Tree Searching Strategy

Outlines

- The Breadth-First Search
- Depth-First Search
- Hill Climbing
- Best-First Search Strategy
- The Branch-and-Bound Strategy
- A Personnel Assignment Problem Solved by the Branch-and-Bound Strategy
- The Traveling Salesperson Optimization Problem Solved by the Branch-and-Bound Strategy
- The 0/1 Knapsack Problem Solved by the Branch-and-Bound Strategy

學習目標

- Tree Searching 策略設計的概念
- Tree Searching 策略設計的應用範例
- Tree Searching 策略分類
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)
  - Hill-Climbing Search
  - Best-First Search

Introduction

- In this chapter, we shall show that the solutions of many problems may be represented by trees and therefore the solving of these problems becomes a tree searching problem.

Boolean basics
Literals, clauses, CNFs, implicates

- Boolean function on n variables is a mapping $\{0,1\}^n \rightarrow \{0,1\}$
- Literal = variable or its negation (eg. **p** or **¬p)**
- Clause = disjunction of literals (no complementary pair)
  - $(\mathbf{p} \vee \mathbf{q})$, $(\mathbf{p} \vee \mathbf{q} \vee \mathbf{r})$, $(\mathbf{p} \vee \neg\mathbf{r})$, …
- Conjunctive Normal Form (CNF) = conjunction of clauses (Fact: Every Boolean function has a CNF representation)
  - $C_1 \wedge C_2 \wedge C_3$
- Every Boolean formula can be transformed into the CNF.
- A formula *G* is a logical consequence of a formula *F* if and only if whenever *F* is true, *G* is true

SAT problem Definition

Input: A CNF *formula* on n Boolean variables $x_1, \ldots, x_n$.
Question: Does there exist a truth assignment to $x_1, \ldots, x_n$ which satisfies *formula*?

- **Satisfiability problem**:
  Given a set of **clauses**, one method of determining whether this set of clauses are *satisfiable* is to examine all possible assignments.

  That is, if then are n variables $x_1, x_2, \ldots, x_n$, then we simply examine all 2^n possible assignment.
  In each assignment, x_i is assigned either $\boldsymbol{T}$ or $\boldsymbol{F}$.

The satisfiability problem

- The satisfiability problem (first NP-complete problem)
  - The logical formula :

$$x_1 \lor x_2 \lor x_3$$
$$\& \ \neg x_1$$
$$\& \ \neg x_2$$

the **assignment** :

$$x_1 \leftarrow F, \ x_2 \leftarrow F, \ x_3 \leftarrow T$$

will make the above formula true .

$(\neg x_1, \neg x_2, x_3)$ represents $x_1 \leftarrow F, x_2 \leftarrow F, x_3 \leftarrow T$

- If an assignment makes a formula true, we shall say that this assignment satisfies the formula; otherwise, it falsifies the formula.

- If there is at least one assignment which satisfies a formula, then we say that this formula is satisfiable; otherwise, it is unsatisfiable.

- An unsatisfiable formula :

$$\begin{array}{l} \quad x_1 \vee x_2 \\ \& \ x_1 \vee \neg x_2 \\ \& \ \neg x_1 \vee x_2 \\ \& \ \neg x_1 \vee -x_2 \end{array}$$

$$\begin{array}{l} \quad x_1 \\ \& \ \neg x_1 \end{array}$$

SAT problem

- SAT is one of the most basic and most studied problems in computer science
- It has many practical applications in VLSI design, network design (and in many other fields where Boolean variables naturally describe the studied problem)
- A procedure which generates resolution closure is enough to solve SAT (but it is exponential).
- How hard it is to solve SAT, i.e. what is the complexity of this problem?

Given (1) $\wedge$ (2) $\wedge$(3) $\wedge$(4) $\wedge$(5) = T, x_i=?

- An instance (a set of clauses):

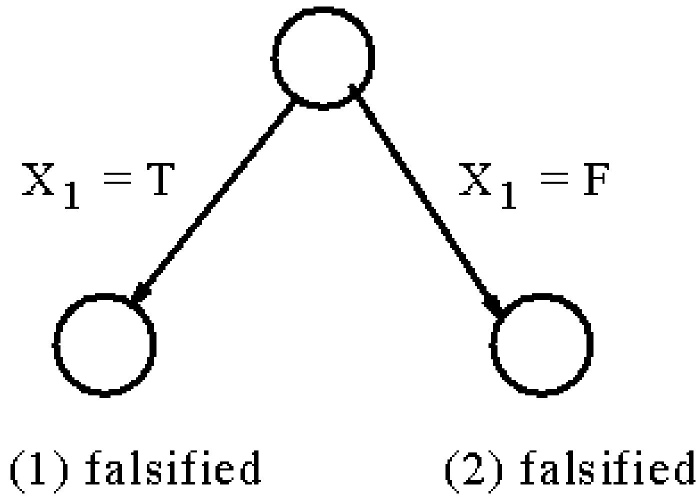
  $\neg x_1$.............. (1)

  x_1.............. (2)

  $x_2 \vee x_5$....... . (3)

  x_3.............. (4)

  $\neg x_2$.............. (5)

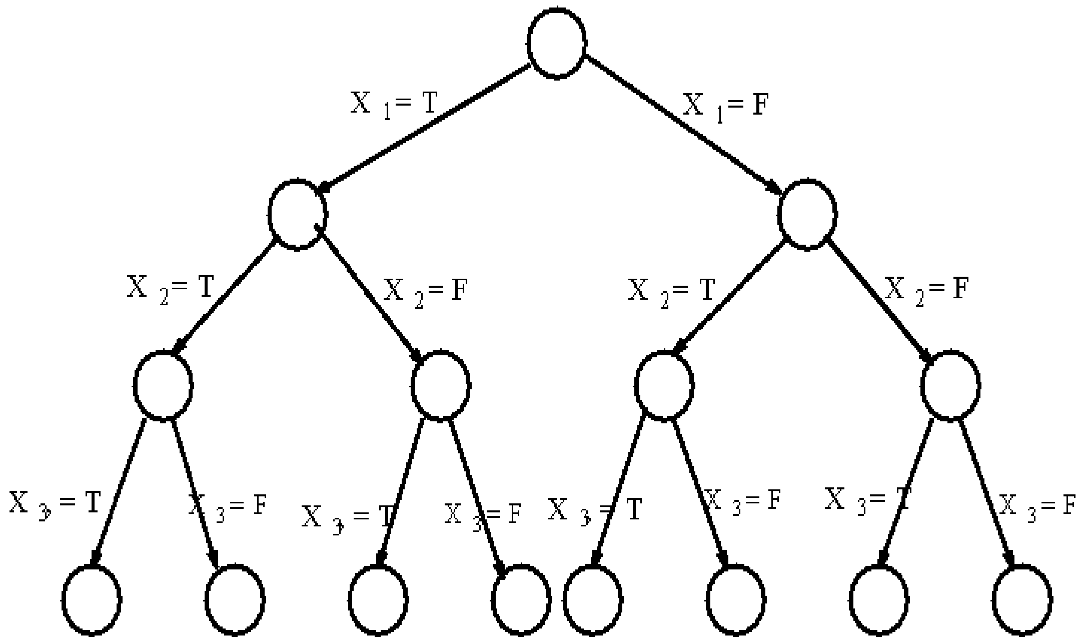


A partial tree to determine the satisfiability problem.

- We may not need to examine all possible assignments.

Satisfiability (SAT) problem

x_1	x_2	x_3
F	F	F
F	F	T
F	T	F
F	T	T
T	F	F
T	F	T
T	T	F
T	T	T

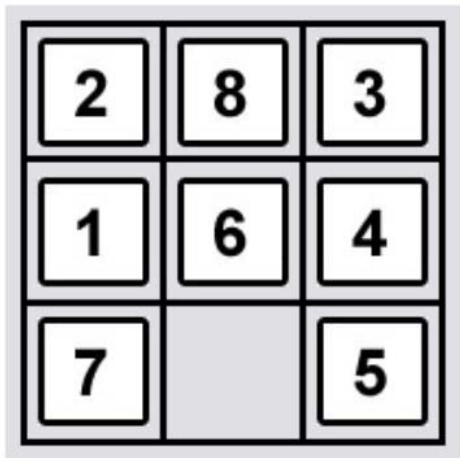


Tree representation of 8 assignments.

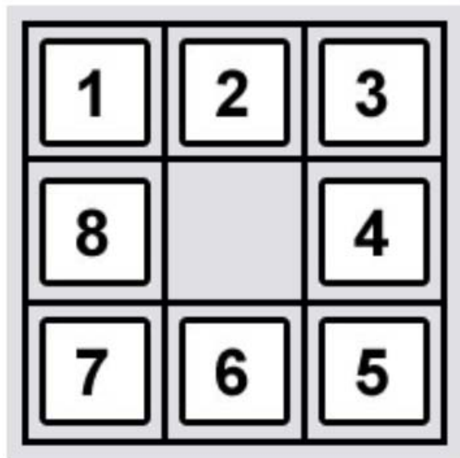
- If there are *n* variables x_1, x_2, …,x_n, then there are 2^n possible assignments.

8-Puzzle Problem

- We show a square frame which can hold 9 items. However, only 8 items exist and therefore there is an empty spot (initial state).
- Our problem is to move these numbered tiles around so that the *final (or goal) state* is reached.
- The numbered tiles can be moved only horizontally or vertically to the empty spot.

