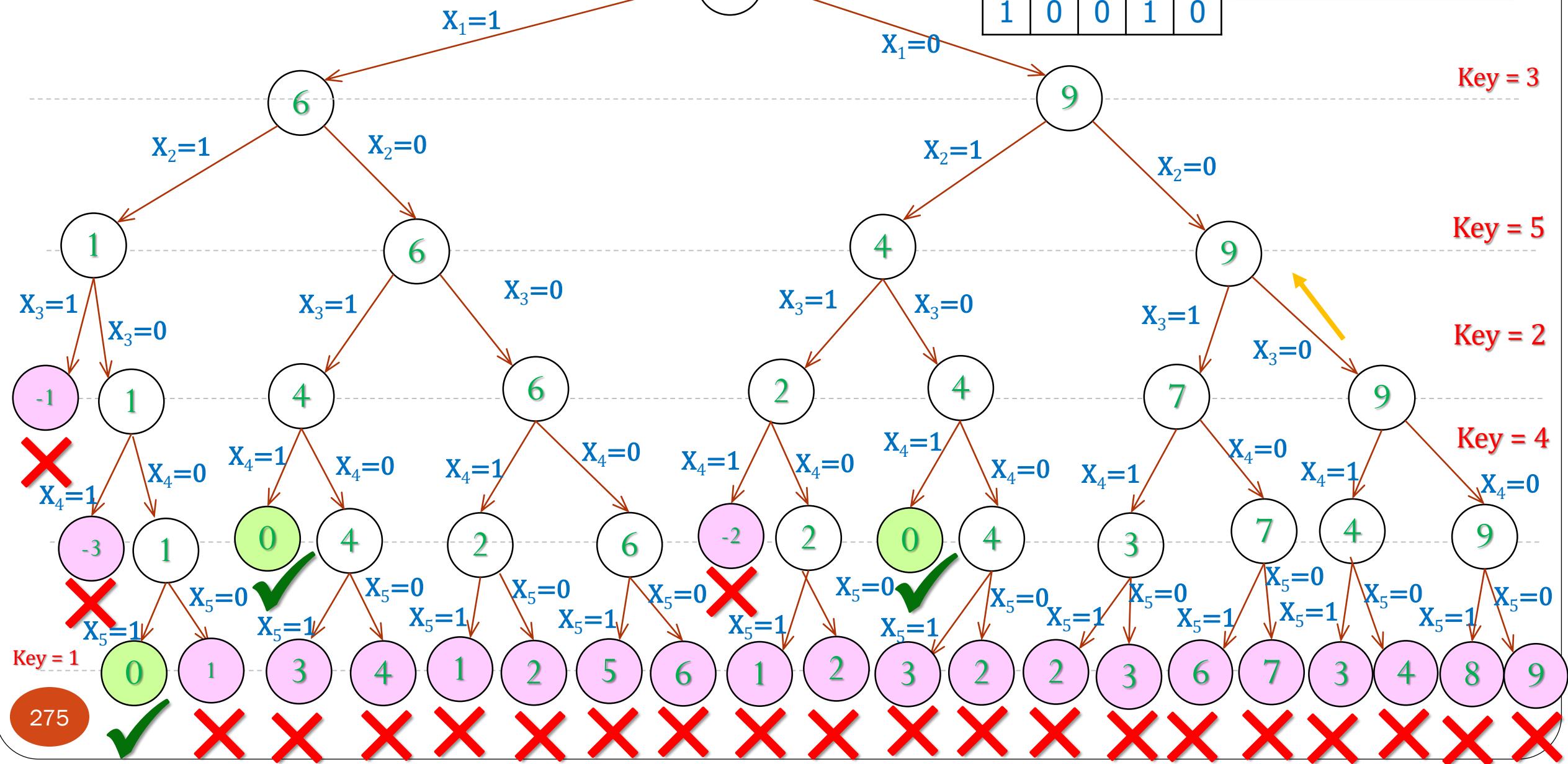
1	2	3	4	5
0	0	0	0	0

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



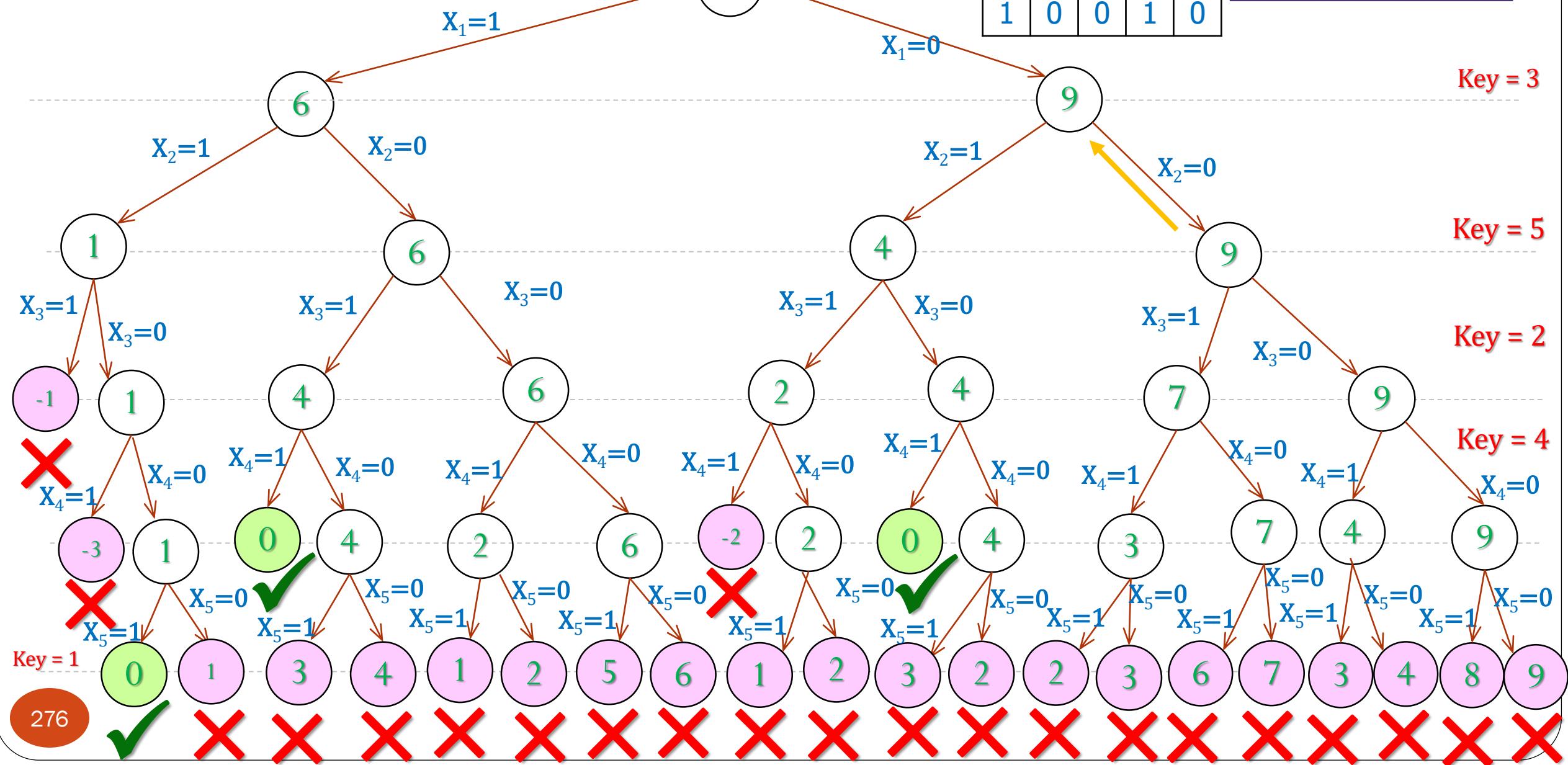
1	2	3	4	5
0	0	0	0	0

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

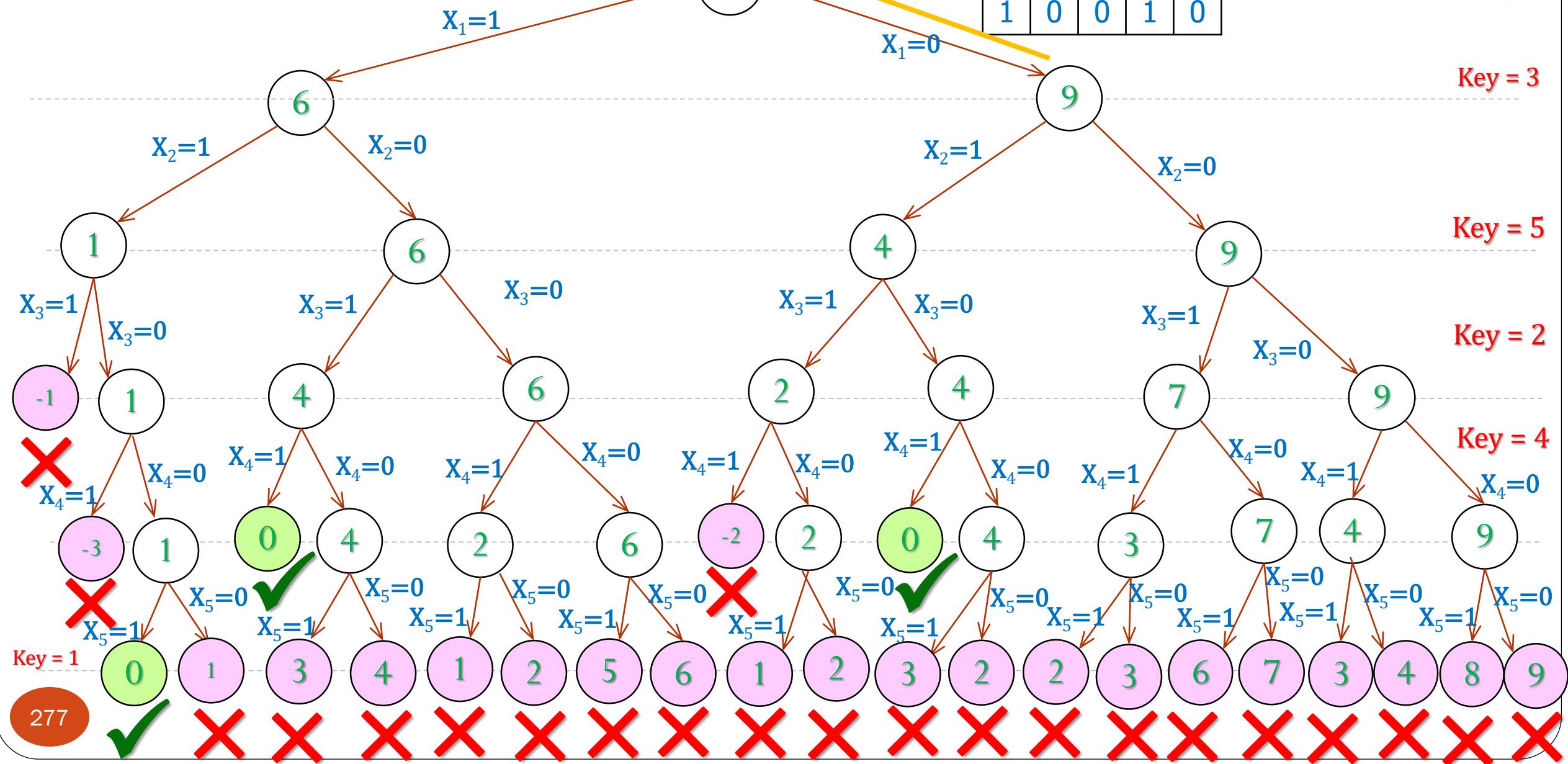
Key

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



X

1	2	3	4	5
0	0	0	0	0

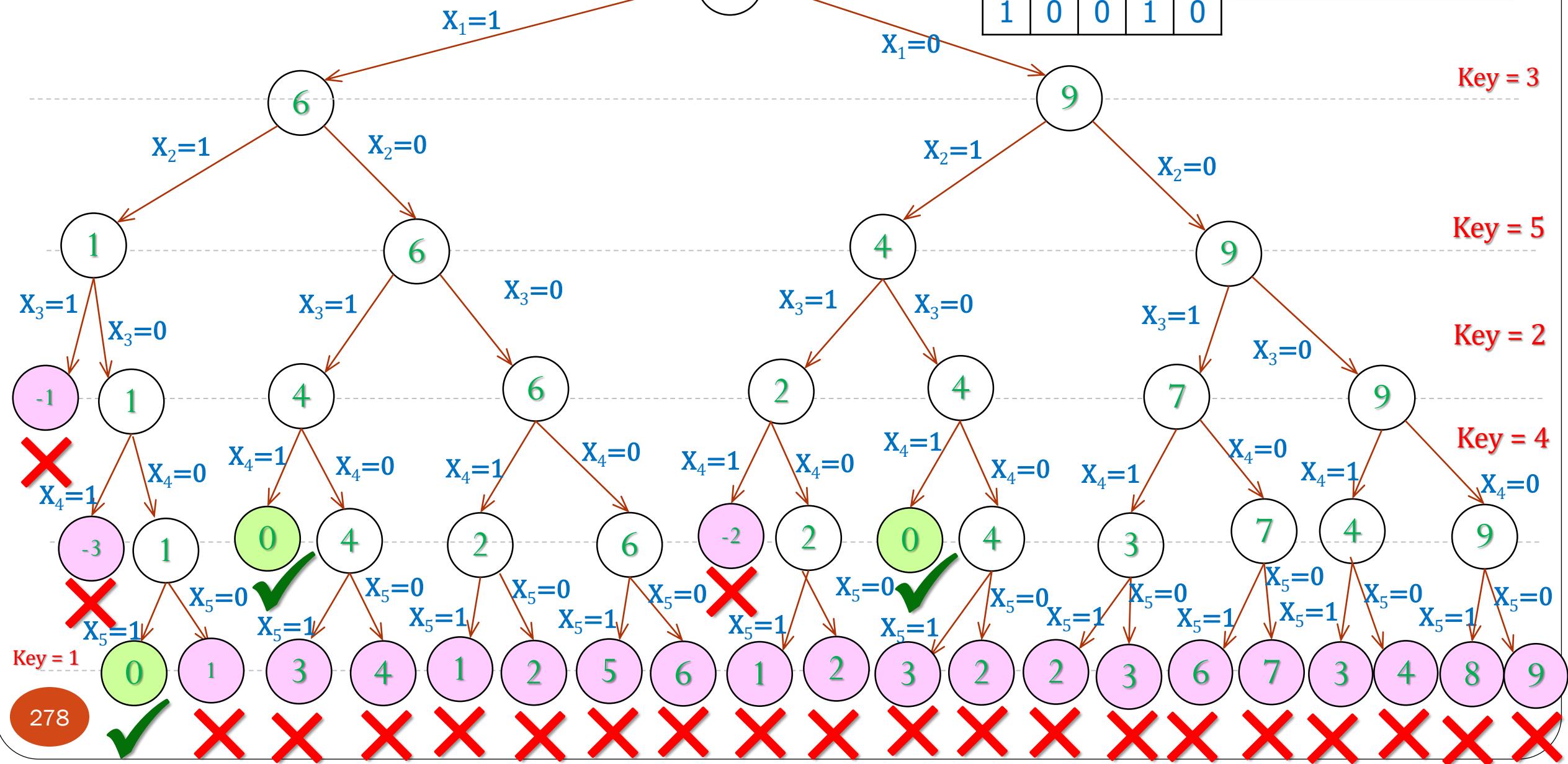
Key

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



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

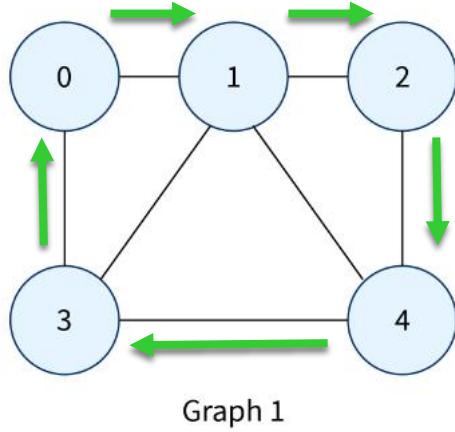
$x2[1..6] = \{0, 0, 1, 0, 0, 1\}$ ($w2+w6=12+18=30$)

Backtracking Approach

Hamiltonian Cycles Problem

Hamiltonian Cycles

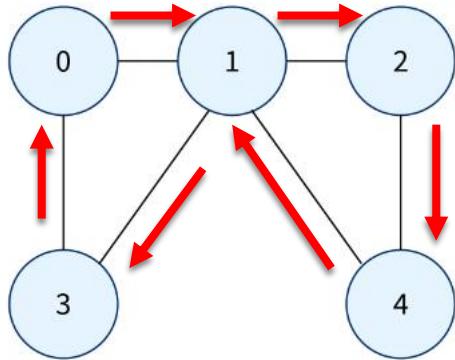
- ✓ It is a cycle in a graph, which visit each vertex exactly once.



Graph 1

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.



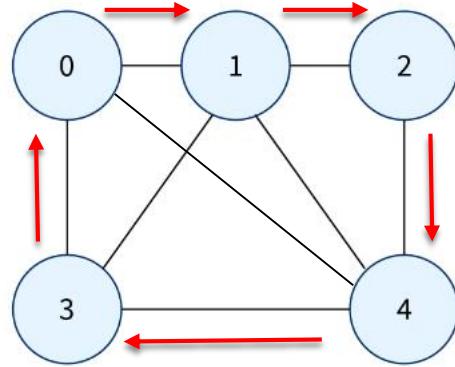
Graph 2

In Graph 2, No Hamiltonian Cycle Present

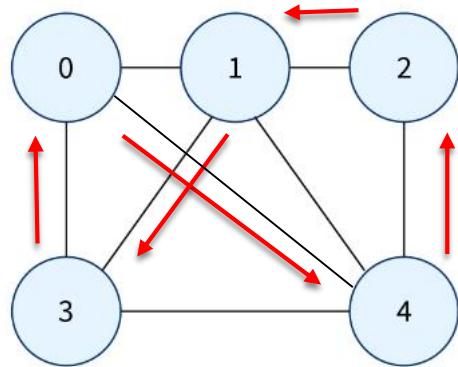
The cycle 0-1-2-4-1-3-0 is not a Hamiltonian cycle. The vertex 1 is visited twice.

Hamiltonian Cycles Problem

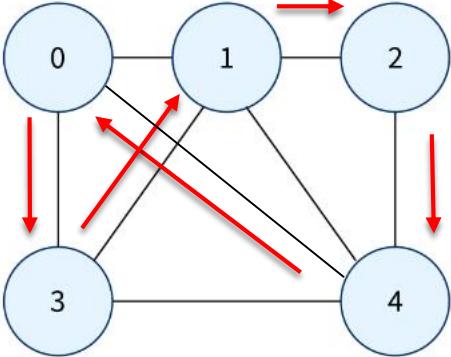
- ✓ Problem of finding all the Hamiltonian cycles present in the graph.



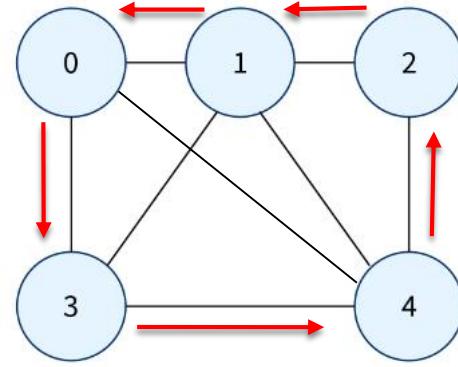
0-1-2-4-3-0



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

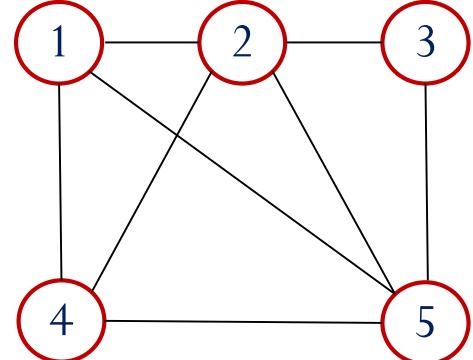


- ✓ Leave the starting vertex
- ✓ Generate the permutations for the remaining vertex
- ✓ Check the generated paths are Hamiltonian cycles or not.
- ✓ Save it, if it is Hamiltonian cycles.

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.

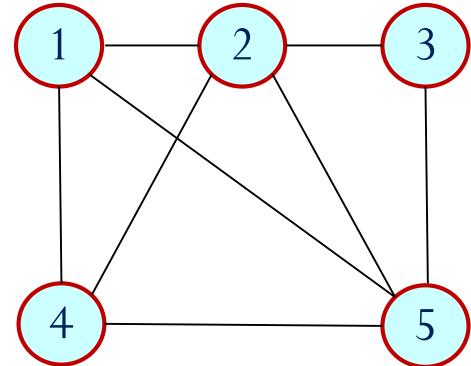
- ✓ So go for next value: 2

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

- ✓ Yes. Edge (1, 2) is presented

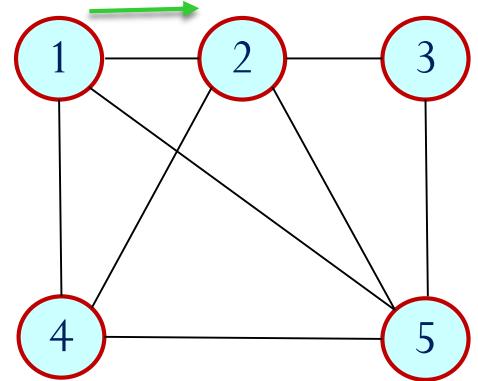
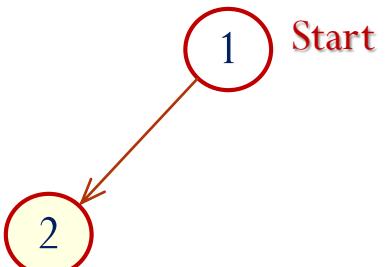
- ✓ So proceed for next level with selected vertex as 2

1 Start



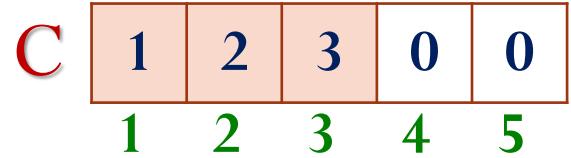
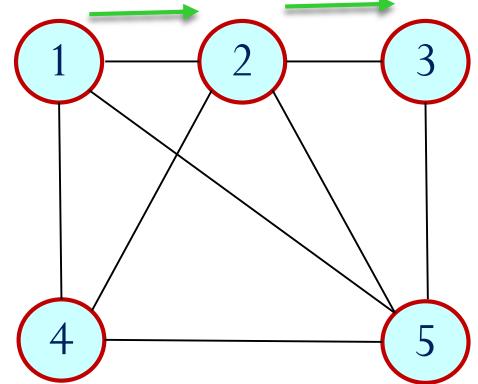
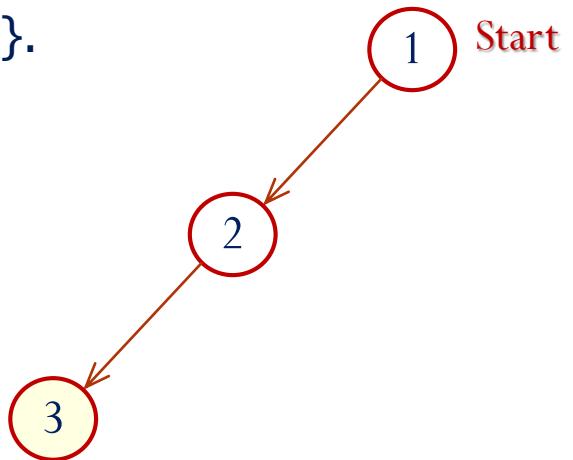
C	1	0	0	0	0
	1	2	3	4	5

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

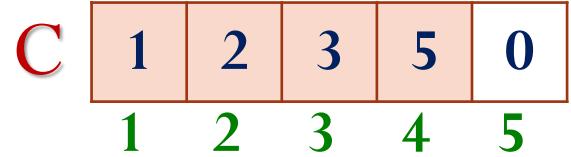
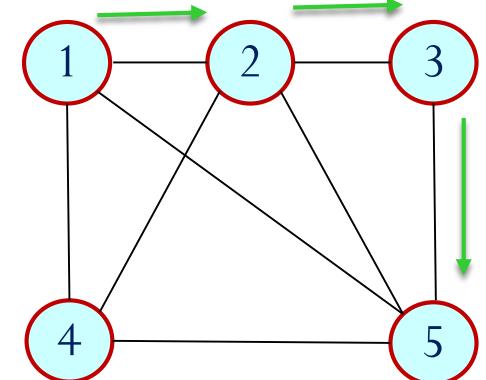
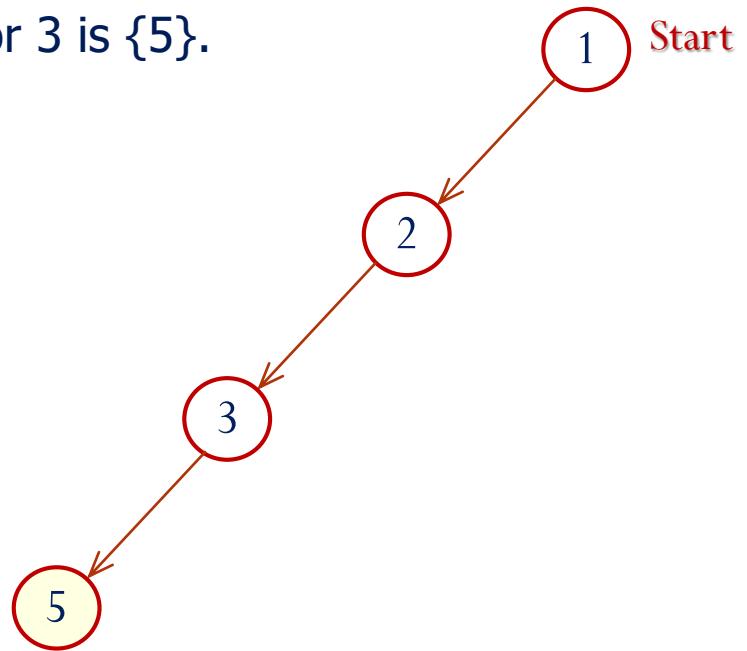


C	1	2	0	0	0
	1	2	3	4	5

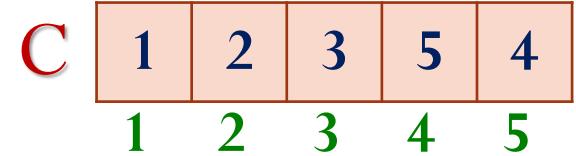
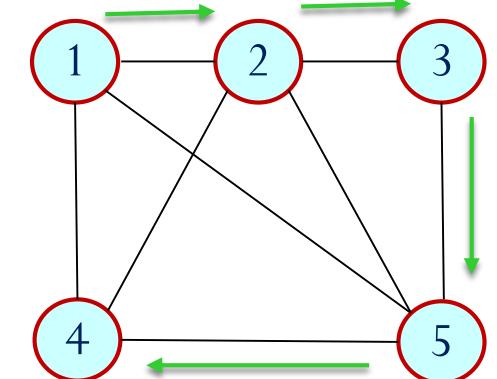
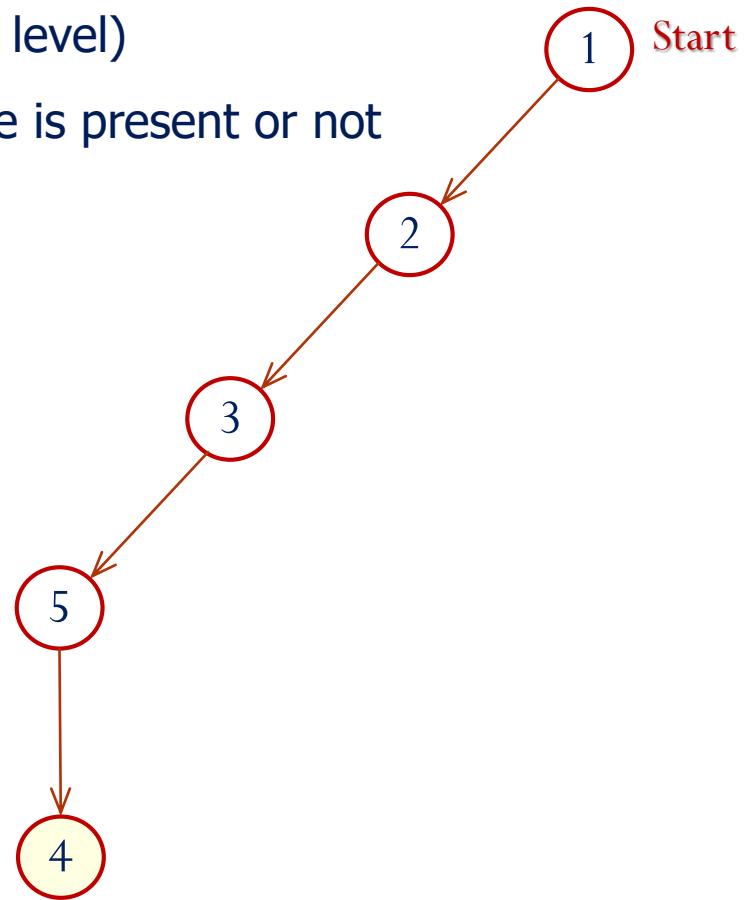
- ✓ The next possible vertices for 3 is {5}.



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



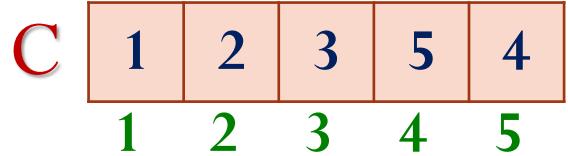
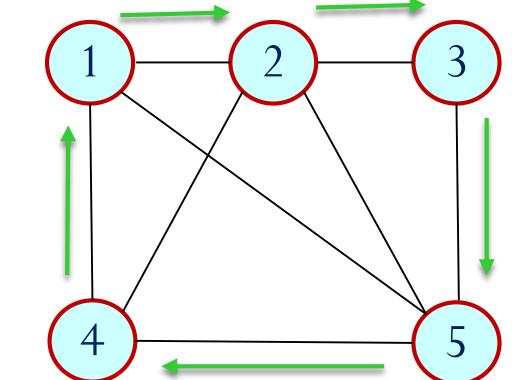
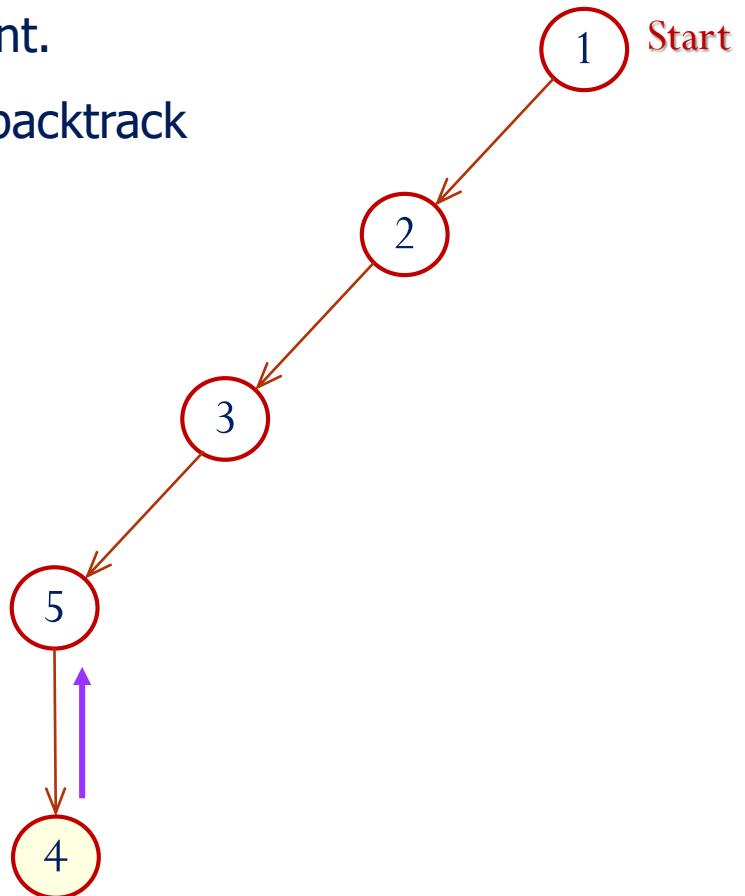
- ✓ Now reached last level. (nth level)
- ✓ So, need to check (4,1) edge is present or not



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

Solutions

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

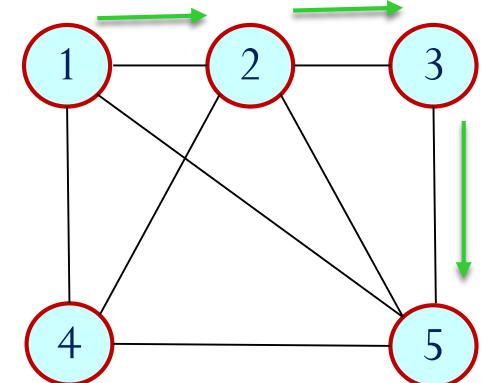
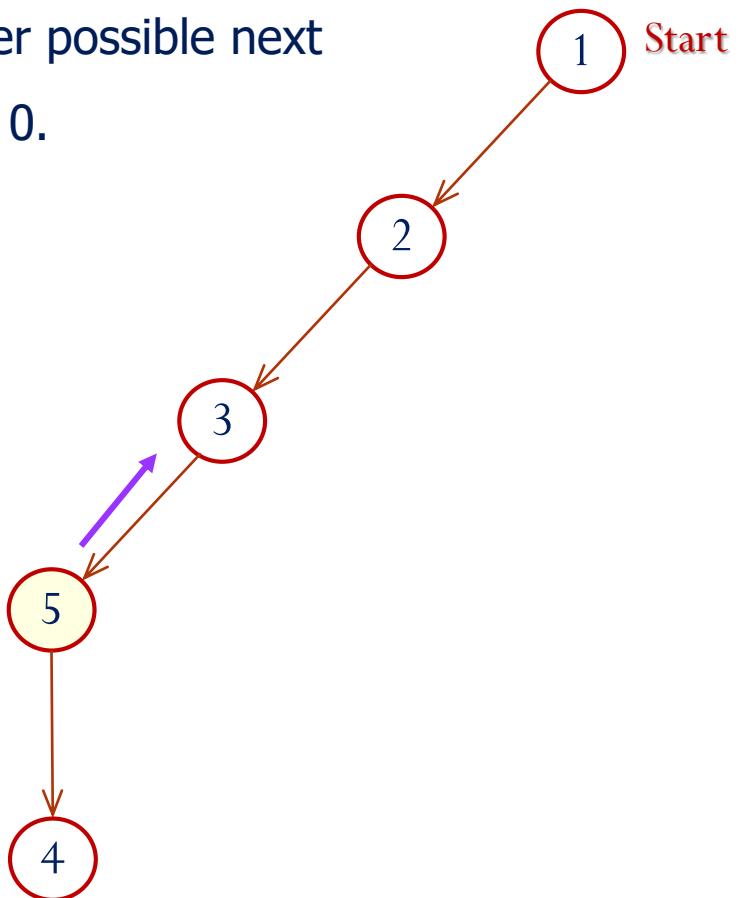


- ✓ For vertex 5, there is no other possible next vertex. So, $C[5]$ will become 0.

- ✓ Backtrack to previous state

Solutions

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



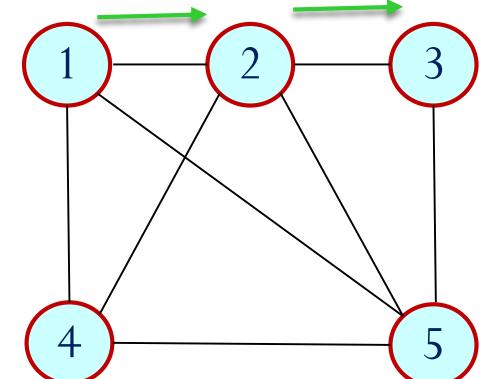
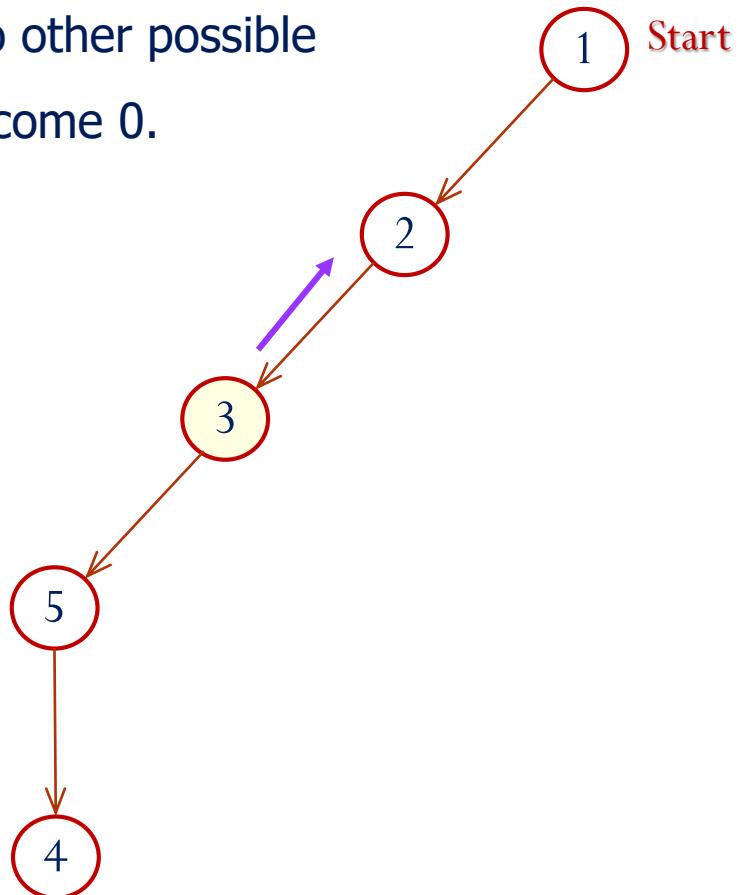
C	1	2	3	5	0
	1	2	3	4	5

✓ 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
---	---	---	---	---

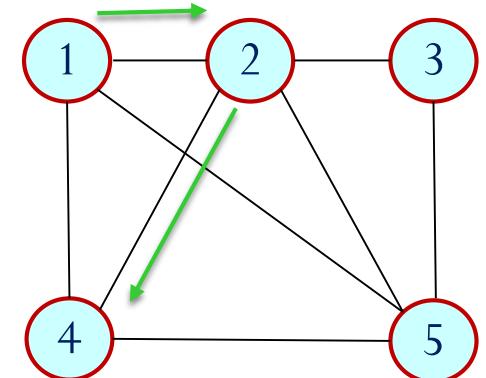
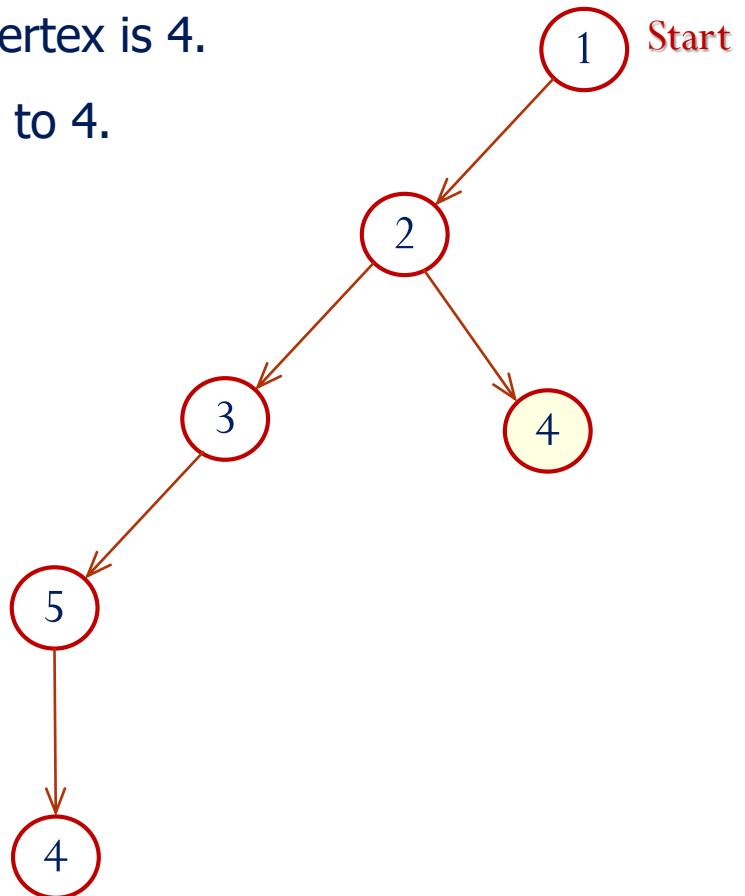


C	1	2	3	0	0
	1	2	3	4	5

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

Solutions

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

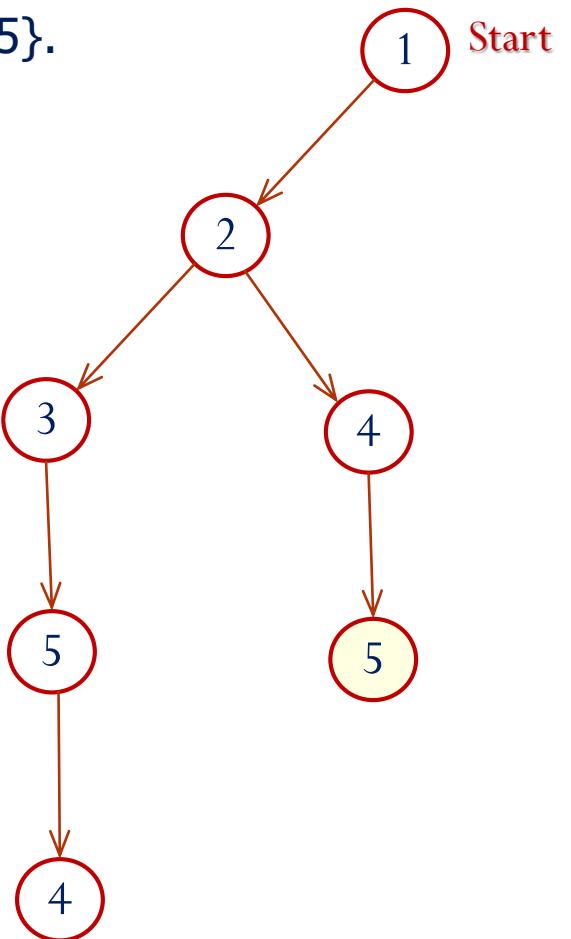


C

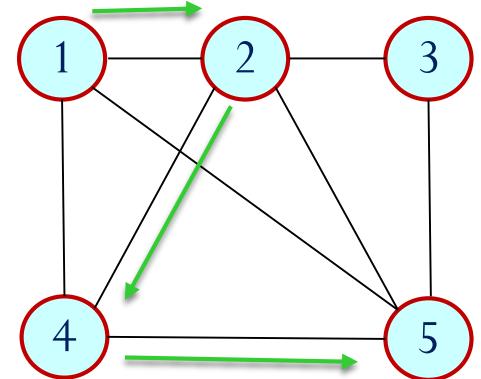
1	2	4	0	0
1	2	3	4	5

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

✓ So, C[4] will be incremented to 5.



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

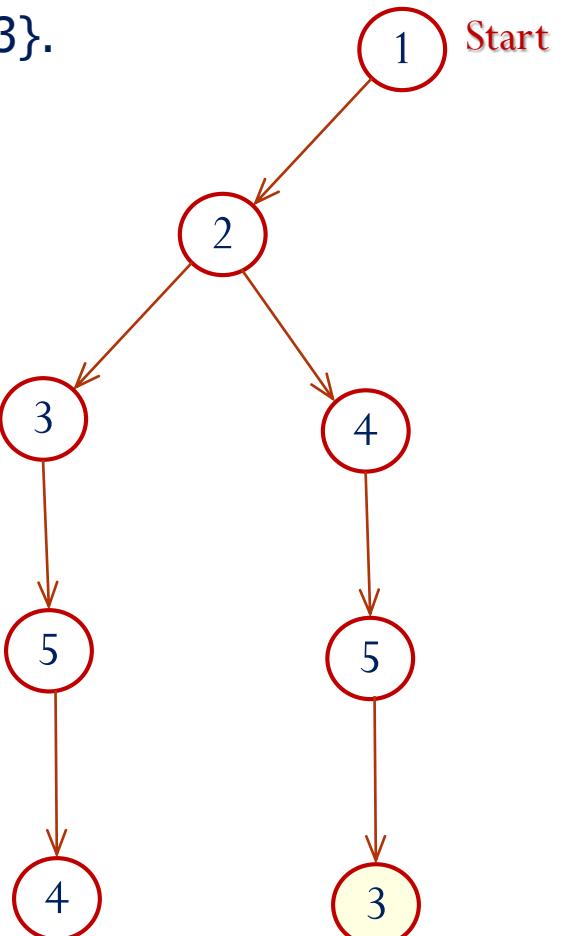


C

1	2	4	5	0
1	2	3	4	5

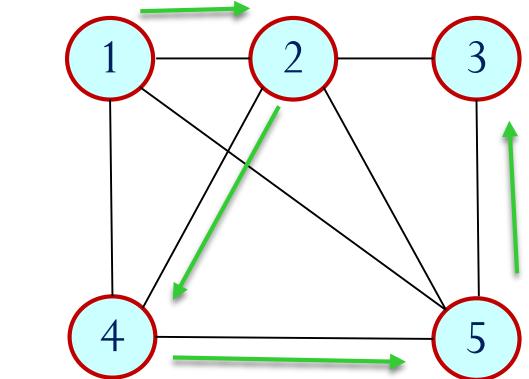
✓ For vertex 5, next possible vertices {3}.

✓ So, C[5] will be incremented to 3.