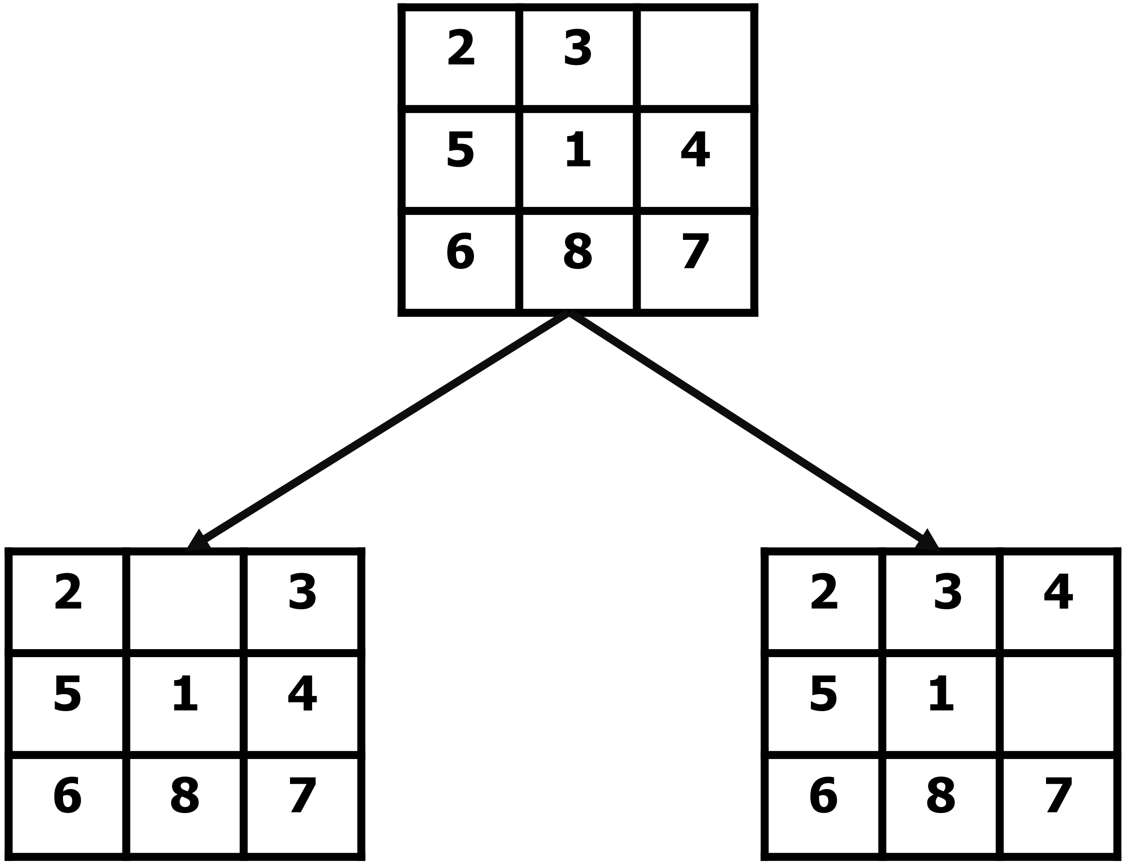


Initial State

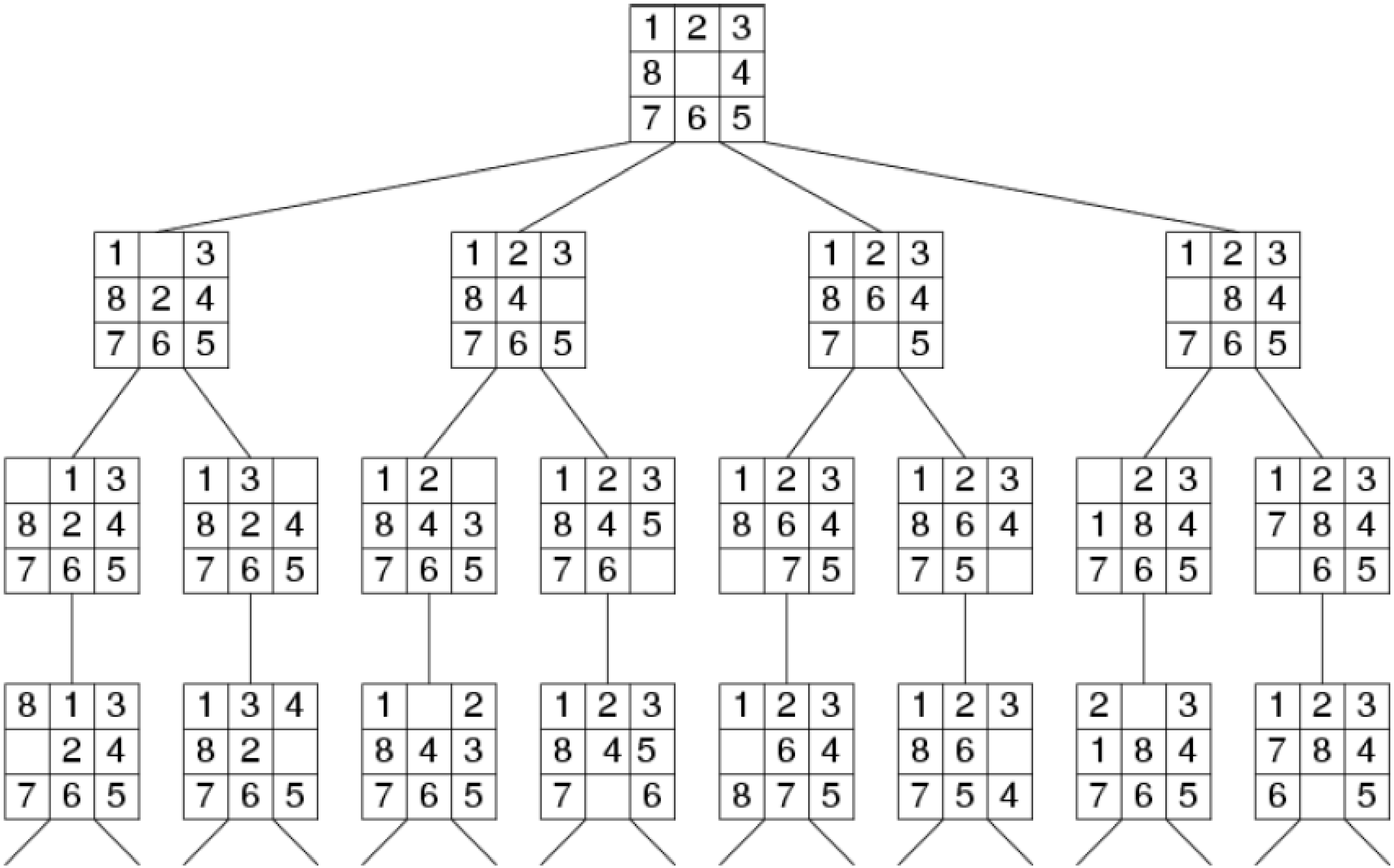


Goal State

Possible Moves

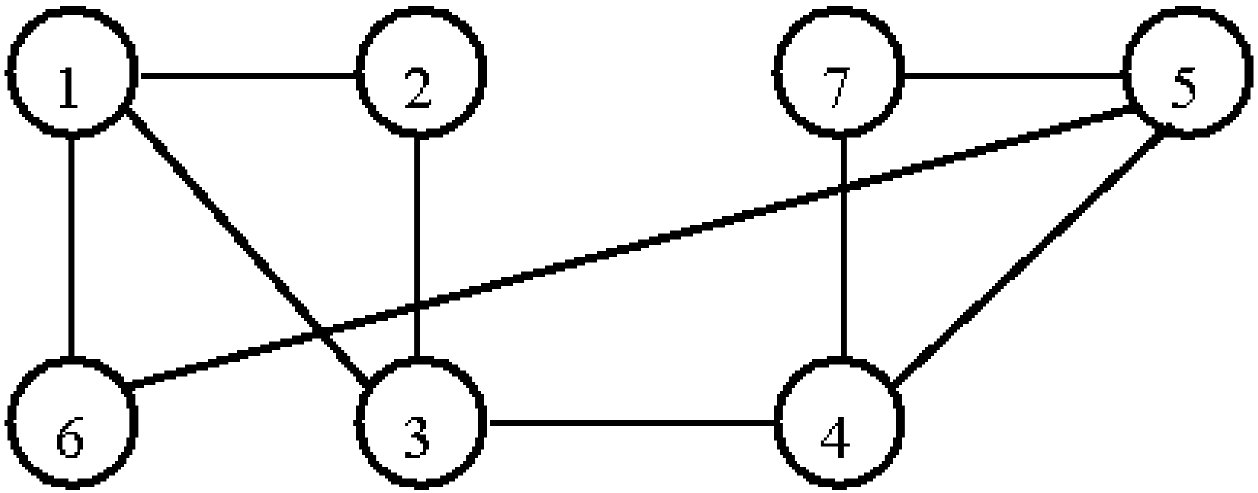


Search Tree for 8-Puzzle



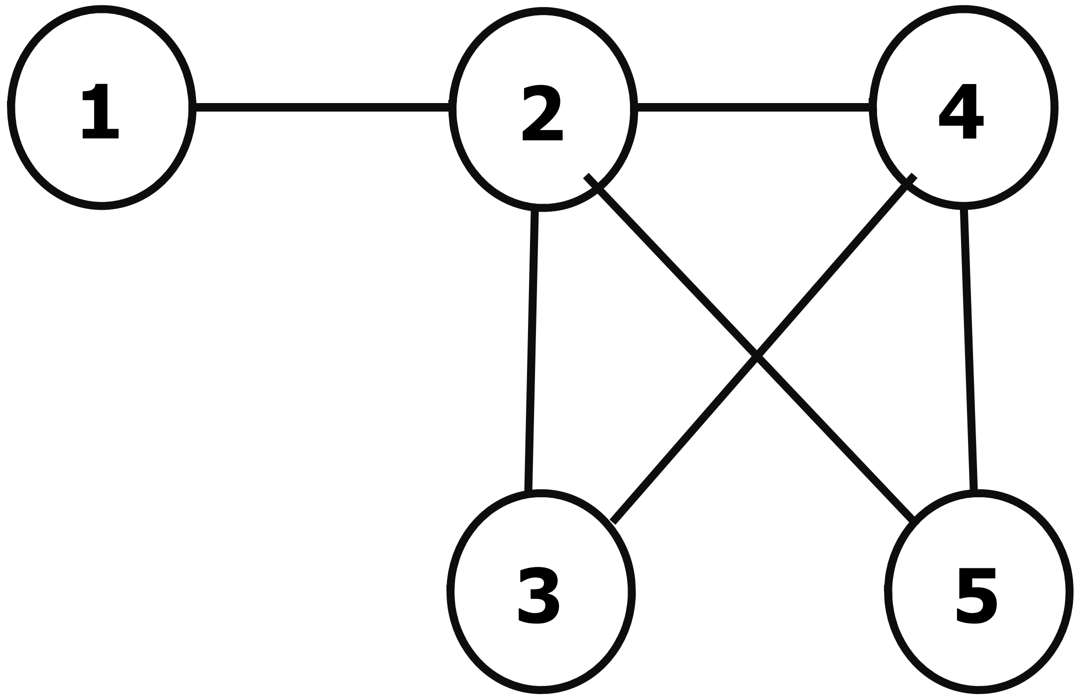
Hamiltonian circuit problem

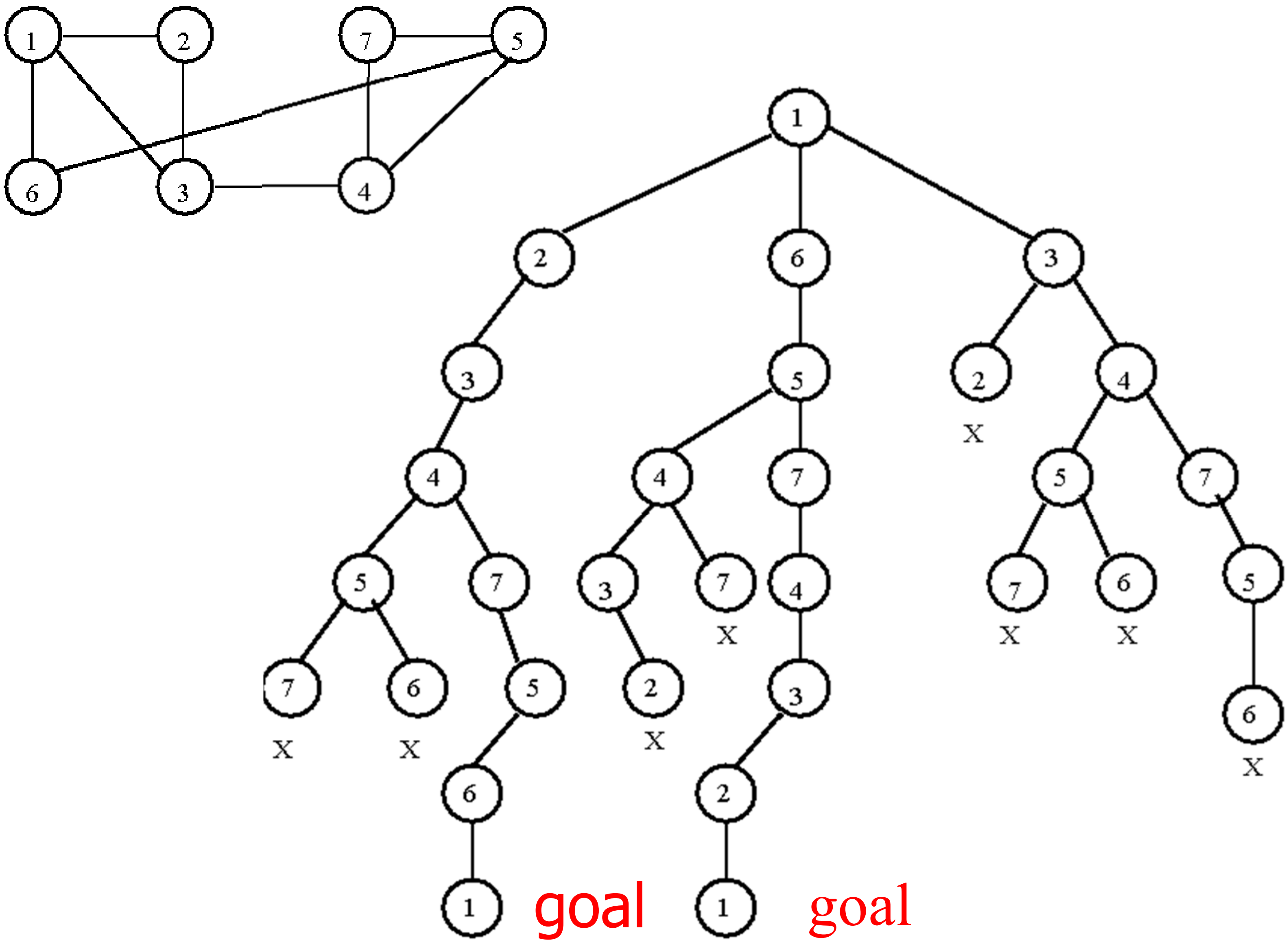
- Given a graph $G=(V,E)$ which is a connected graph with n vertices, a *Hamiltonian circuit* is a round trip path along n edges of G which visits every vertex once and returns to its starting position.



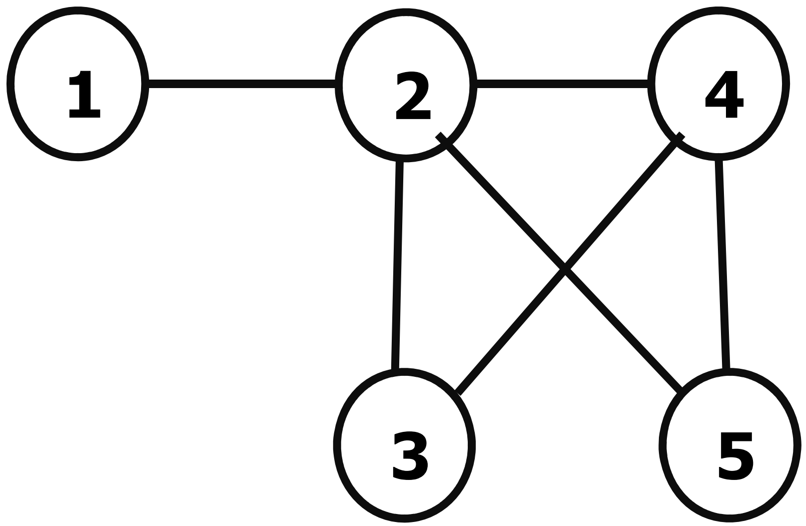
Example

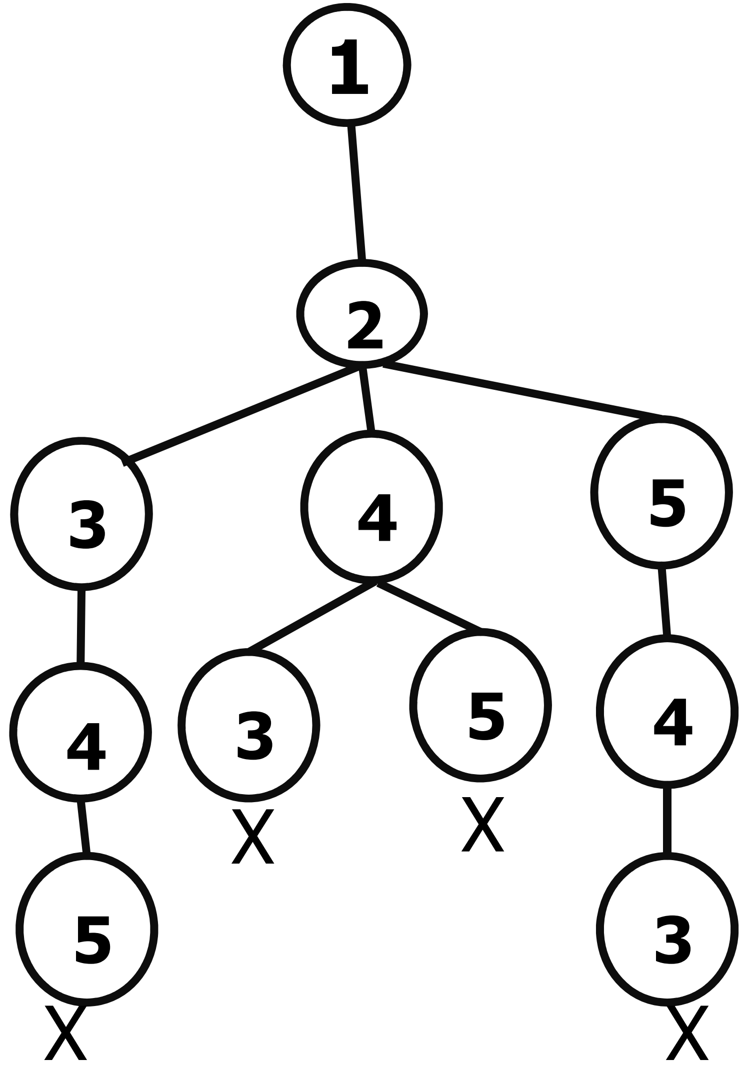
- Find HC





The tree representation of whether there exists a Hamiltonian circuit.


1
2
4
3
5

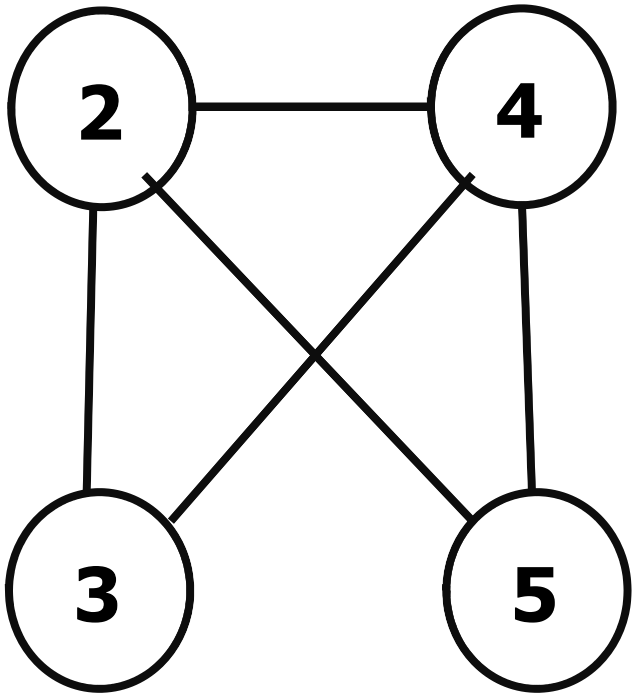

1
2
3
4
5
4
3
5
4
5
X
X
X
3
X

Question:

- Is there any *Hamiltonian circuit* in the given graph?