Solutions

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



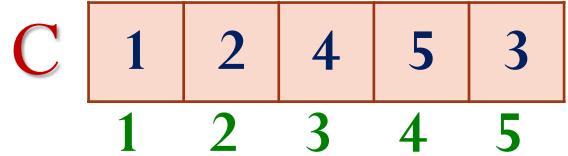
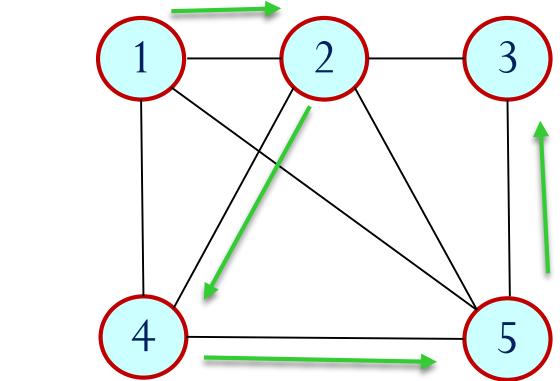
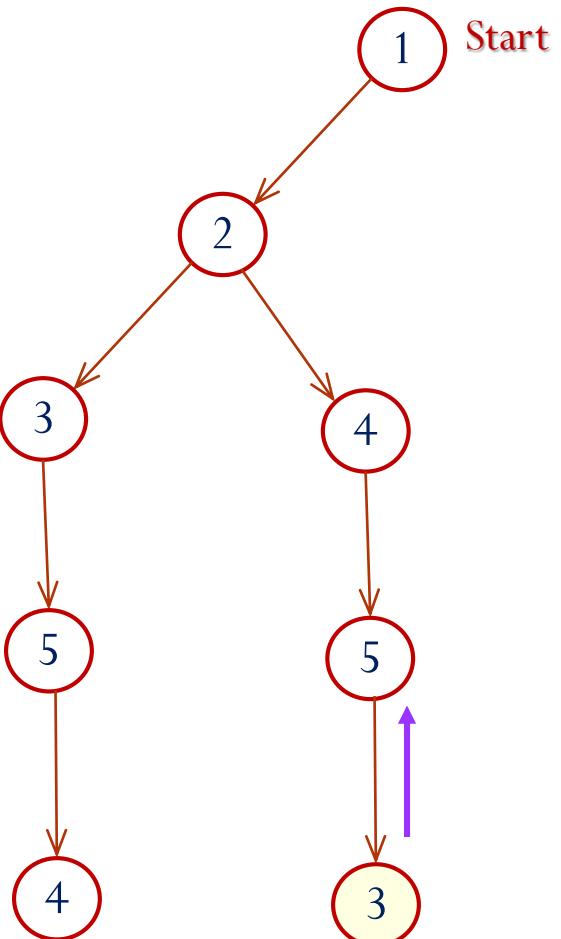
C

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
---	---	---	---	---



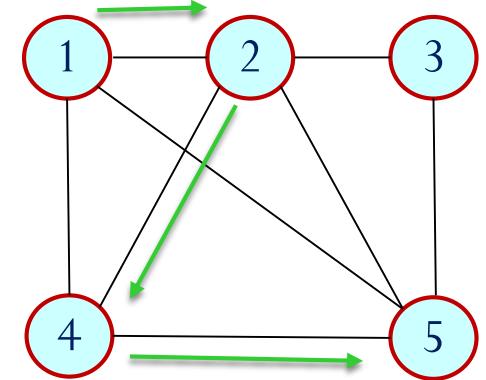
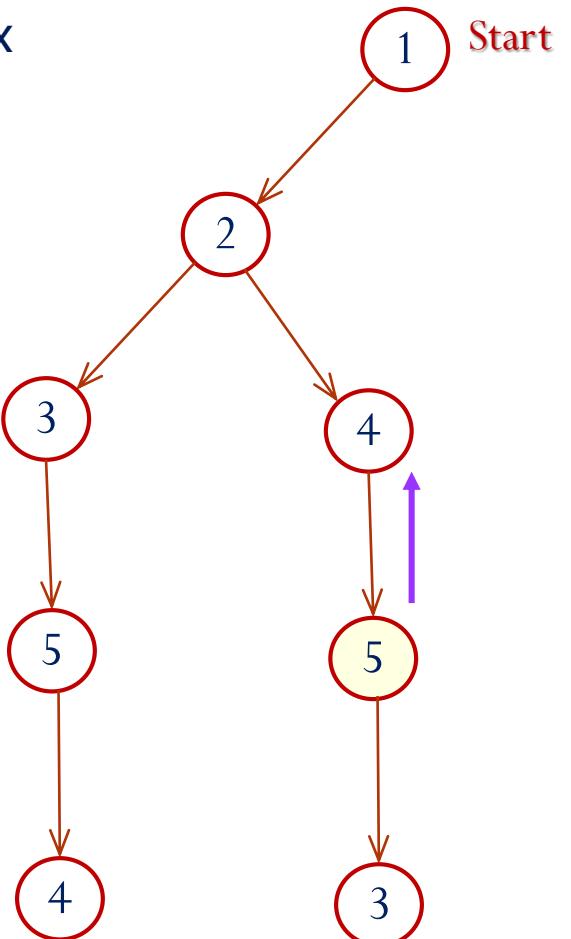
- ✓ For vertex 5, no other possible vertex
- ✓ So backtrack



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

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



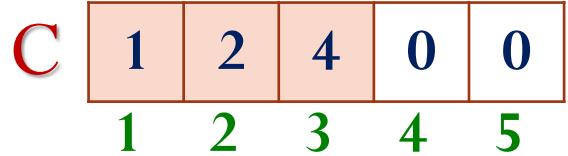
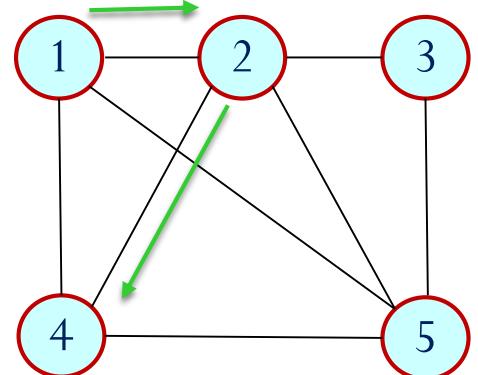
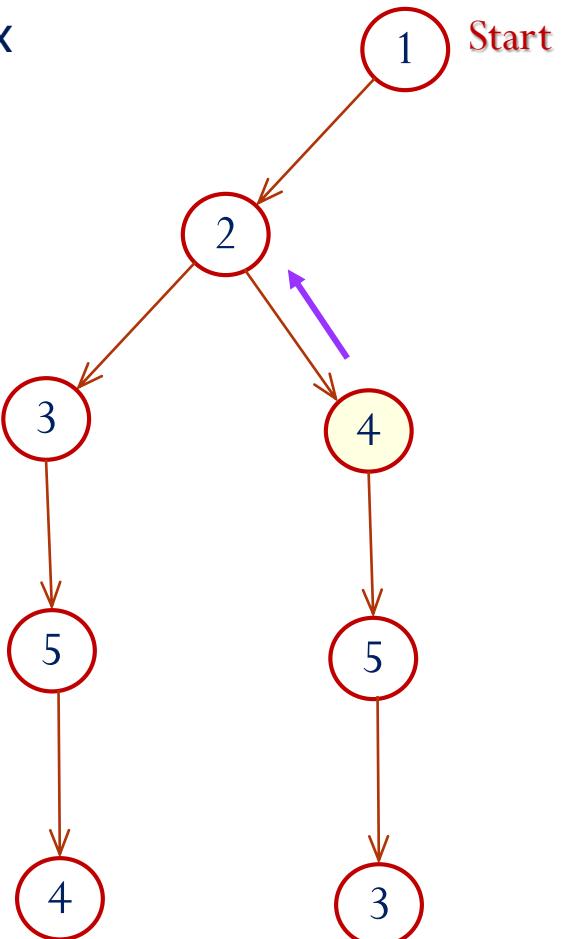
C

1	2	4	5	0
1	2	3	4	5

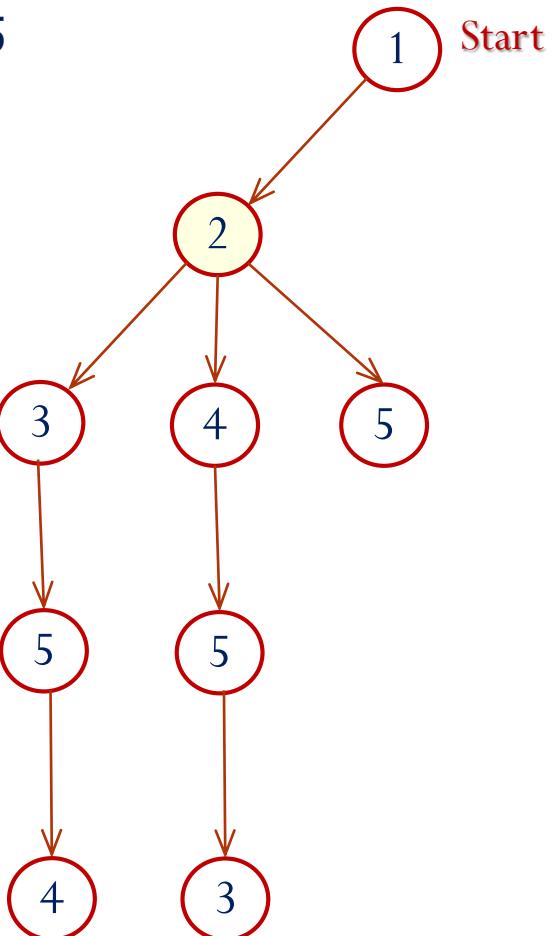
- ✓ For vertex 4, no other possible vertex
- ✓ So backtrack

Solutions

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

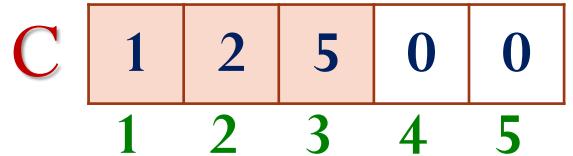
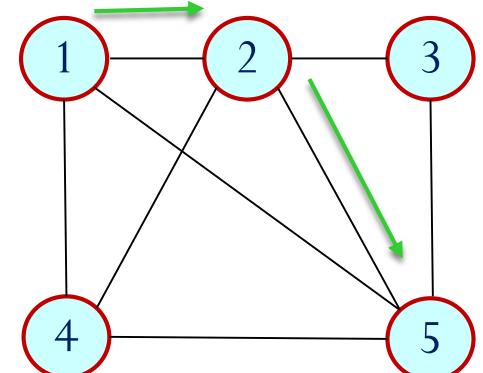


- ✓ For vertex 2, next possible vertex is 5



Solutions

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

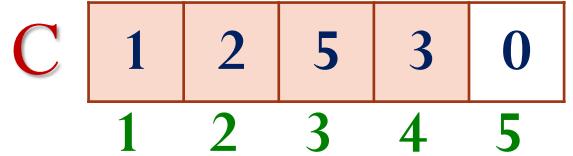
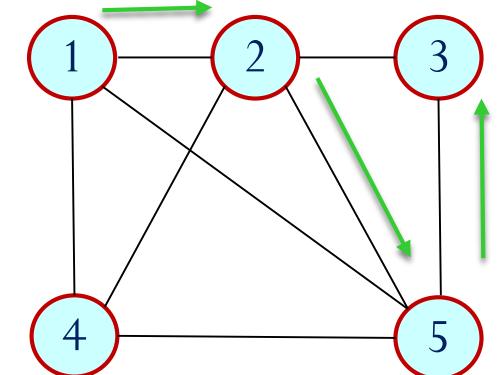
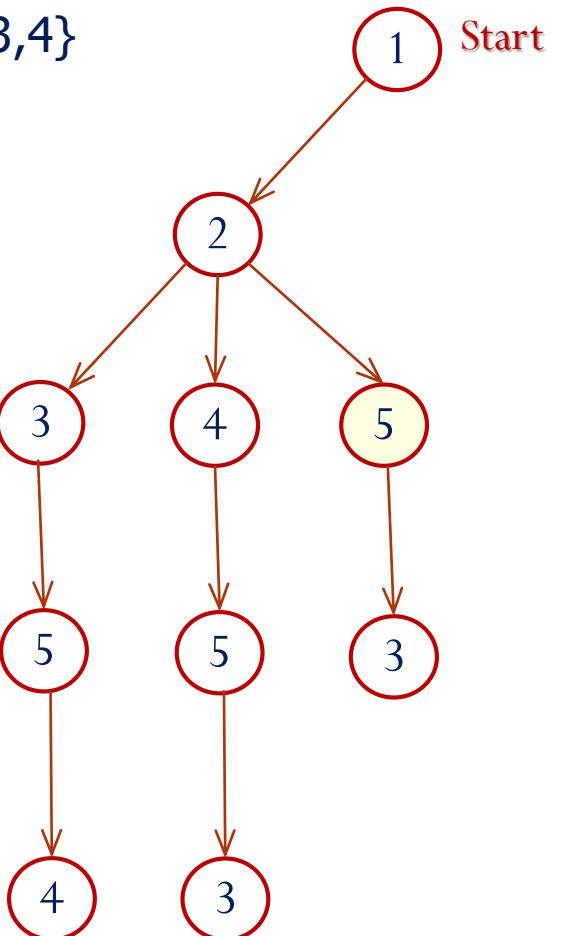


✓ For vertex 5, next possible vertices {3,4}

✓ Proceed with 3

Solutions

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

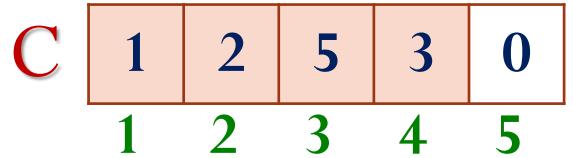
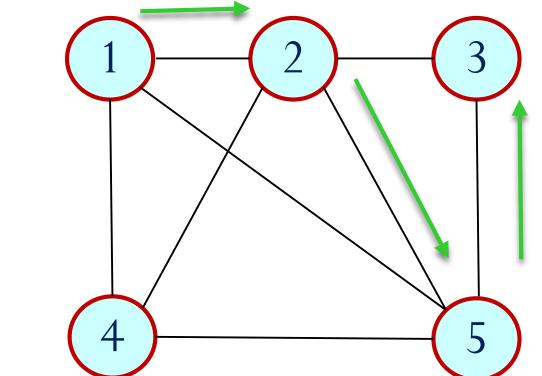
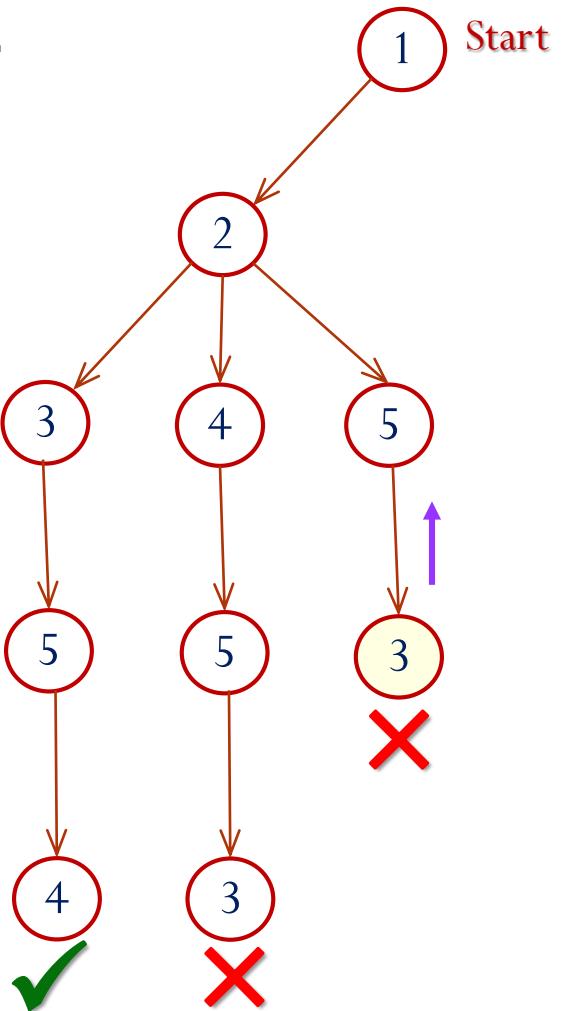


✓ For vertex 3, no possible next vertex.

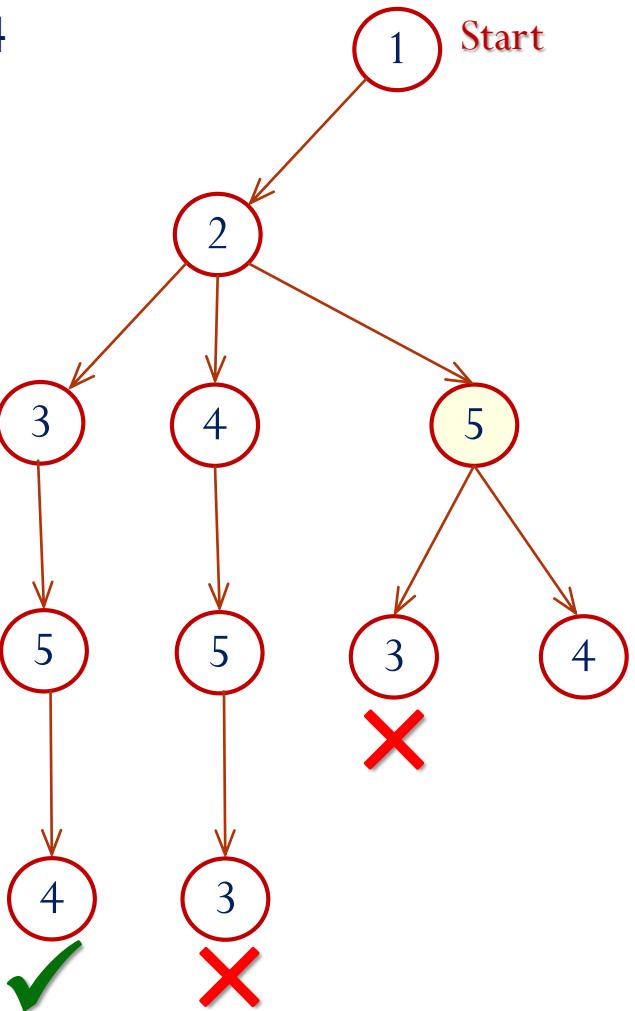
✓ So, backtrack

Solutions

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

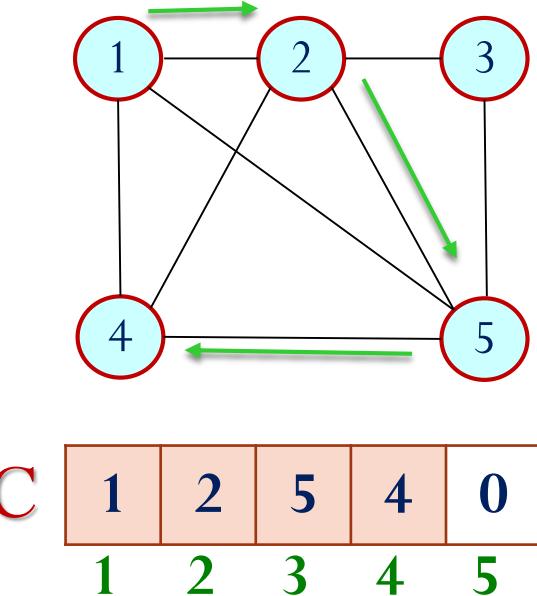


- ✓ For vertex 5, possible next vertex is 4



Solutions

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

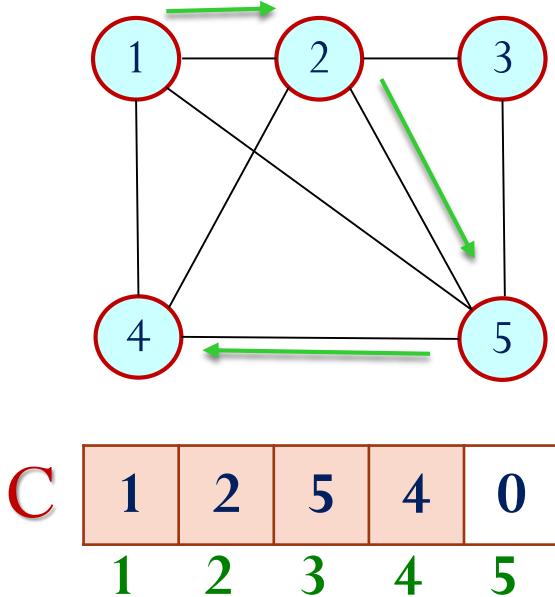
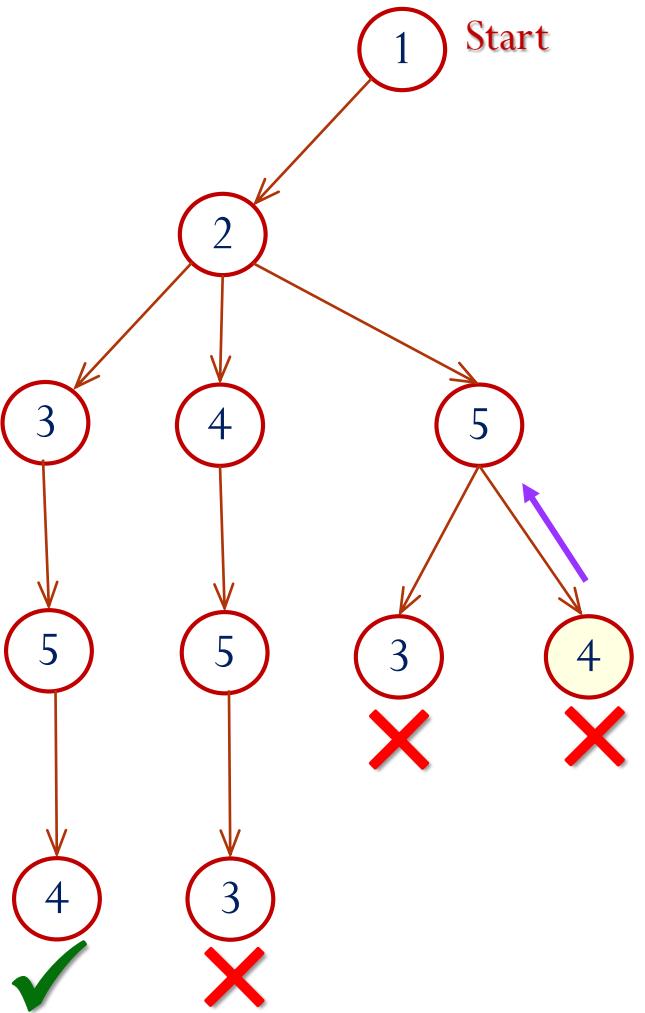


- ✓ For vertex 4, no possible vertex
- ✓ So, backtrack



Solutions

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

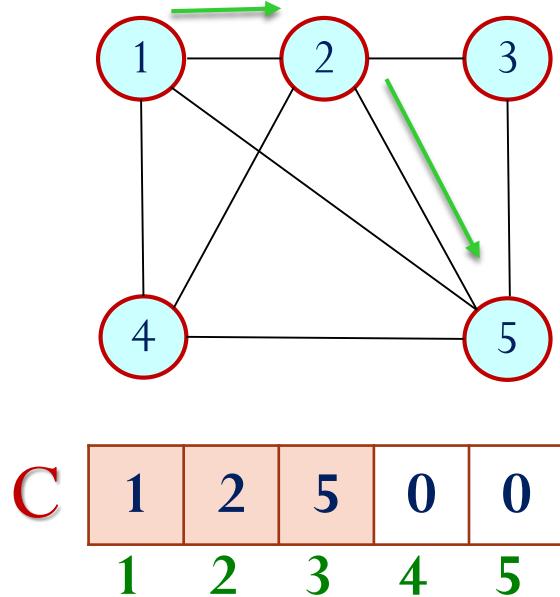
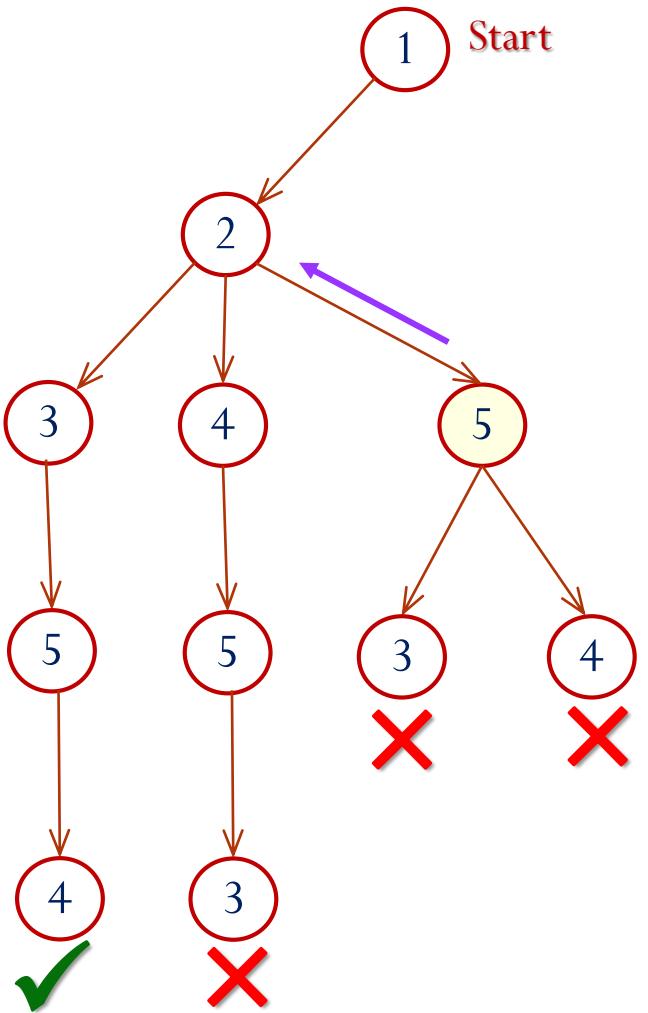


- ✓ For vertex 5, no possible vertex
- ✓ So, backtrack



Solutions

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

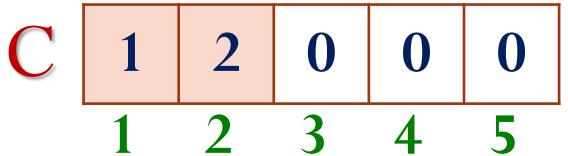
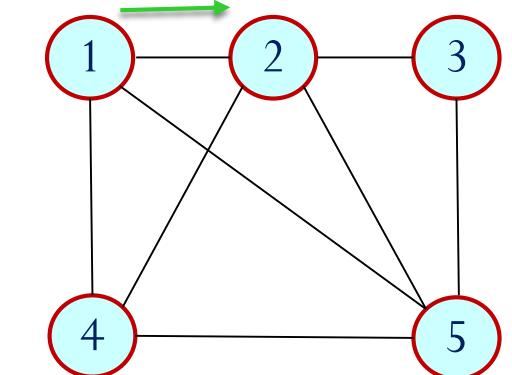
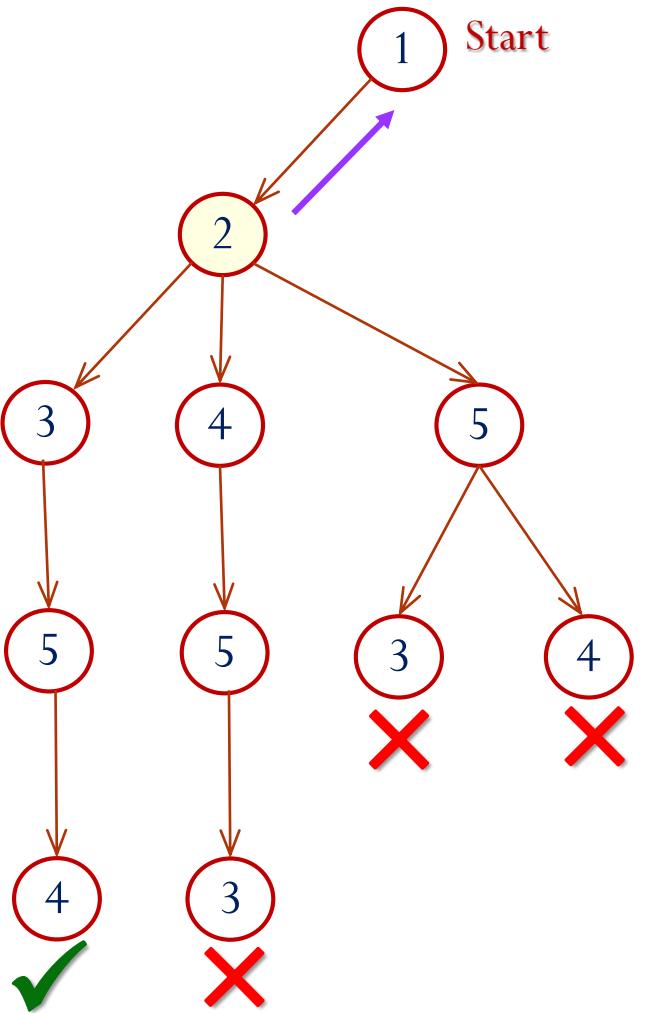


- ✓ For vertex 2, no possible vertex
- ✓ So, backtrack

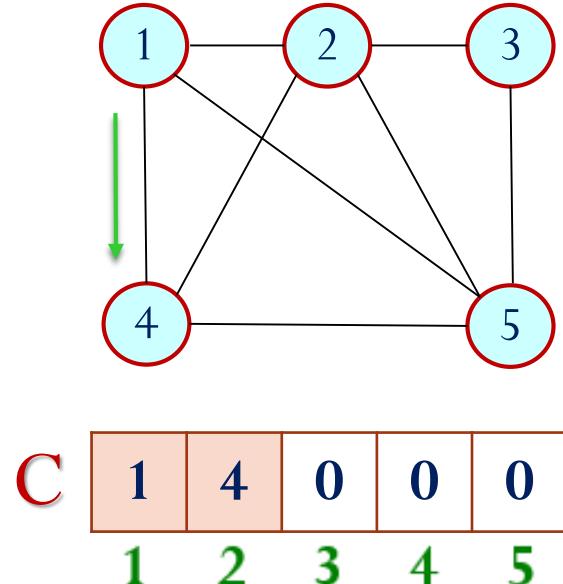
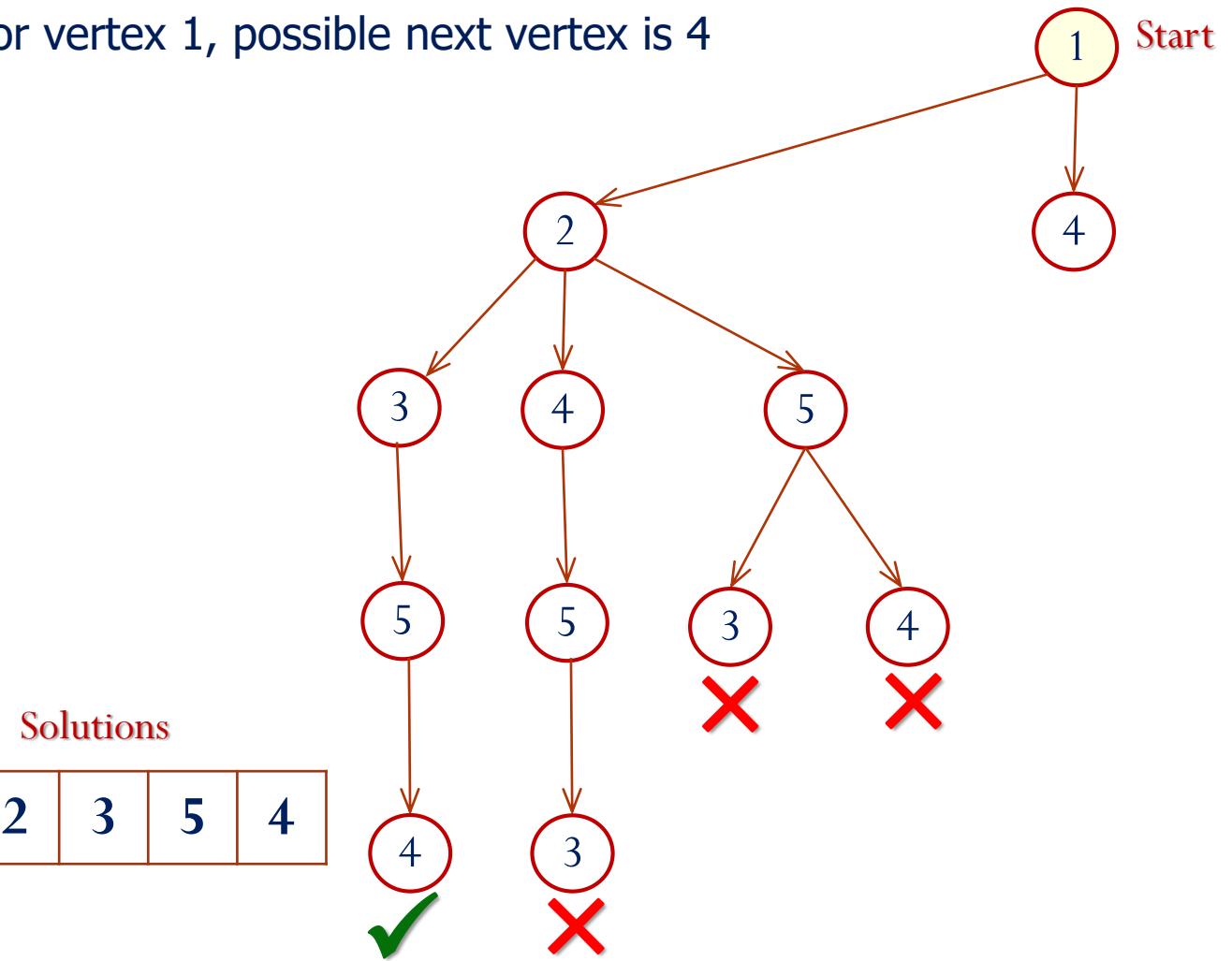


Solutions

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

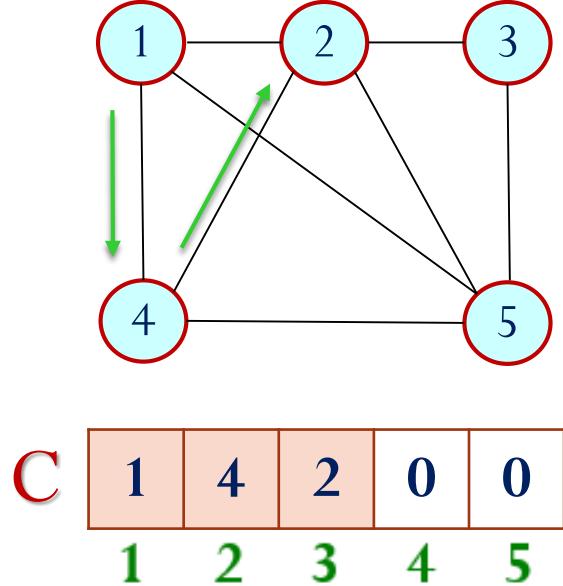
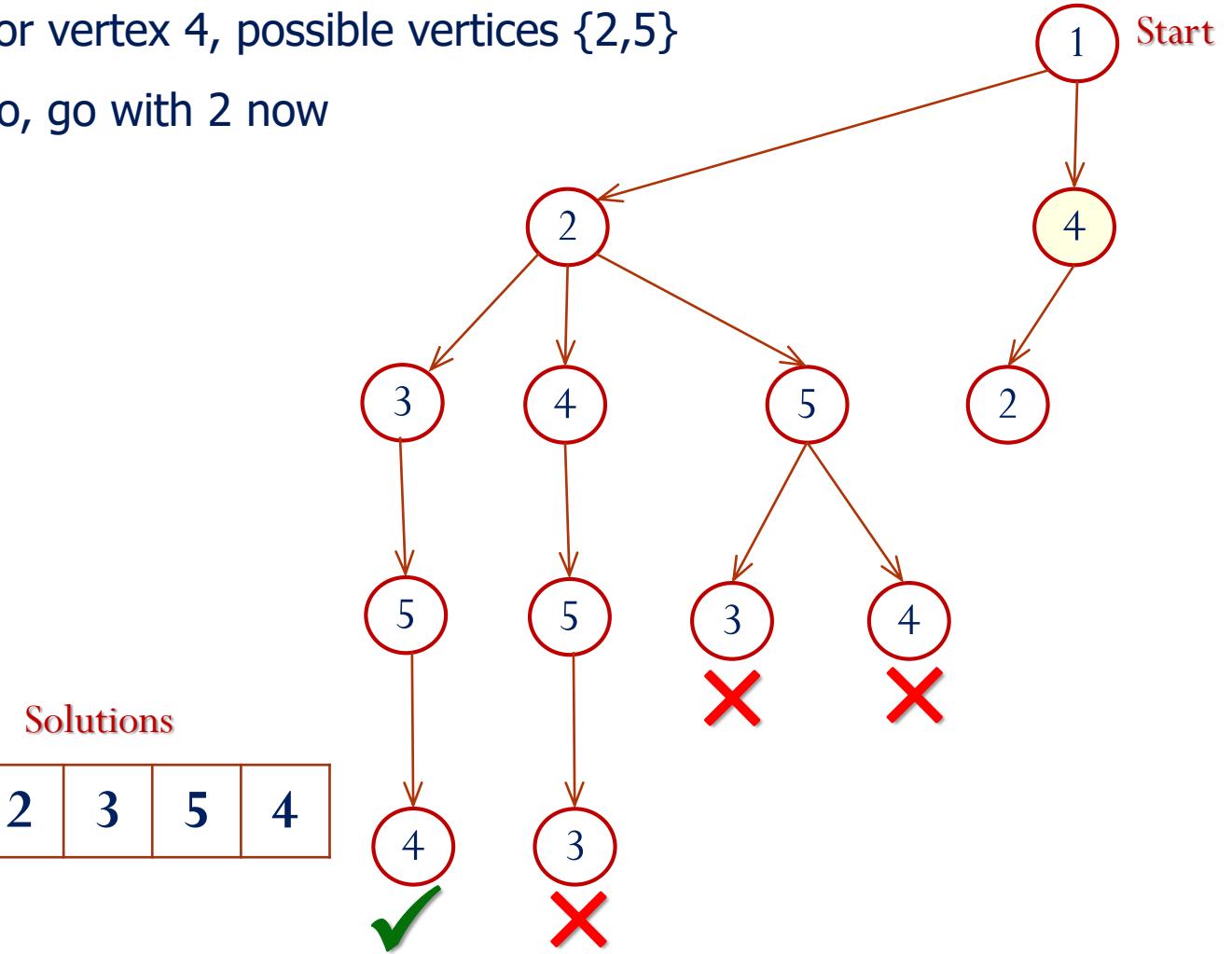


- ✓ For vertex 1, possible next vertex is 4



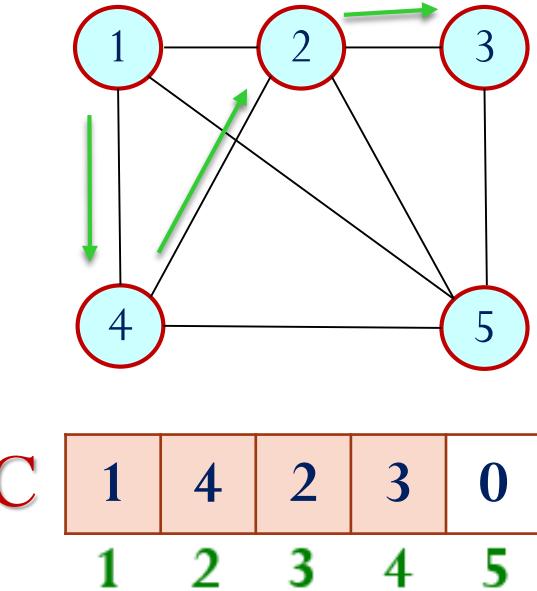
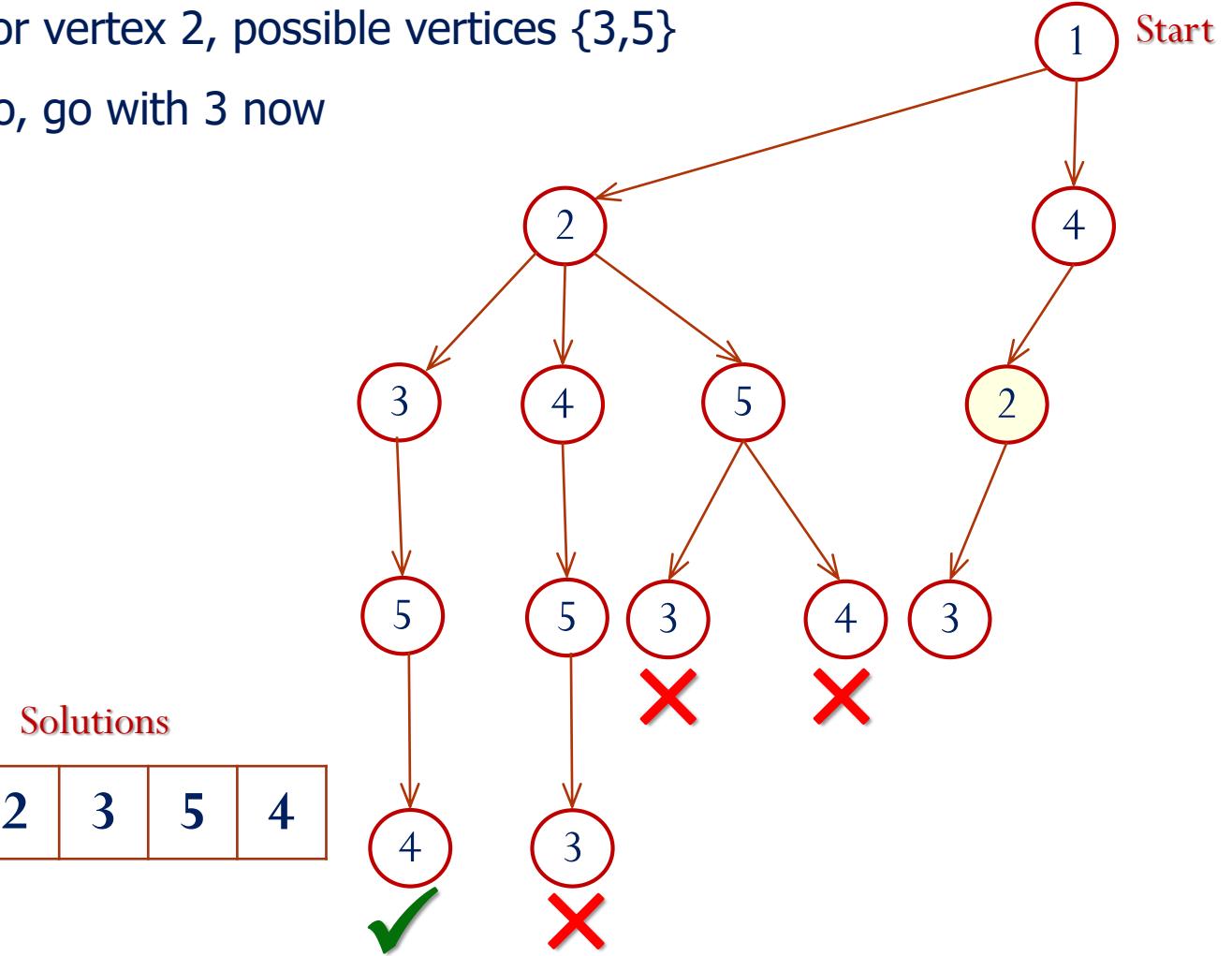
✓ For vertex 4, possible vertices {2,5}

✓ So, go with 2 now



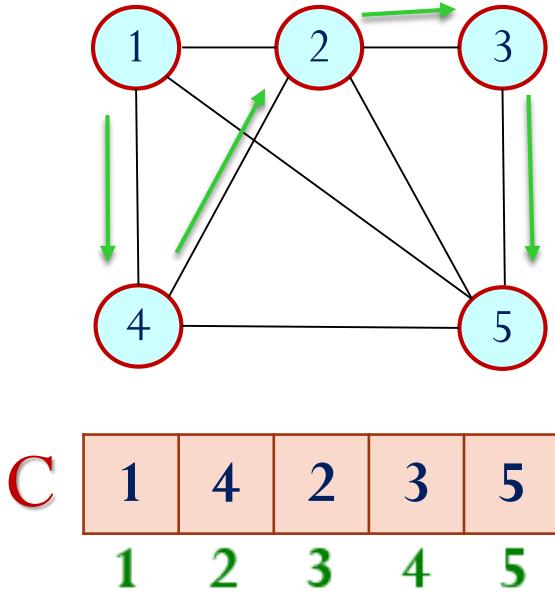
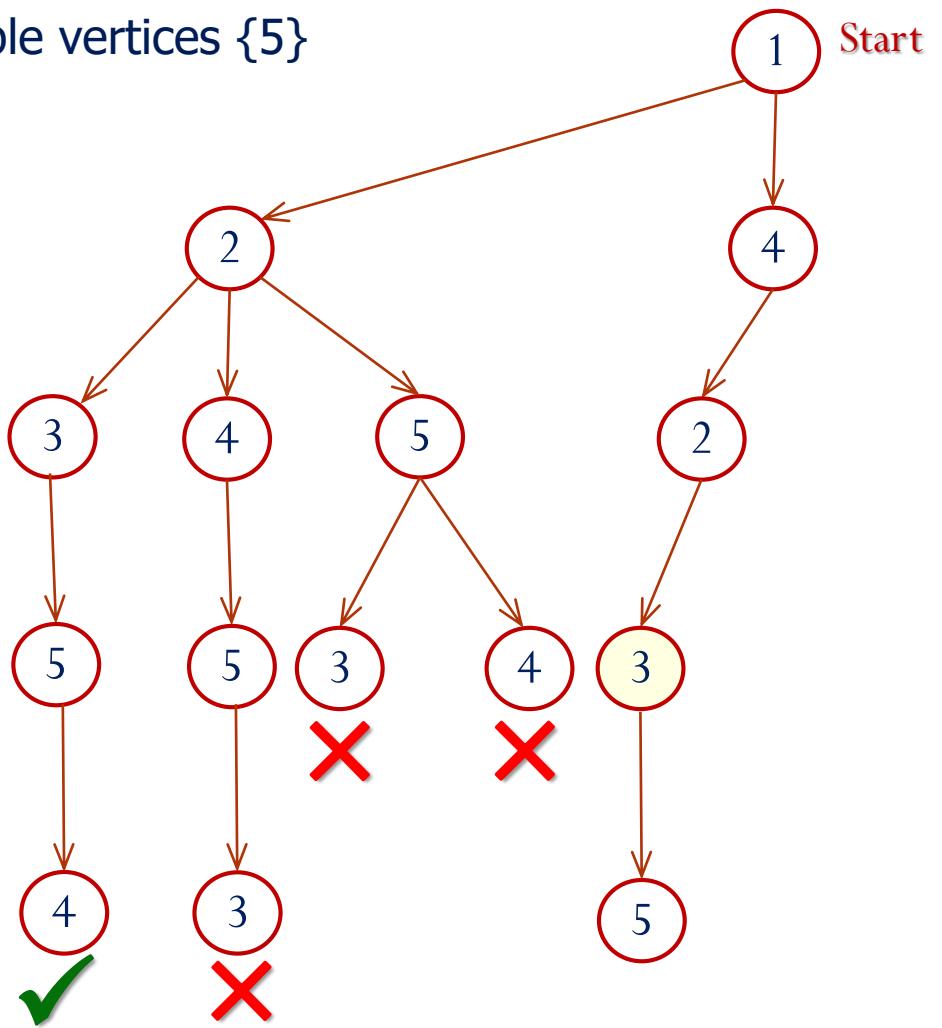
✓ For vertex 2, possible vertices {3,5}