(1) yes

(2) no

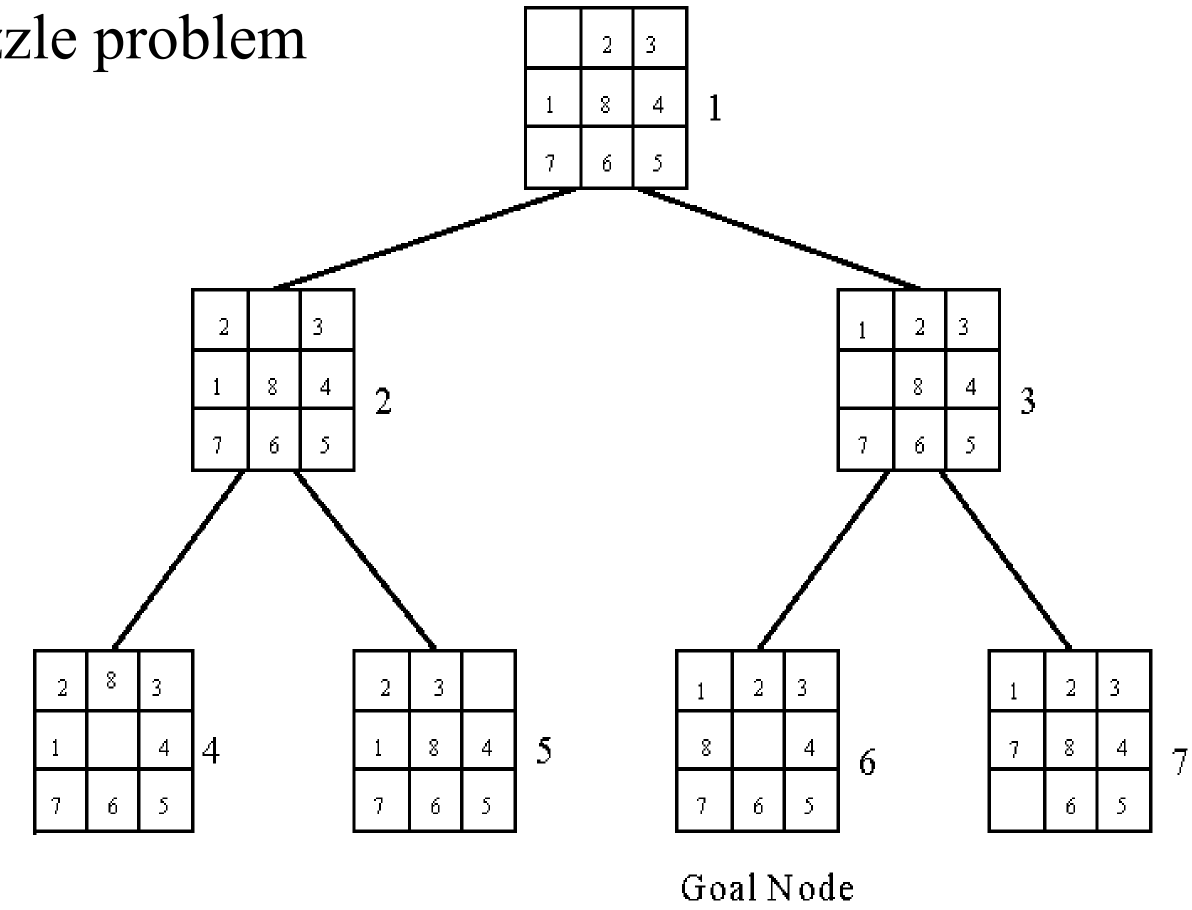


Ans. 1

Breadth-First Search

Breadth-first search (BFS)

- 8-puzzle problem



- The breadth-first search uses a queue to holds all expanded nodes.

Breadth-First Search

- **Step 1.** Form a one-element queue consisting of the root node.
- **Step** 2. Test to see if the first element in the queue is a goal node. If it is, stop. Otherwise, go to Step 3.
- **Step** 3. Remove the first element from the queue. Add the first element's descendants, if any, to the end of the queue.
- **Step** 4. If the queue is empty. then failure. Otherwise, go to Step 2.

Depth-first search (DFS)

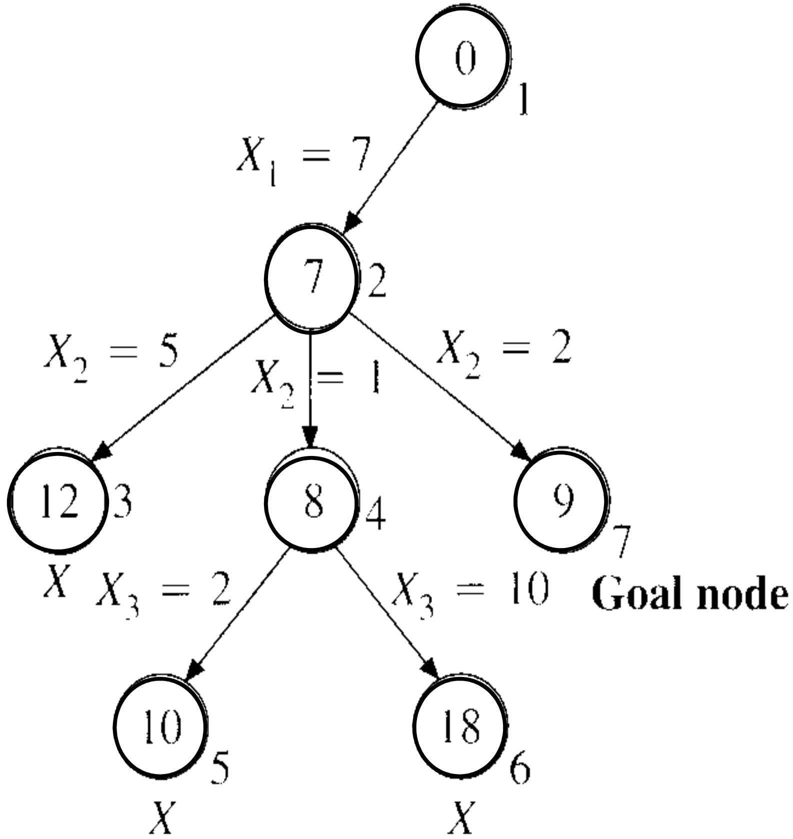
Depth-first search (DFS)

- e.g. **sum of subset problem**

  S={7, 5, 1, 2, 10}

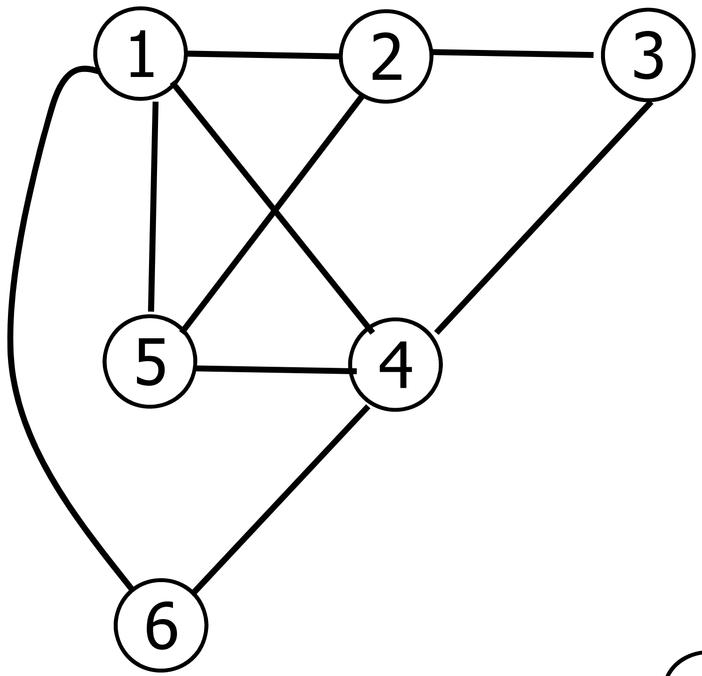
  ∃ S' ⊆ S ∍ sum of S' = 9 ?

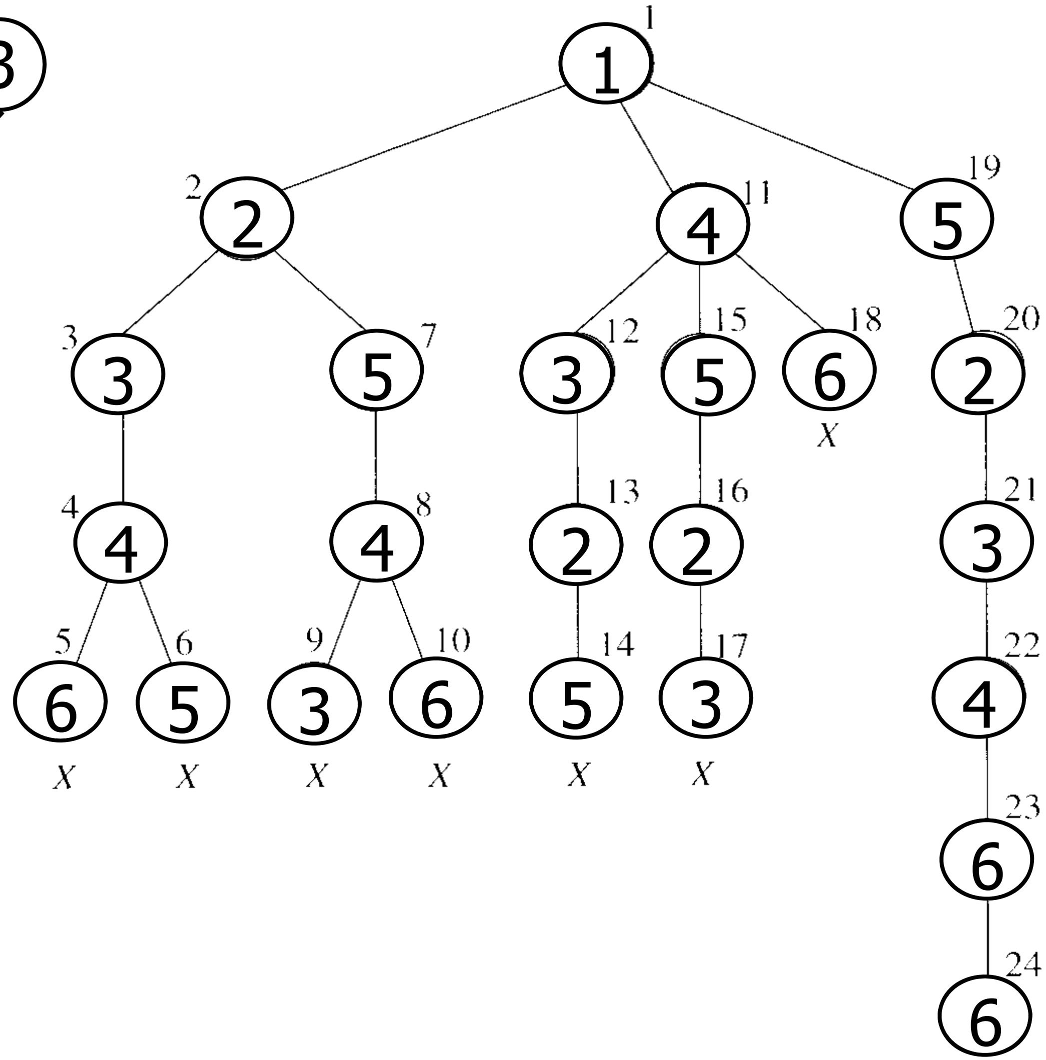
- A stack can be used to guide the depth-first search.



A sum of subset problem solved by depth-first search.

DFS-tree for Hamiltonian cycle





DFS (**Depth-First Search**)

- **Step 1.** Form a one-element stack consisting of the root node.
- **Step** 2. Test to see if the top element in the stack is a goal node. If it is, then stop: otherwise, go to Step 3.
- **Step 3**. Remove the top element from the stack and add the first elements descendants, if any, to the top of the stack.
- **Step** 4. If the stack is empty, then failure. Otherwise, go to Step 2.

Question:

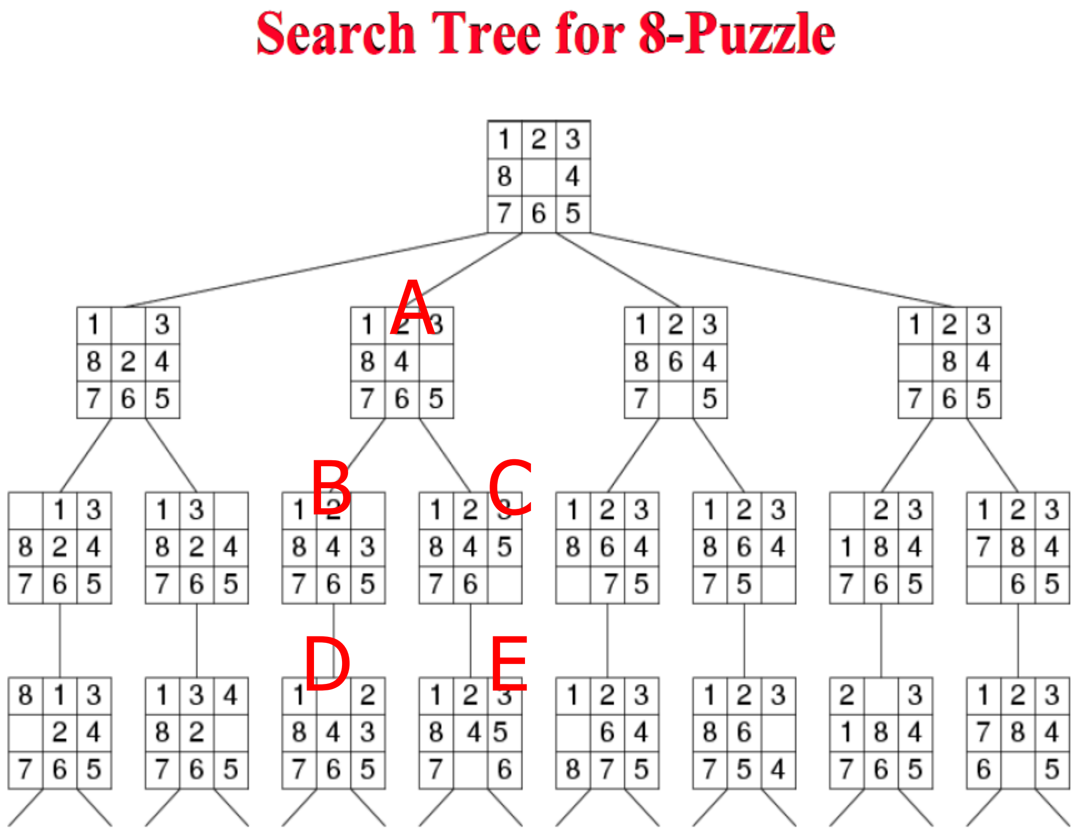
- Which is next state by using the BFS, if you are at the state A?

(1) B,

(2) C,

(3) D,

(4) E



Ans. 1

Hill climbing

Hill climbing

- Among all the descendants, which node should be selected by us to expand?
- Hill climbing is a variant of depth-first search in which some greedy method is used to help us decide which direction to move in the search space.
- Usually, the greedy method uses some heuristics measure to order the choices. And, the better the heuristics, the better the hill climbing is.

Hill Climbing

- A variant of depth-first search

  The method selects the locally optimal node to expand.
- e.g. 8-puzzle problem

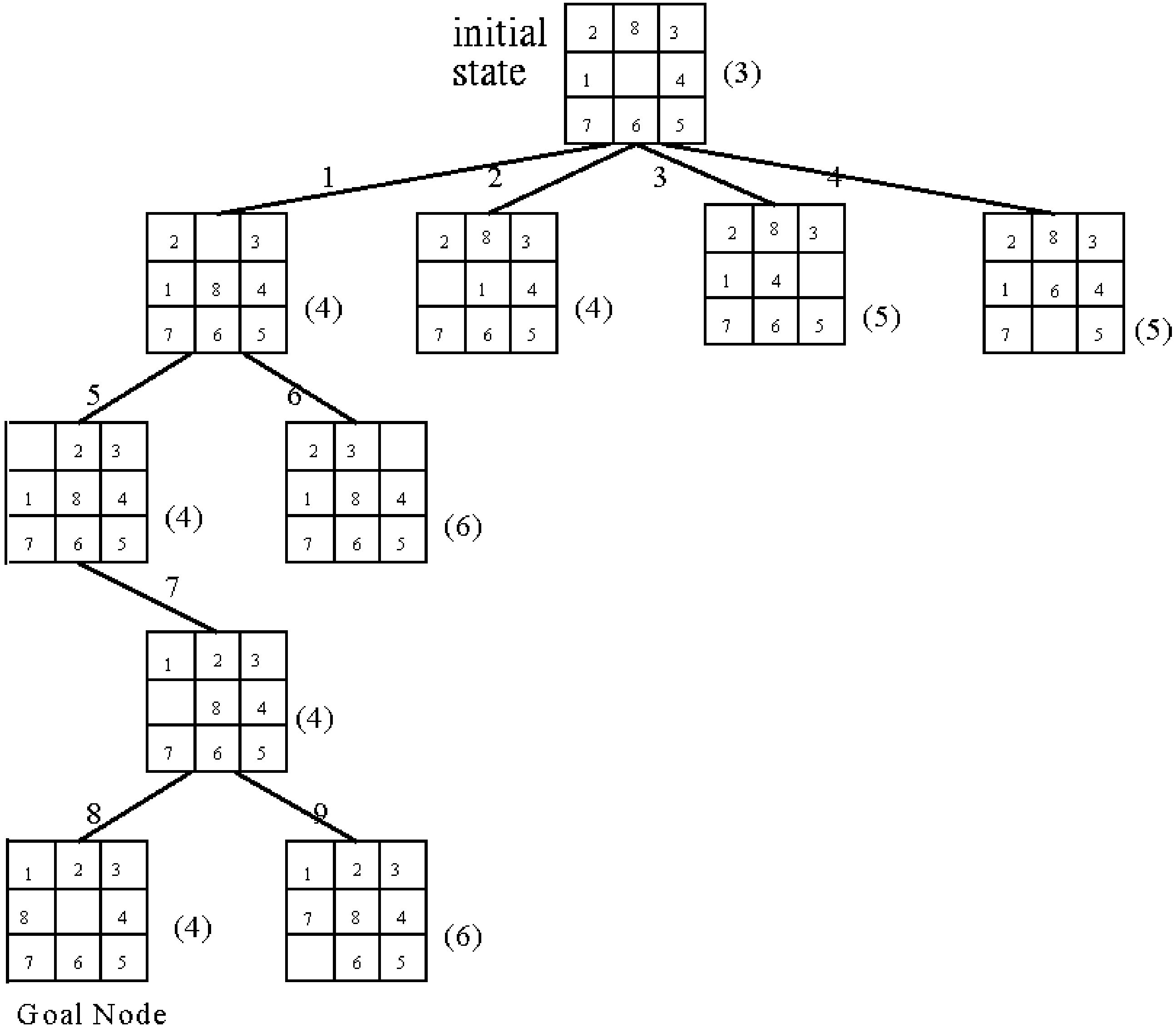
  evaluation function $f(n) = d(n) + w(n)$

  where $d(n)$ is the depth of node n

  $w(n)$ is # of misplaced tiles in node n.