✓ So, go with 3 now

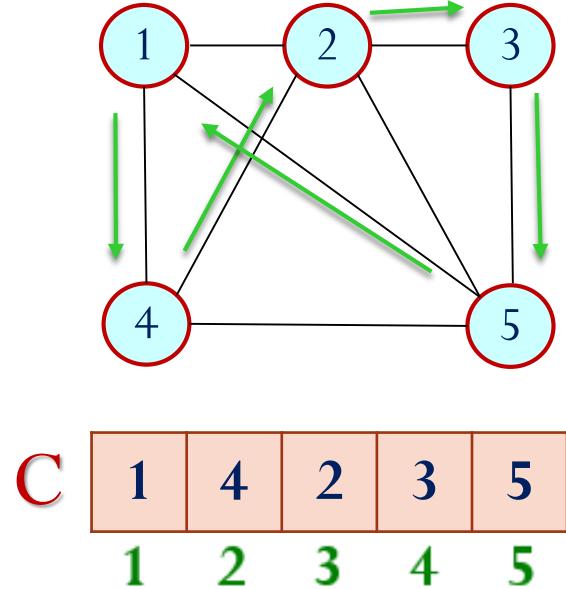
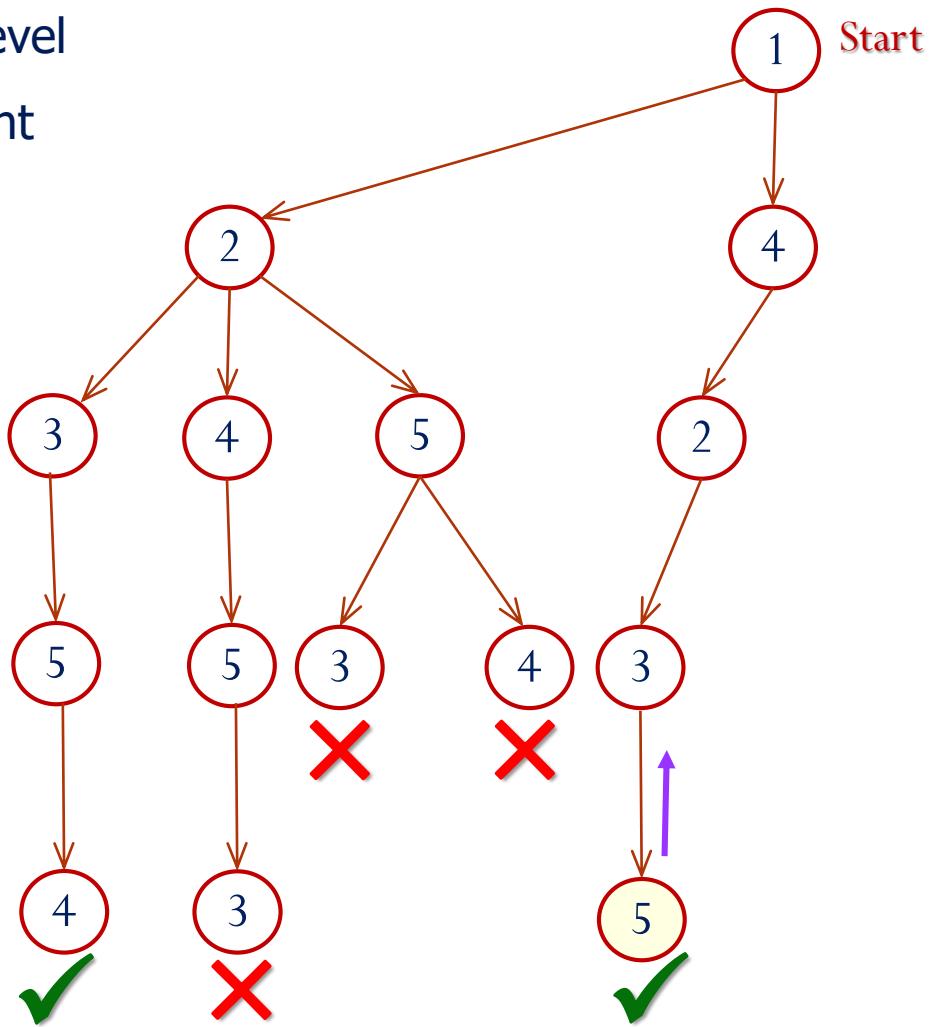


✓ For vertex 3, possible vertices {5}

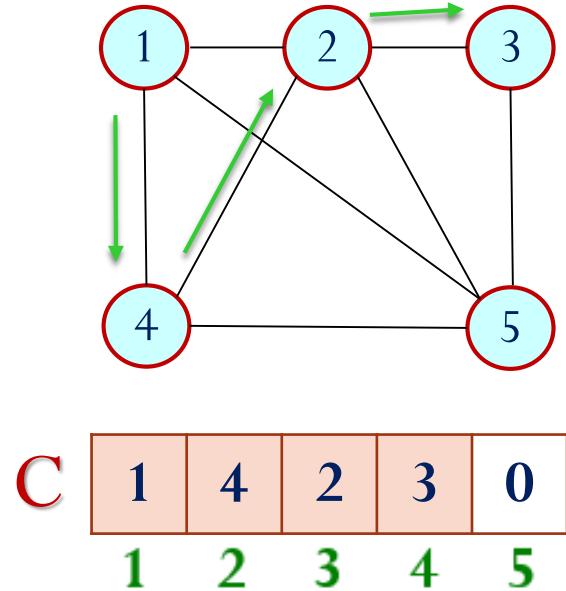
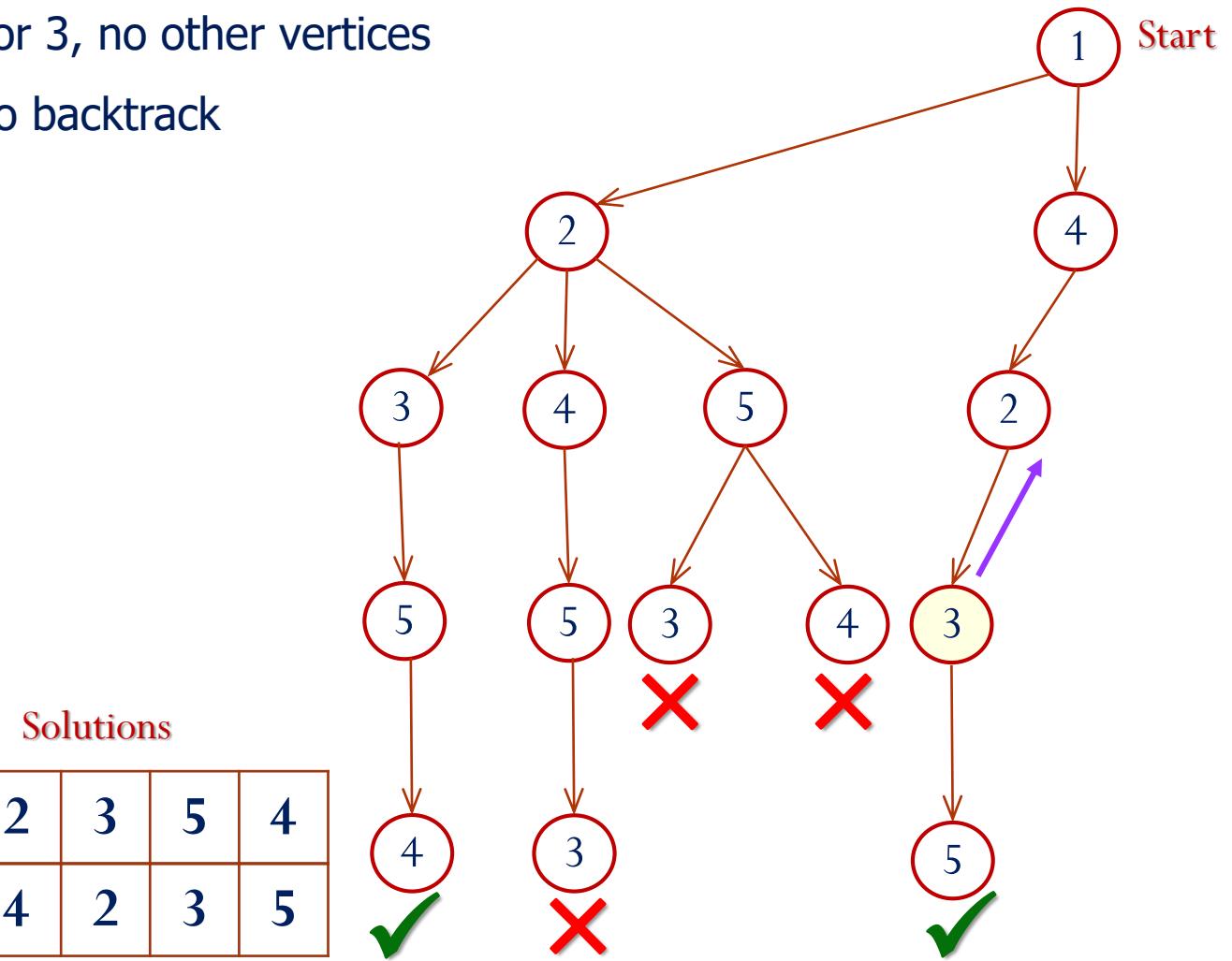
✓ So, go with 5 now



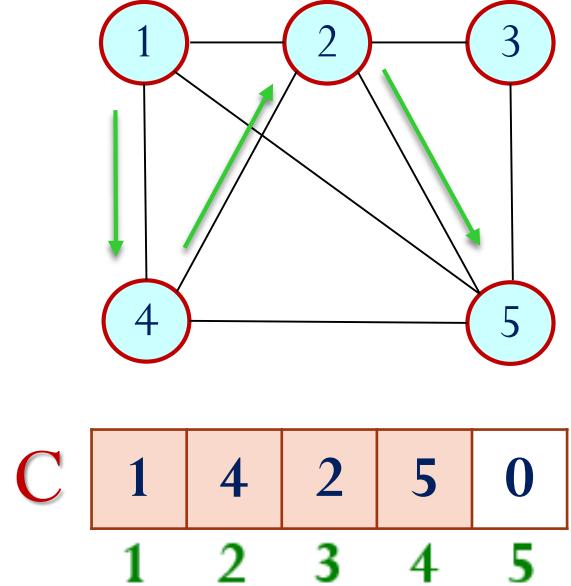
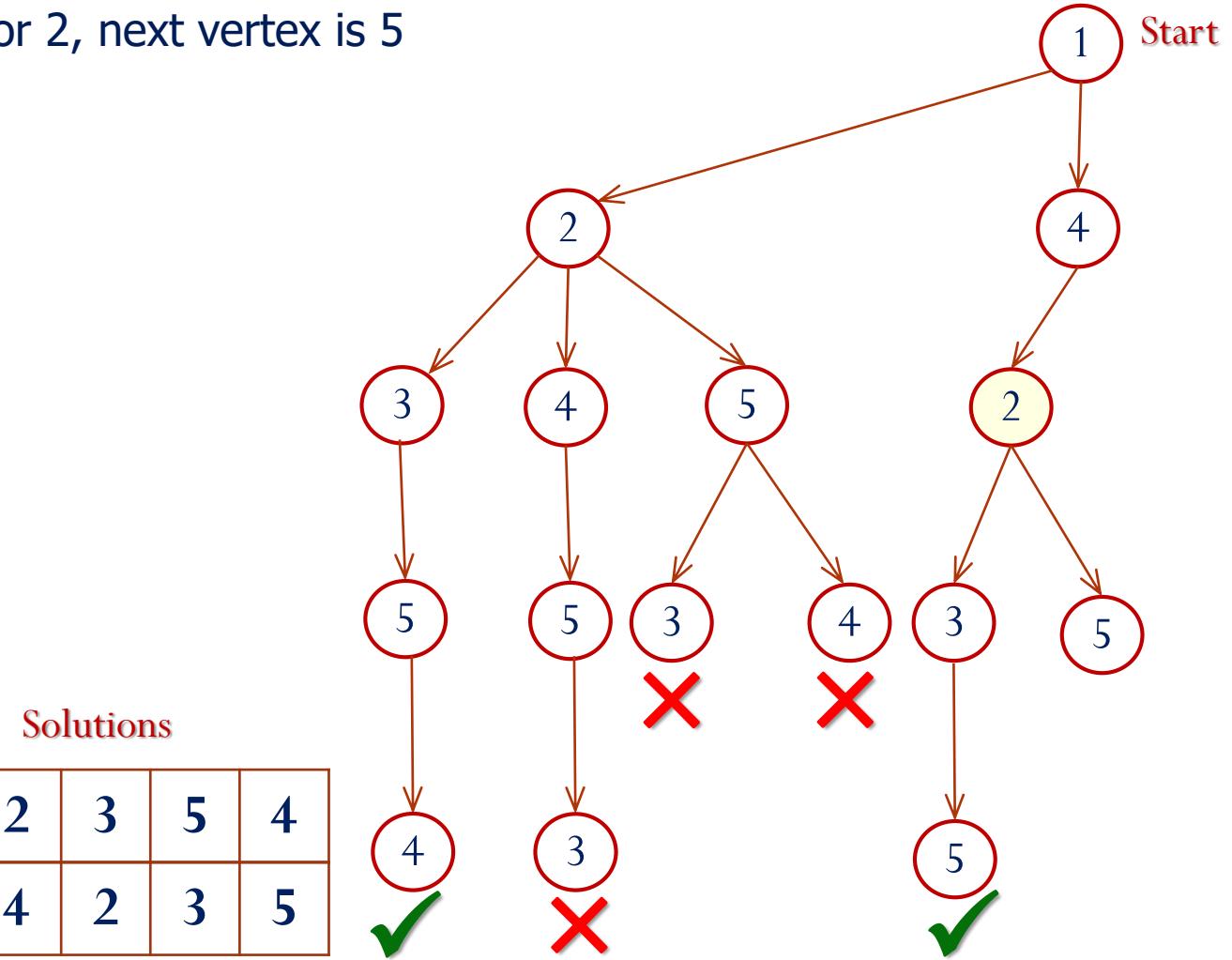
- ✓ Now reached last level
- ✓ (5,1) edge is present
- ✓ So, record solution
- ✓ And backtrack



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

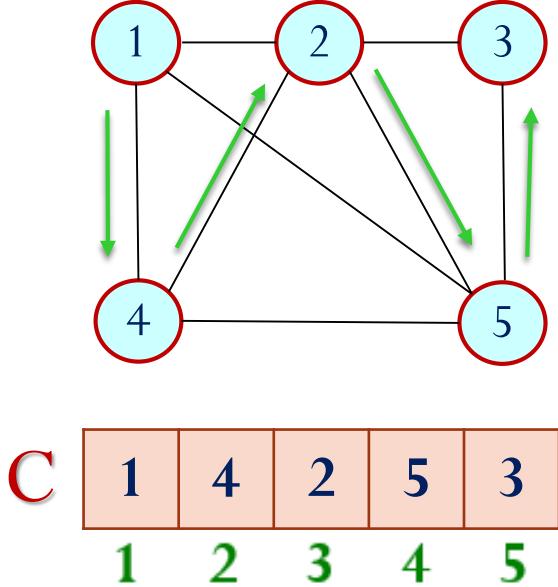
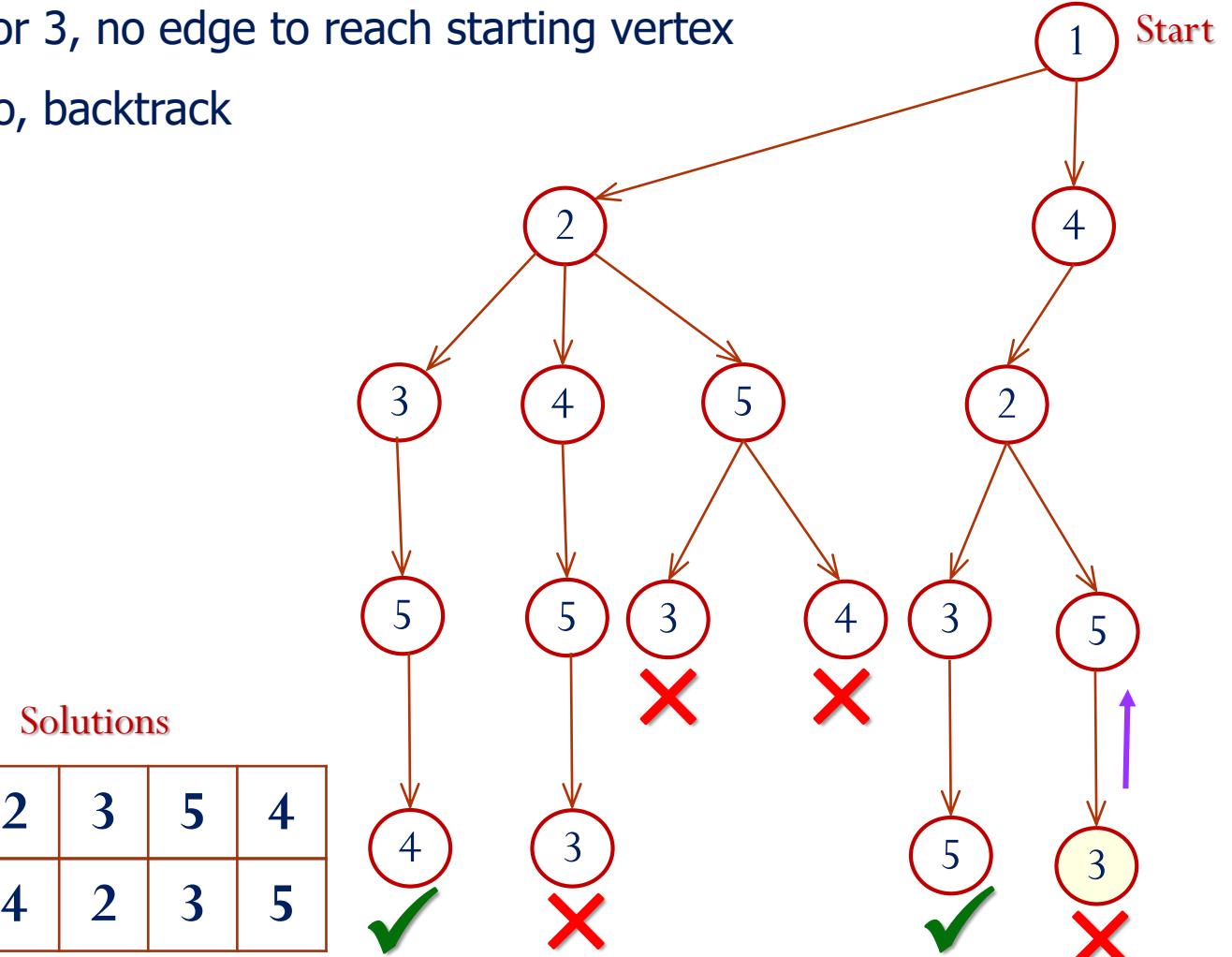


✓ For 2, next vertex is 5



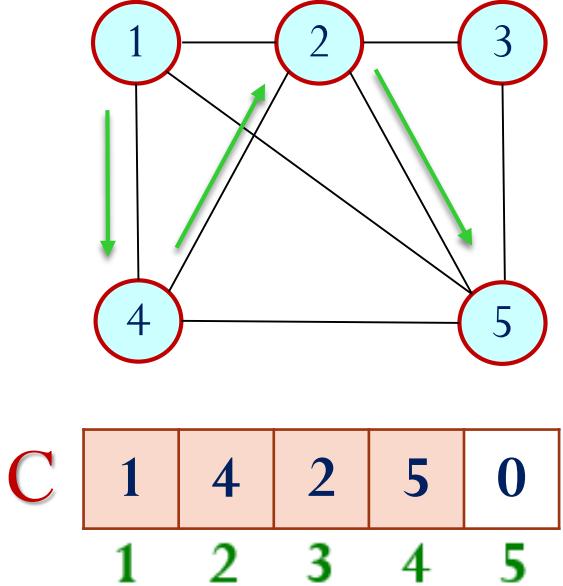
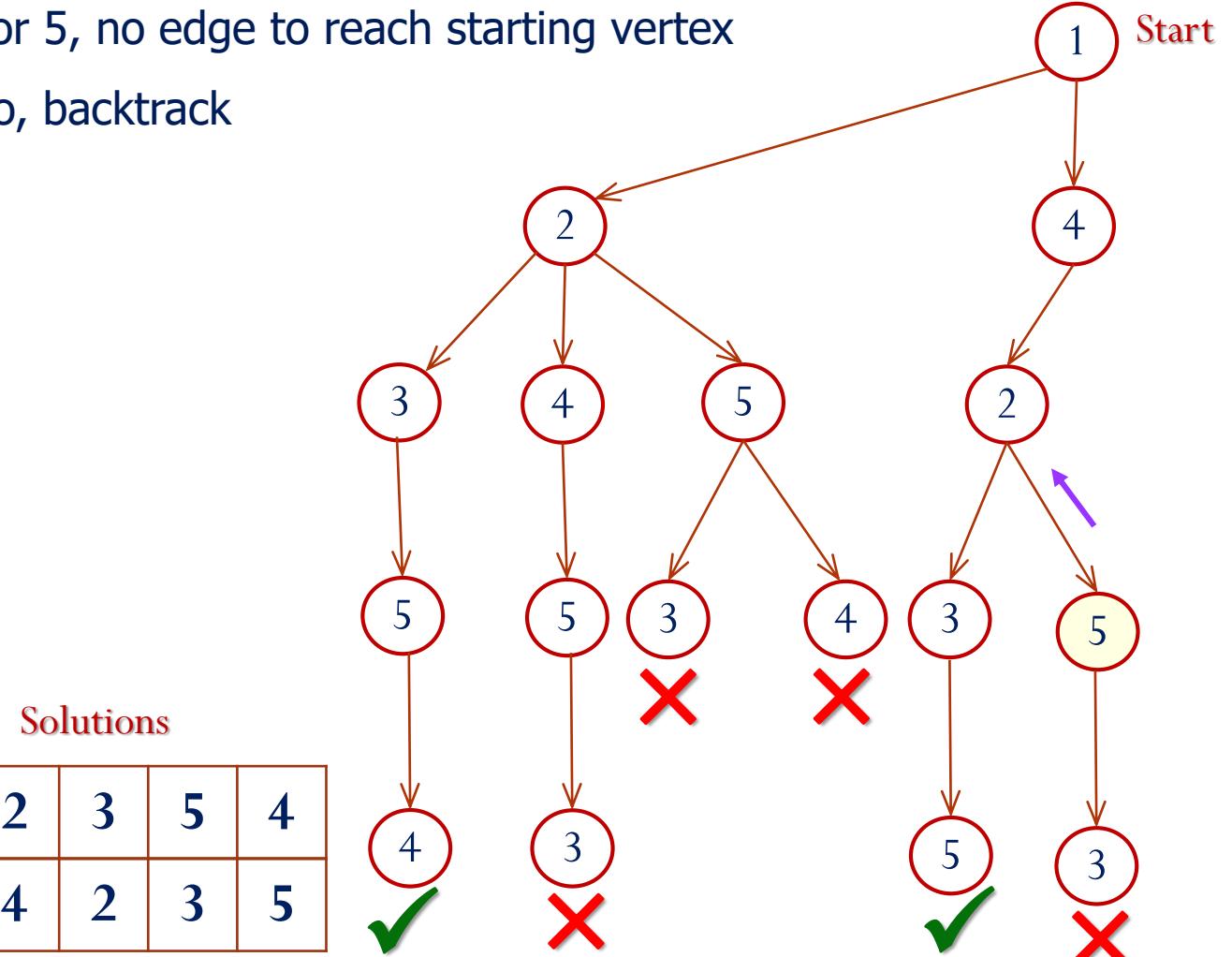
✓ For 3, no edge to reach starting vertex

✓ So, backtrack



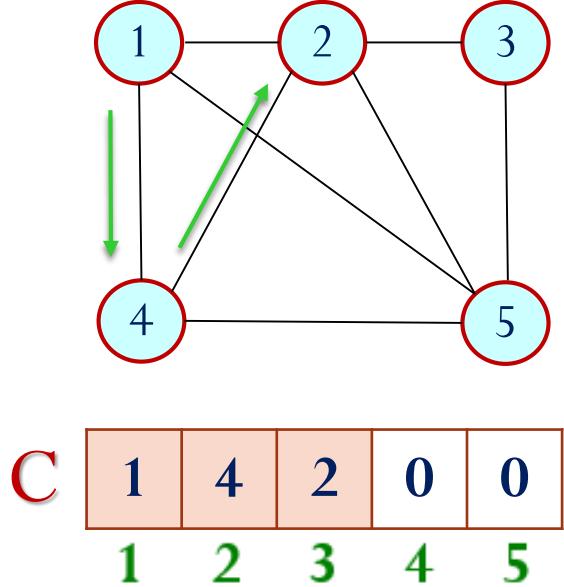
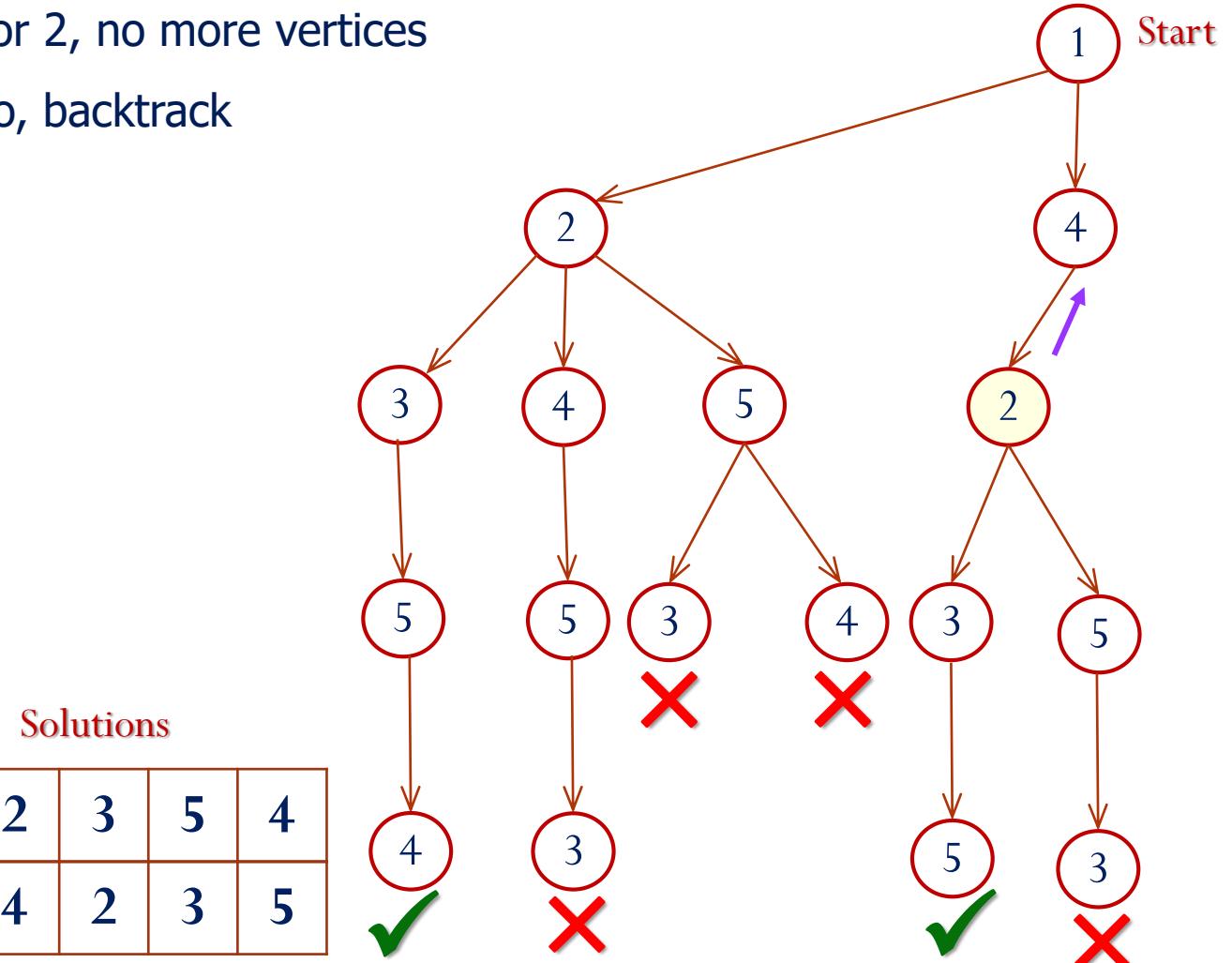
✓ For 5, no edge to reach starting vertex

✓ So, backtrack

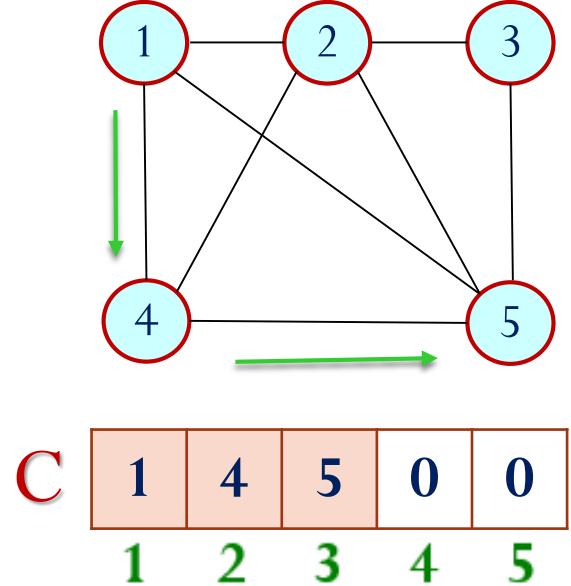
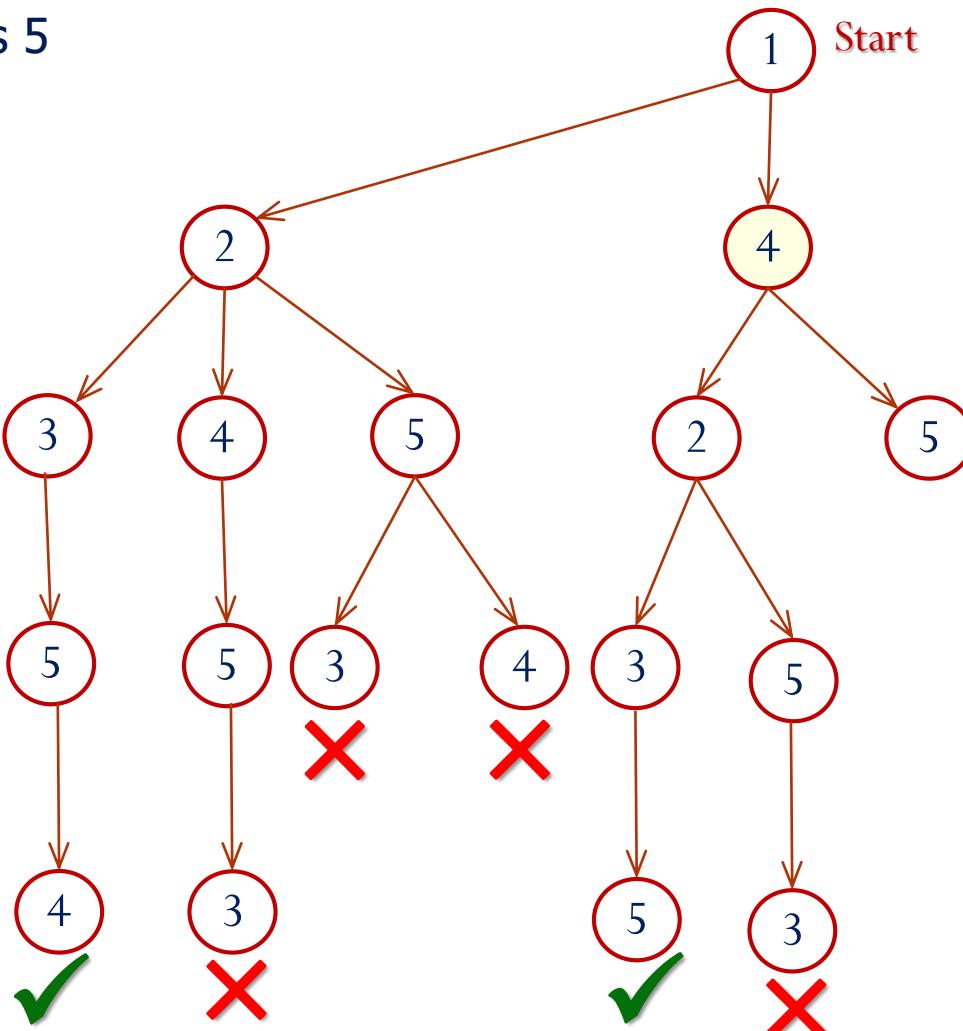


✓ For 2, no more vertices

✓ So, backtrack



✓ For 4, next vertex is 5

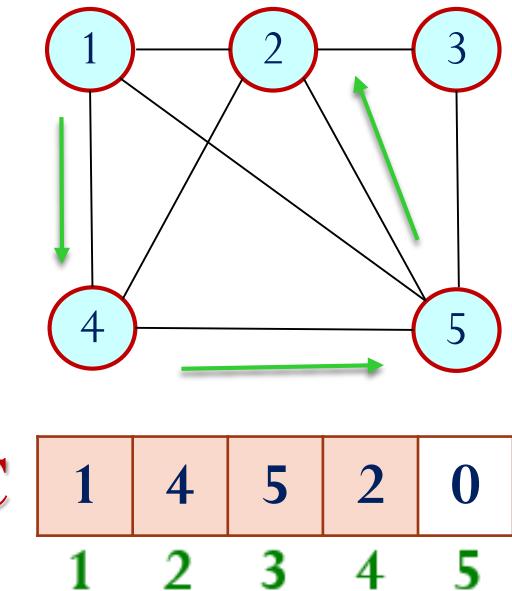
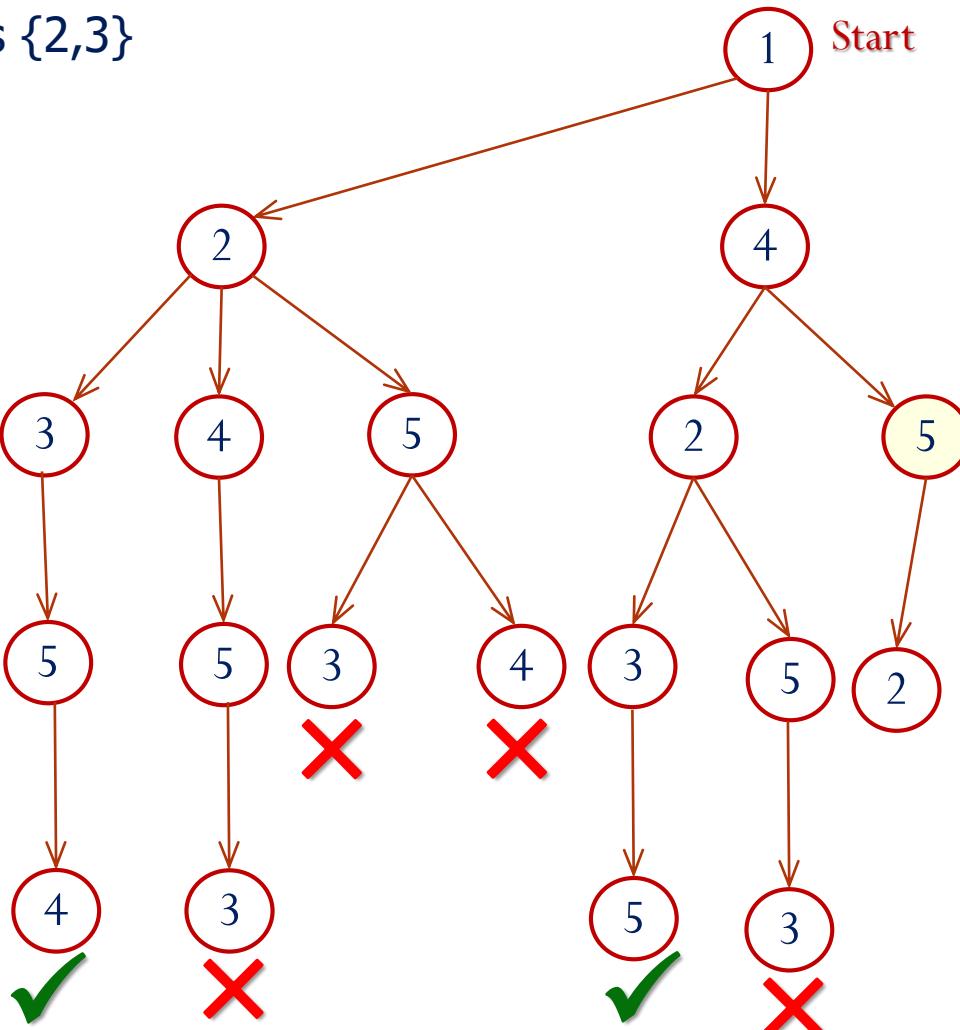


✓ For 5, next vertex is {2,3}

✓ So, proceed with 2

Solutions

1	2	3	5	4
1	4	2	3	5

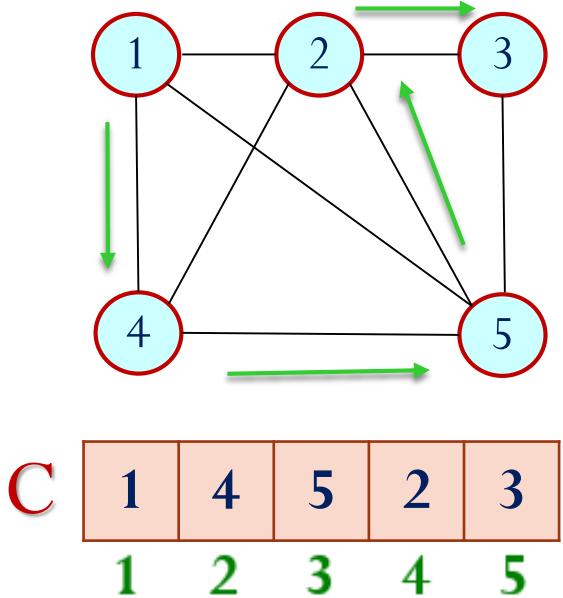
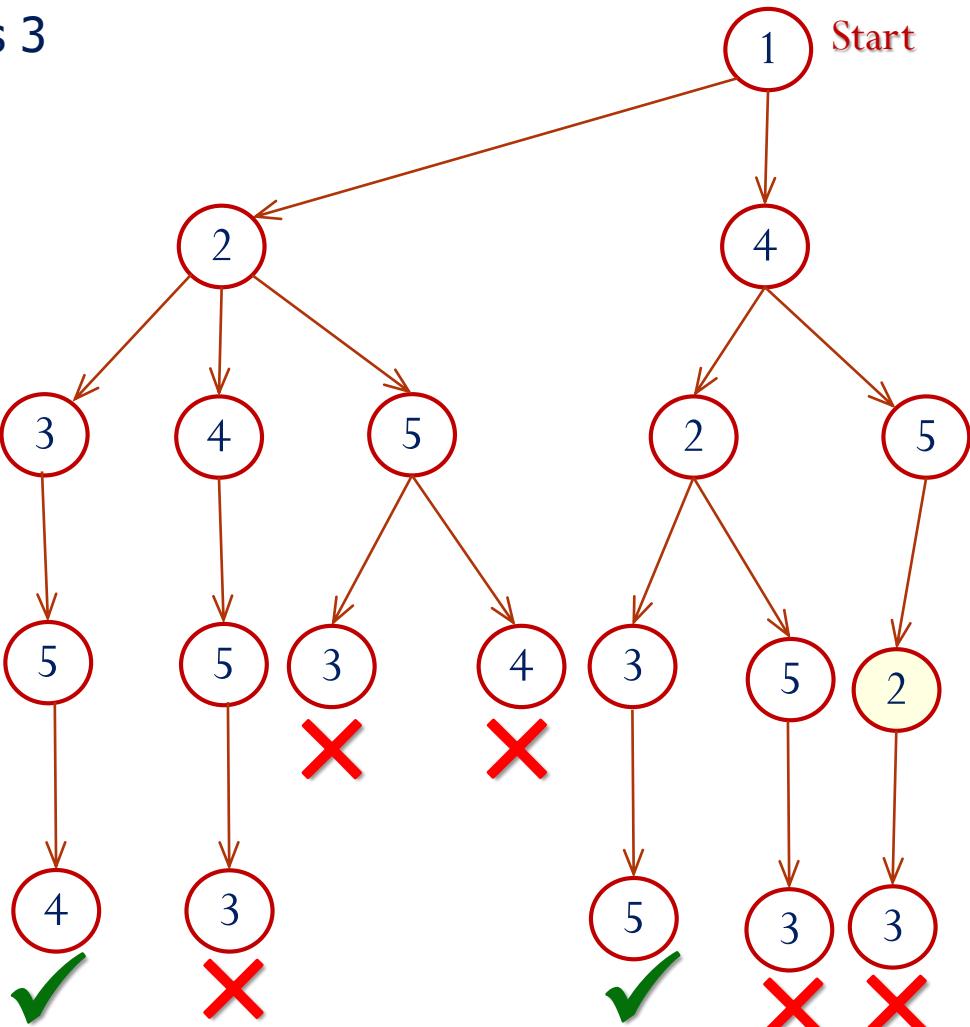


✓ For 2, next vertex is 3

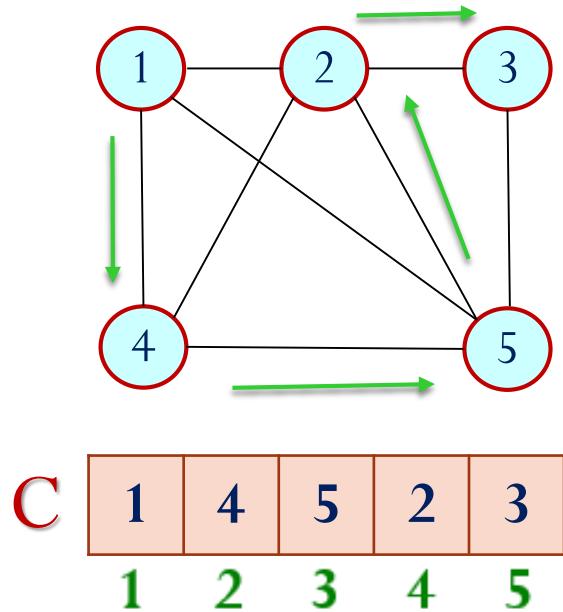
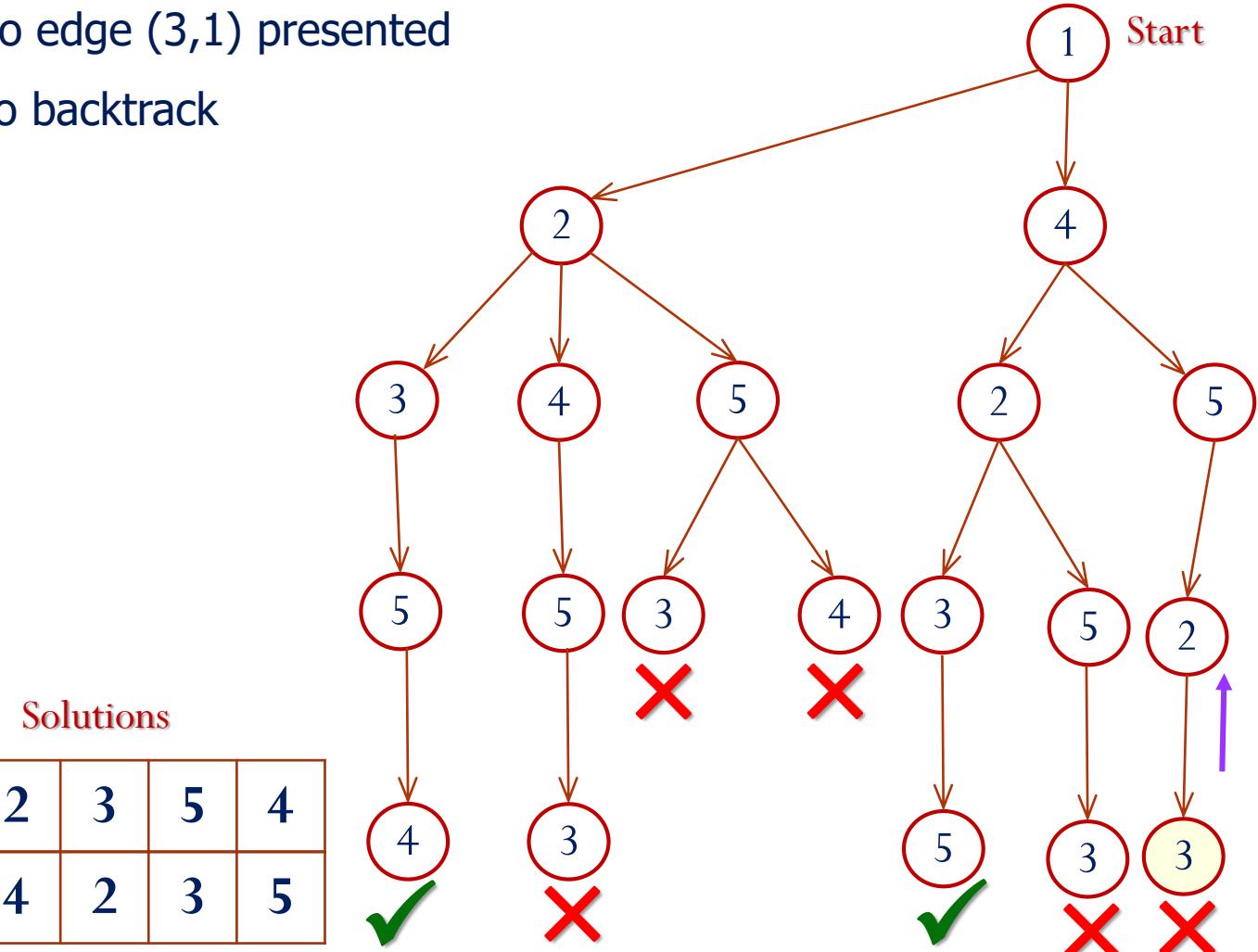
✓ So, proceed with 3