Scheme of Hill Climbing

- **Step 1.** Form a one-element stack consisting of the root node.
- **Step** 2. Test to see if the top element in the stack is a goal node. If it is. Stop: otherwise, go to Step 3.
- **Step 3.** Remove the top element from the stack and expand the element. Add the descendants of the element into the stack ordered by the evaluation function.
- **Step** 4. If the list is empty. failure. Otherwise, go to Step 2.



An 8-puzzle problem solved by a hill climbing method.

Best-first search strategy

Best-first search strategy

- Combing depth-first search and breadth-first search.
- Selecting the node with the best estimated cost among all nodes.
- This method has a global view.

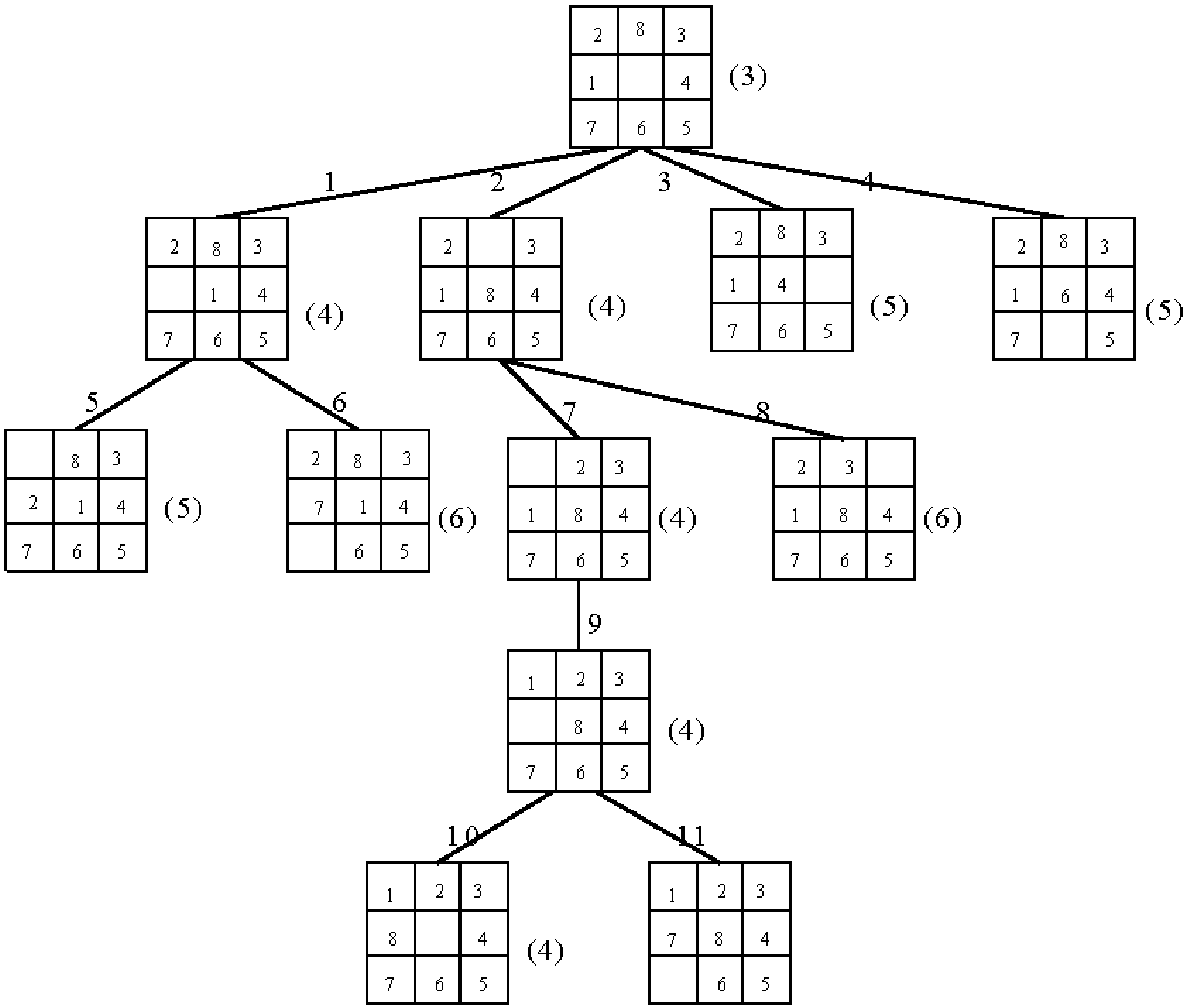
Best-First Search Scheme

Step1: Construct a heap by using the evaluation function. First, form a 1-element heap consisting of the root node.

Step2: Test to see if the root element in the heap is a goal node. If it is, stop; otherwise, go to Step 3.

Step3: Remove the root element from the heap and expand the element. Add the descendants of the element into the heap.

Step4: If the heap is empty, then failure. Otherwise, go to Step 2.



An 8-puzzle problem solved by a best-first search scheme.

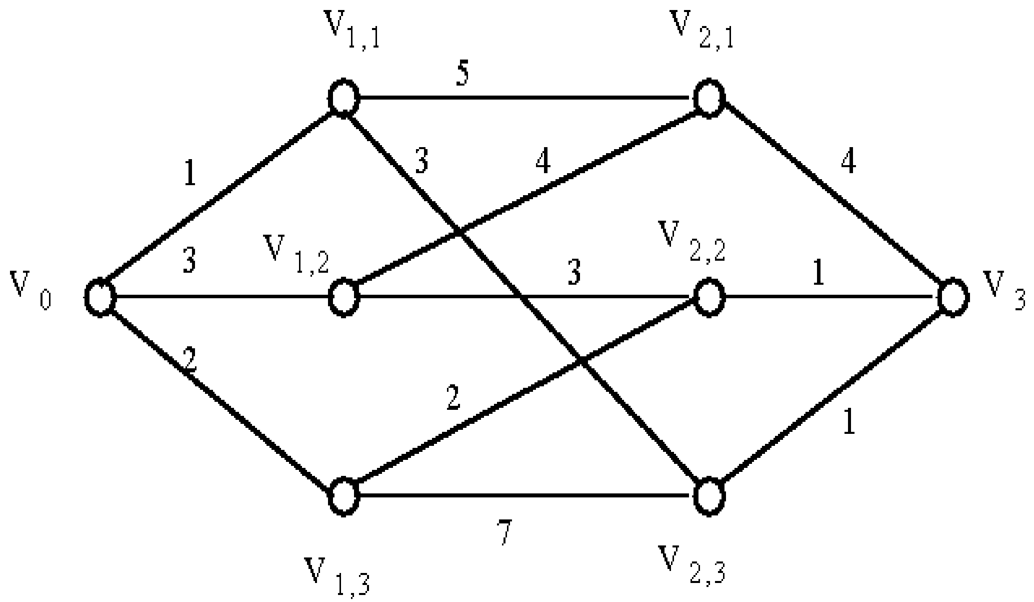
Branch-and-bound strategy

學習目標

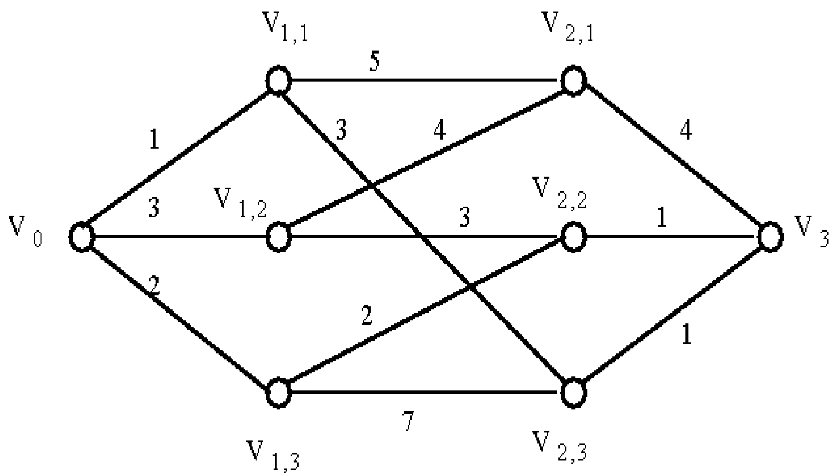
- **Branch-and-bound strategy**設計的概念
- **Branch-and-bound strategy**設計的應用範例

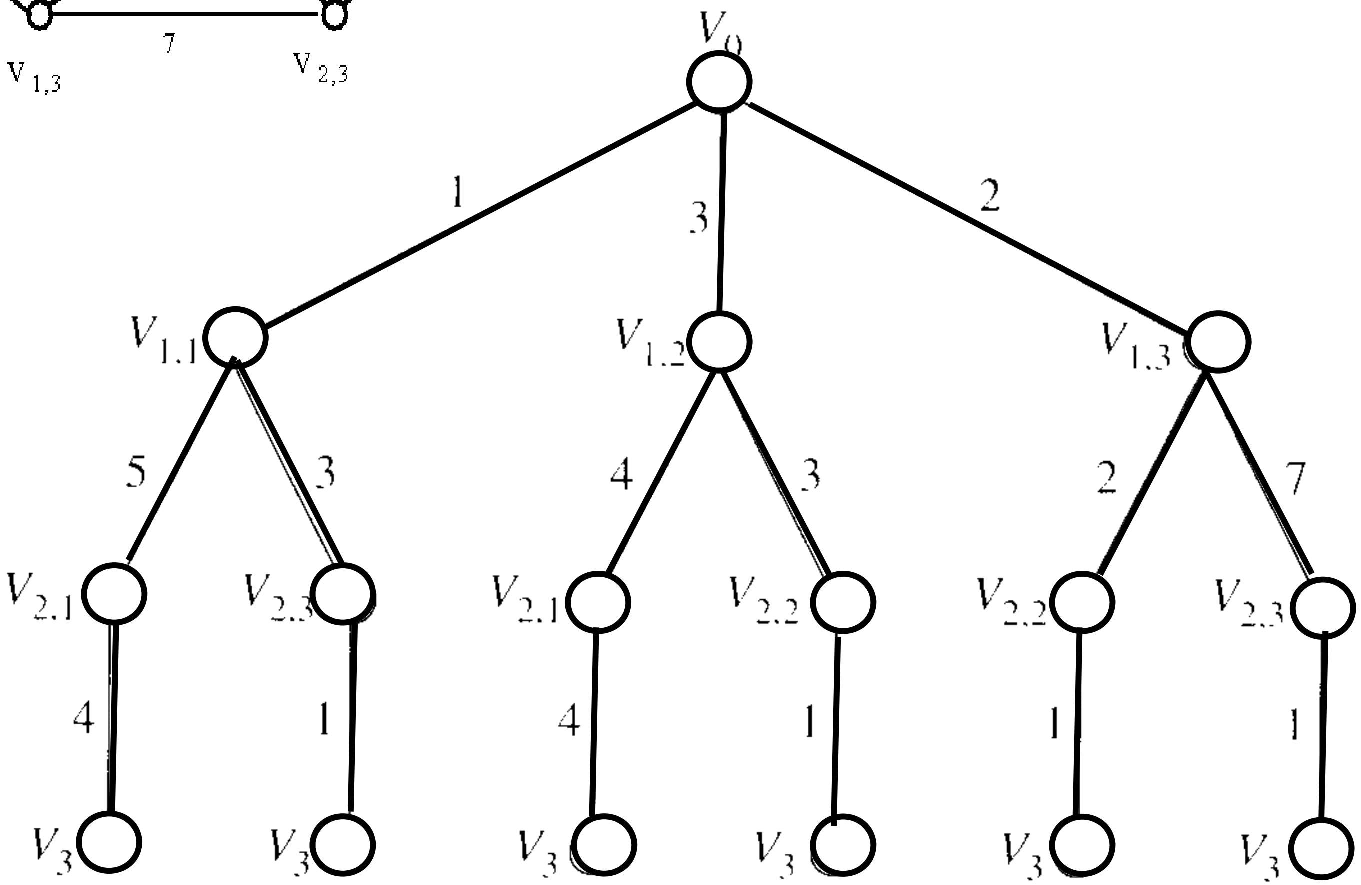
Branch-and-bound strategy

- This strategy can be used to efficiently solve optimization problems.
- One of the most efficient strategies to solve a large combinatorial problem.
- Basically, it suggests that a problem may have feasible solutions. However, one should try to cut down the solution space by finding out that many feasible solutions can not be optimal solutions.

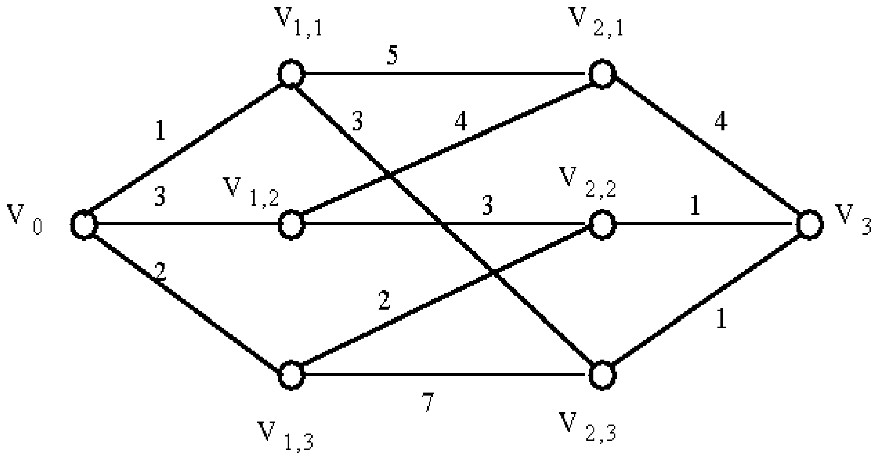


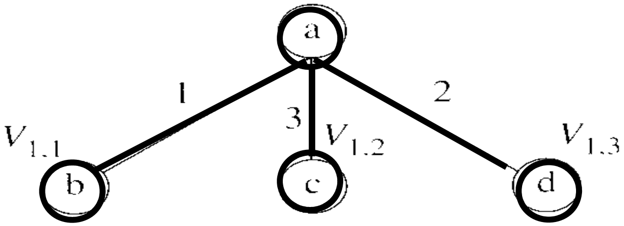
Example
-exhaustive search



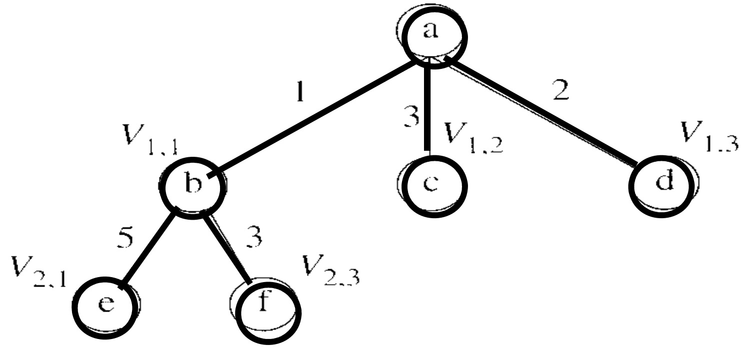


Hill climbing

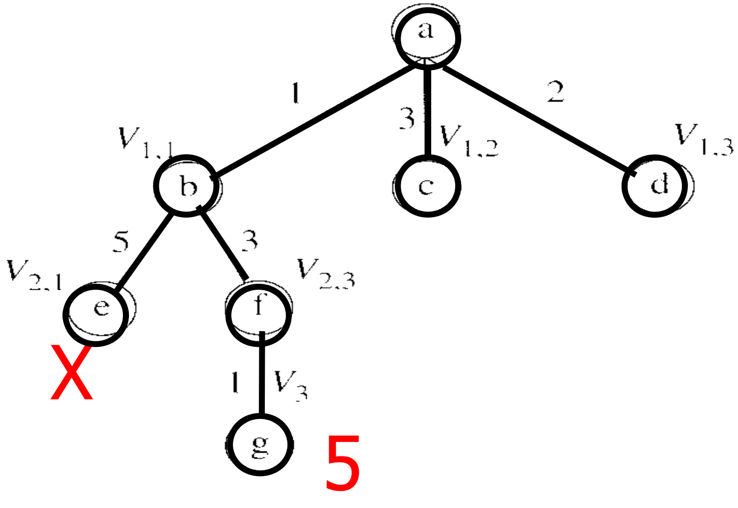




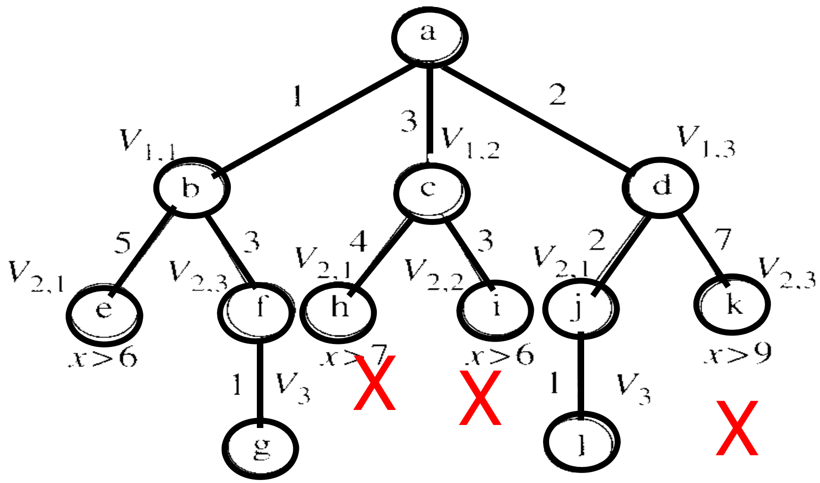
(a)



(b)