Solutions

1	2	3	5	4
1	4	2	3	5

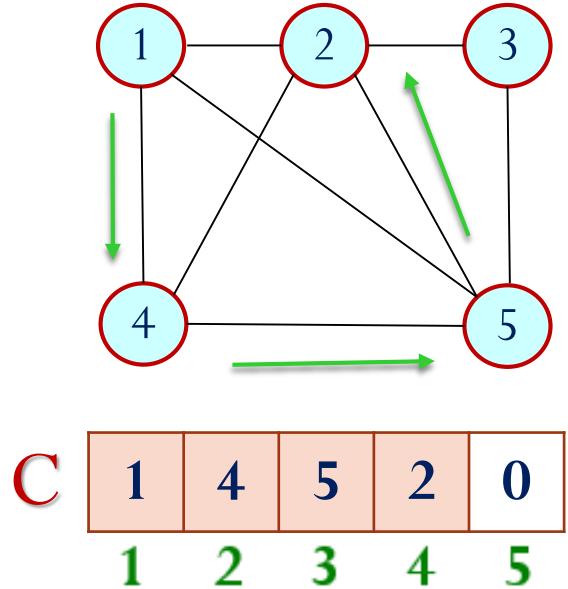
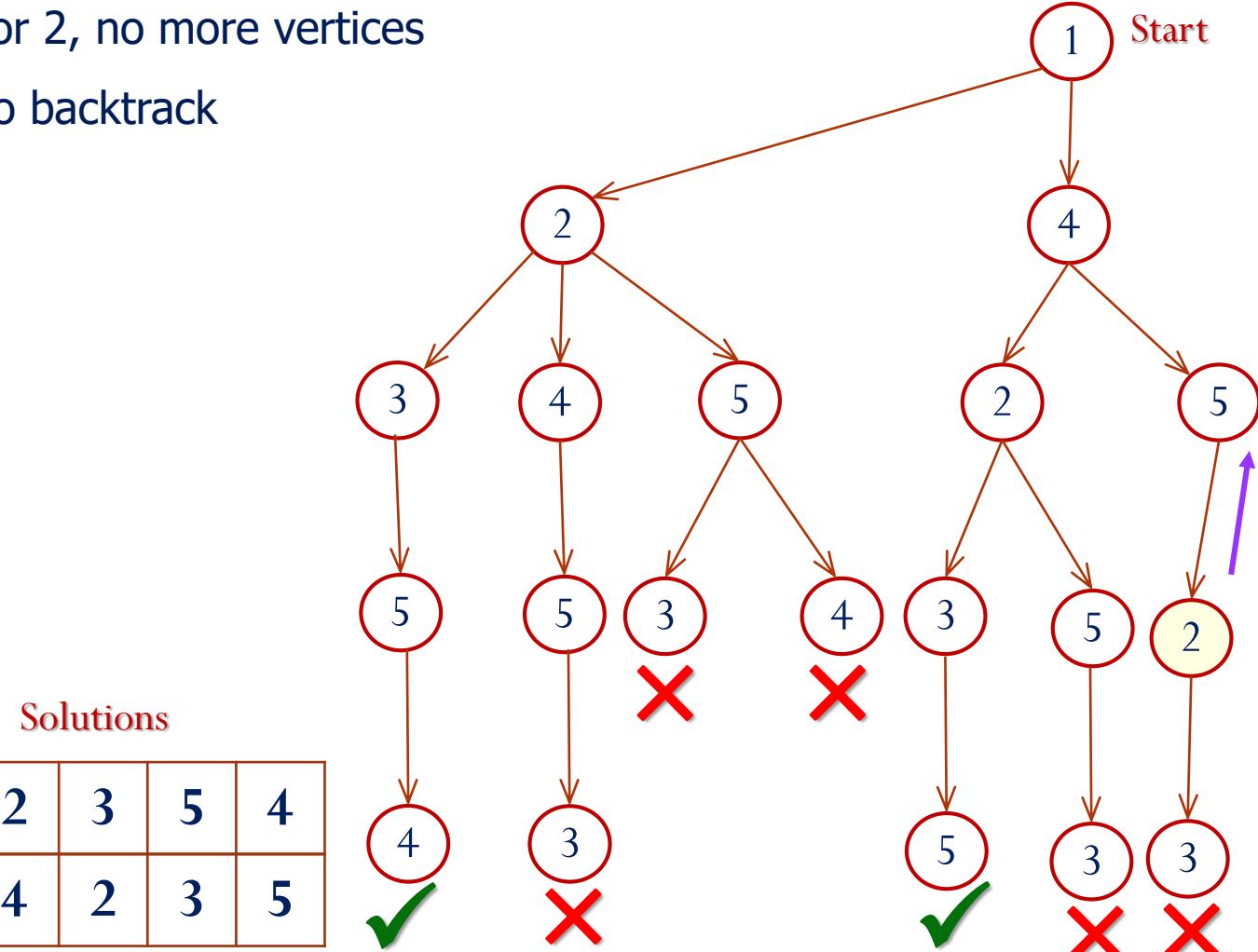


- ✓ No edge (3,1) presented
- ✓ So backtrack



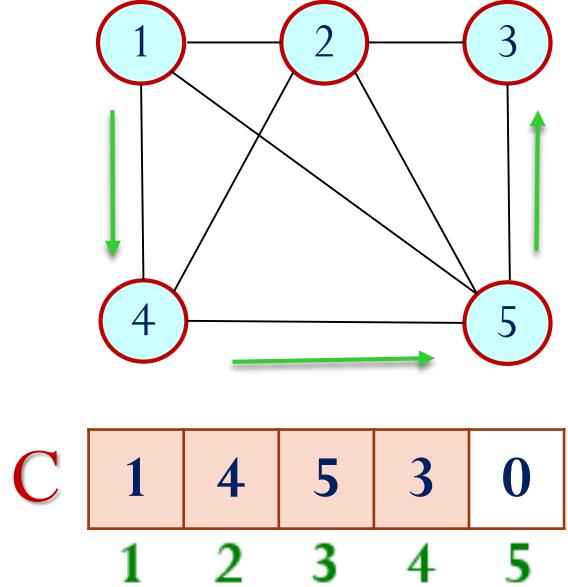
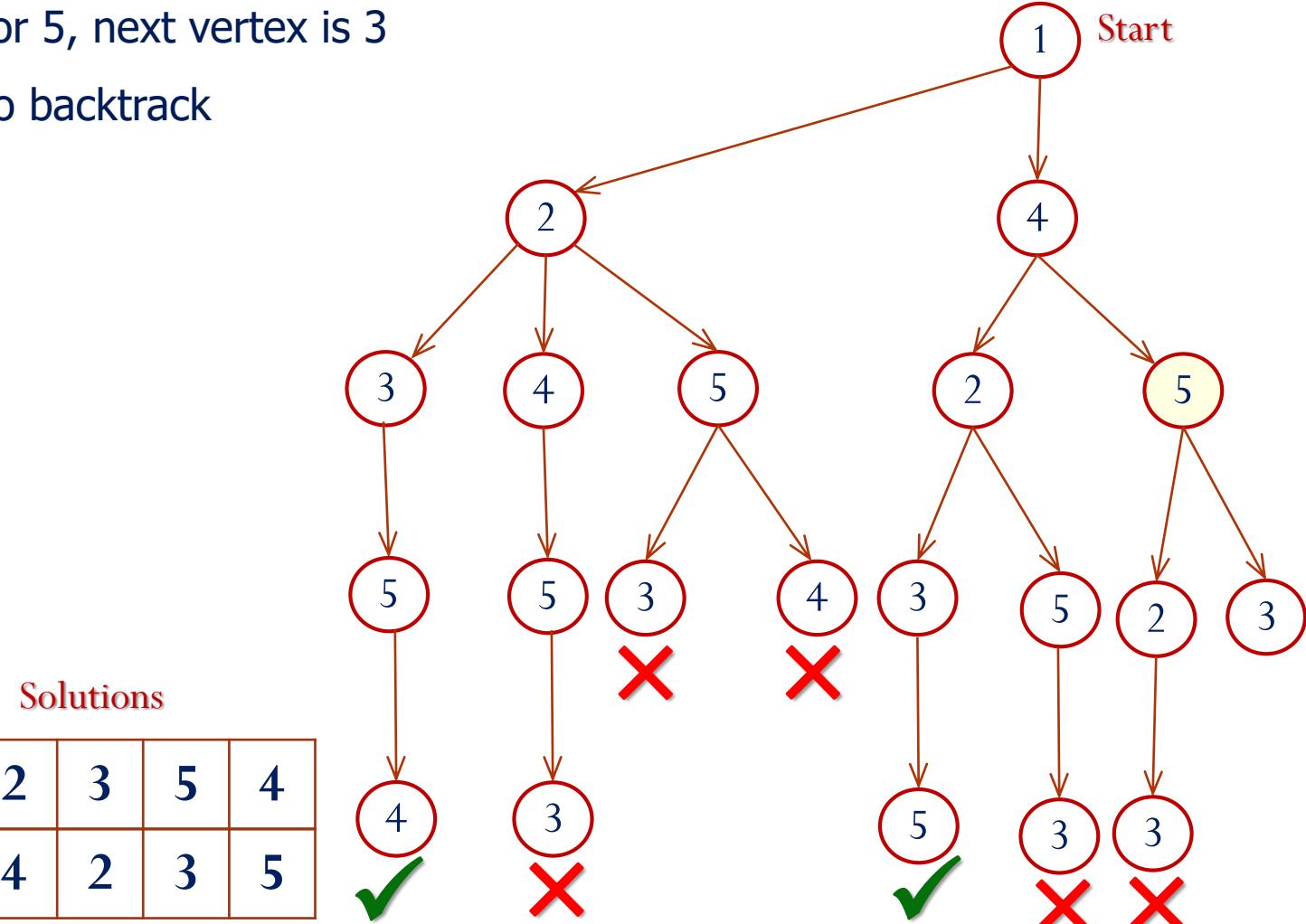
✓ For 2, no more vertices

✓ So backtrack

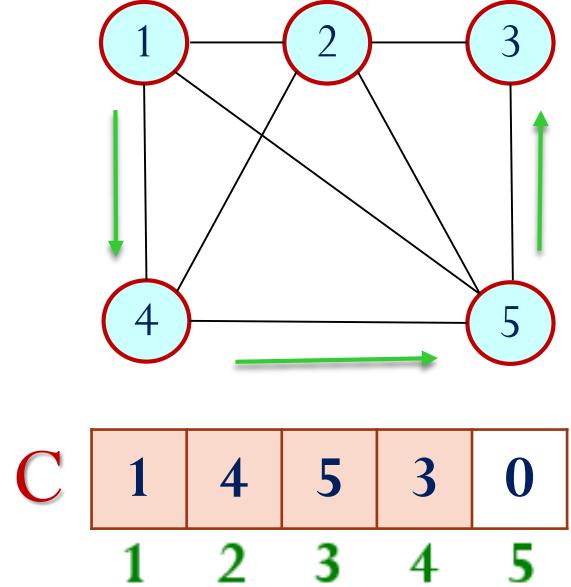
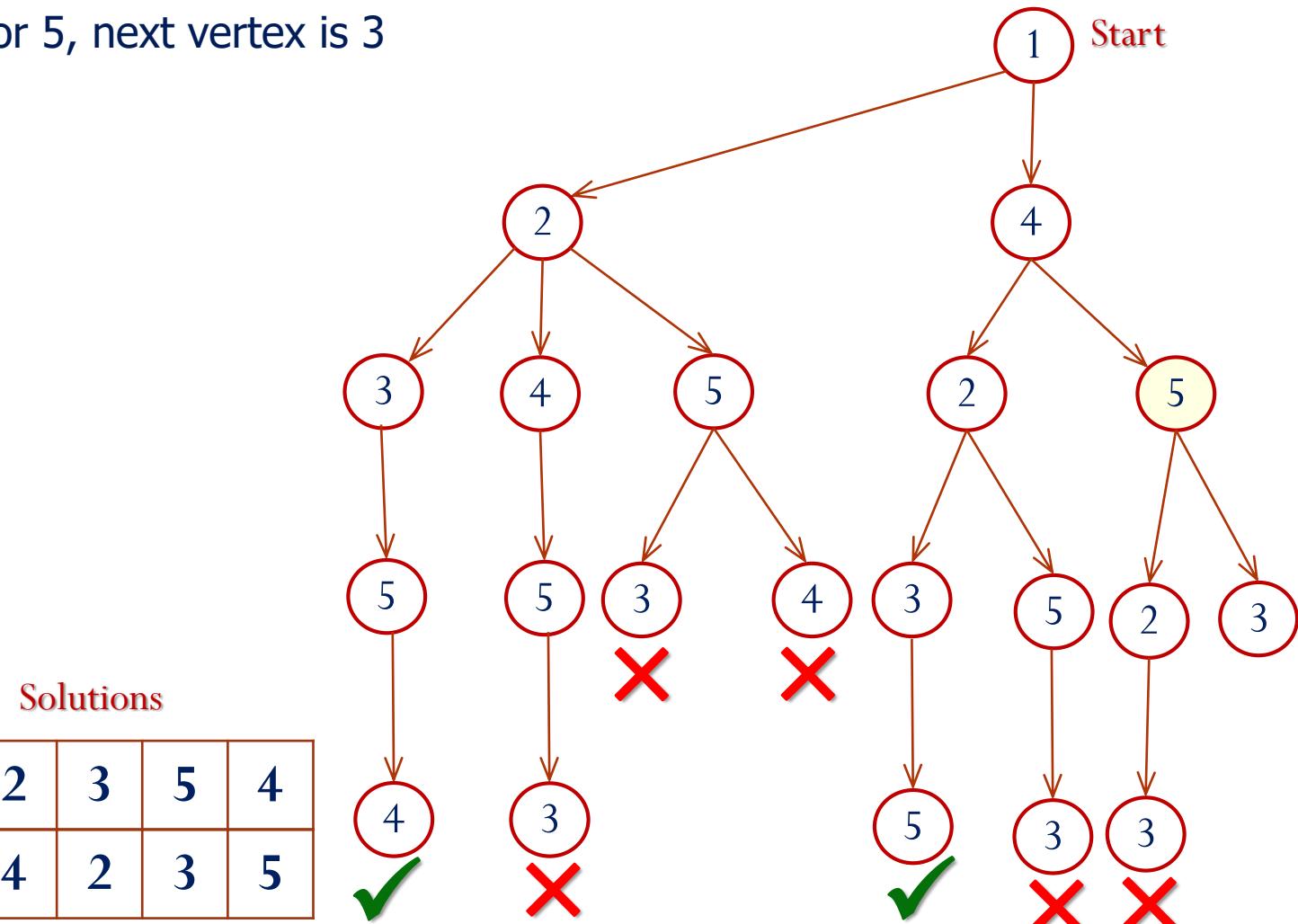


✓ For 5, next vertex is 3

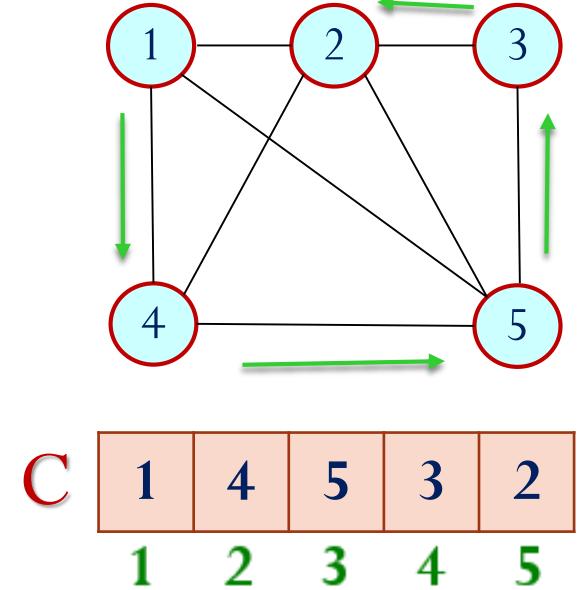
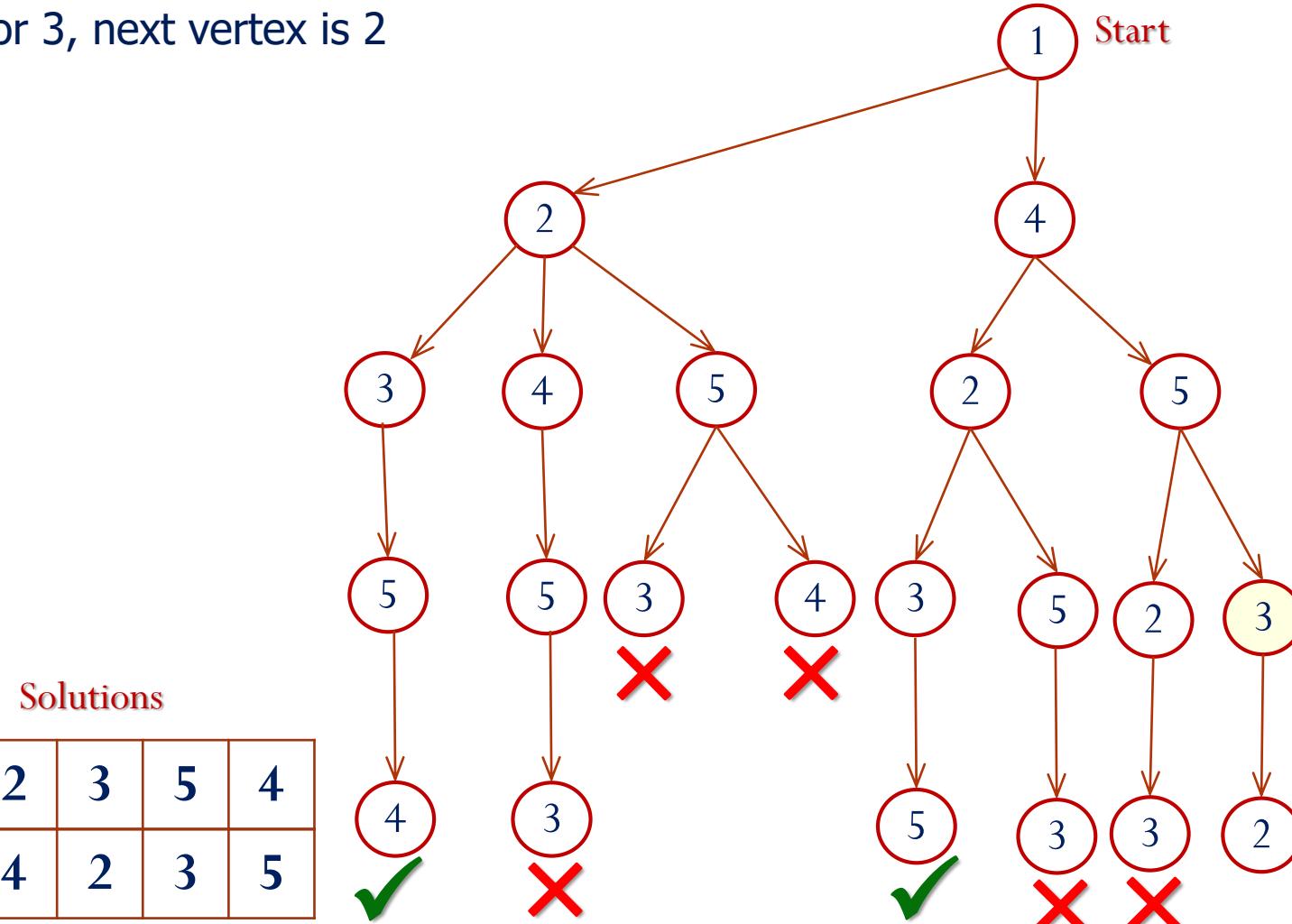
✓ So backtrack



✓ For 5, next vertex is 3



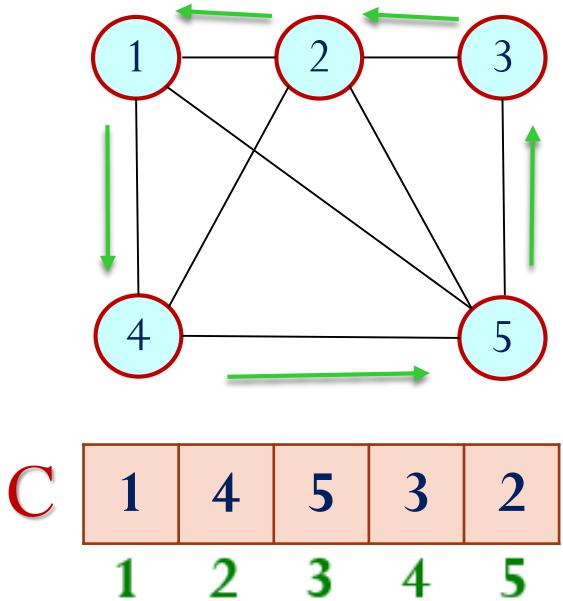
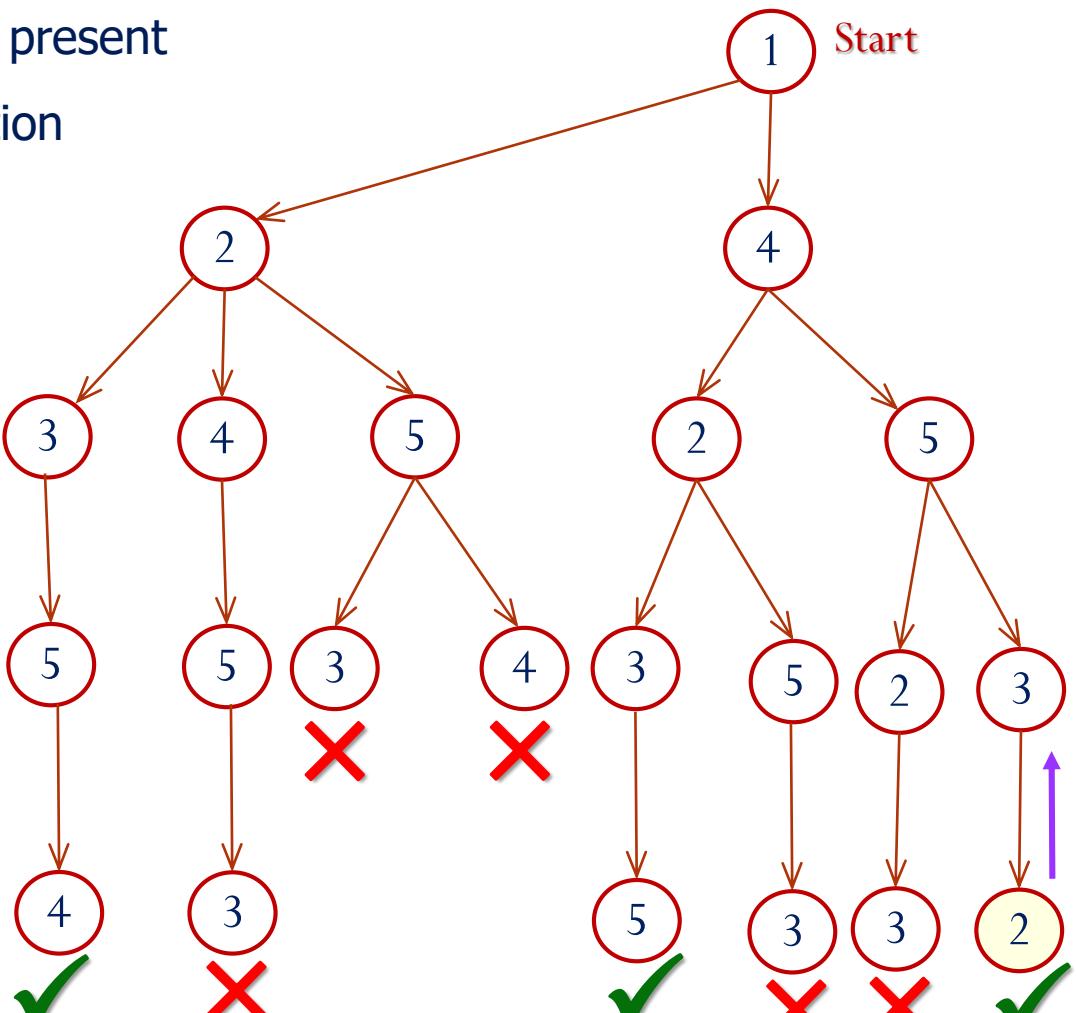
✓ For 3, next vertex is 2



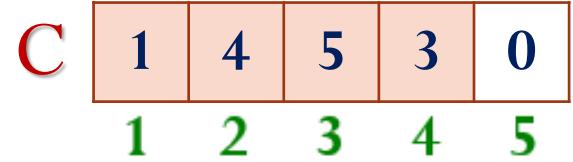
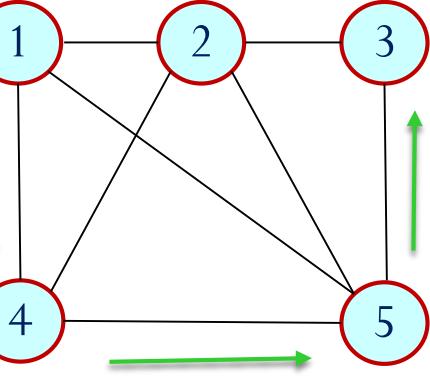
✓ For 2, (2,1) edge is present

✓ So, record the solution

✓ And backtrack

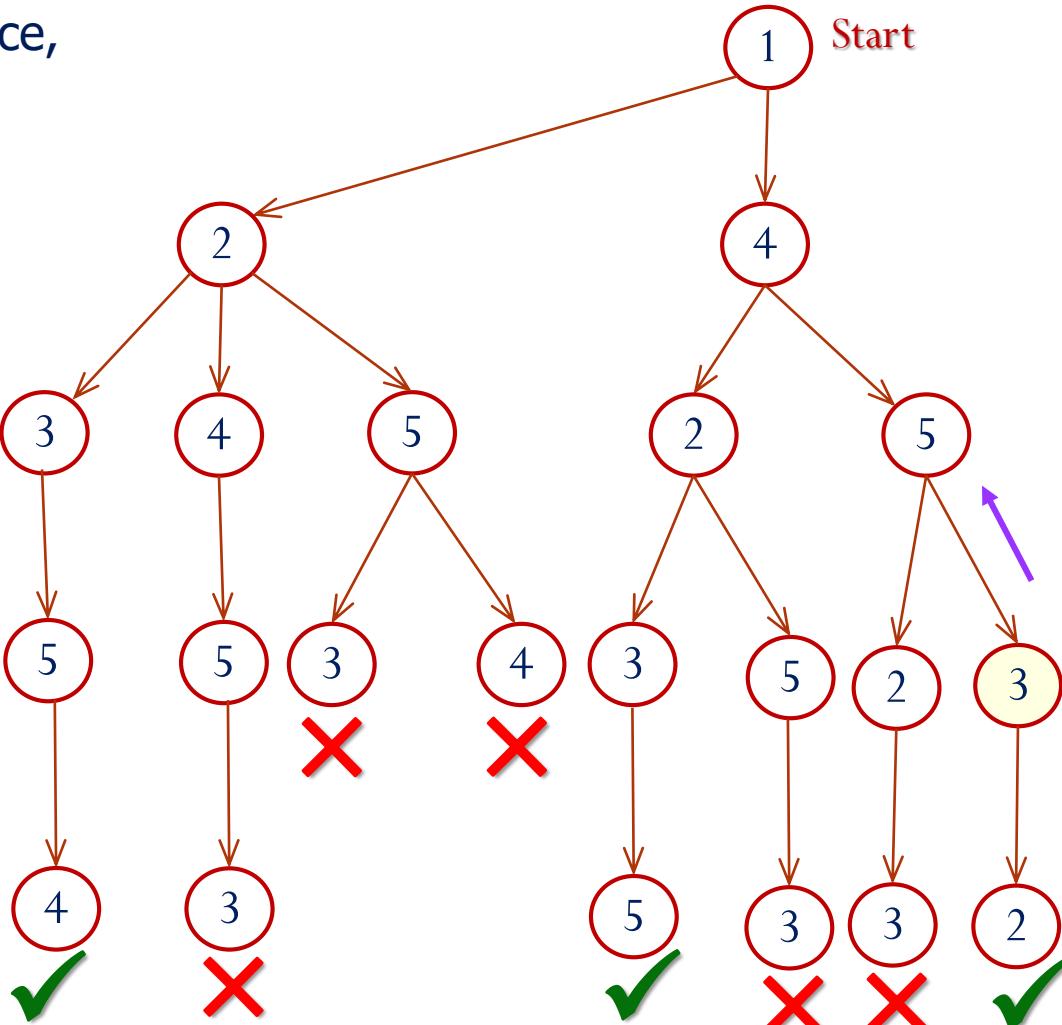


- ✓ For 3, no more choice,
- ✓ So, backtrack

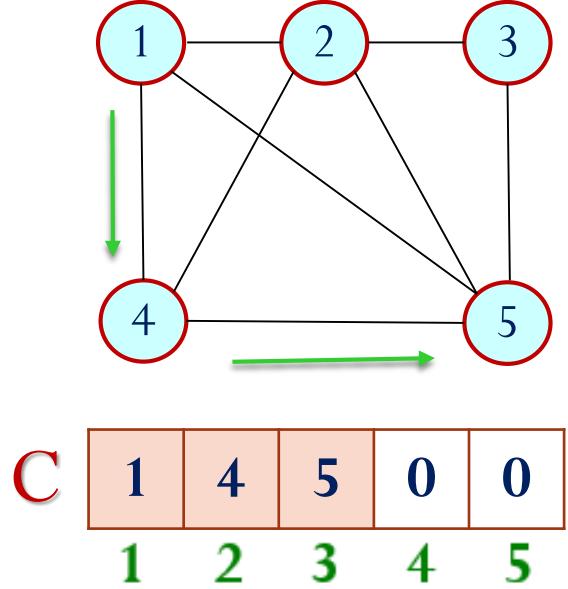
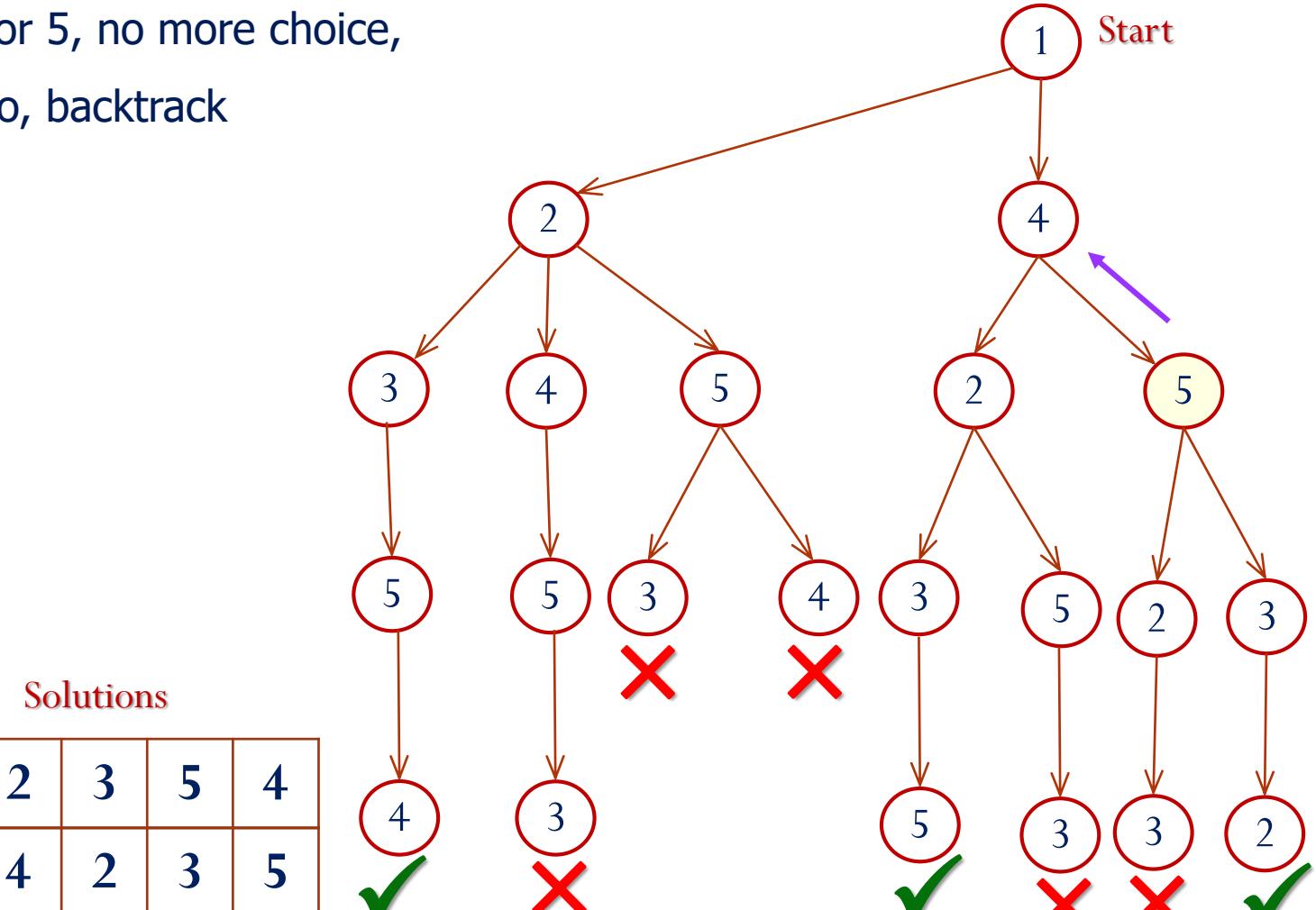


Solutions

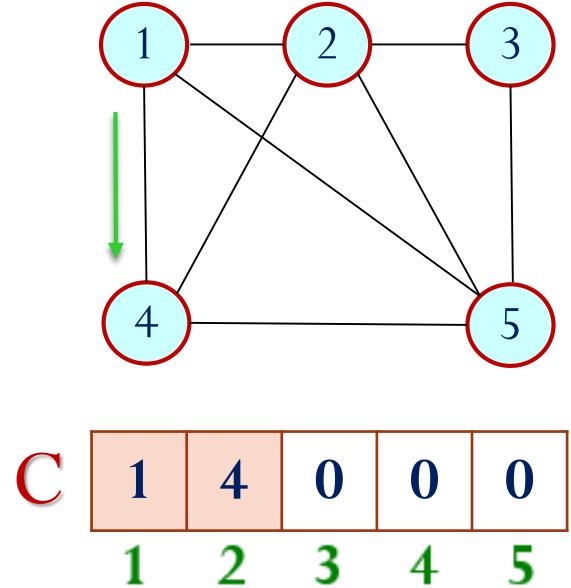
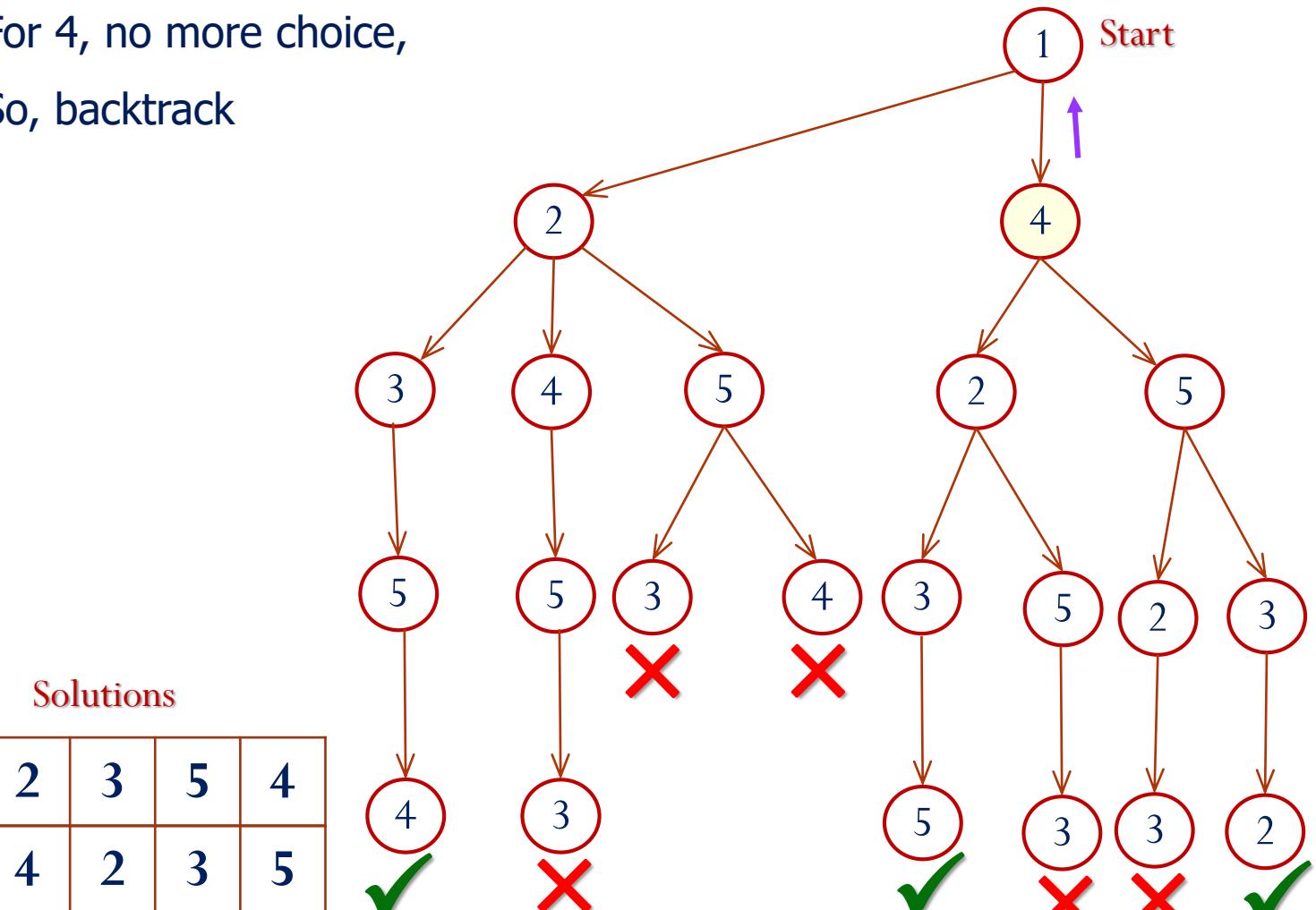
1	2	3	5	4
1	4	2	3	5
1	4	5	3	2



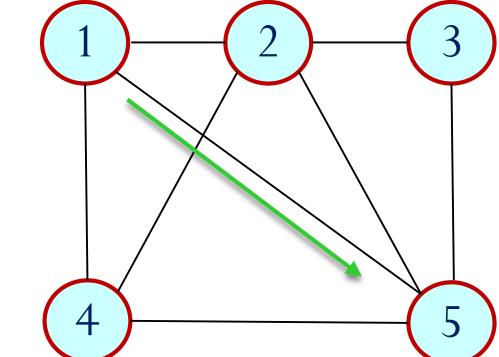
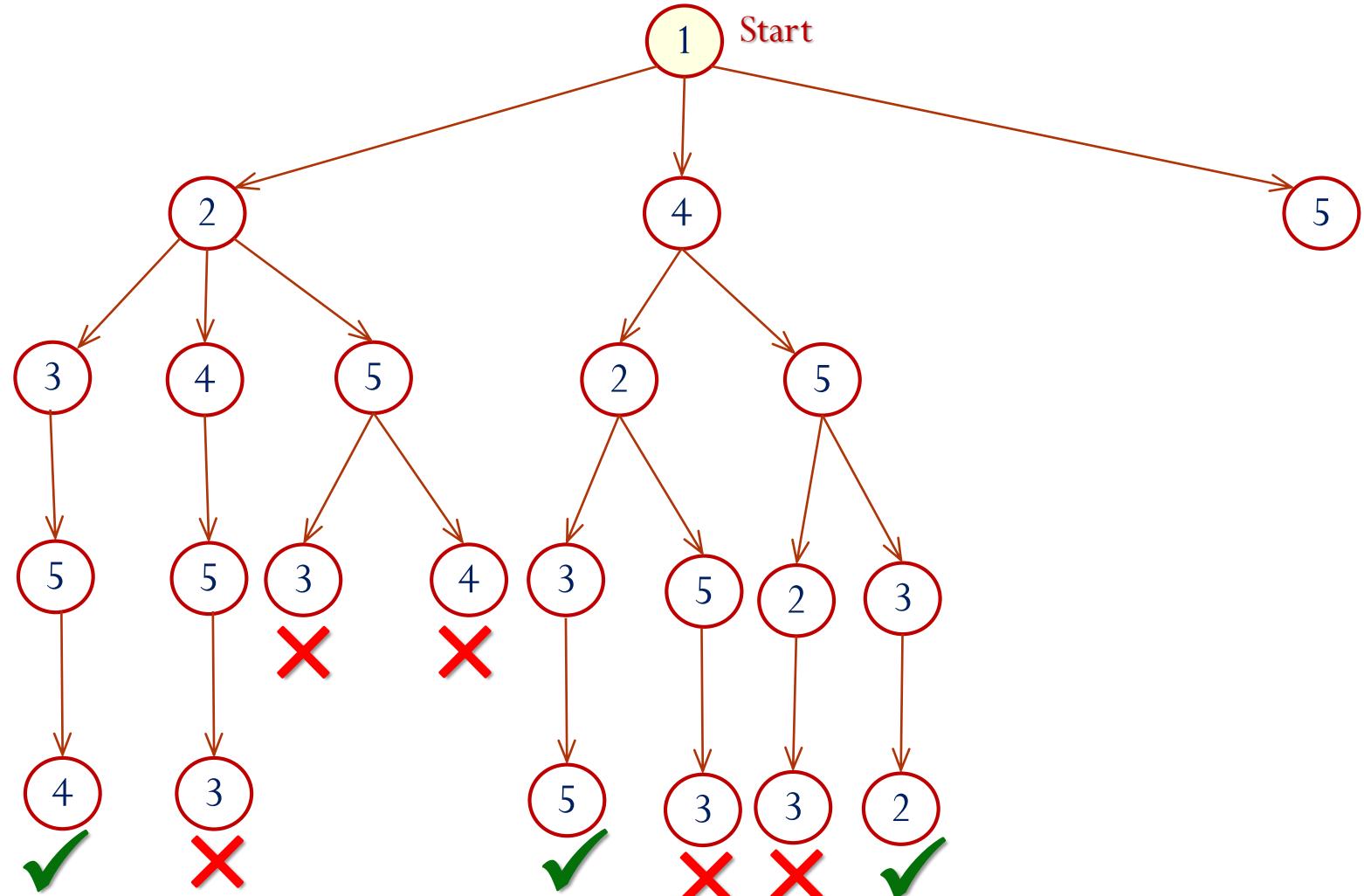
- ✓ For 5, no more choice,
- ✓ So, backtrack



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



✓ For 1, next choice is vertex 5,



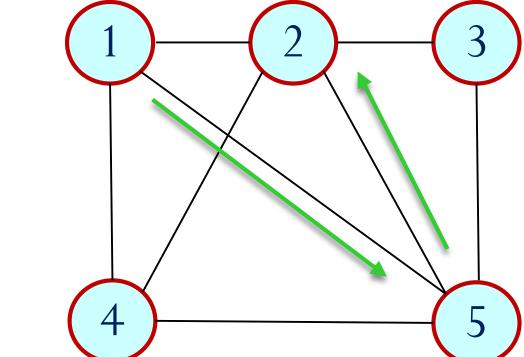
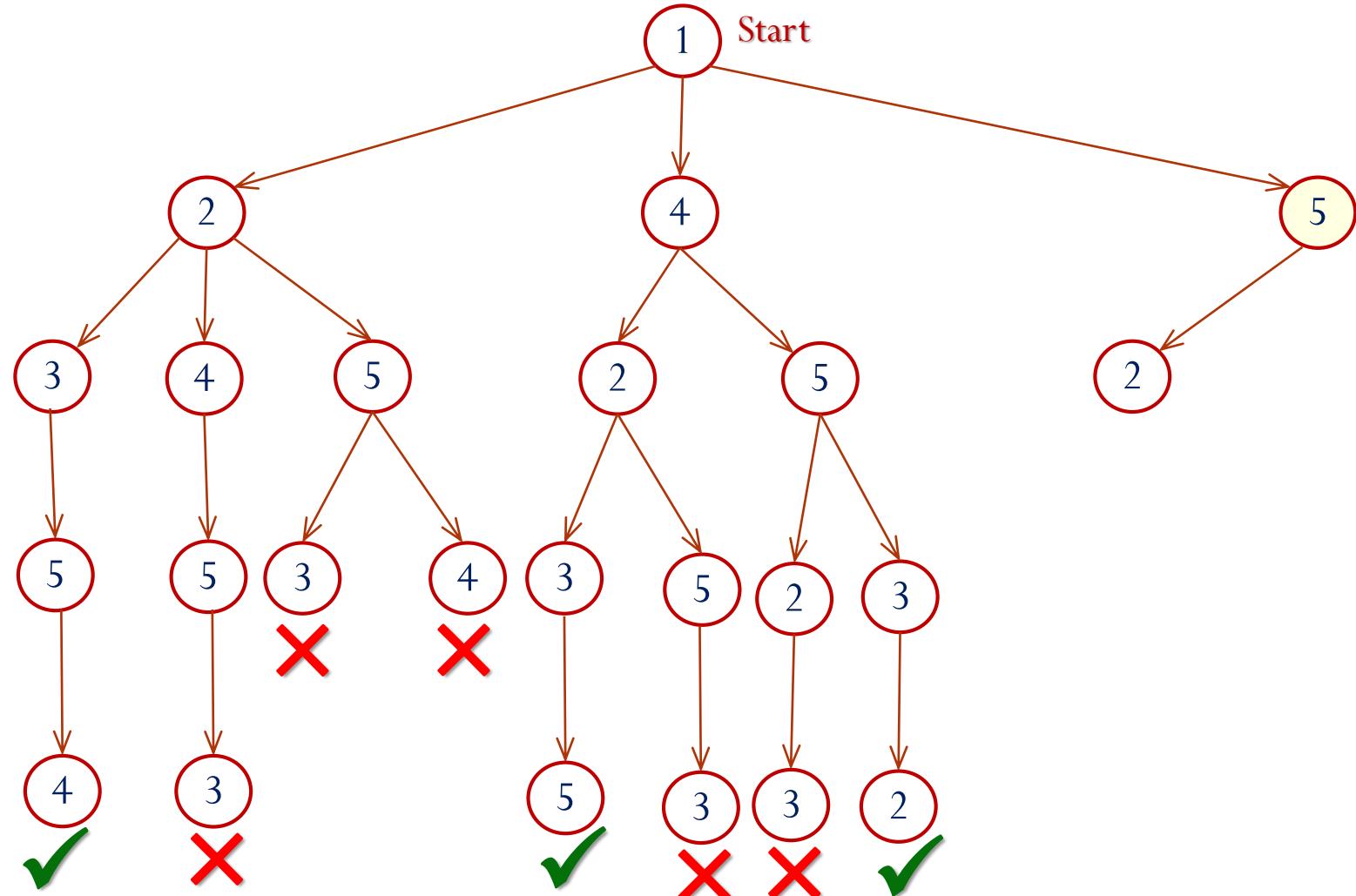
C

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,



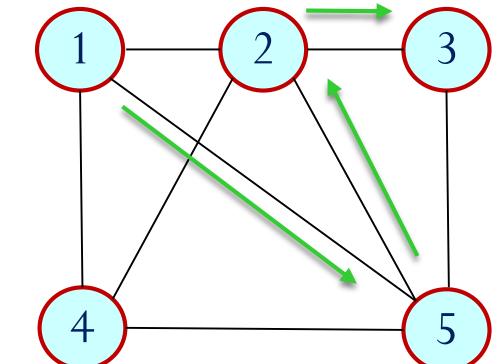
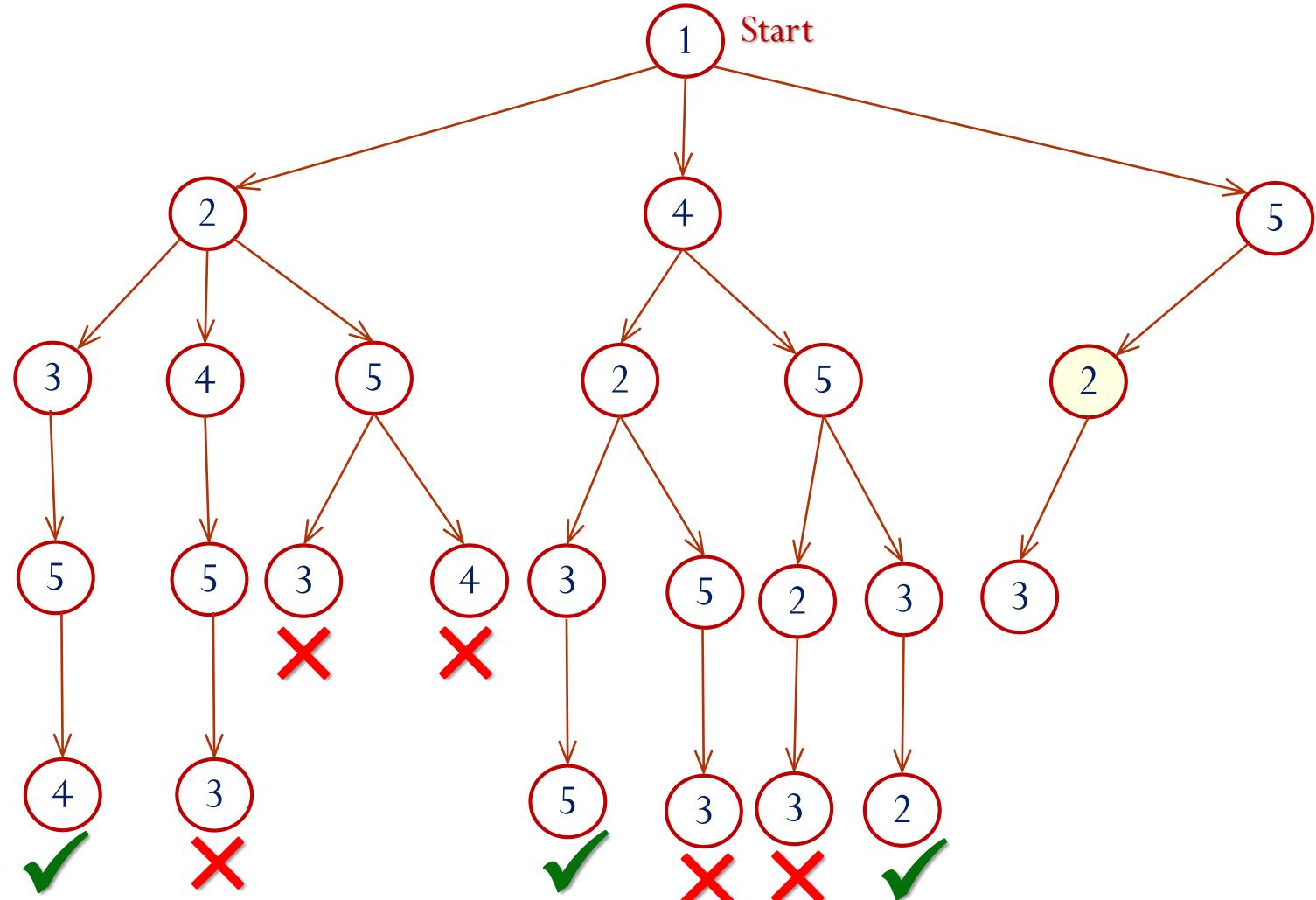
C

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,



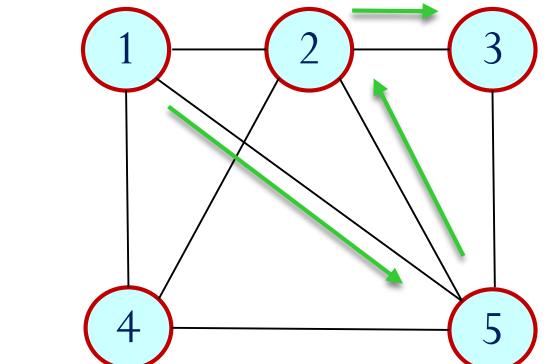
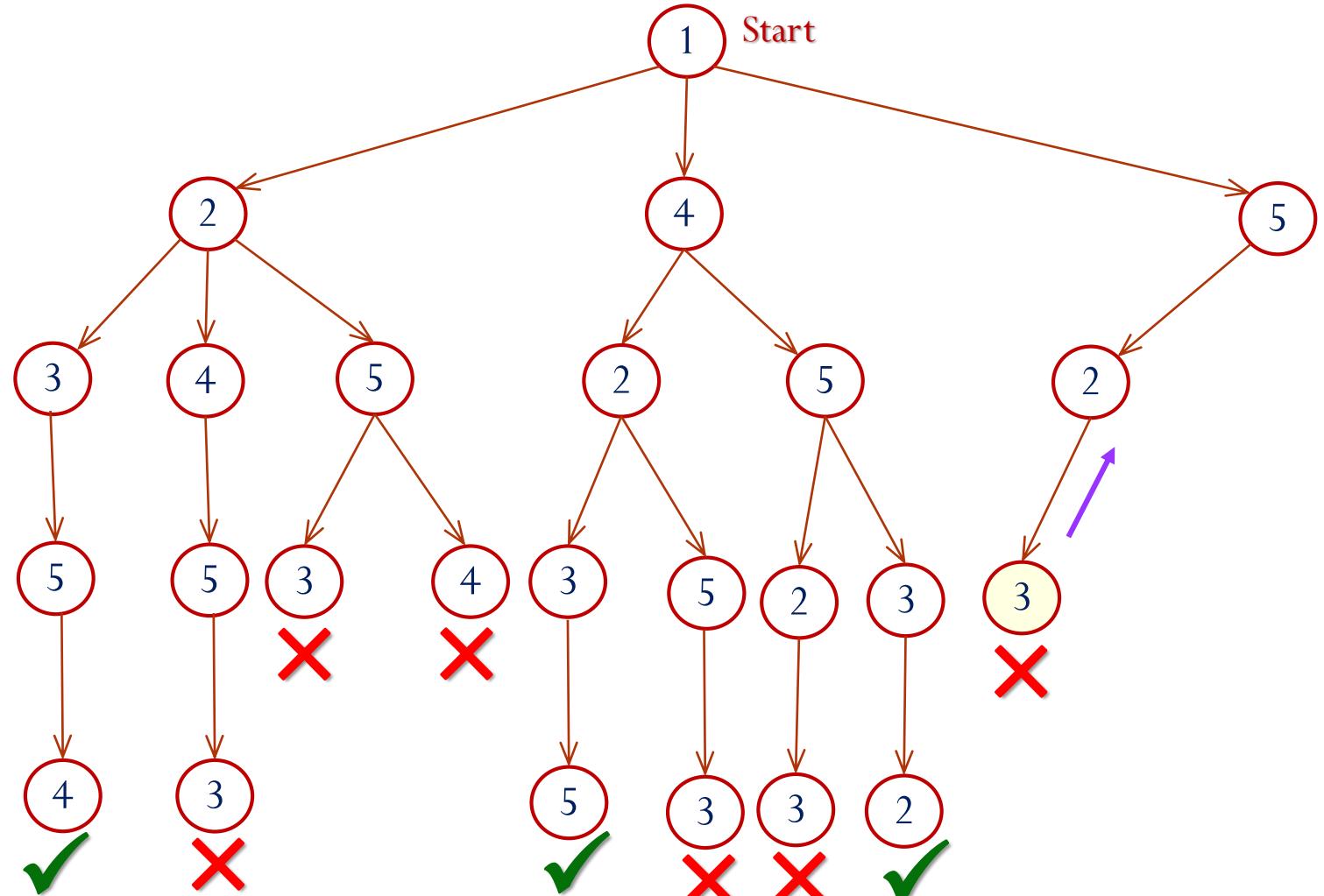
C

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

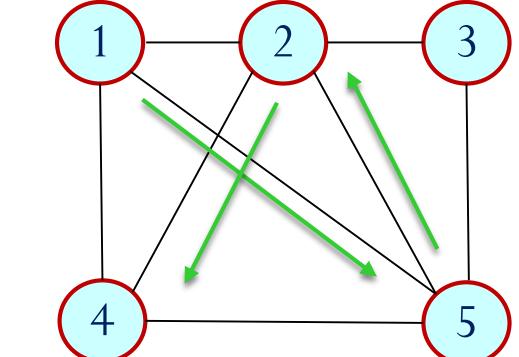
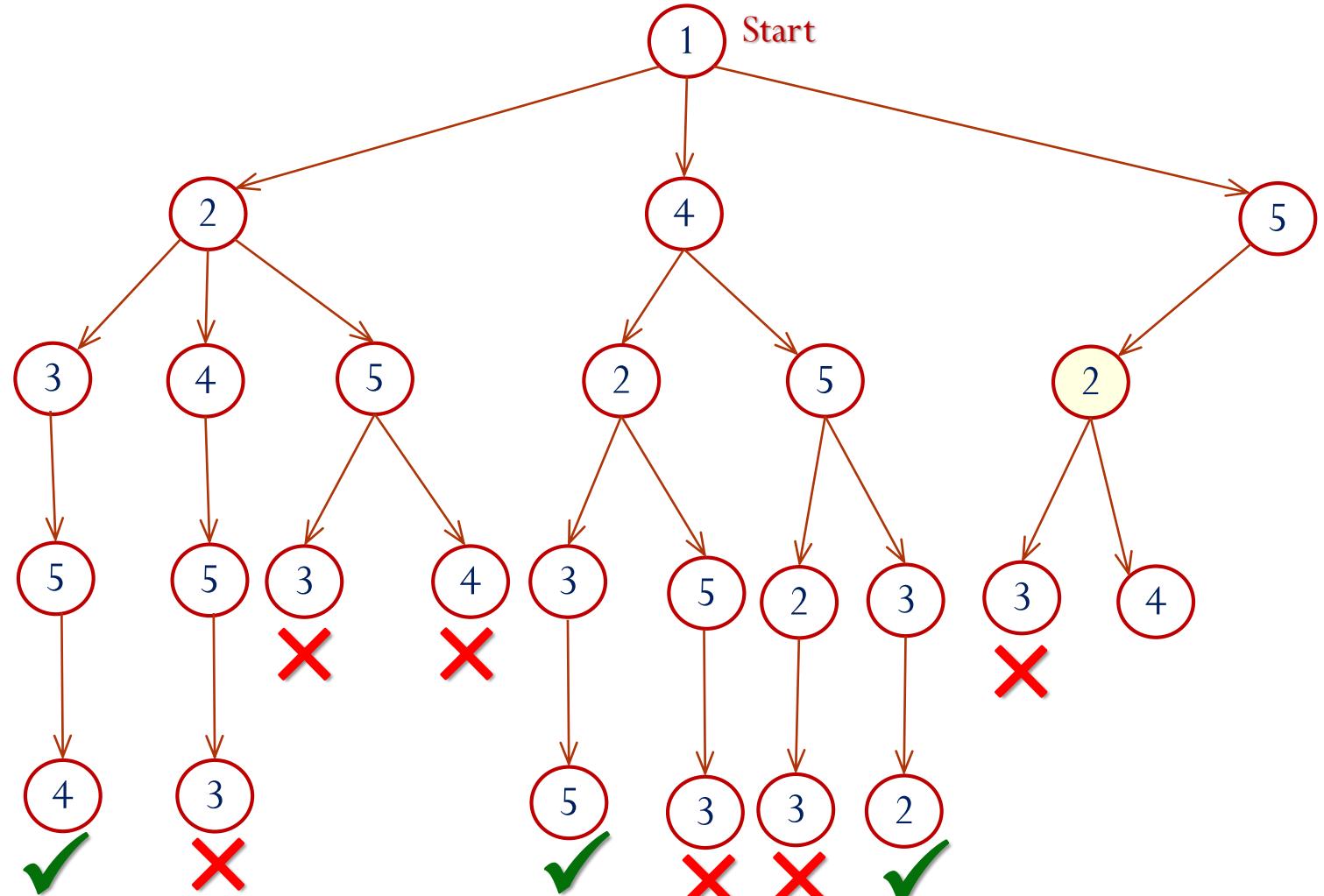


C	1	5	2	3	0
	1	2	3	4	5

Solutions

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

✓ For 2, next choice is vertex 4

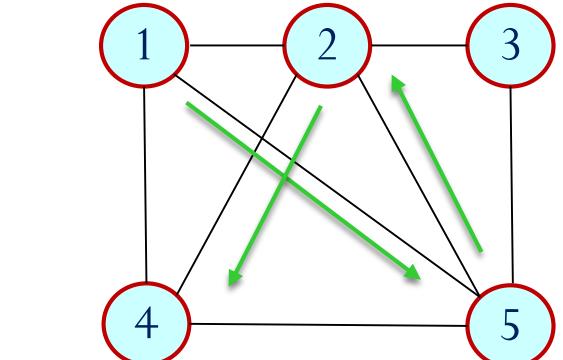
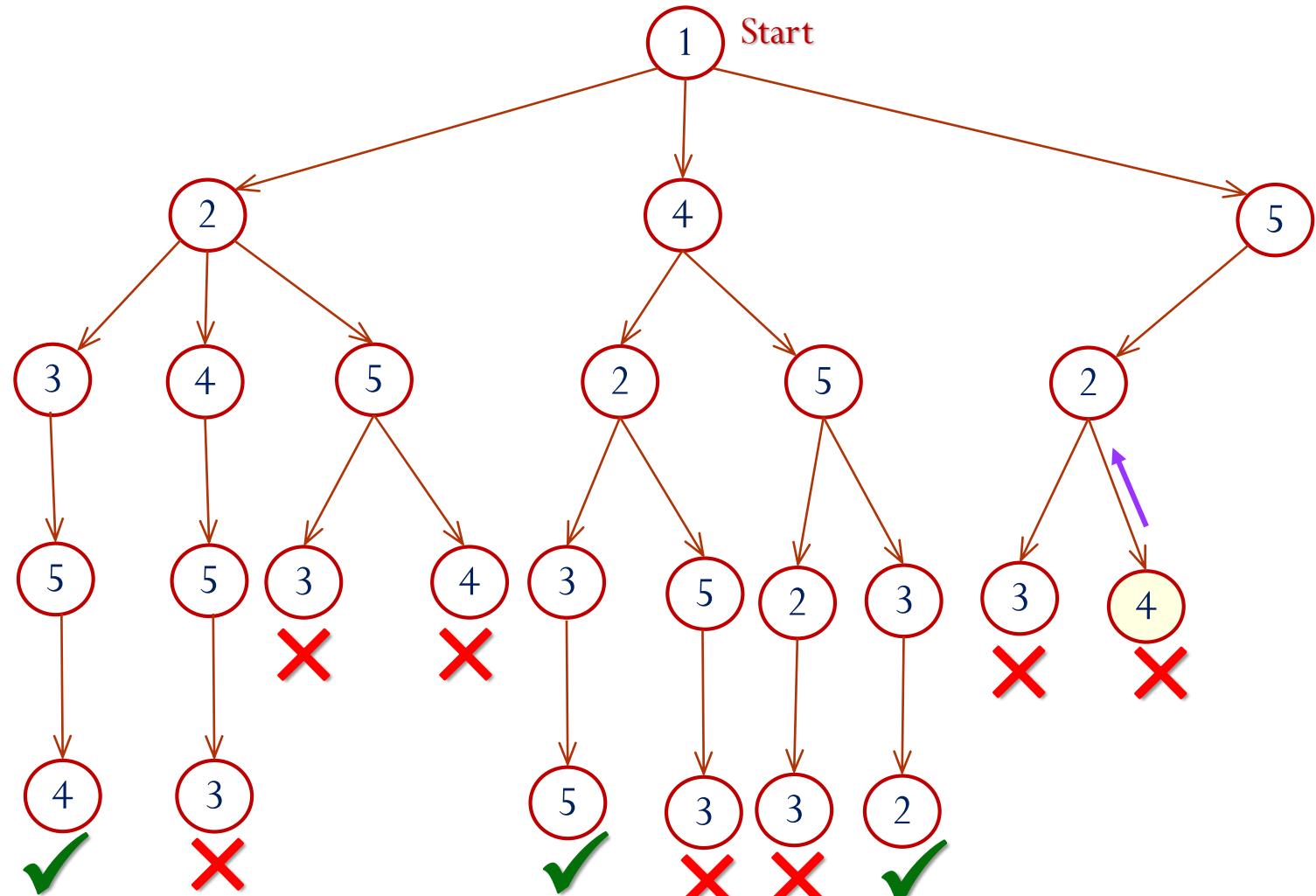


C	1	5	2	4	0
	1	2	3	4	5

Solutions

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

✓ For 4, no choice, so, backtrack

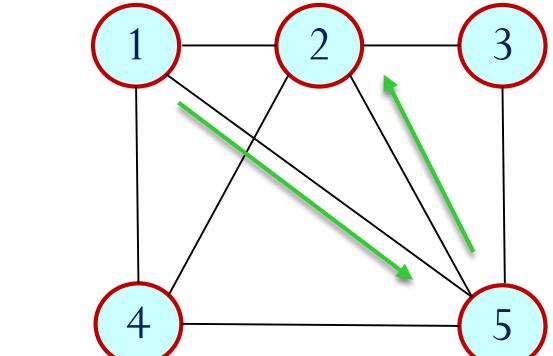
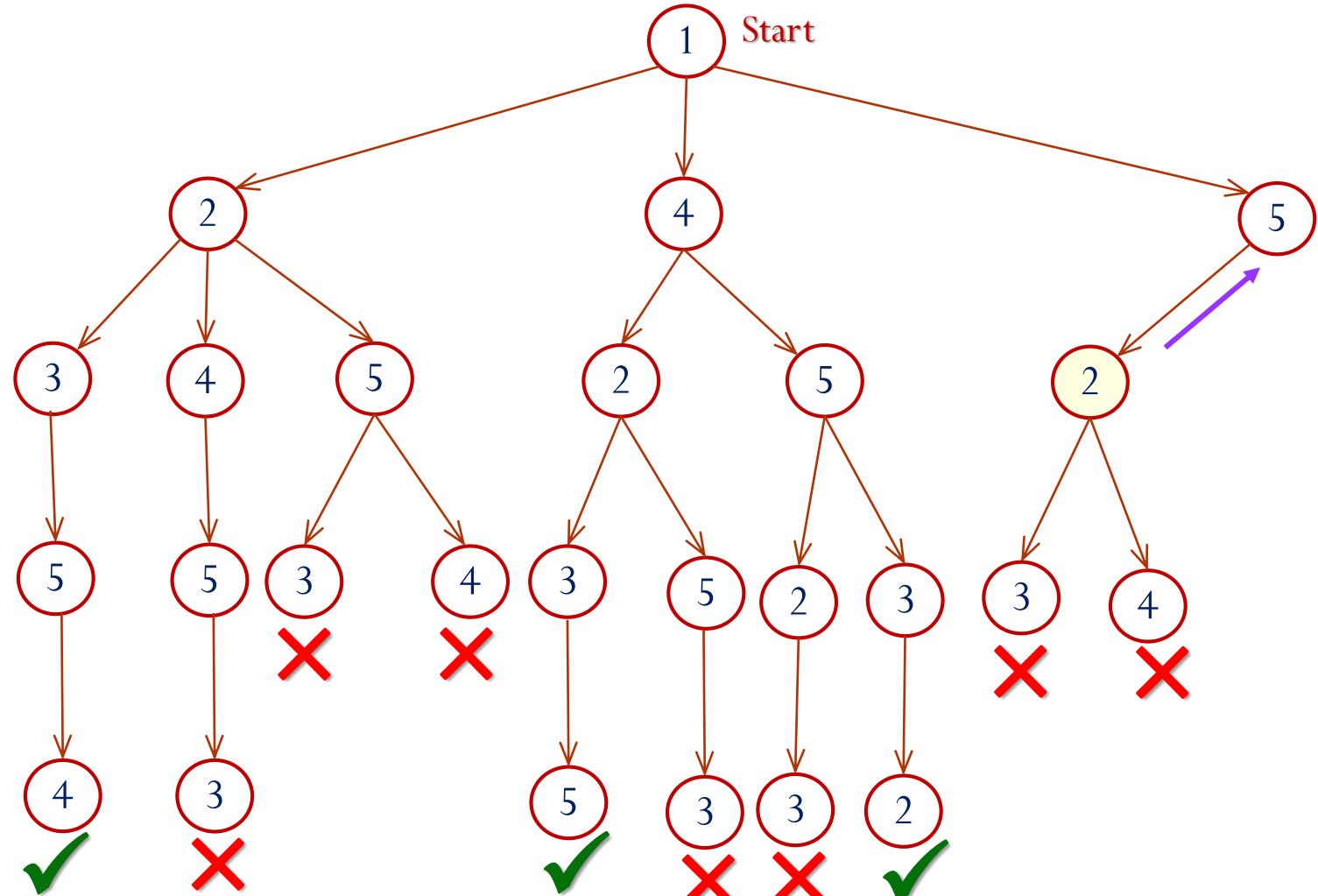


C	1	5	2	4	0
	1	2	3	4	5

Solutions

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

✓ For 2, no choice, so, backtrack

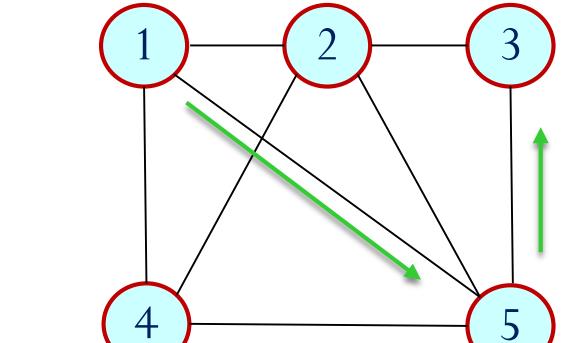
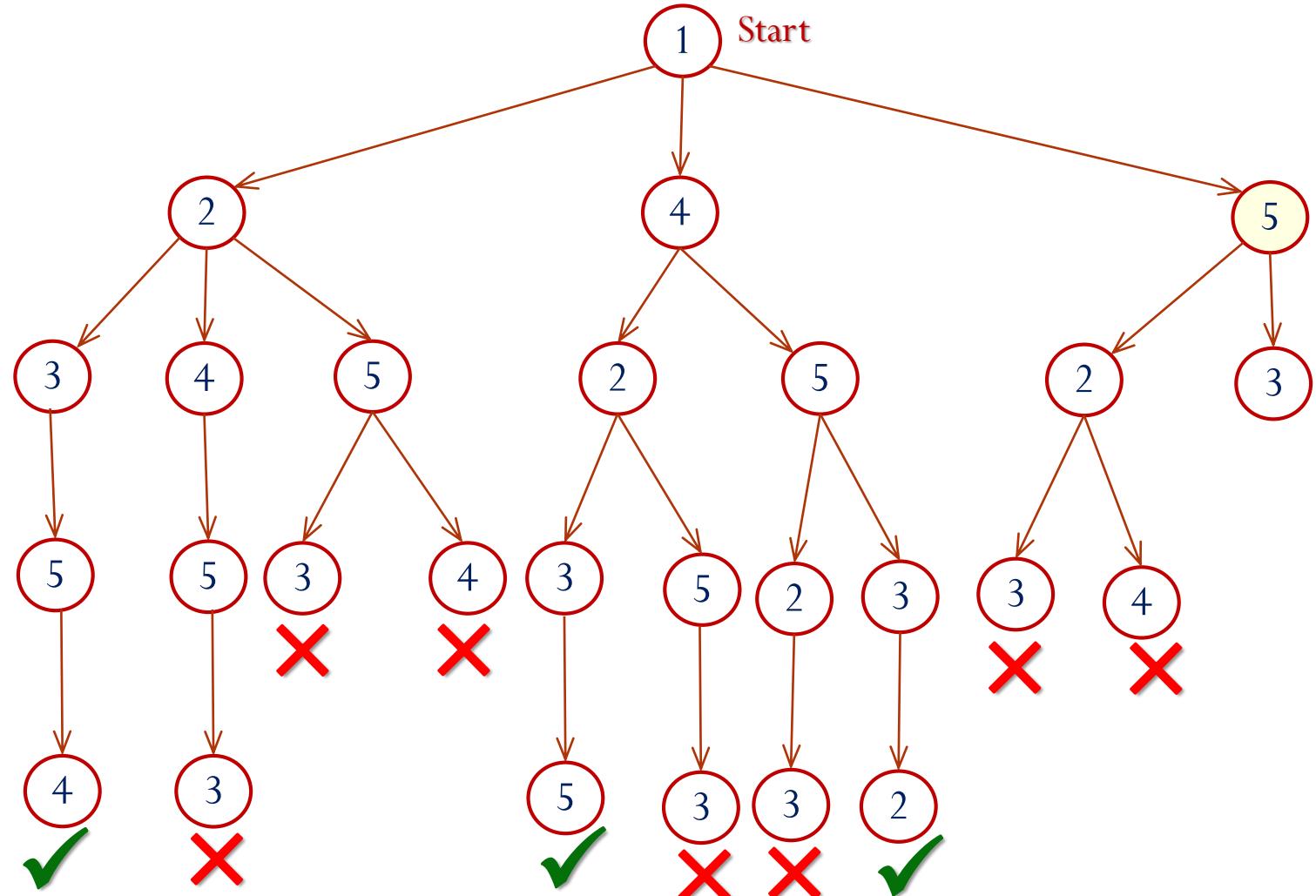


C	1	5	2	0	0
	1	2	3	4	5

Solutions

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

- ✓ For 5, next possibility is vertex 3

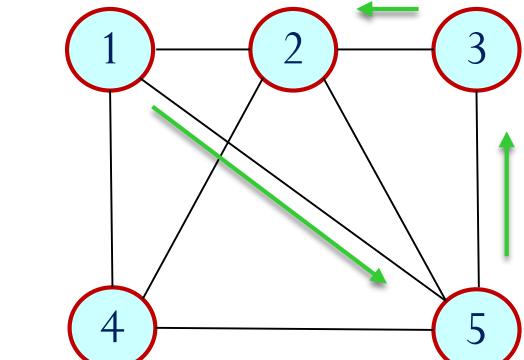
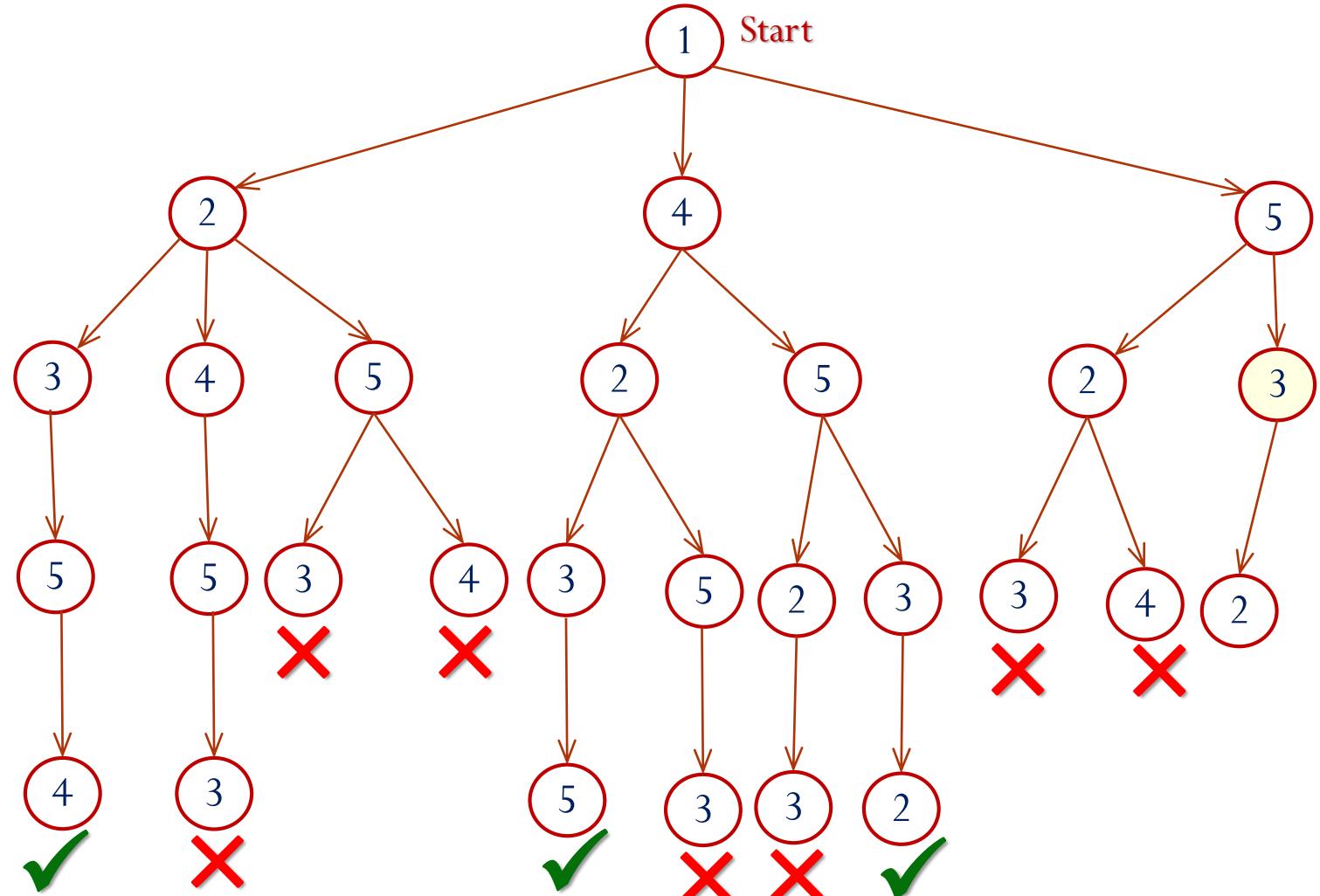


C	1	5	3	0	0
	1	2	3	4	5

Solutions

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

- ✓ For 3 next possibility is vertex 2



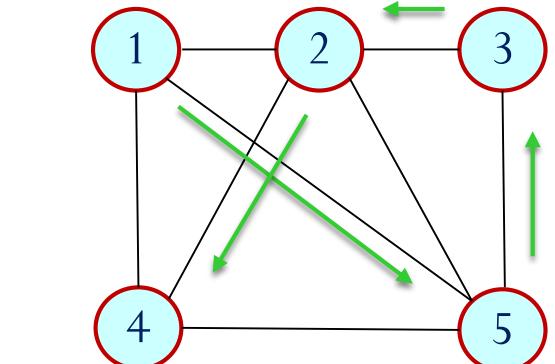
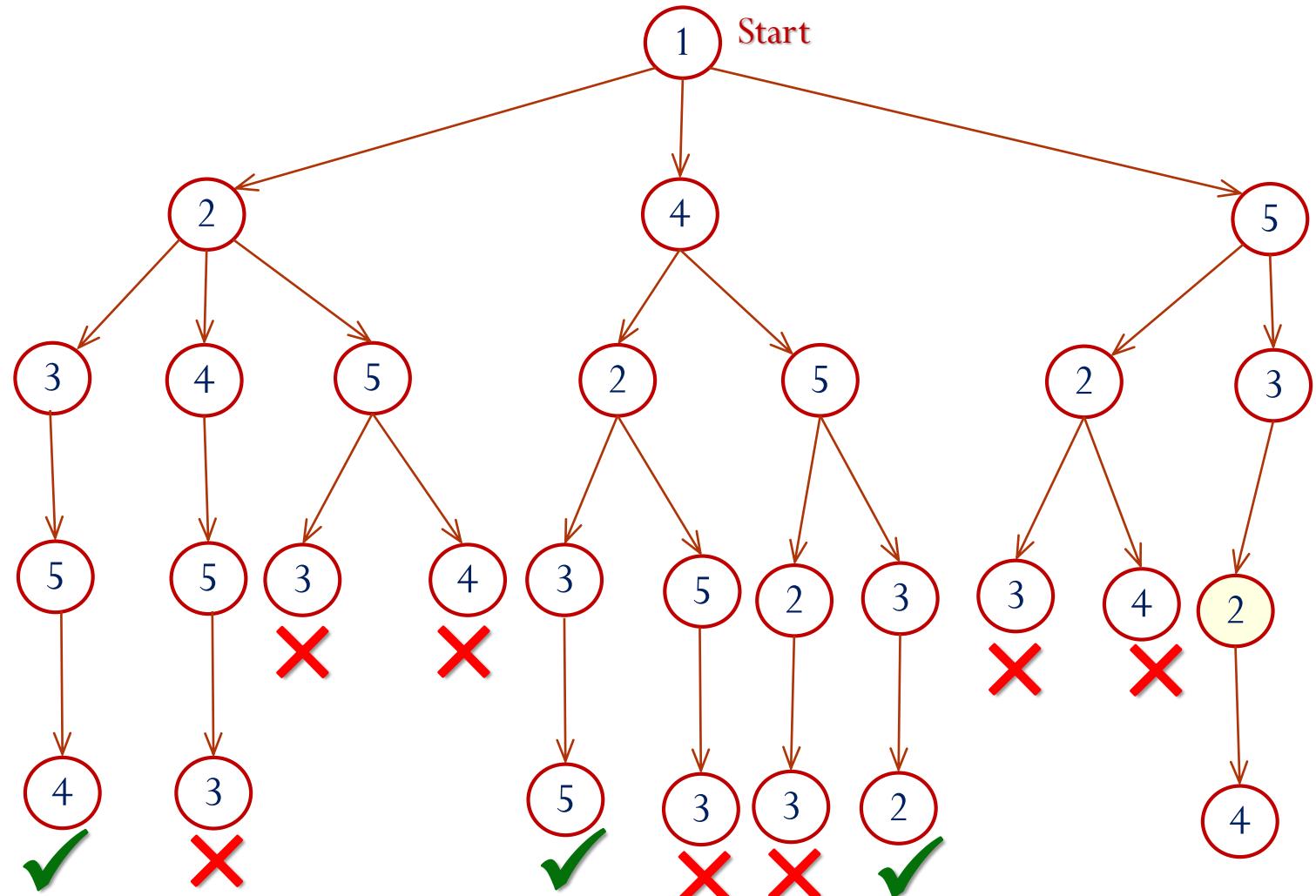
C

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

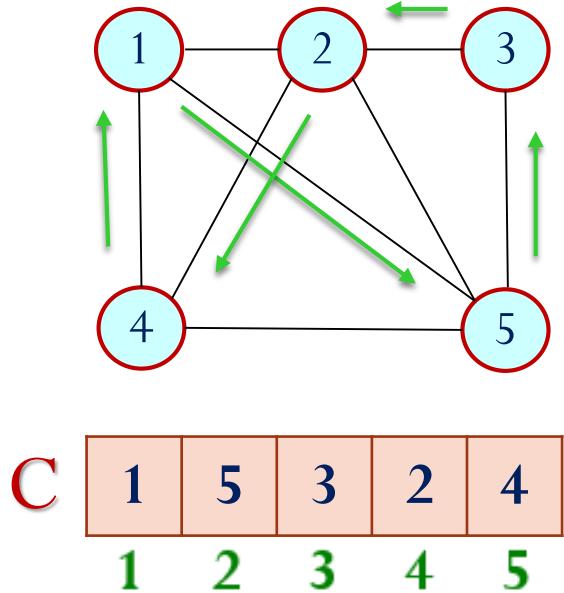
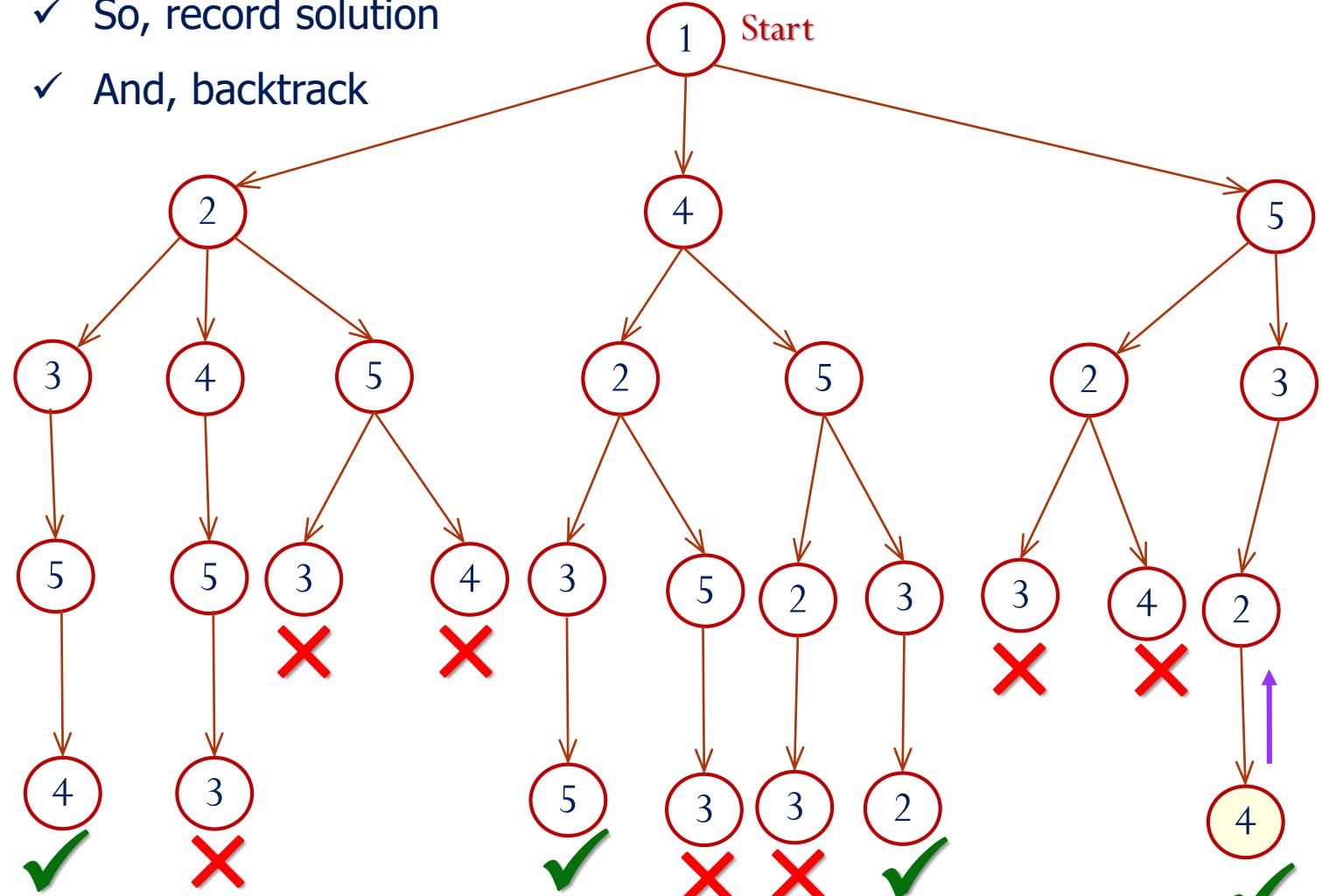
Solutions

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

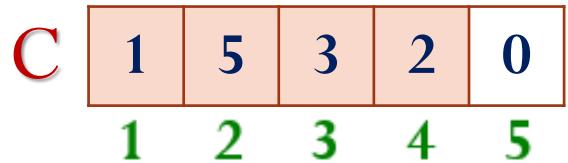
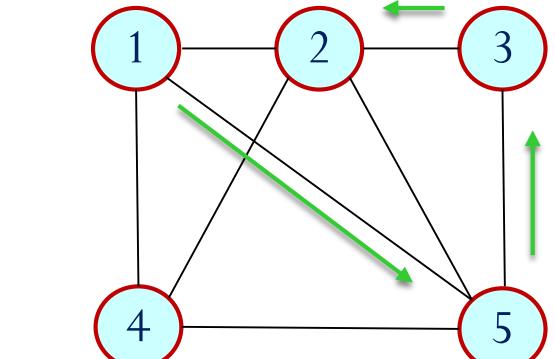
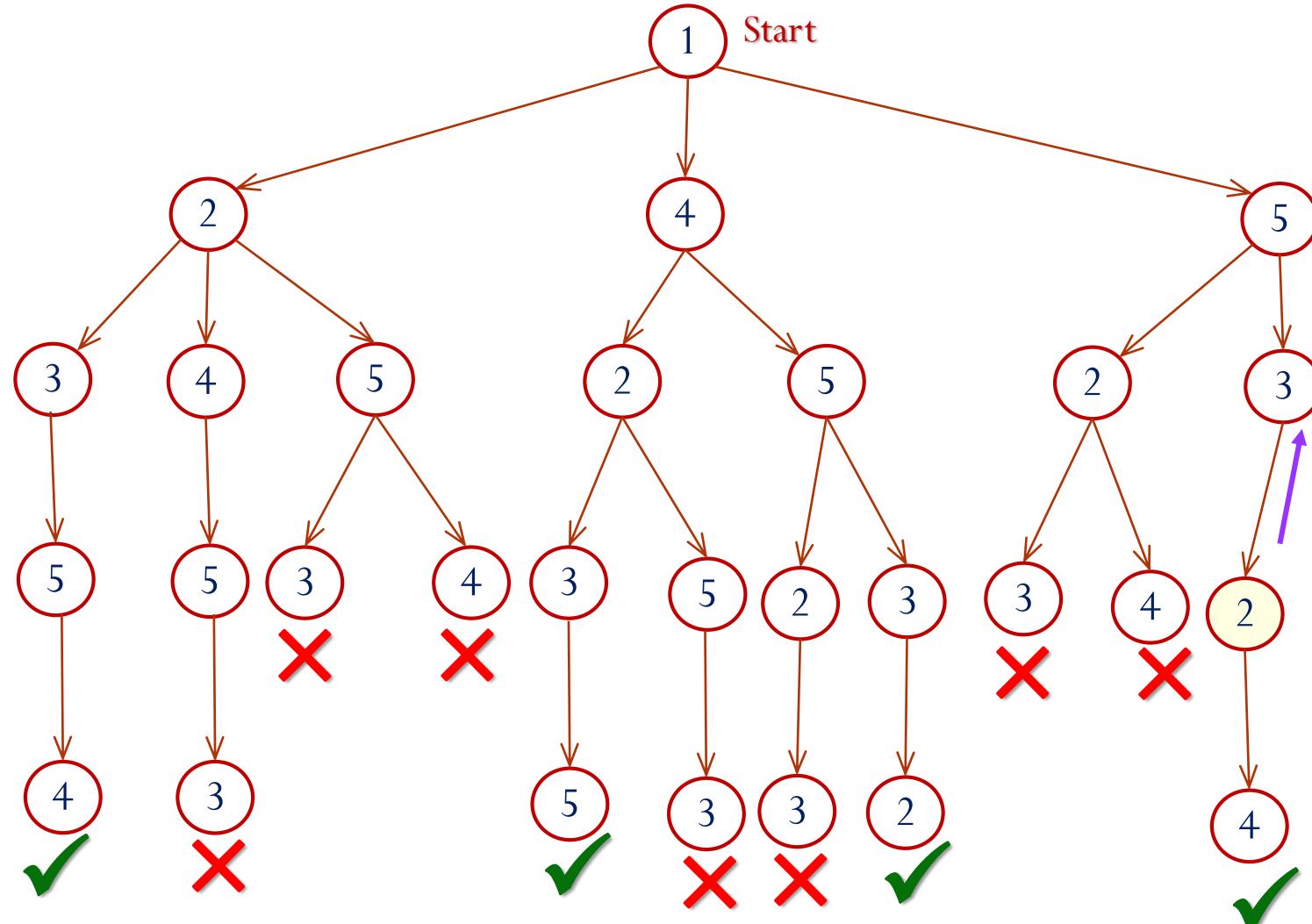
✓ And, backtrack



Solutions

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

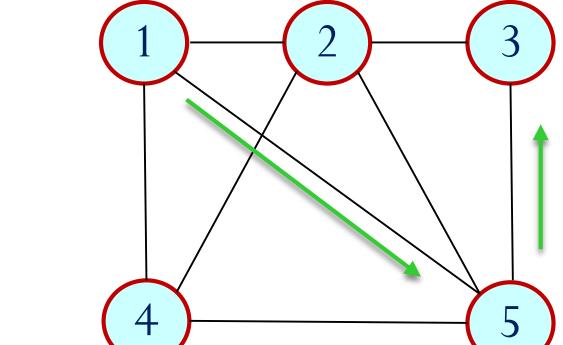
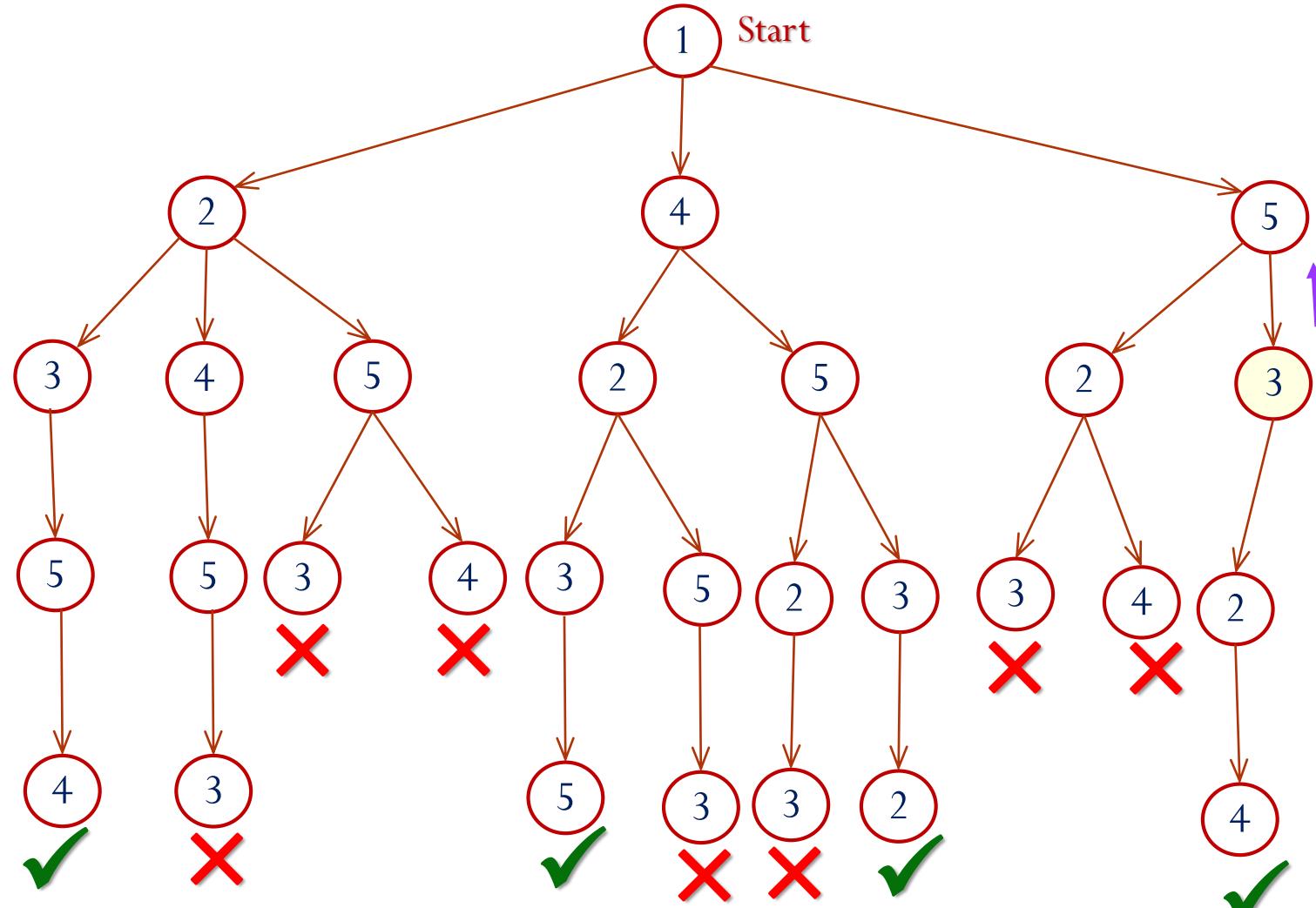
✓ For 2, no choice, backtrack



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



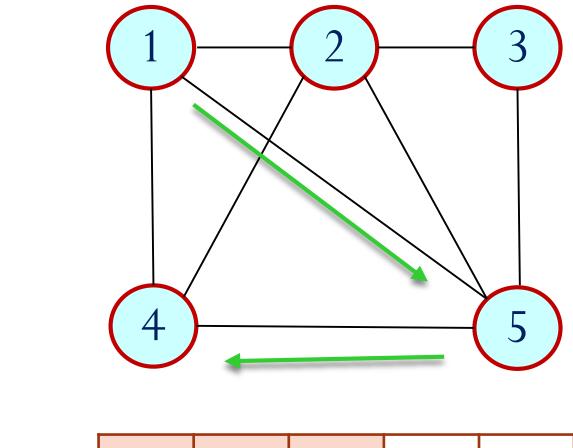
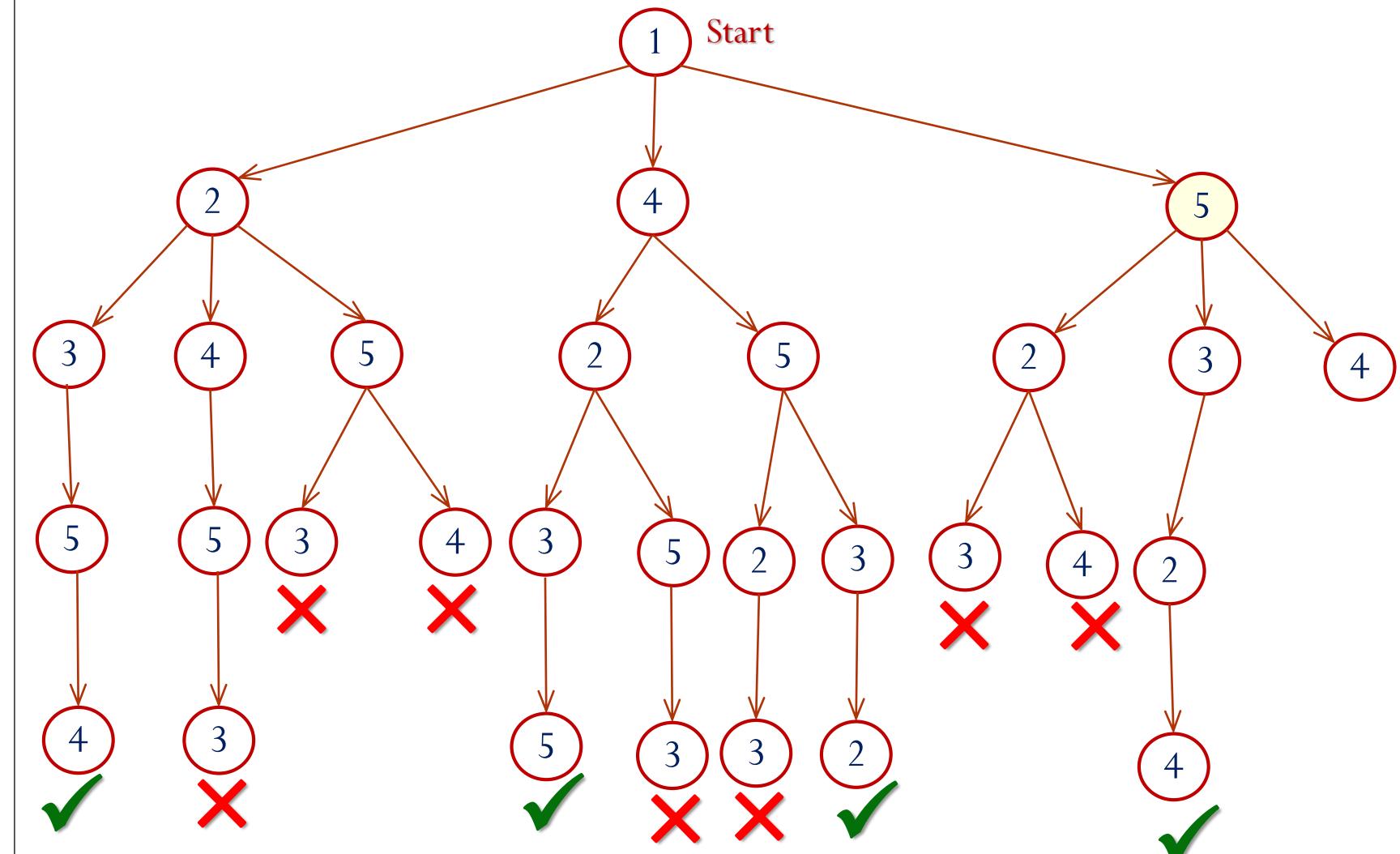
C

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

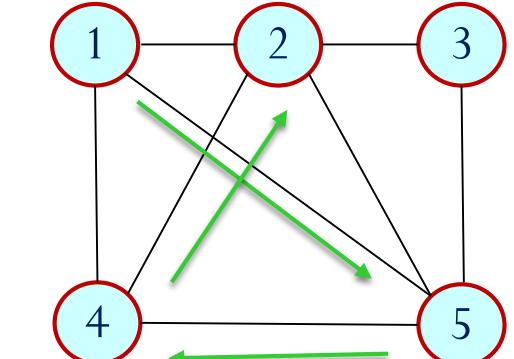
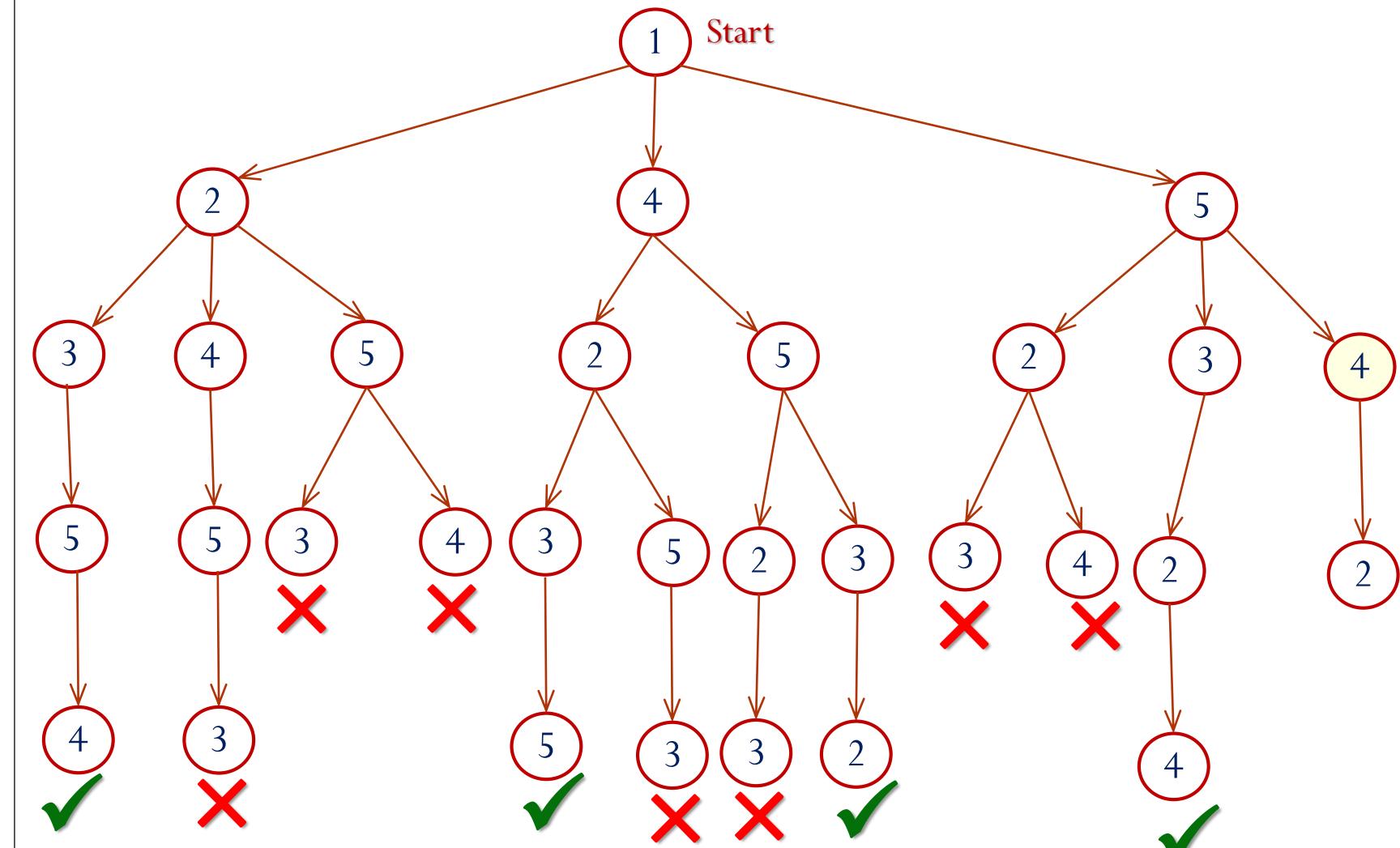
✓ For 5, next choice is 4



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

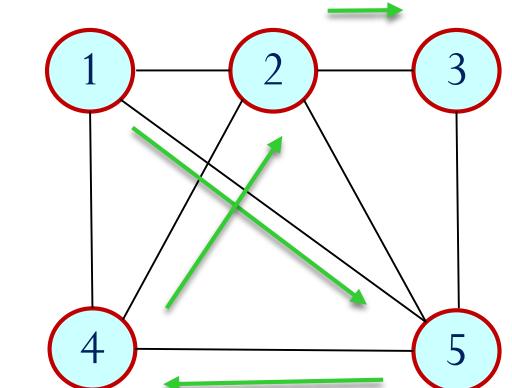
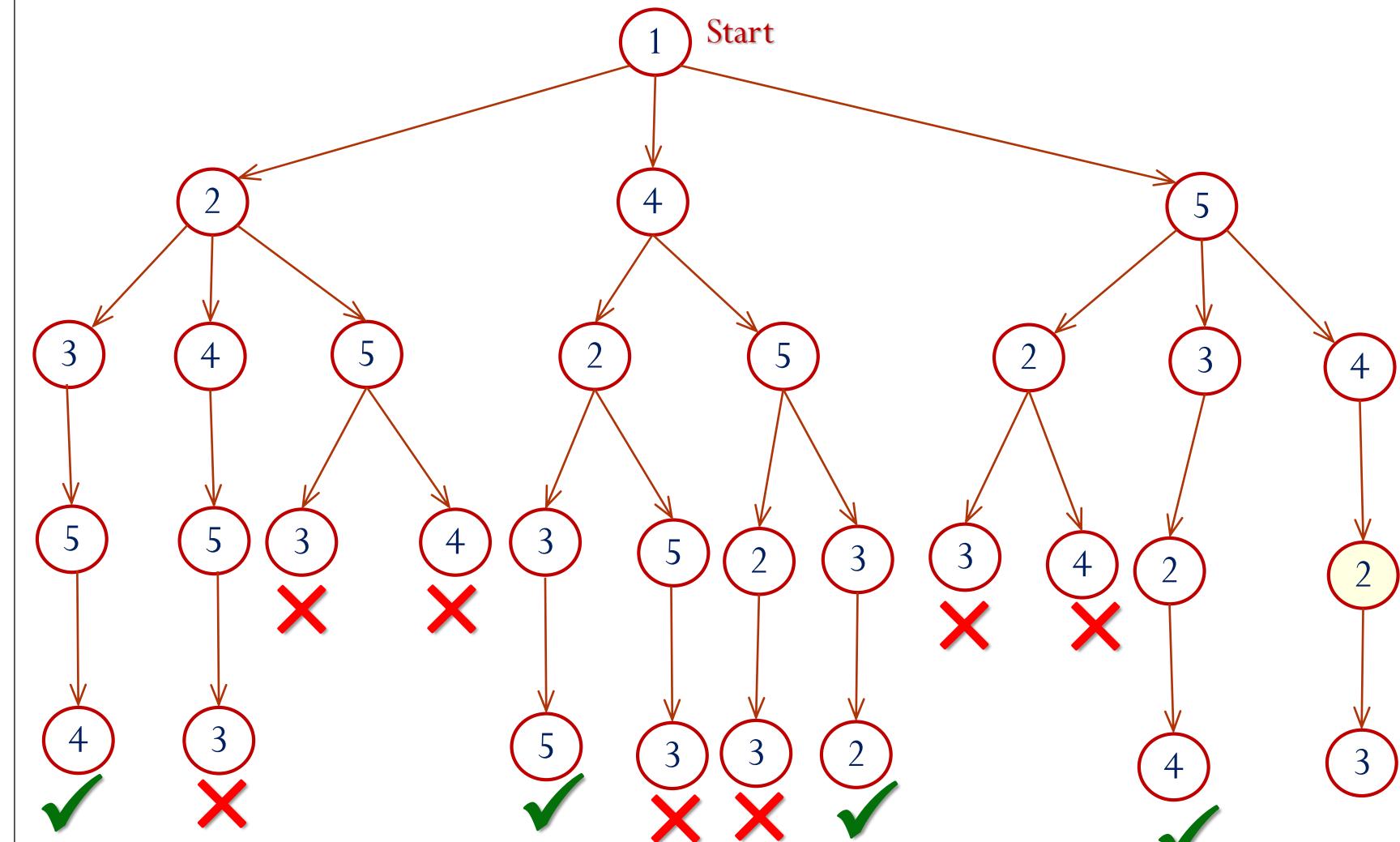


C	1	5	4	2	0
	1	2	3	4	5

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



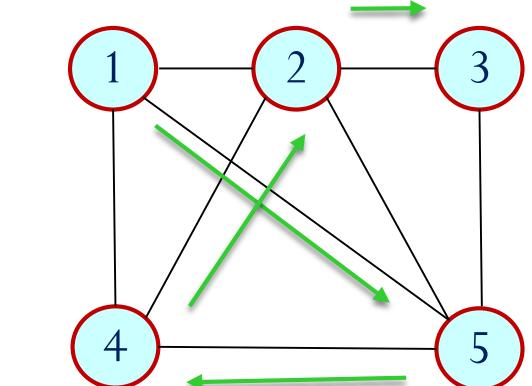
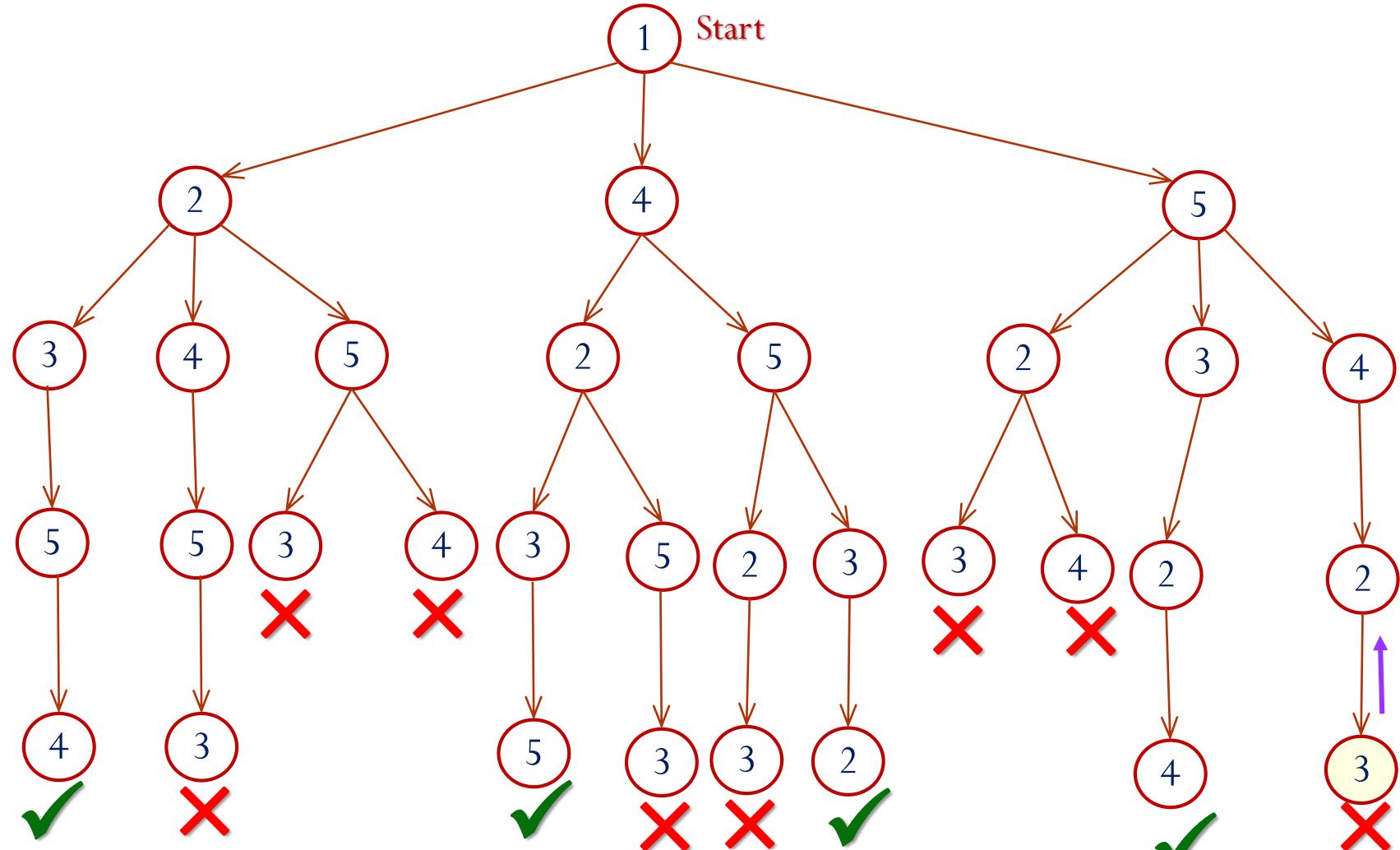
C

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

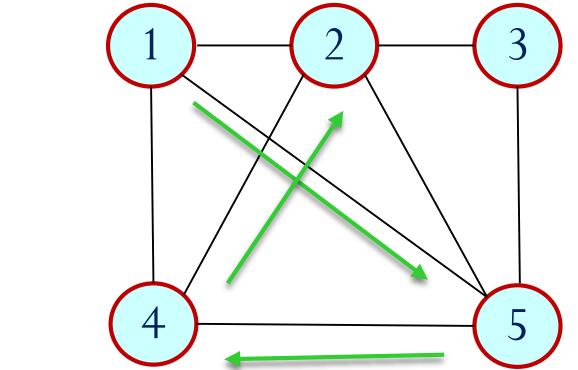
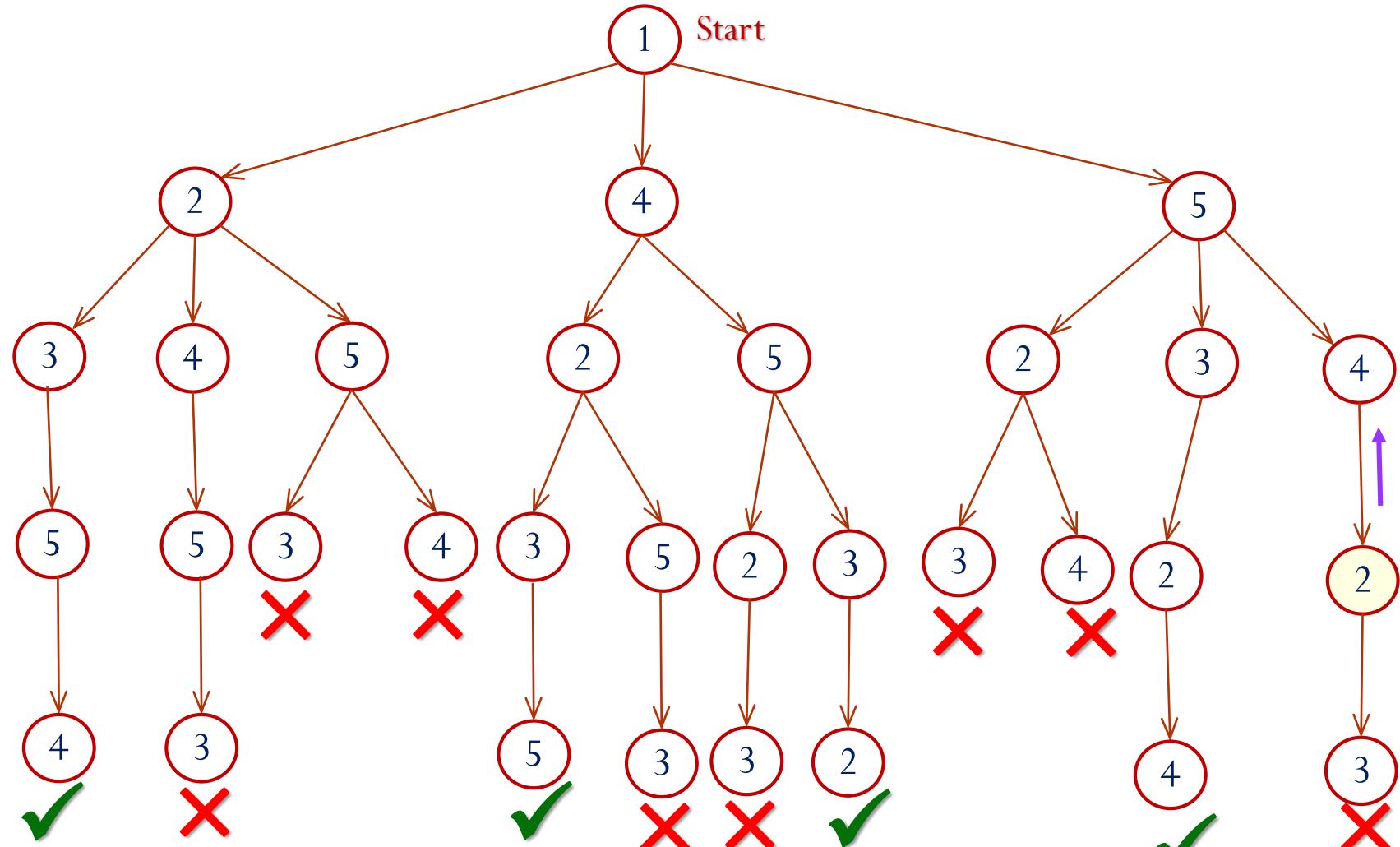


C	1	5	4	2	3
	1	2	3	4	5

Solutions

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

✓ For 2, no more choice, so backtrack

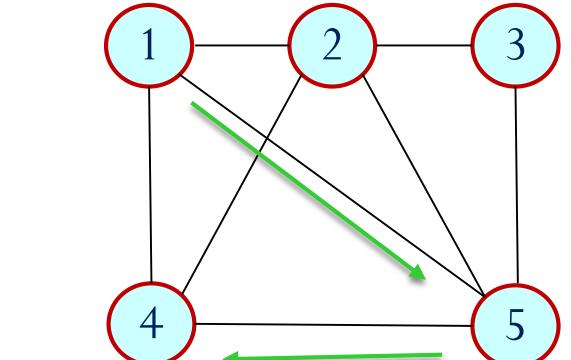
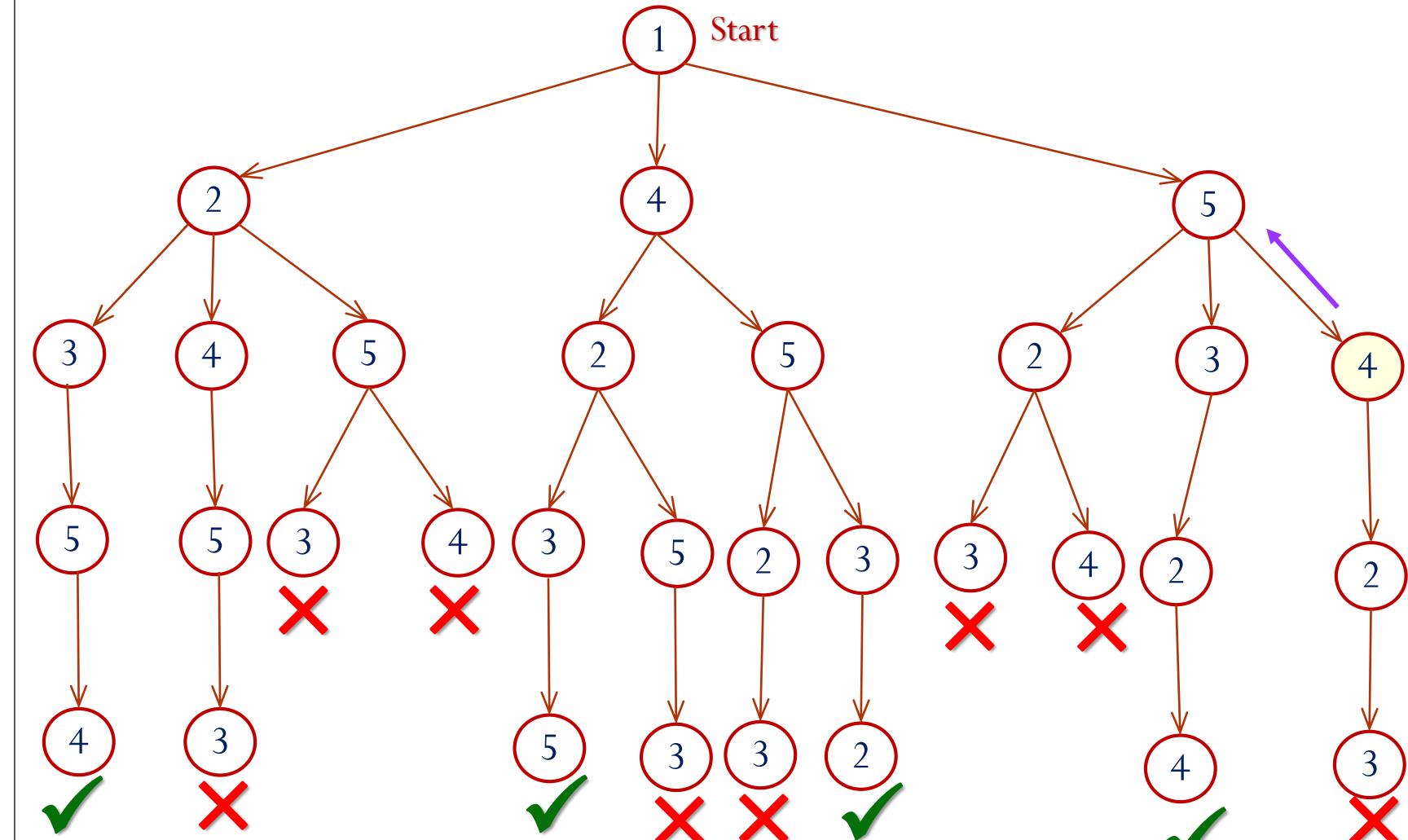


C	1	5	4	2	0
	1	2	3	4	5

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



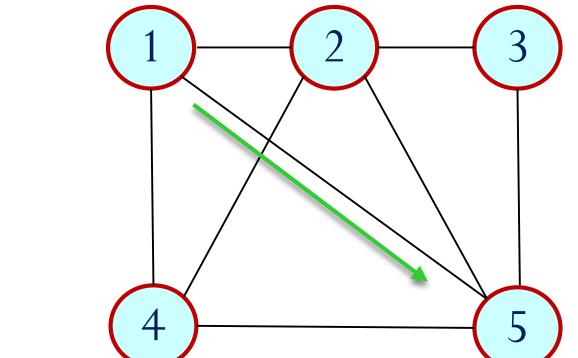
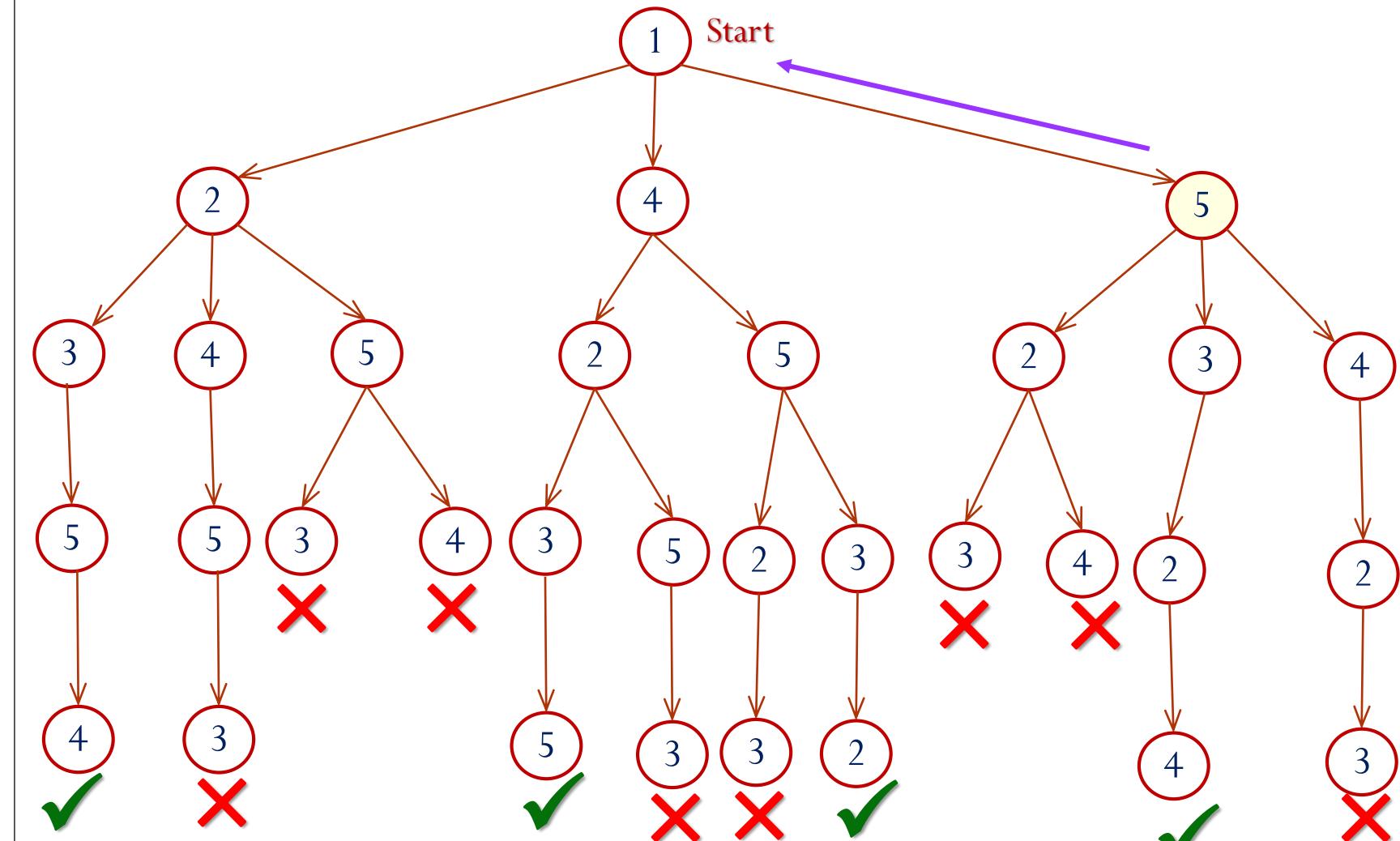
C

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

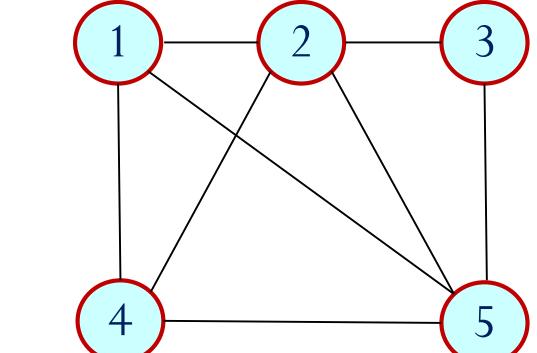
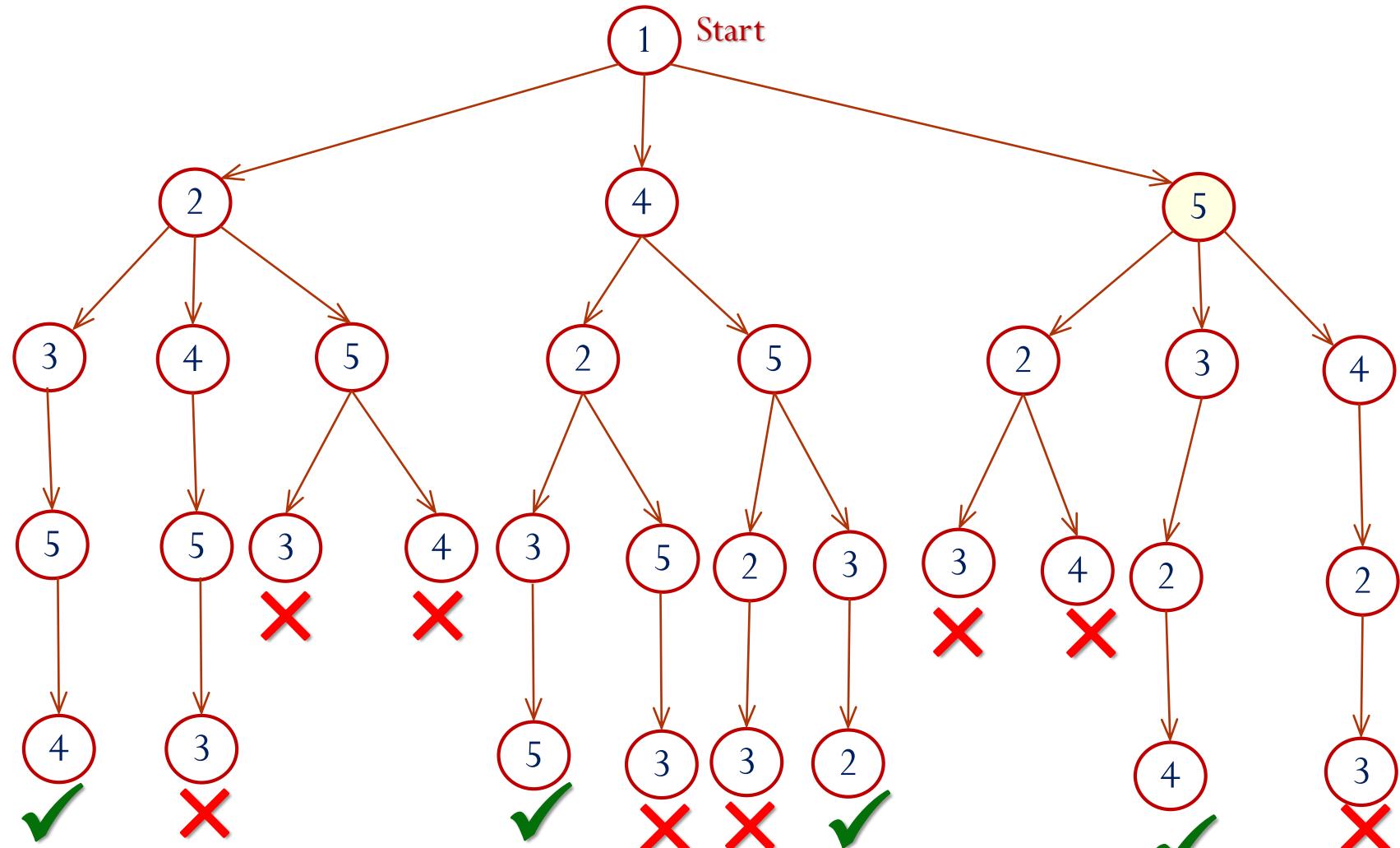


C	1	5	0	0	0
	1	2	3	4	5

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**

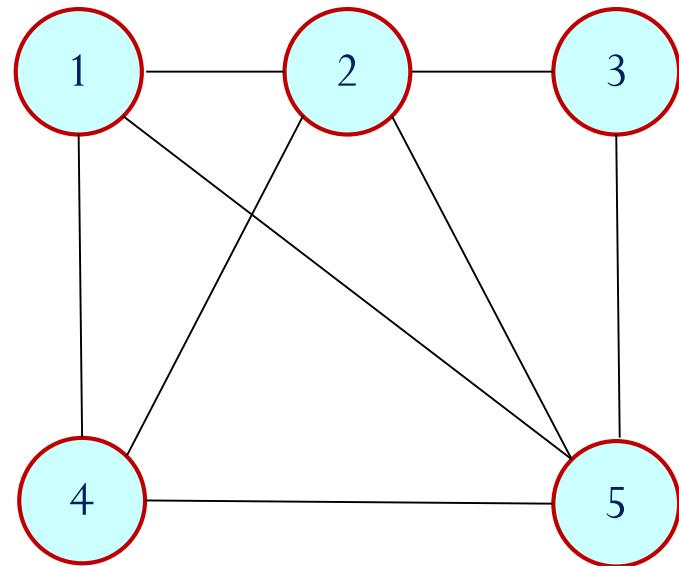


C	1	0	0	0	0
	1	2	3	4	5

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

Branch & Bound Approach

Branch & Bound - Introduction

- ✓ This is an enhancement of backtracking.
- ✓ The branch & bound design strategy is very similar to backtracking in that a state space tree is used to solve the problem.
- ✓ The backtracking uses **depth first search** whereas branch and bound uses **breadth first search**.
- ✓ Backtracking will find all the feasible solutions whereas the branch and bound is used **for optimization problem and only for minimization problem**. If we need to solve maximization problem, we need to convert it to minimization problem and need to apply branch and bound strategy.

Branch & Bound - Introduction

- ✓ The branch & bound algorithm computes a number (**bound**) at a node to determine **whether the node is promising or not**
- ✓ If the bound is not better than the value of the best solution found so far, the node is nonpromising. Otherwise it is promising.
- ✓ Once a node is identified as nonpromising, it will not be explored further for the solution. It means, the node will be killed.

Branch & Bound - Introduction

- ✓ Based on the order of next node exploration, BB can be classified as
 1. **FIFO BB** – Queue is used for maintaining the order of nodes to be explored for the next level
 2. **LIFO BB** – Stack is used for maintaining the order of nodes to be explored for the next level
 3. **Least Cost BB (LC-BB)** – Node with LEAST COST will be explored for the next level.

Branch & Bound Approach

0/1 Knapsack Problem

0/1 Knapsack Problem:

(Branch & Bound Method)

(Least Cost BB)

Example:

	x_1	x_2	x_3	x_4
profit	10	10	12	18
weight	2	4	6	9

$$m = 15$$

$$n = 4$$

Upper Bound, $u = \sum_{i=1}^n p_i x_i \leq m$

Cost, $c = \sum_{i=1}^n p_i \cdot x_i$ (with fraction)

Calculate cost & upper bound for node 1

①
 $\text{Upper Bound} = \alpha$

Starting with node 1

Since the problem is maximizing profit (maximization problem), it is converted to minimization problem by taking -ve values of u and cost

$$\begin{aligned}
 u &= 10 & 10 & 12 \Rightarrow u = 32 \\
 &\quad 2 & 4 & 6 \\
 c &= 10 & 10 & 12 & 18 \\
 &\quad 2 & 4 & 6 & 3/9 \\
 &= 10 + 10 + 12 + 18 \times \frac{3}{9} \\
 &= 38 \Rightarrow c = -38
 \end{aligned}$$

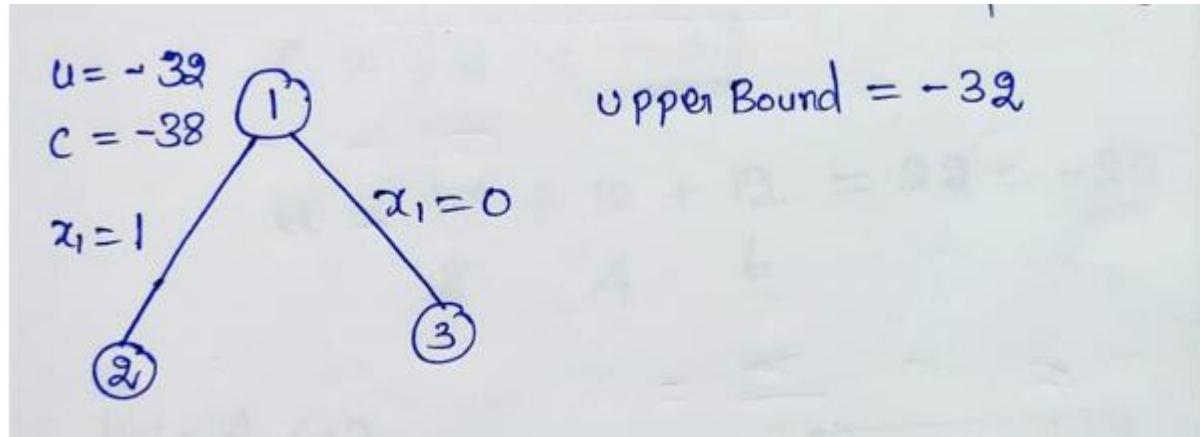
(For minimization problem)

① $u = -32$
 $c = -38$
 $\text{Upper Bound} = \alpha$

If $u <$ Upper bound, update upper bound

① $u = -32$
 $c = -38$
 $\text{Upper Bound} = -32$

Explore node 1: 2 options – 1. x1 included (x1=1) 2. x1 excluded (x1=0)



Calculate c and u for node 2 (if x1=1)

$$\underline{x_1=1:} \quad c = \frac{10}{2} + \frac{10}{4} + \frac{12}{6} + \frac{18 \times 3}{9} = 38 \\ = -38$$

$$u = \frac{10}{2} + \frac{10}{4} + \frac{12}{6} = 32 = \underline{-32}$$

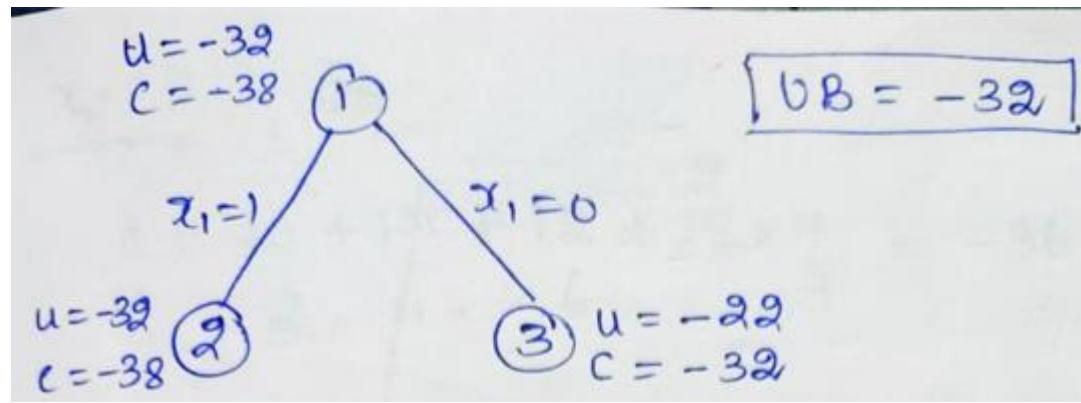
Calculate c and u for node 3 (if x1=0)

$$\underline{x_1=0:} \quad c = \cancel{\frac{10}{2}} + \frac{10}{4} + \frac{12}{6} + \frac{18 \times 5}{9} = 32$$

$$c = 32 = \underline{-32}$$

$$u = \cancel{\frac{10}{2}} + \frac{10}{4} + \frac{12}{6} = 22 = \underline{-22}$$

Need to update UB(upper bound) only
If new u is less than current upper bound



1. Update UB:

$$u < UB$$

No change in UB,

Because, $u > UB$, $-22 > -32$,

So no need to update UB

2. Check and Kill Non-Promising Nodes

After updating UB, need to check for any non-promising node exist.

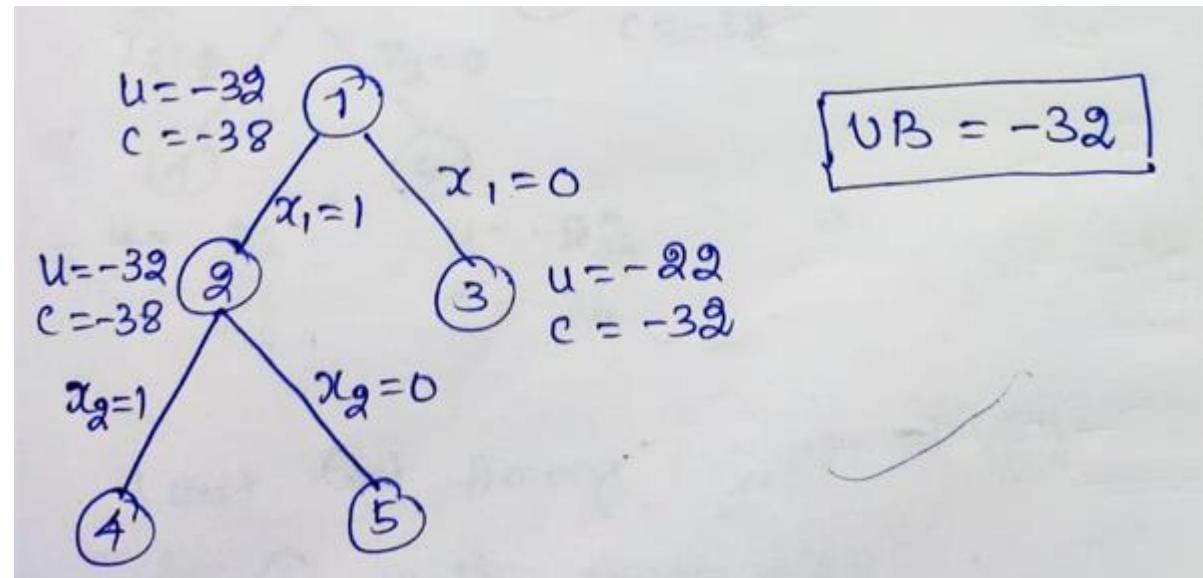
If exist, need to kill the node.

If the cost (c) of a node is greater than UB, that node is said to be non-promising node. Kill that node. i.e., No use in exploring that node.

At this stage, there is no non-promising node among 2 and 3.

3. Choose Least Cost Node and Explore:

Need to explore one node from the nodes 2 and 3. As per LC-BB, the node 2 will be selected for exploration, because it is having least cost as -38



For node 2, two more options,

1. x2 included ($x_2=1$)
2. x2 excluded ($x_2=0$)

Calculate c and u for node 4 (if $x_2=1$)

$x_2=1$: No change in u & c



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

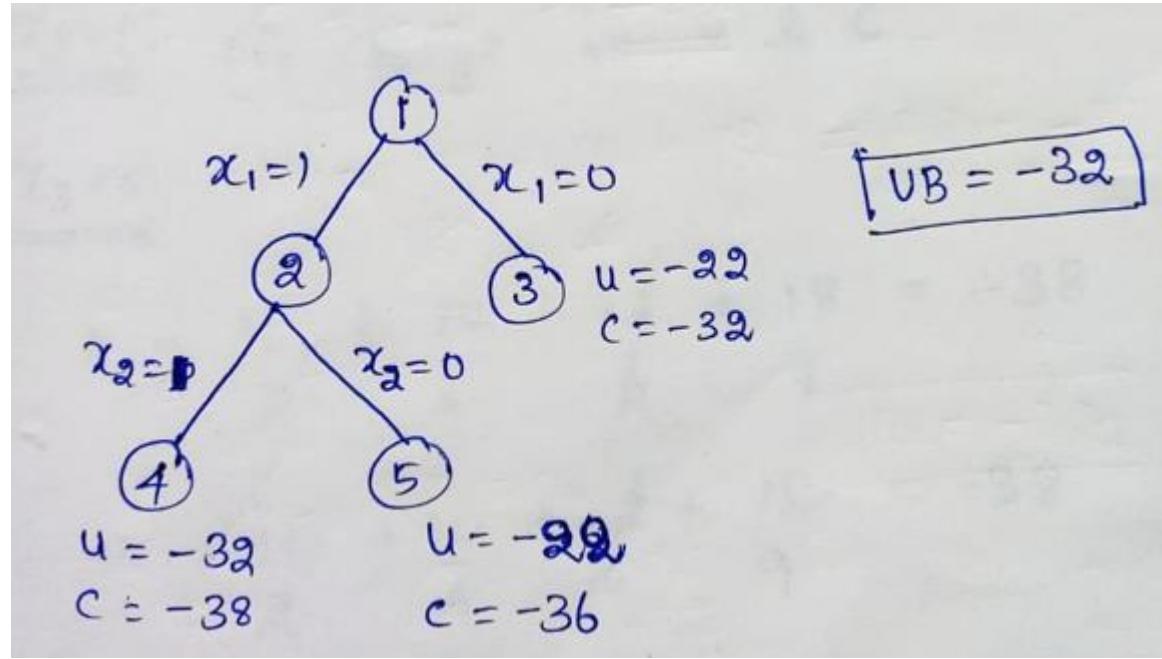
Calculate c and u for node 5 (if $x_2=0$)

$x_2=0$: ~~10~~

$$c = \frac{10}{2} + \frac{10}{4} + \frac{12}{6} + \frac{18}{9} \times \frac{7}{9} = -36$$

$$u = \frac{10}{2} + \frac{10}{4} + \frac{12}{6} = -22$$

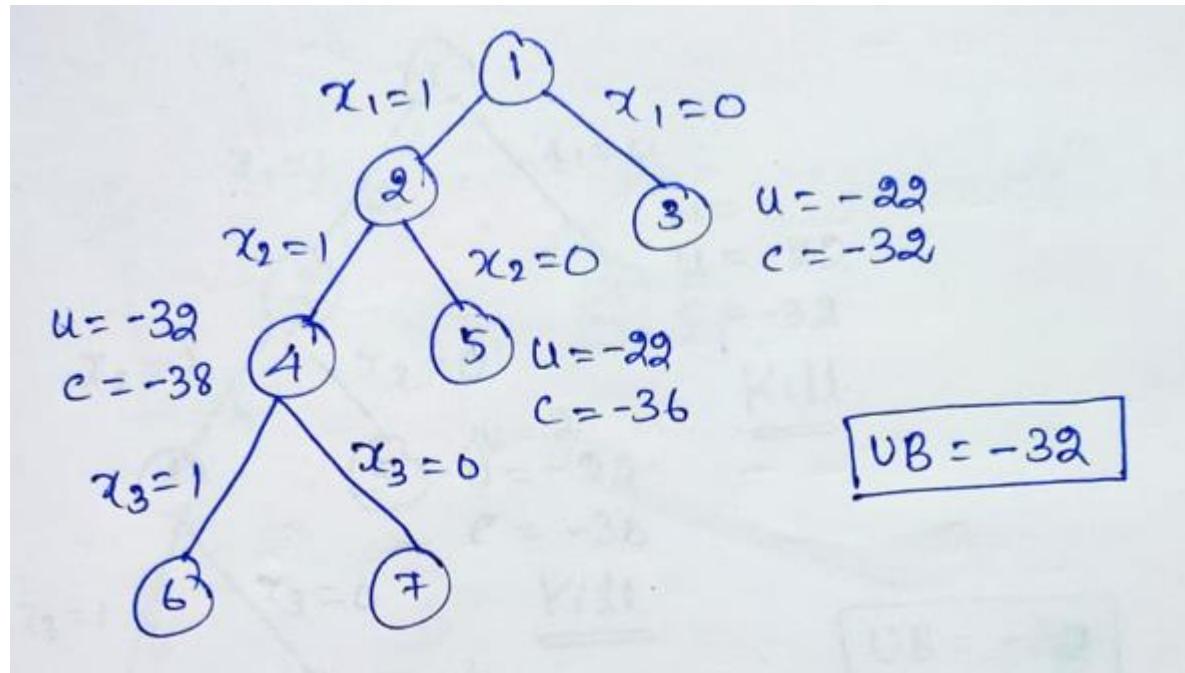
Need to update UB(upper bound) only
If new u is less than current upper bound



1. No change in UB,
Because, $u > UB$, $-22 > -32$,
So no need to update UB.
2. No non-promising node
3. Need to explore one node from the nodes 4, 5 and 3.
As per LC-BB, the node 4 will be selected for exploration, because it is having least cost as -38

For node 4, two more options,

1. x_3 included ($x_3=1$)
2. x_3 excluded ($x_3=0$)



Calculate c and u for node 6 (if $x_3=1$)

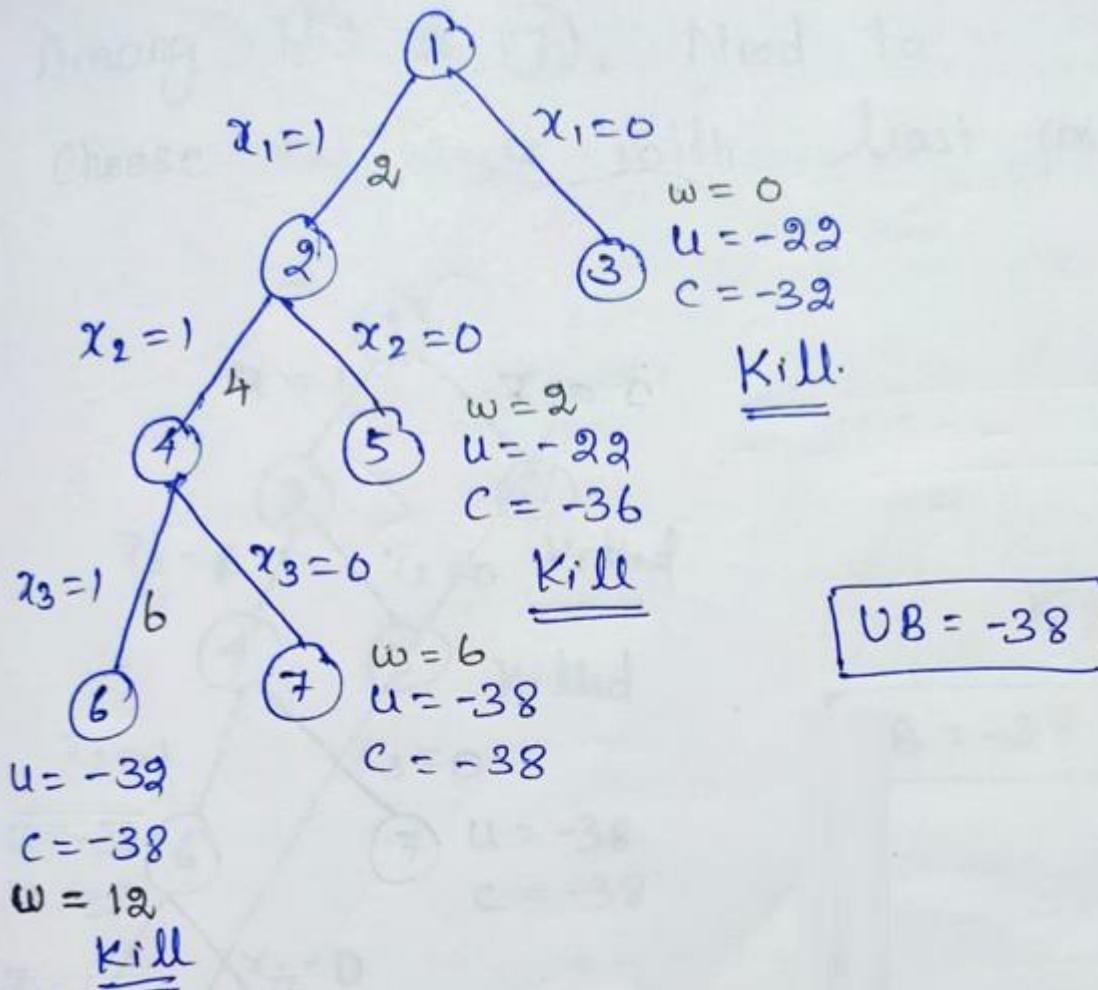
$x_3=1$: No change in u & c

Calculate c and u for node 7 (if $x_3=0$)

$x_3=0$:

$$c = \frac{10}{2} + \frac{10}{4} + \cancel{\frac{12}{6}} + \frac{18}{9} = -38$$

$$u = \frac{10}{2} + \frac{10}{4} + \cancel{\frac{12}{6}} + \frac{18}{9} = -38$$



1. There is a change in UB,
Because, $u < UB$, $-38 < -32$,
So need to update UB as -38

2. Check for non-promising nodes:
Before exploration of next node, need to check the bounding condition for the nodes.

Next exploration nodes nominees: 3, 5, 6 & 7

Bounding condition: If $c > UB$, then the node is nonpromising node, so KILL the node, no need of exploring such a node.

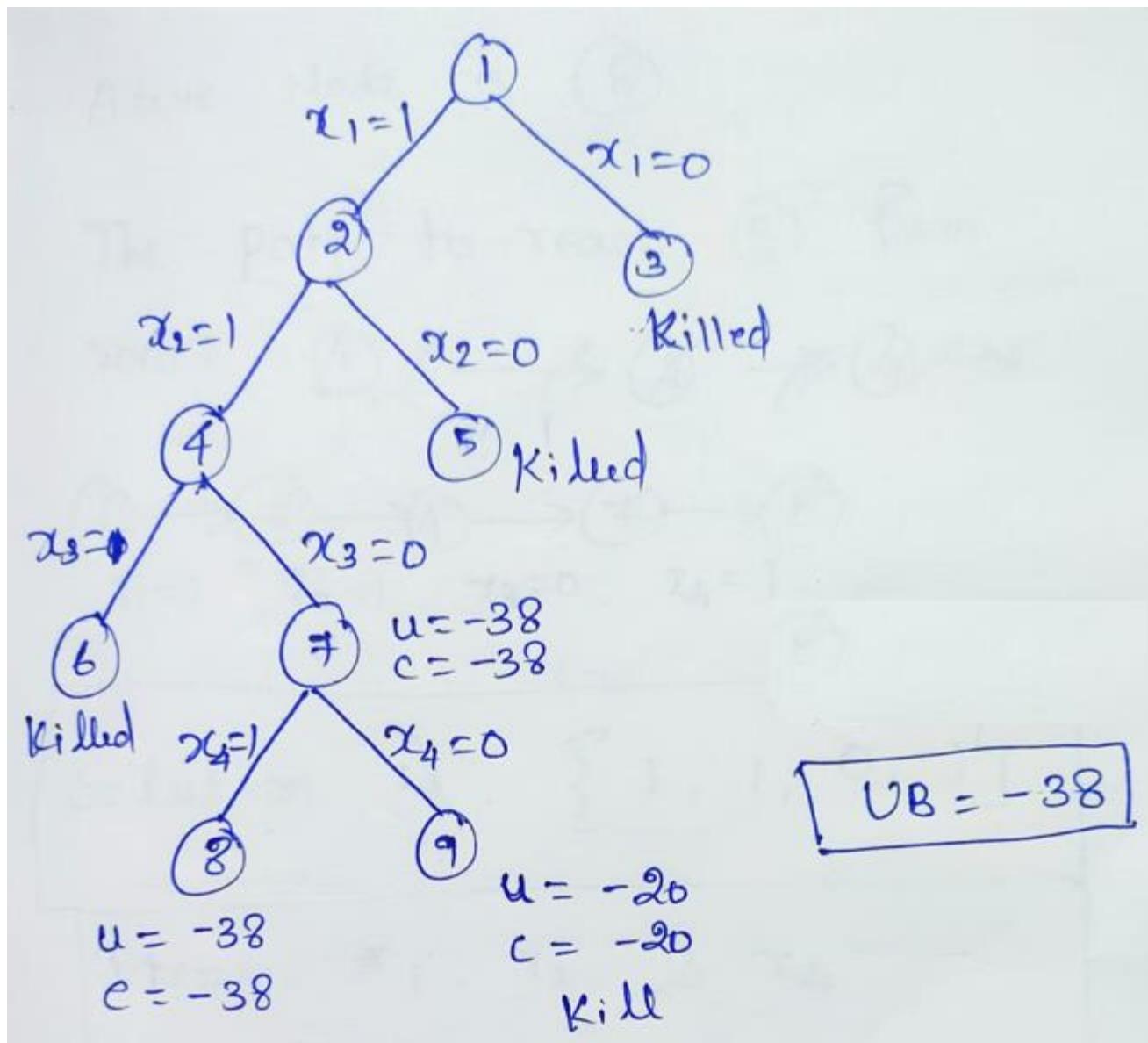
Node 3: $c > UB$, $-32 > -38$, KILL node 3

Node 5: $c > UB$, $-36 > -38$, KILL node 5

For Node 6 & 7: $c = UB$, both are -38, so no need to kill.

But, for the node 6, total weight so far, is 12 ($2+4+6$),
If next item is included, total weight $12+9=21$ will exceed
the bag capacity, 15. So **node 6 also will get killed**.

So, need to explore only node 7 now

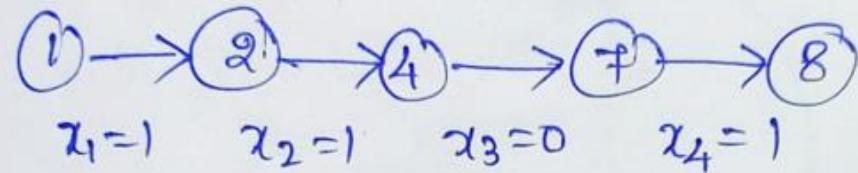


$$\begin{aligned}
 & \underline{x_4 = 0 :} \\
 & c = \frac{10}{2} + \frac{10}{4} + \cancel{\frac{12}{6}} + \cancel{\frac{18}{9}} = -20 \\
 & u = -20
 \end{aligned}$$

⑨ $\Rightarrow c = -20 > -38$, Kill ⑨

Alive Node is ⑧

The path to reach ⑧ from root:



Solution is: {1, 1, 0, 1}

Items: x_1 , x_2 & x_4

$$\text{Profit} = 10 + 10 + 18 = 38$$

$$\text{Weight} = 2 + 4 + 9 = 15$$

Branch & Bound Approach

Travelling Salesman Problem

Travelling Salesman Problem: [TSP]



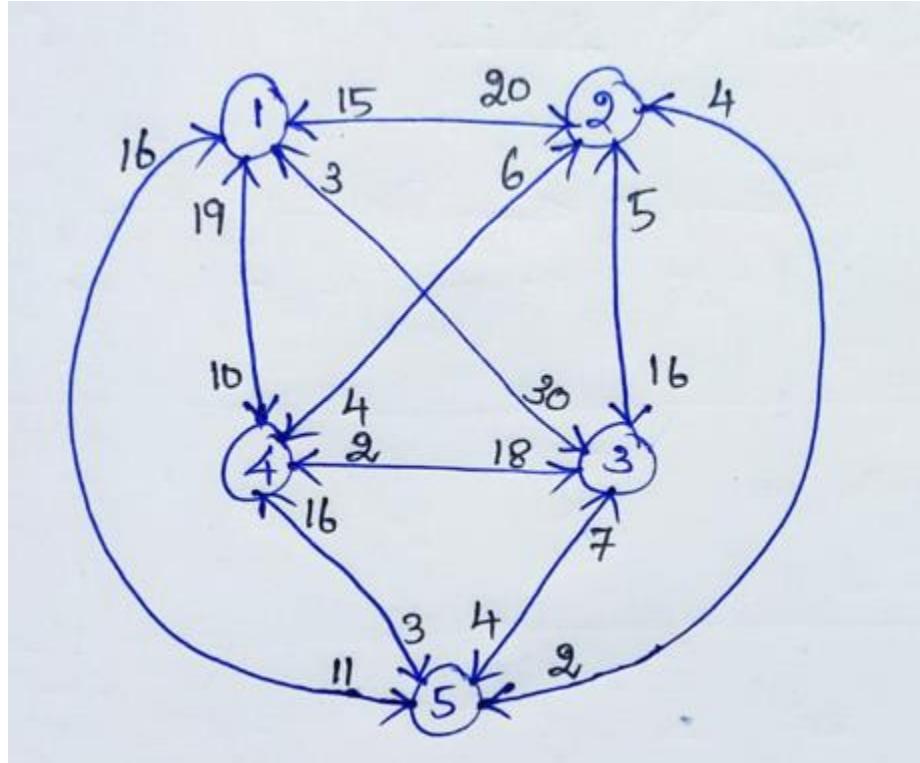
SASTRA
ENGINEERING • MANAGEMENT • LAW • SCIENCES • HUMANITIES EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Problem:

→ Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visit every city exactly ^{once} and returns to the starting point.

→ In other words, the TSP problem is to find a minimum weight Hamiltonian Cycle.

Example



	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

Start

1

Finding Cost of Node 1 (1 to 1)

	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

Start

1 c=25

	1	2	3	4	5	min
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

	1	2	3	4	5	min
1	∞	10	20	0	1	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	15	∞	0	3
5	12	0	3	12	∞	4

Matrix after row-wise reducing

	1	2	3	4	5	min
1	∞	10	20	0	1	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	15	∞	0	3
5	12	0	3	12	∞	4
min	1	0	3	0	0	

	1	2	3	4	5	min
1	∞	10	17	0	1	10
2	12	∞	11	2	0	2
3	0	3	∞	0	2	2
4	15	3	12	∞	0	3
5	11	0	0	12	∞	4
min	1	0	3	0	0	25

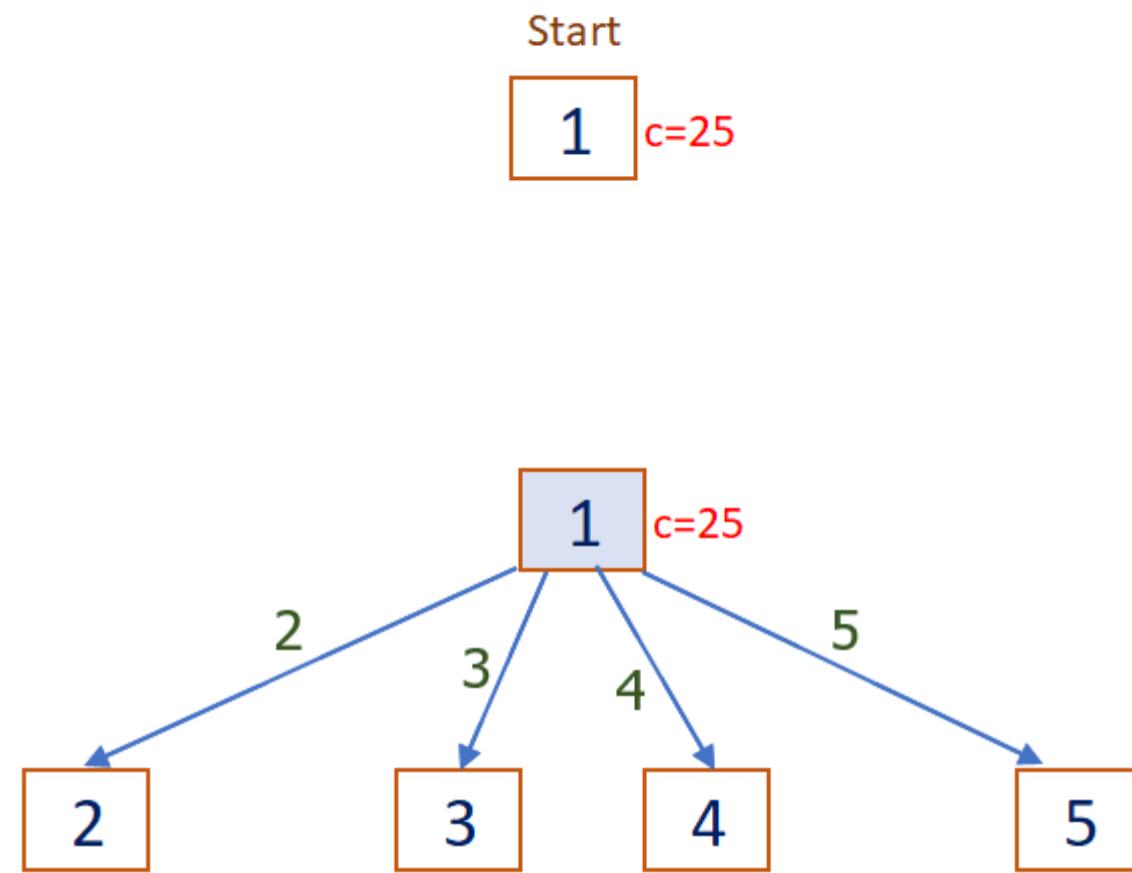
Matrix after column-wise reducing

Cost of Matrix = 25

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

Matrix for Node 1

Reduced Cost = 25



Finding Cost of Node 2 (1 to 2)

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

Matrix for Node 1

Reduced Cost = 25

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

Row - 1, Column - 2, Cell - (2,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	2	0	0
3	0	∞	∞	0	2	0
4	15	∞	12	∞	0	0
5	11	∞	0	12	∞	0

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	2	0	0
3	0	∞	∞	0	2	0
4	15	∞	12	∞	0	0
5	11	∞	0	12	∞	0

No Change after row-wise reducing

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	2	0	0
3	0	∞	∞	0	2	0
4	15	∞	12	∞	0	0
5	11	∞	0	12	∞	0
min	0	--	0	0	0	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	2	0	0
3	0	∞	∞	0	2	0
4	15	∞	12	∞	0	0
5	11	∞	0	12	∞	0
min	0	--	0	0	0	0

No Change after column-wise reducing

Cost of Reduced Matrix = 0

Cost of Node 2 = $C(1,2) + C(\text{Node 1 Matrix}) + C(\text{reduced matrix})$

Cost of Node 2 = $10 + 25 + 0 = 35$

1	2	3	4	5	
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

Matrix for Node 1

Reduced Cost = 25

1	2	3	4	5	min	
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	2	0	0
3	0	∞	∞	0	2	0
4	15	∞	12	∞	0	0
5	11	∞	0	12	∞	0
min	0	--	0	0	0	0

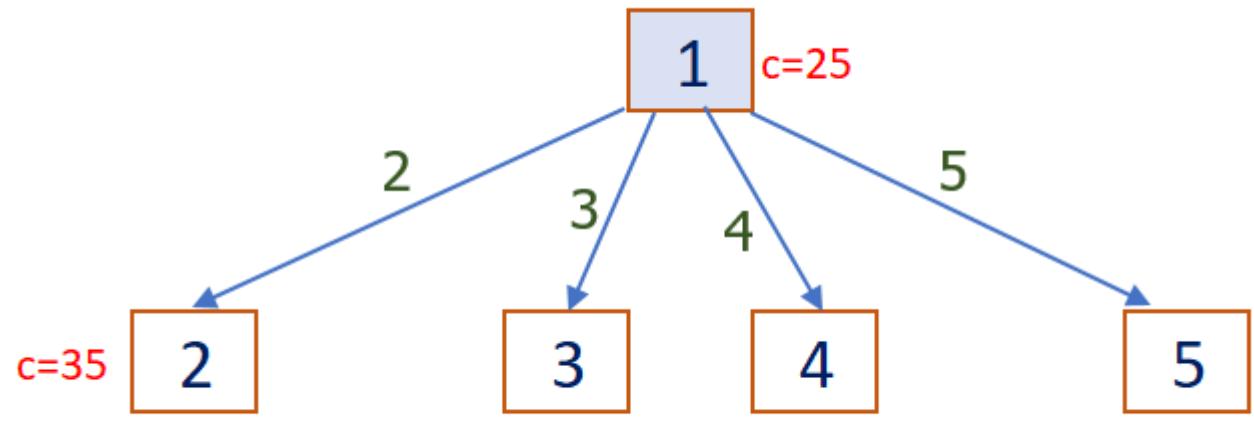
No Change after column-wise reducing

Cost of Reduced Matrix = 0

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

Matrix for Node 2

Reduced Cost = 35



Finding Cost of Node 3 (1 to 3)

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

Matrix for Node 1

Reduced Cost = 25

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	2	0
3	∞	3	∞	0	2
4	15	3	∞	∞	0
5	11	0	∞	12	∞

Row - 1, Column - 3, Cell - (3,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	∞	2	0	0
3	∞	3	∞	0	2	0
4	15	3	∞	∞	0	0
5	11	0	∞	12	∞	0

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	∞	2	0	0
3	∞	3	∞	0	2	0
4	15	3	∞	∞	0	0
5	11	0	∞	12	∞	0

No Change after row-wise reducing

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	∞	2	0	0
3	∞	3	∞	0	2	0
4	15	3	∞	∞	0	0
5	11	0	∞	12	∞	0
min	11	0	--	0	0	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	1	∞	∞	2	0	0
3	∞	3	∞	0	2	0
4	4	3	∞	∞	0	0
5	0	0	∞	12	∞	0
min	11	0	--	0	0	11

Matrix after column-wise reducing

Cost of Reduced Matrix = 11

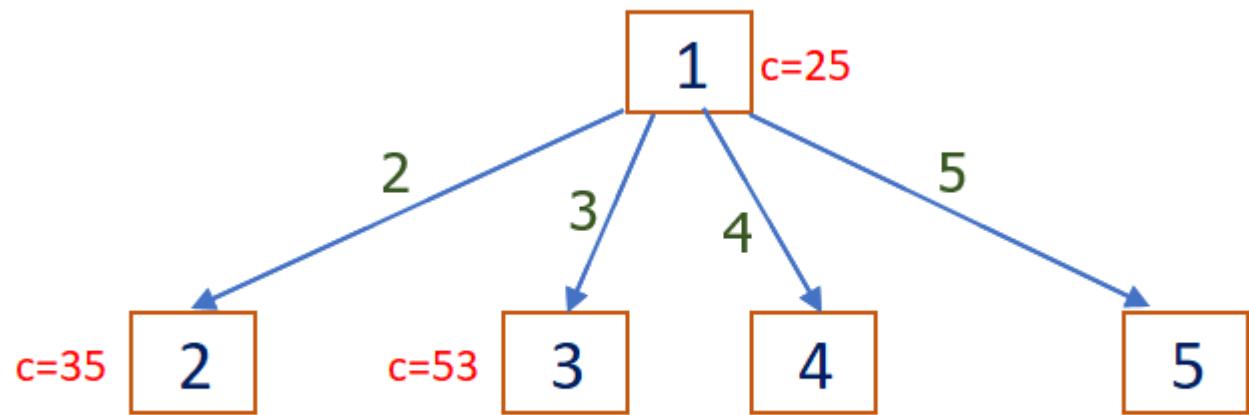
Cost of Node 3 = C(1,3)+C(Node 1 Matrix)+C(reduced matrix)

Cost of Node 3= 17 + 25 + 11 = 53

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	2	0
3	∞	3	∞	0	2
4	4	3	∞	∞	0
5	0	0	∞	12	∞

Matrix for Node 3

Reduced Cost = 53



Finding Cost of Node 4 (1 to 4)

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

Matrix for Node 1

Reduced Cost = 25

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

Row - 1, Column - 4, Cell - (4,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	11	∞	0	0
3	0	3	∞	∞	2	0
4	∞	3	12	∞	0	0
5	11	0	0	∞	∞	0

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	11	∞	0	0
3	0	3	∞	∞	2	0
4	∞	3	12	∞	0	0
5	11	0	0	∞	∞	0

No Change after row-wise reducing

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	11	∞	0	0
3	0	3	∞	∞	2	0
4	∞	3	12	∞	0	0
5	11	0	0	∞	∞	0
min	0	0	0	--	0	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	11	∞	0	0
3	0	3	∞	∞	2	0
4	∞	3	12	∞	0	0
5	11	0	0	∞	∞	0
min	0	0	0	--	0	0

No Change after column-wise reducing

Cost of Reduced Matrix = 0

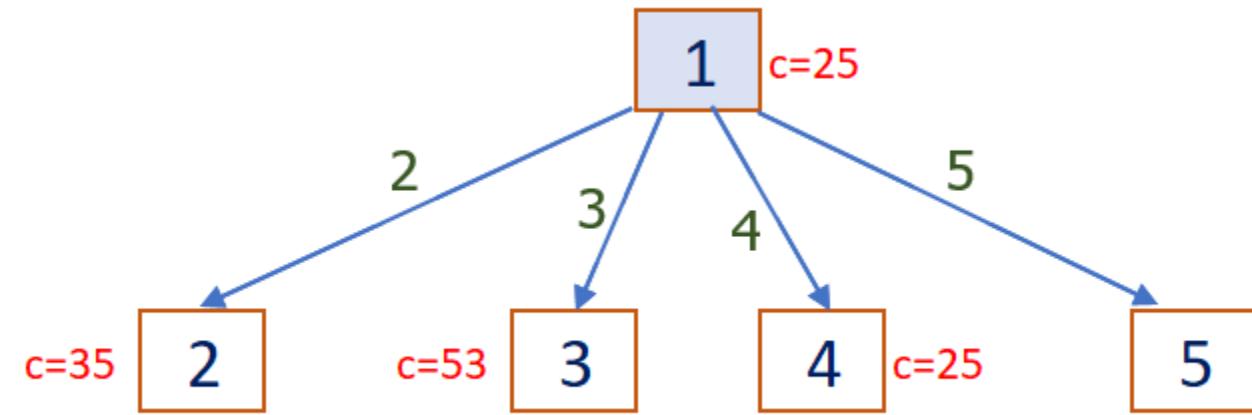
Cost of Node 4 = $C(1,4) + C(\text{Node 1 Matrix}) + C(\text{reduced matrix})$

Cost of Node 4 = $0 + 25 + 0 = 25$

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

Matrix for Node 4

Reduced Cost = 25



Finding Cost of Node 5 (1 to 5)

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

Matrix for Node 1

Reduced Cost = 25

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	2	∞
3	0	3	∞	0	∞
4	15	3	12	∞	∞
5	∞	0	0	12	∞

Row - 1, Column - 5, Cell - (5,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	11	2	∞	2
3	0	3	∞	0	∞	0
4	15	3	12	∞	∞	3
5	∞	0	0	12	∞	0

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	10	∞	9	0	∞	2
3	0	3	∞	0	∞	0
4	12	0	9	∞	∞	3
5	∞	0	0	12	∞	0

Matrix after row-wise reduction

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	10	∞	9	0	∞	2
3	0	3	∞	0	∞	0
4	12	0	9	∞	∞	3
5	∞	0	0	12	∞	0
min	0	0	0	0	--	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	10	∞	9	0	∞	2
3	0	3	∞	0	∞	0
4	12	0	9	∞	∞	3
5	∞	0	0	12	∞	0
min	0	0	0	0	--	5

No Change after column-wise reducing

Cost of Reduced Matrix = 5

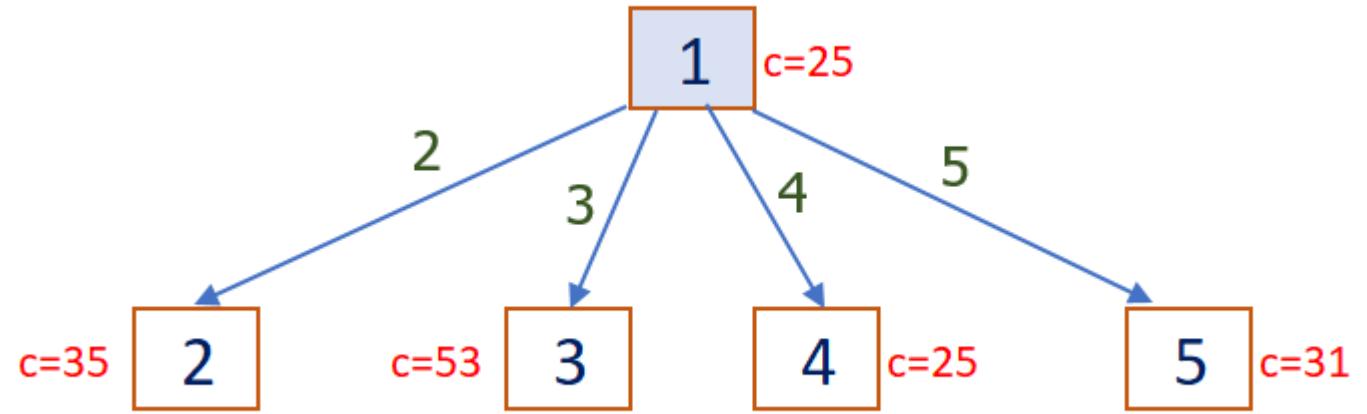
Cost of Node 5 = C(1,5)+C(Node 1 Matrix)+C(reduced matrix)

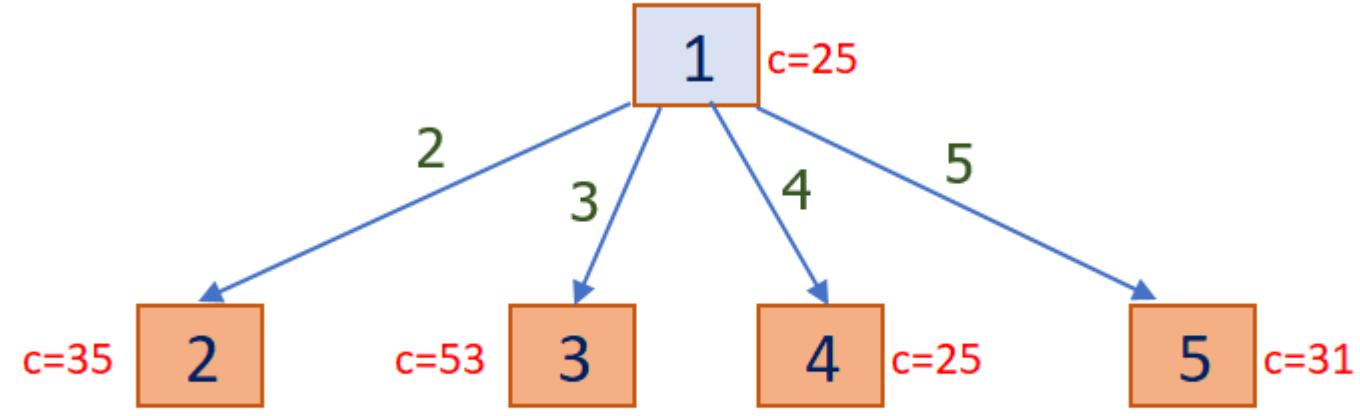
Cost of Node 5= 1 + 25 + 5 = 31

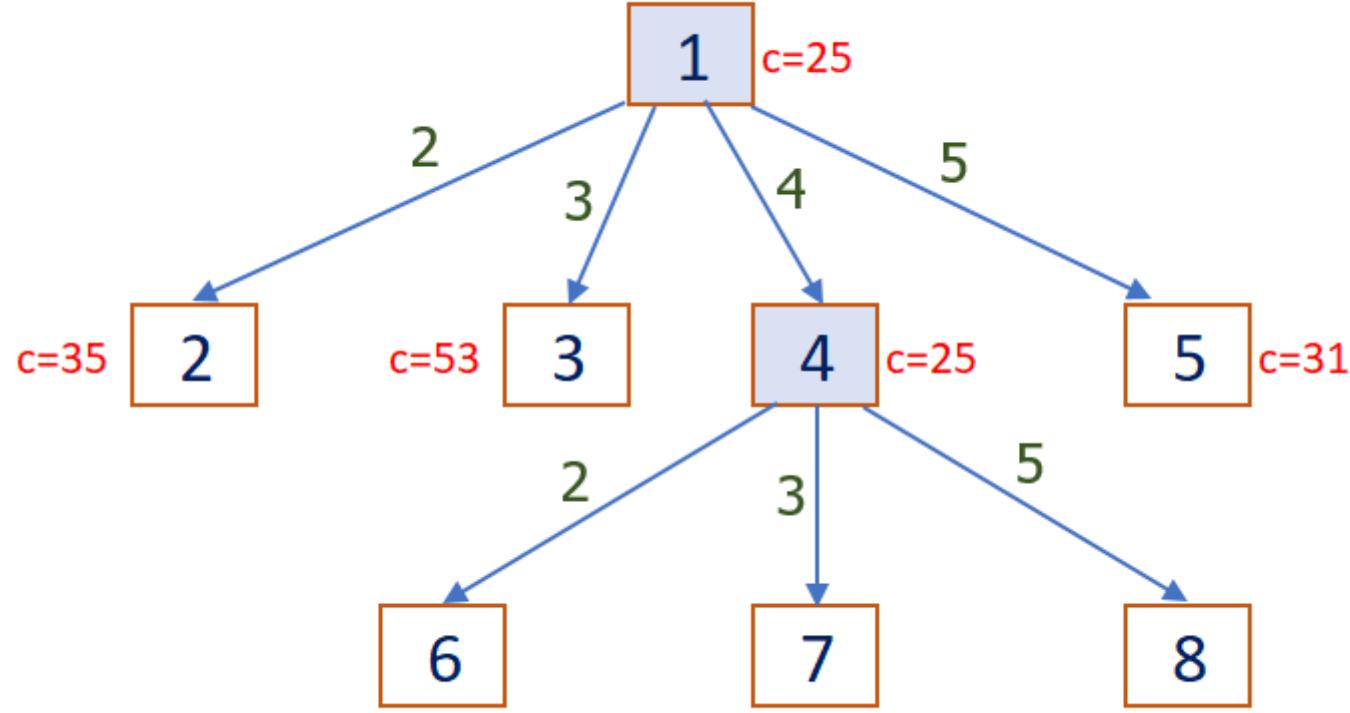
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	10	∞	9	0	∞
3	0	3	∞	0	∞
4	12	0	9	∞	∞
5	∞	0	0	12	∞

Matrix for Node 5

Reduced Cost = 31







Finding Cost of Node 6 (4 to 2)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

Matrix for Node 4

Reduced Cost = 25

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

Row - 4, Column - 2, Cell - (2,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	∞	0	0
3	0	∞	∞	∞	2	0
4	∞	∞	∞	∞	∞	--
5	11	∞	0	∞	∞	0

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	∞	0	0
3	0	∞	∞	∞	2	0
4	∞	∞	∞	∞	∞	--
5	11	∞	0	∞	∞	0

No Change after row-wise reducing

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	∞	0	0
3	0	∞	∞	∞	2	0
4	∞	∞	∞	∞	∞	--
5	11	∞	0	∞	∞	0
min	0	--	0	--	0	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	11	∞	0	0
3	0	∞	∞	∞	2	0
4	∞	∞	∞	∞	∞	--
5	11	∞	0	∞	∞	0
min	0	--	0	--	0	0

No Change after column-wise reducing

Cost of Reduced Matrix = 0

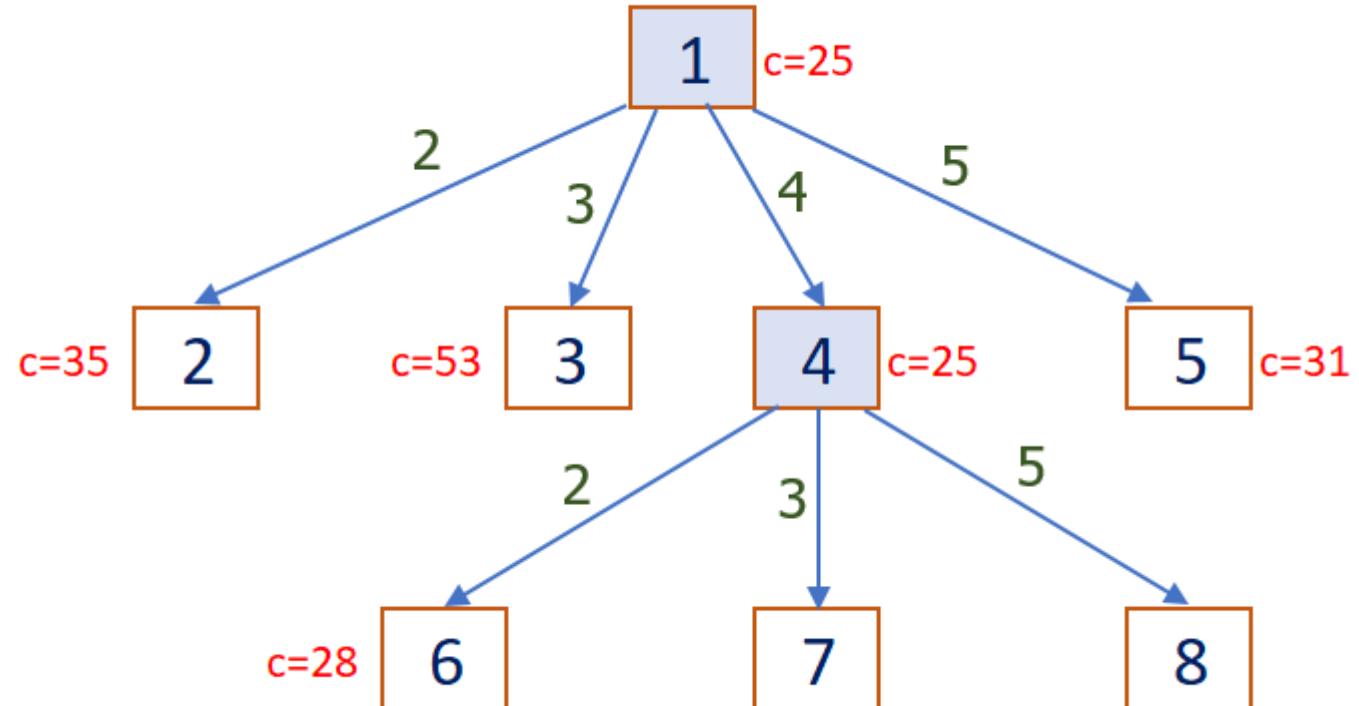
Cost of Node 6 = $C(4,2) + C(\text{Node 4 Matrix}) + C(\text{reduced matrix})$

Cost of Node 6 = $3 + 25 + 0 = 28$

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

Matrix for Node 6

Reduced Cost = 28



Finding Cost of Node 7 (4 to 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

Matrix for Node 4

Reduced Cost = 25

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	∞	0
3	∞	3	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	0	∞	∞	∞

Row - 4, Column - 3, Cell - (3,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	∞	∞	0	0
3	∞	3	∞	∞	2	2
4	∞	∞	∞	∞	∞	--
5	11	0	∞	∞	∞	0

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	∞	∞	0	0
3	∞	1	∞	∞	0	2
4	∞	∞	∞	∞	∞	--
5	11	0	∞	∞	∞	0

Matrix after row-wise reduction

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	∞	∞	0	0
3	∞	1	∞	∞	0	2
4	∞	∞	∞	∞	∞	--
5	11	0	∞	∞	∞	0
min	11	0	--	--	0	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	1	∞	∞	∞	0	0
3	∞	1	∞	∞	0	2
4	∞	∞	∞	∞	∞	--
5	0	0	∞	∞	∞	0
min	11	0	--	--	0	13

Matrix after column-wise reducing

Cost of Reduced Matrix = 13

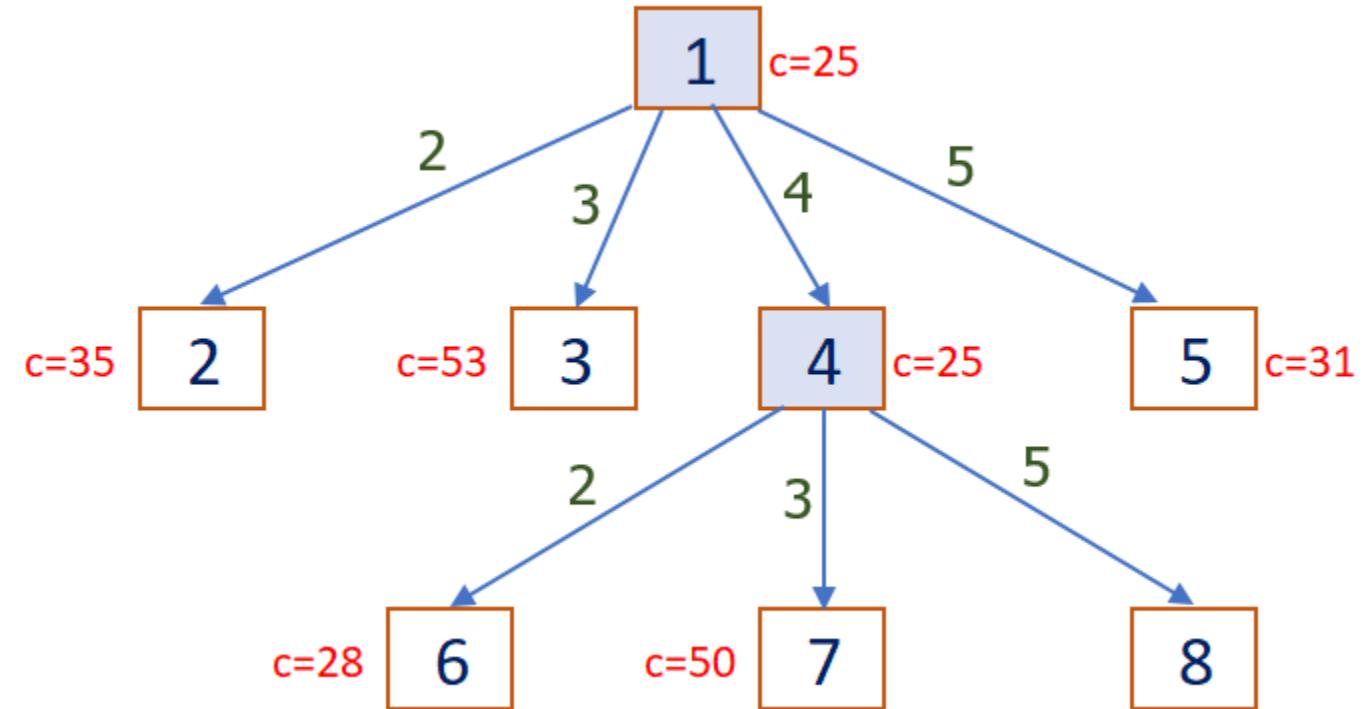
Cost of Node 7 = $C(4,3) + C(\text{Node 4 Matrix}) + C(\text{reduced matrix})$

Cost of Node 7 = $12 + 25 + 13 = 50$

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	∞	0
3	∞	1	∞	∞	0
4	∞	∞	∞	∞	∞
5	0	0	∞	∞	∞

Matrix for Node 7

Reduced Cost = 50



Finding Cost of Node 8 (4 to 5)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

Matrix for Node 4

Reduced Cost = 25

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	∞
3	0	3	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	0	∞	∞

Row - 4, Column - 5, Cell - (5,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	12	∞	11	∞	∞	11
3	0	3	∞	∞	∞	0
4	∞	∞	∞	∞	∞	--
5	∞	0	0	∞	∞	0

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	1	∞	0	∞	∞	11
3	0	3	∞	∞	∞	0
4	∞	∞	∞	∞	∞	--
5	∞	0	0	∞	∞	0

Matrix after row-wise reduction

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	1	∞	0	∞	∞	11
3	0	3	∞	∞	∞	0
4	∞	∞	∞	∞	∞	--
5	∞	0	0	∞	∞	0
min	0	0	0	--	--	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	1	∞	0	∞	∞	11
3	0	3	∞	∞	∞	0
4	∞	∞	∞	∞	∞	--
5	∞	0	0	∞	∞	0
min	0	0	0	--	--	11

No change after column-wise reducing

Cost of Reduced Matrix =11

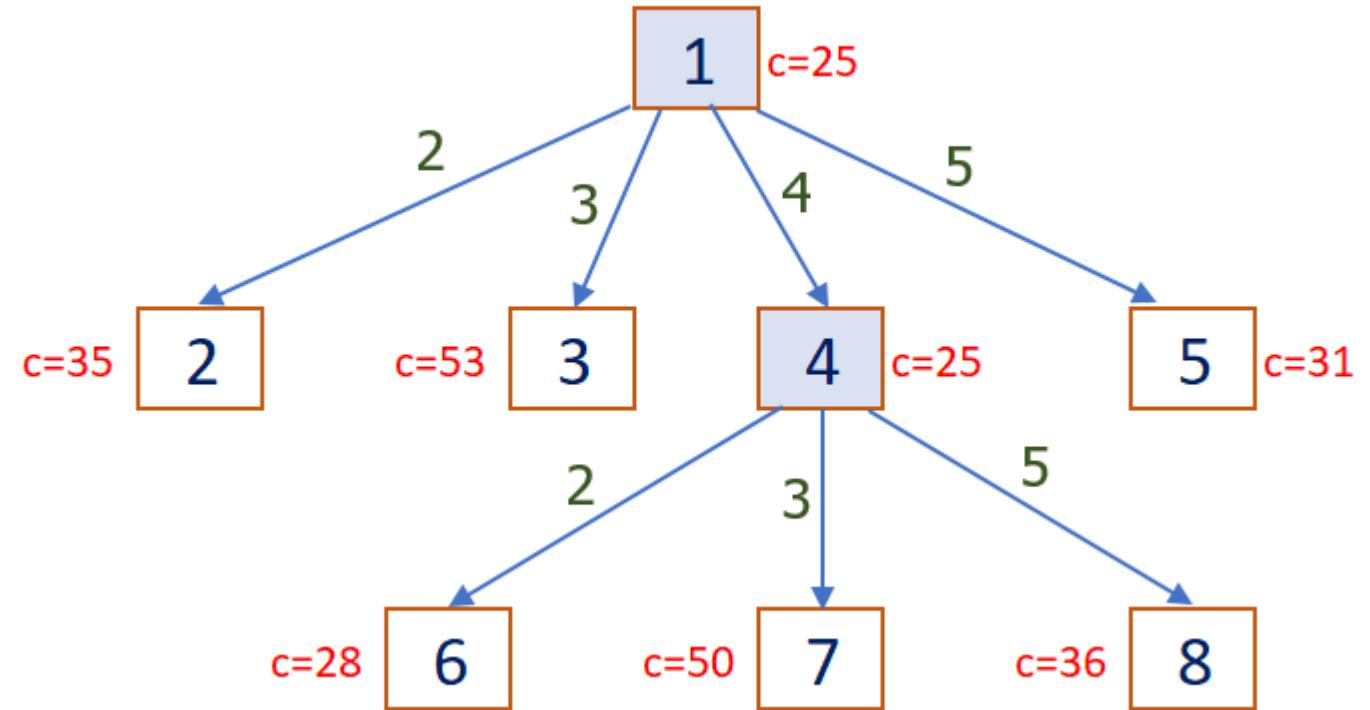
Cost of Node 8 = $C(4,5) + C(\text{Node 4 Matrix}) + C(\text{reduced matrix})$

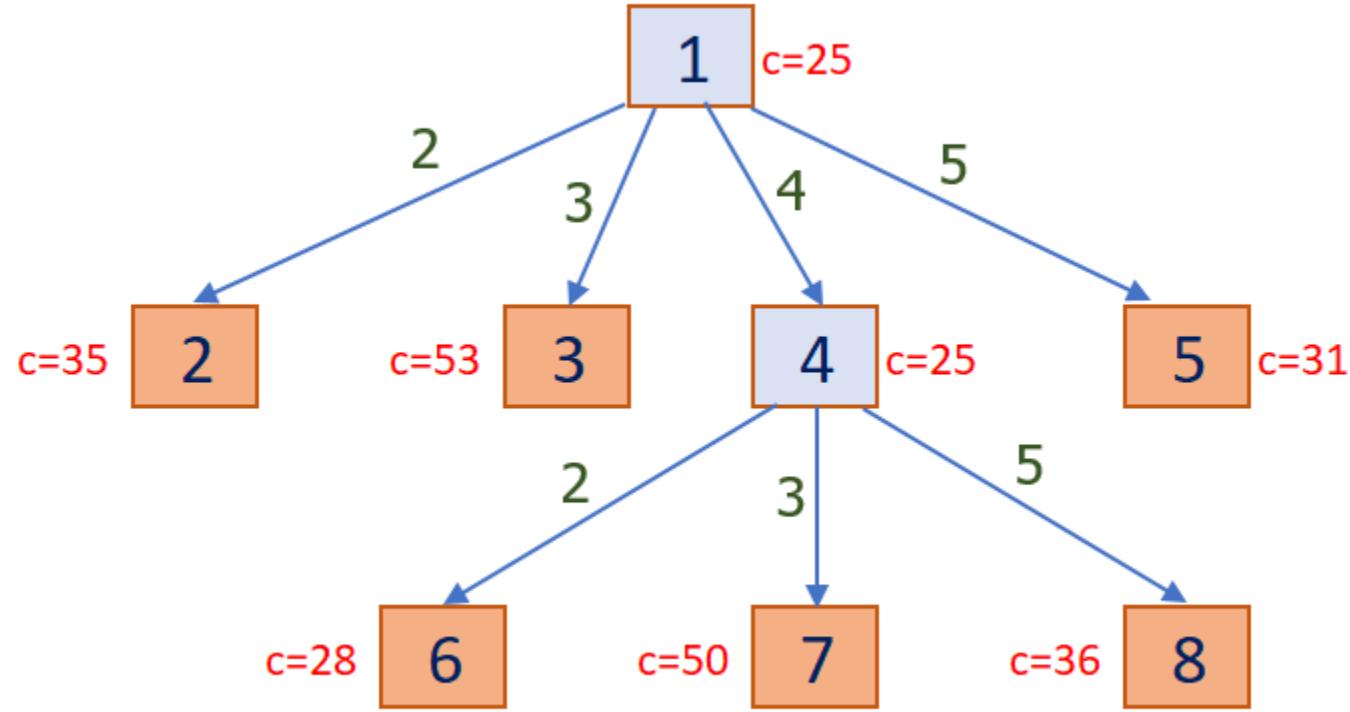
Cost of Node 8 = $0 + 25 + 11 = 36$

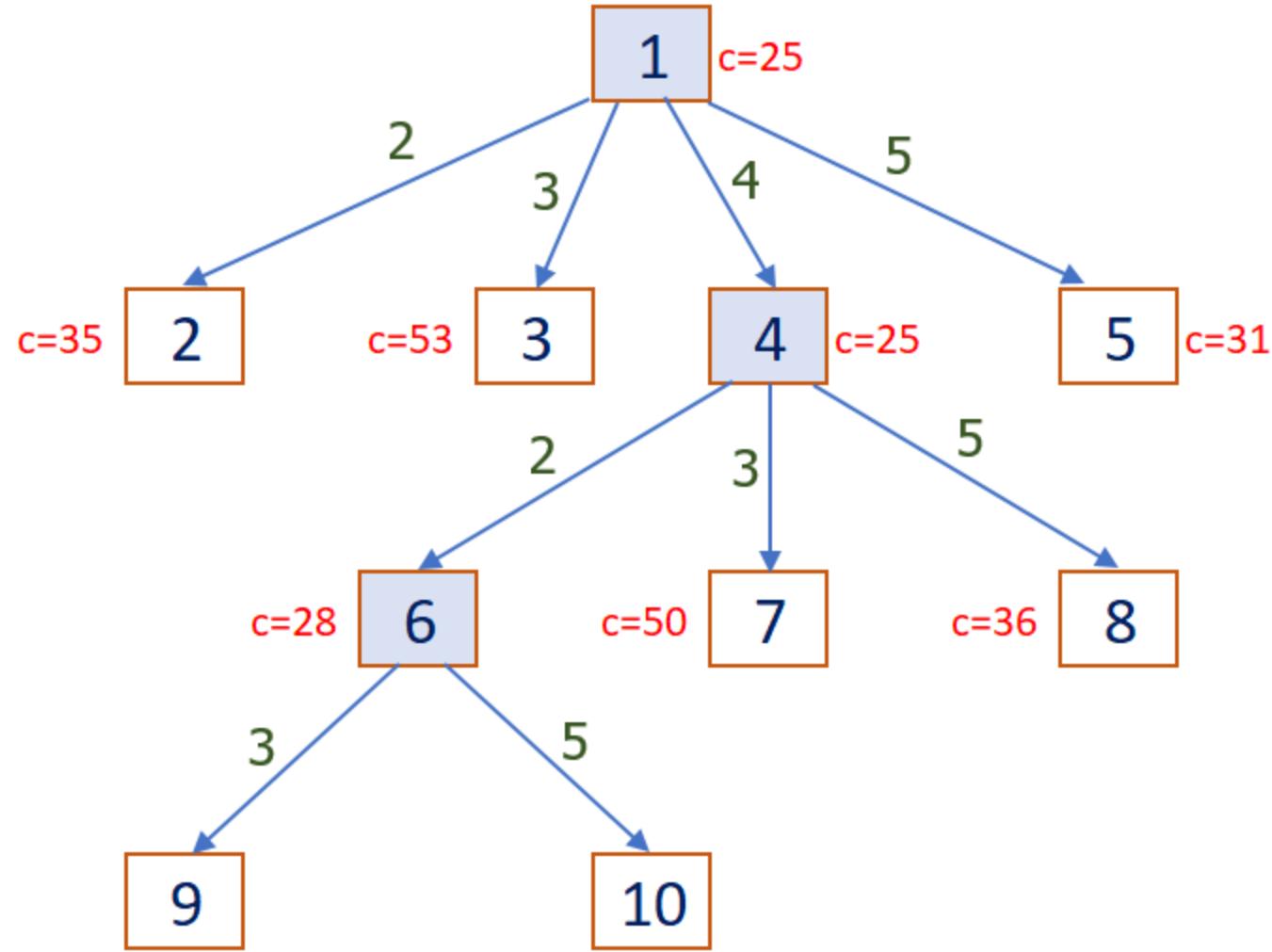
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	0	∞	∞
3	0	3	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	0	∞	∞

Matrix for Node 8

Reduced Cost = 36







Finding Cost of Node 9 (2 to 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

Matrix for Node 6

Reduced Cost = 28

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	∞	∞	∞

Row - 2, Column - 3, Cell - (3,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	∞	∞	∞	--
3	∞	∞	∞	∞	2	2
4	∞	∞	∞	∞	∞	--
5	11	∞	∞	∞	∞	11

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	∞	∞	∞	--
3	∞	∞	∞	∞	0	2
4	∞	∞	∞	∞	∞	--
5	0	∞	∞	∞	∞	11

Matrix after row-wise reduction

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	∞	∞	∞	--
3	∞	∞	∞	∞	0	2
4	∞	∞	∞	∞	∞	--
5	0	∞	∞	∞	∞	11
min	0	--	--	--	0	

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	∞	∞	∞	--
3	∞	∞	∞	∞	0	2
4	∞	∞	∞	∞	∞	--
5	0	∞	∞	∞	∞	11
min	0	--	--	--	0	13

No change after column-wise reducing
Cost of Reduced Matrix = 13

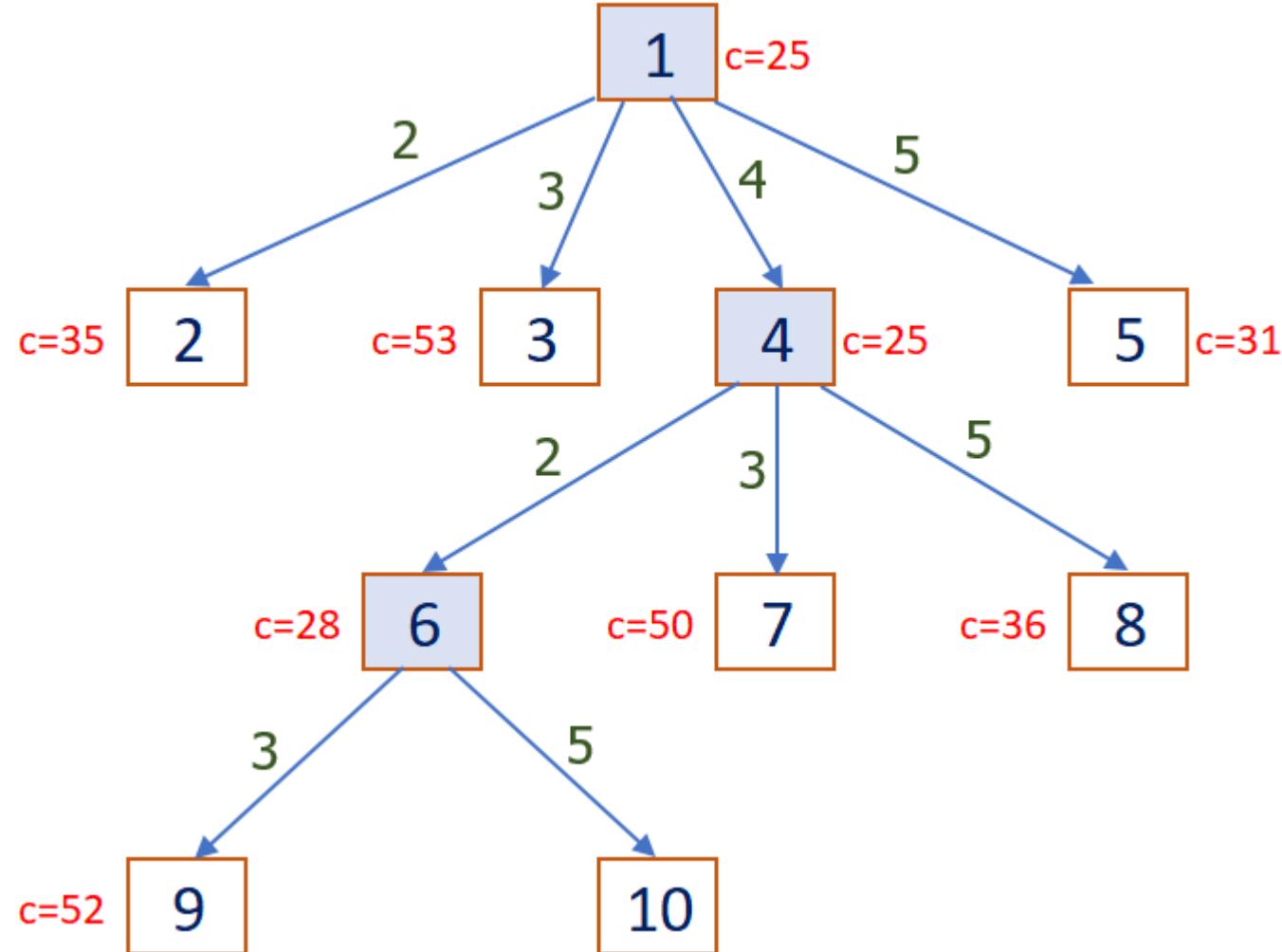
Cost of Node 9 = $C(2,3) + C(\text{Node 6 Matrix}) + C(\text{reduced matrix})$

Cost of Node 9 = $11 + 28 + 13 = 52$

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞
5	0	∞	∞	∞	∞

Matrix for Node 9

Reduced Cost = 52



Finding Cost of Node 10 (2 to 5)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

Matrix for Node 6

Reduced Cost = 28

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

Row - 2, Column - 5, Cell - (5,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	∞	∞	∞	--
3	0	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞	--
5	∞	∞	0	∞	∞	0
min	0	--	0	--	--	0

No change after reducing

Cost of Reduced Matrix = 0

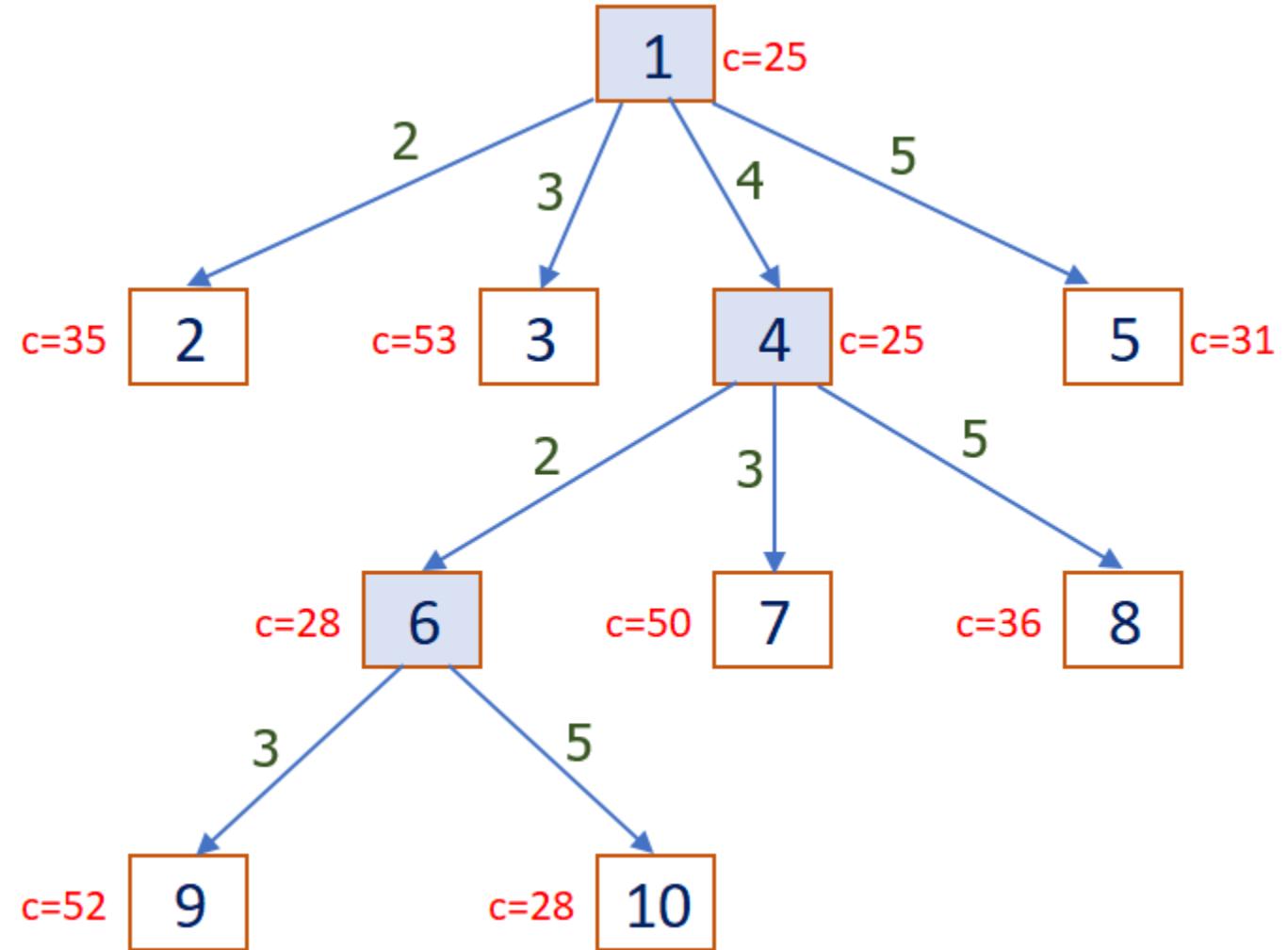
Cost of Node 10 = C(2,5)+C(Node 6 Matrix)+C(reduced matrix)

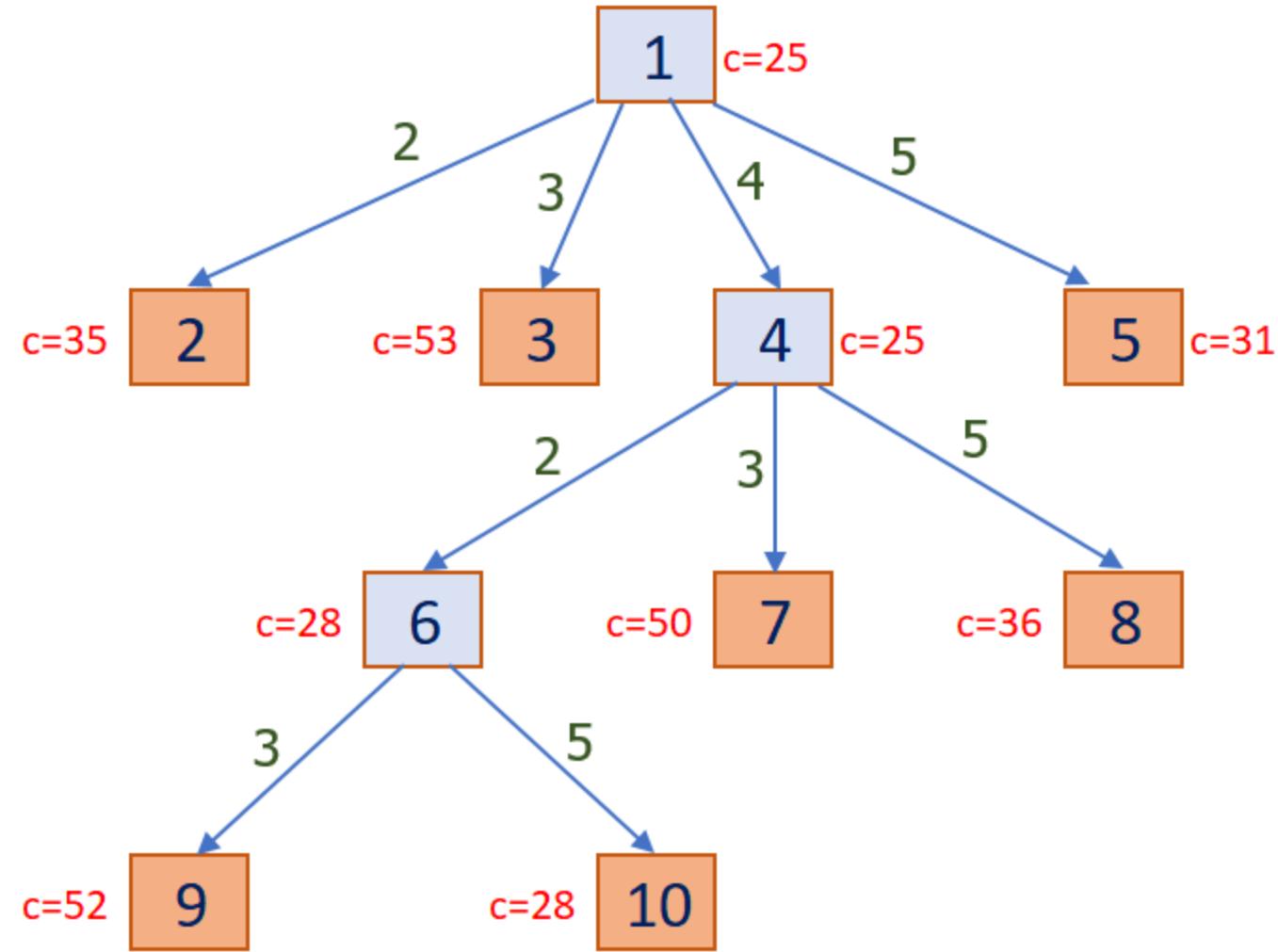
Cost of Node 10= 0 + 28 + 0 = 28

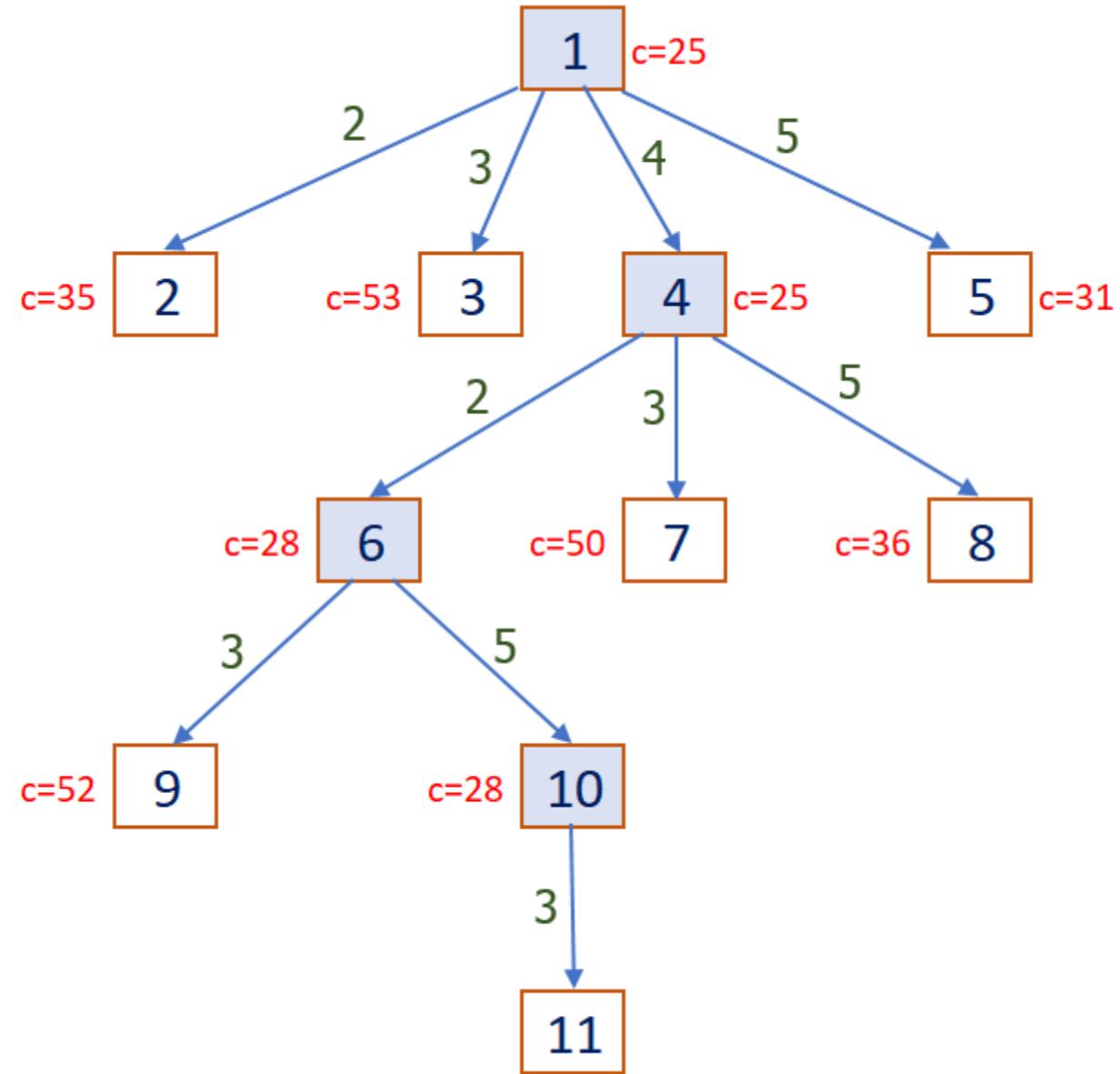
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

Matrix for Node 10

Reduced Cost = 28







Finding Cost of Node 11 (5 to 3)

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

Matrix for Node 10

Reduced Cost = 28

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

Row - 5, Column - 3, Cell - (3,1)

	1	2	3	4	5	min
1	∞	∞	∞	∞	∞	--
2	∞	∞	∞	∞	∞	--
3	∞	∞	∞	∞	∞	--
4	∞	∞	∞	∞	∞	--
5	∞	∞	∞	∞	∞	--
min	--	--	--	--	--	0

No change after reducing

Cost of Reduced Matrix = 0

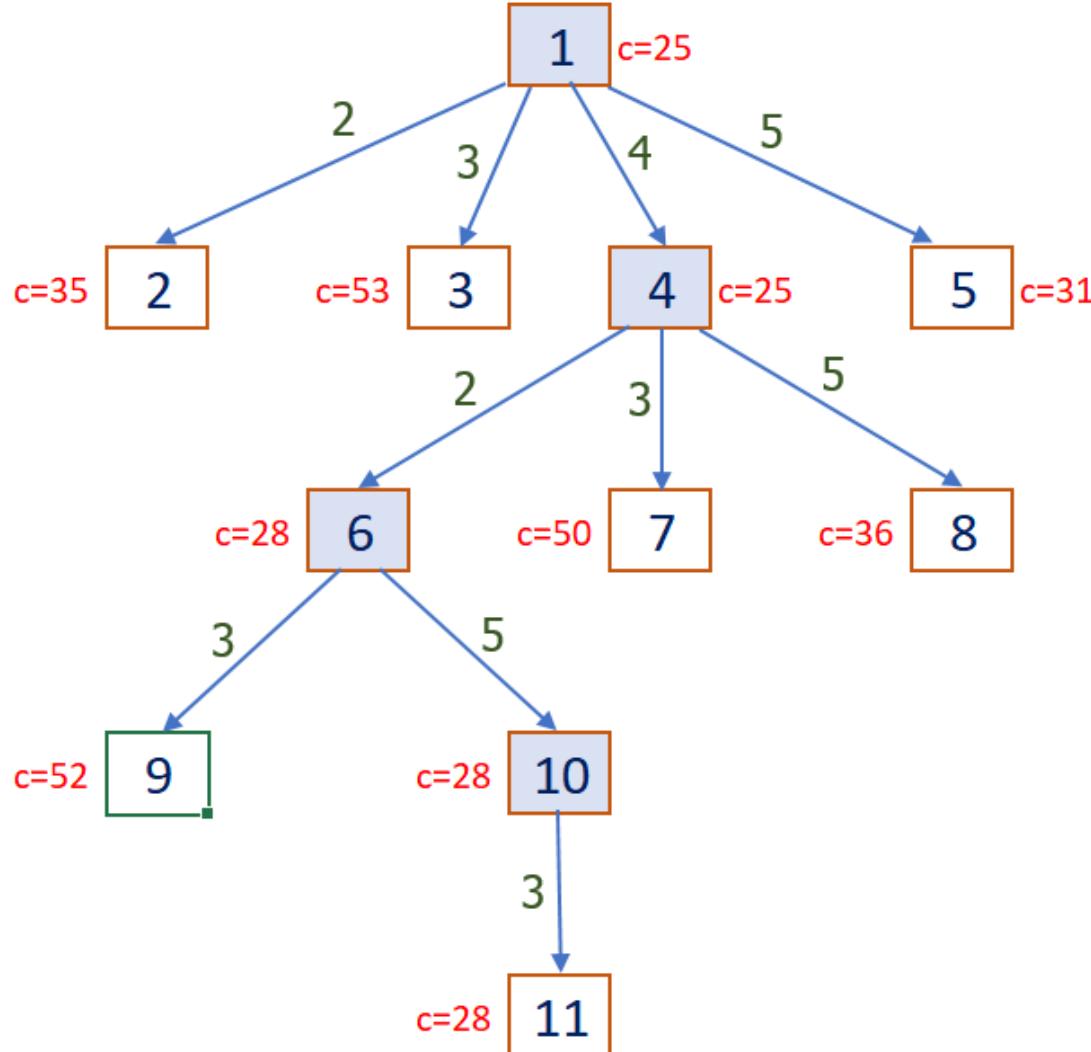
Cost of Node 11 = $C(5,3) + C(\text{Node 10 Matrix}) + C(\text{reduced matrix})$

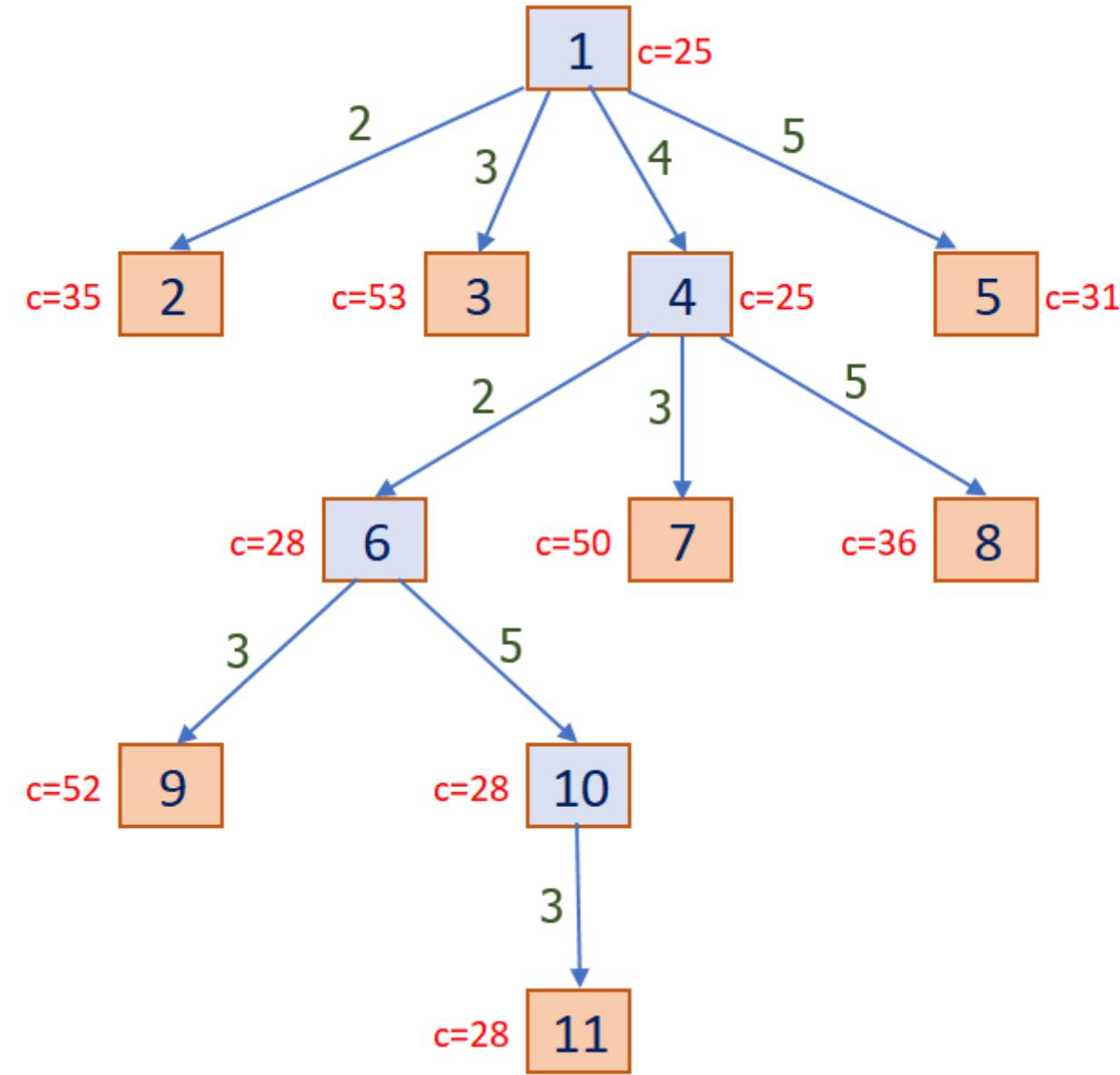
Cost of Node 11 = $0 + 28 + 0 = 28$

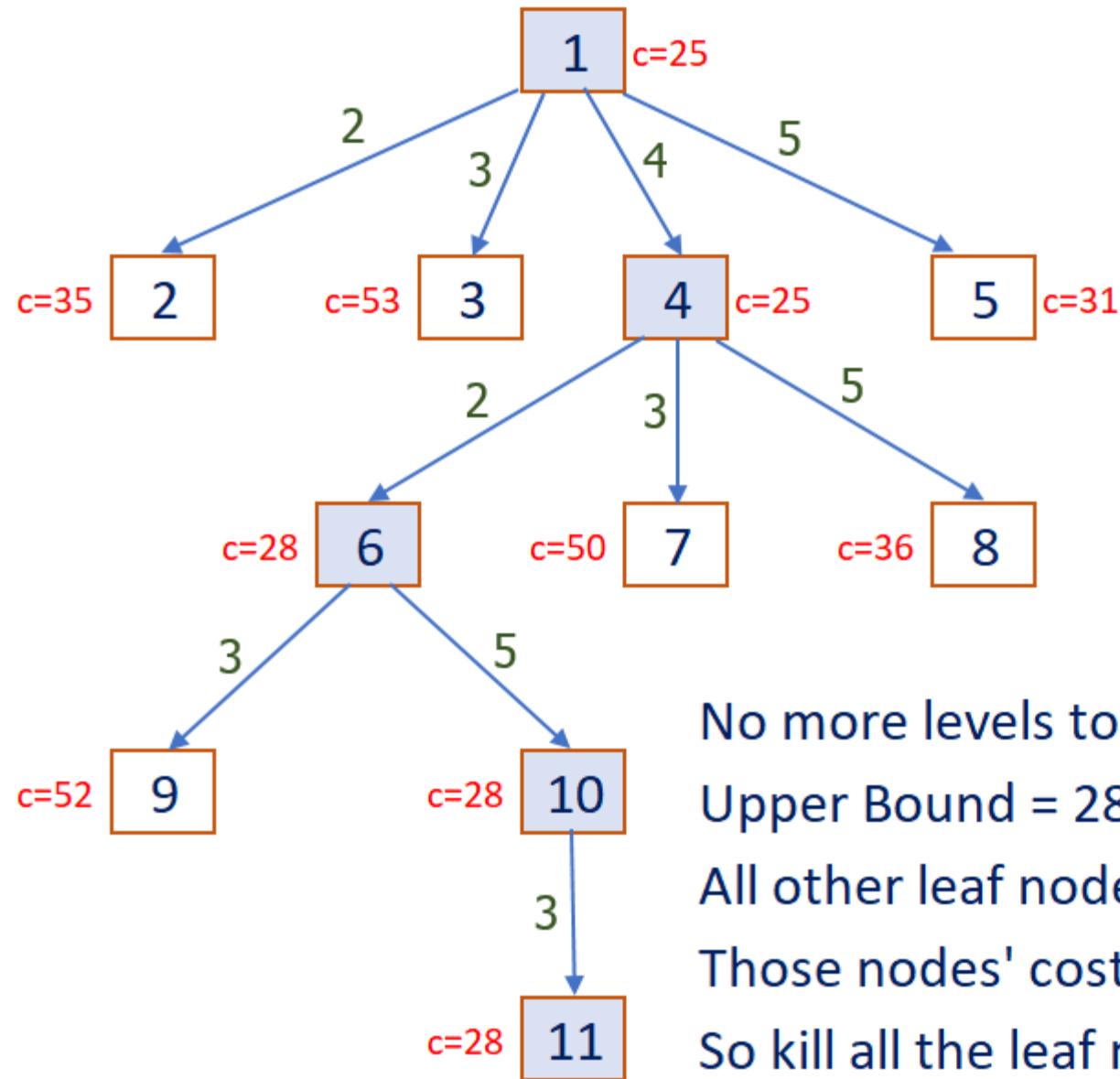
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞

Matrix for Node 11

Reduced Cost = 28

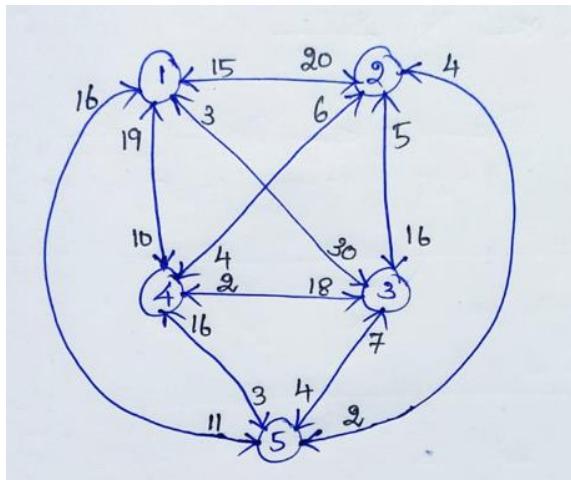






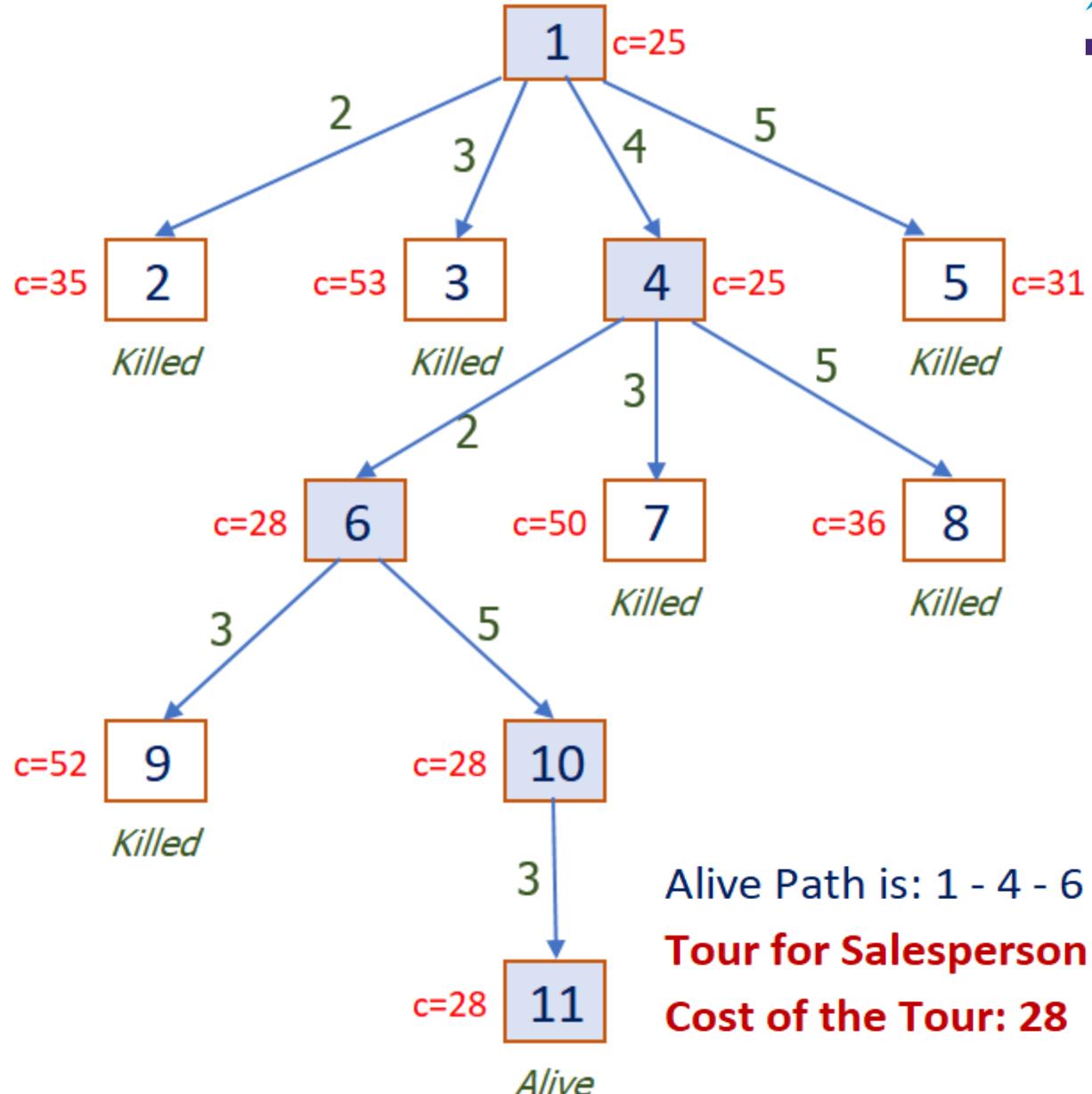
No more levels to explore further, So update upper bound
 Upper Bound = 28
 All other leaf nodes, 2, 3, 9, 7, 8 and 5.
 Those nodes' cost are greater than Upper Bound,
 So kill all the leaf nodes except node 11

Input Graph



	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

Result



Summary

- ✓ Comparisons of followings
 - ✓ Divide & Conquer Vs Dynamic Programming
 - ✓ Dynamic Programming Vs Greedy
 - ✓ Brute-force Vs Backtracking
 - ✓ Backtracking Vs Branch & Bound
- ✓ Problems & Algorithms for the following
 - ✓ TSP – Branch & Bound
 - ✓ Optimal BST – Dynamic Programming
 - ✓ 4-Queen Problem – Backtracking
 - ✓ 0/1 Knapsack Problem – Dynamic Programming
 - ✓ 0/1 Knapsack Problem – Branch & Bound
 - ✓ TSP – Dynamic Programming
 - ✓ String Editing – Dynamic Programming