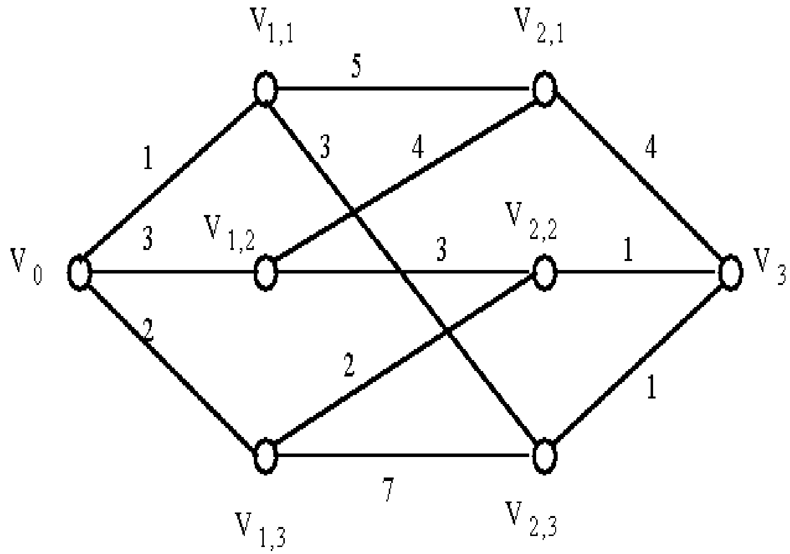


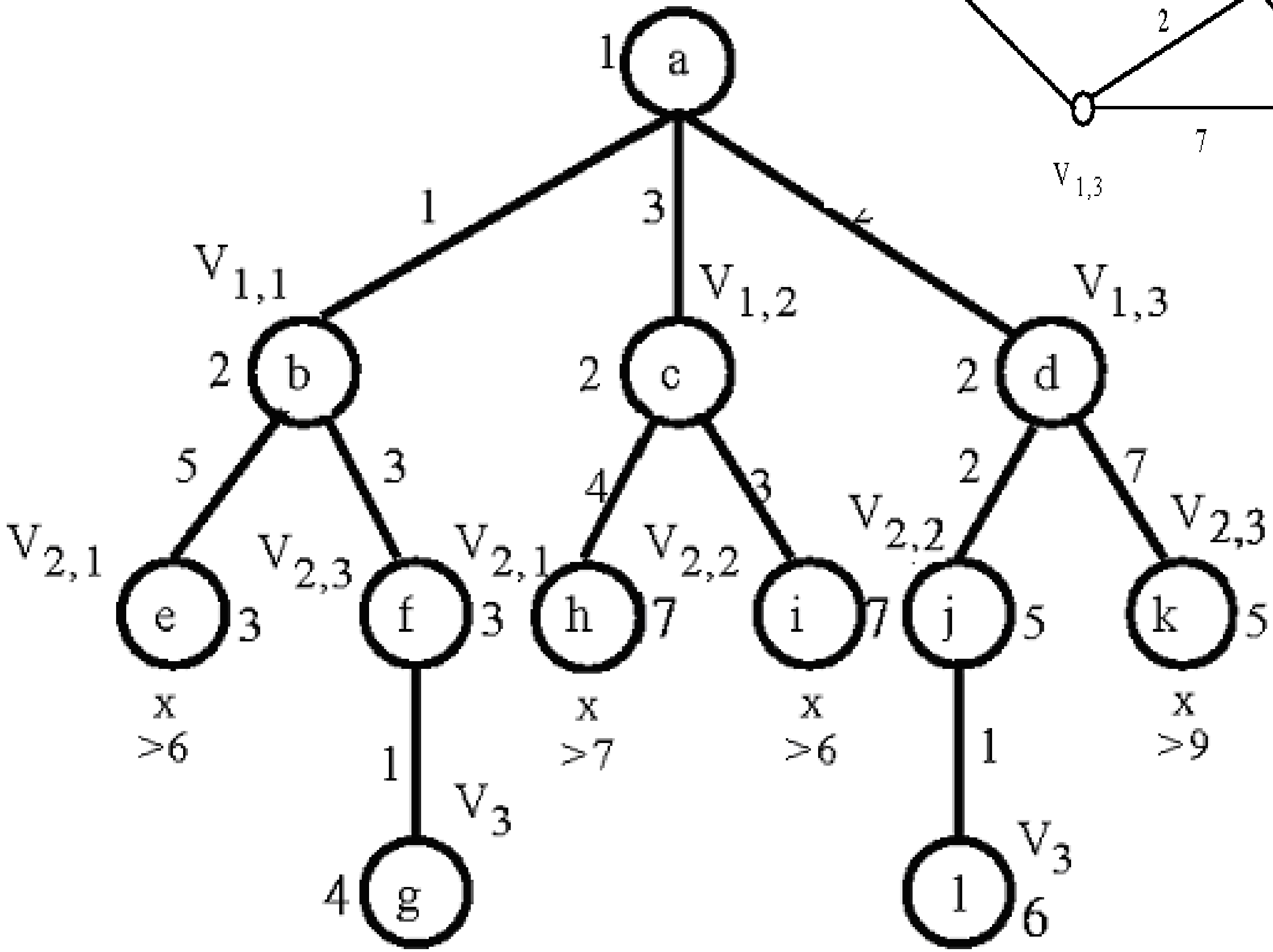
(c)



(d)

- Solved by branch-and-bound





Branch & Bound

- This strategy consists of two important mechanisms:
  - A mechanism to generate branchings and
  - a mechanism to generate a bound so that many branchings can be terminated.
- Although the branch-and-bound strategy is usually very efficient. in worst cases, a very large tree may still be generated.
- Thus, we must realize that the branch-and-bound strategy is efficient in the sense of average cases.

Personnel assignment problem

學習目標

- **Personnel assignment problem 問題定義**
- **Topological sorting 定義**
- **Branch-and-bound strategy 演算法設計**

Personnel assignment problem

- A linearly ordered set of persons $P=\{P_1, P_2, \ldots, P_n\}$ where $P_1<P_2<\ldots<P_n$
- A partially ordered set of jobs $J=\{J_1, J_2, \ldots, J_n\}$
- Suppose that P_i and P_j are assigned to jobs $f(P_i)$ and $f(P_j)$ respectively.
- **Constraint**:
  - If $f(P_i) \le f(P_j)$, then $P_i \le P_j$. If $P_i \ne P_j$, then $f(P_i) \ne f(P_j)$.

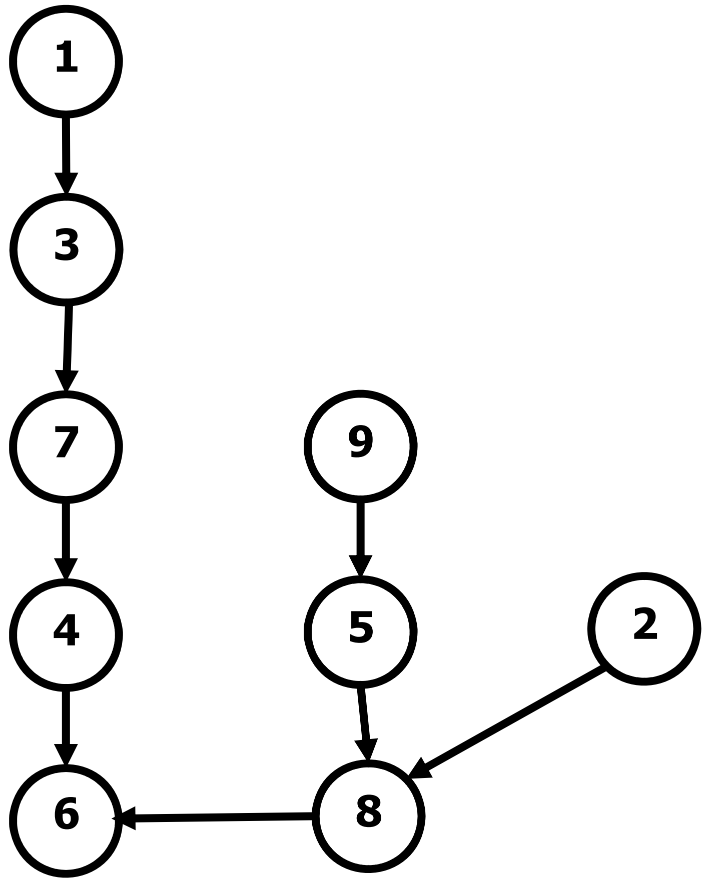
- Cost C_{ij} is the cost of assigning P_i to J_j.
- We want to find a feasible assignment with the minimum cost. i.e.

  $X_{ij} = 1$ if P_i is assigned to J_j

  $X_{ij} = 0$ otherwise.
- **Goal: Minimize** $\sum_{\forall i,j} C_{ij} X_{ij}$

Topological sorting (拓樸排序)

- For a *n* partial ordering set S, a linear sequence S_1, S_2, … , S_n, is topologically sorted respect to *S* if $S_i < S_j$ in the partial ordering implies that S_i is located before S_j in the sequence.



One possible topologically sorted sequence is 1, 3, 7, 4, 9, 2, 5, 8, 6.

A feasible assignment

- Let $P_1 \rightarrow J_{k1}$, $P_2 \rightarrow J_{k2}$, …, $P_n \rightarrow J_{kn}$, be a feasible assignment.
- According to our problem definition, the jobs are partially ordered and persons are linearly ordered.
- Therefore, J_{k1} , J_{k2} , …, J_{kn} must be a topologically sorted sequence with respect to the partial ordering of jobs.
- Let us illustrate our idea by an example. Consider $J = \{J_1, J_2, J_3, J_4\}$ and $P = \{P_1, P_2, P_3, P_4\}$.

- e.g. A partial ordering of jobs

$$
\begin{tikzcd}
J_1 \arrow[d] \arrow[dr] & J_2 \arrow[d] \\
J_3 & J_4
\end{tikzcd}
$$

- After topological sorting, one of the following topologically sorted sequences will be generated:

J_1,	J_2,	J_3,	J_4
J_1,	J_2,	J_4,	J_3
J_1,	J_3,	J_2,	J_4
J_2,	J_1,	J_3,	J_4
J_2,	J_1,	J_4	J_3

- One of feasible assignments:

  $P_1 \to J_1$, $P_2 \to J_2$, $P_3 \to J_3$, $P_4 \to J_4$

Question:

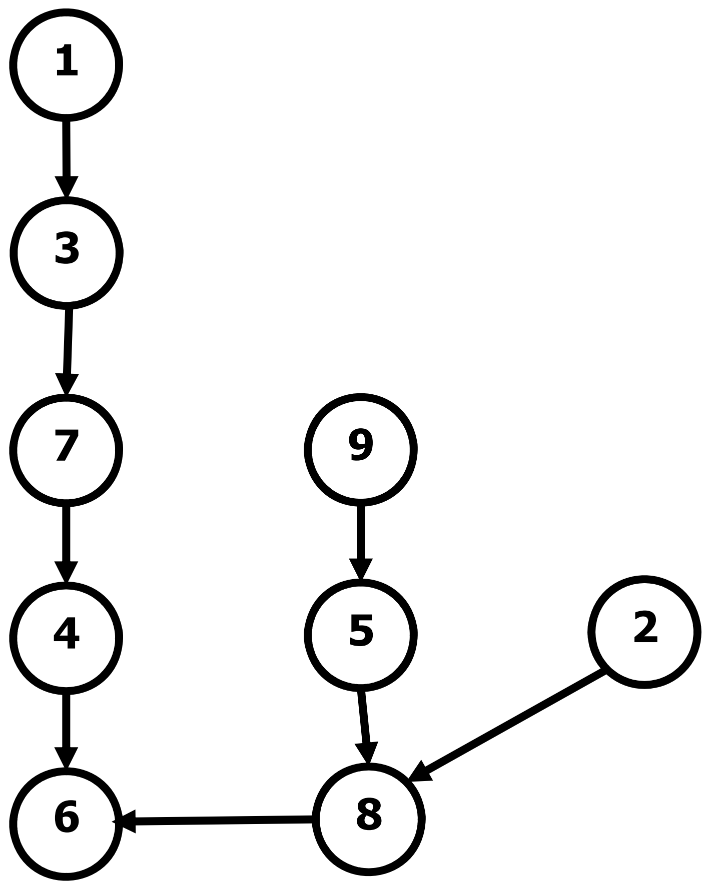
- Which one is a feasible topological sorting sequence of the given graph?

(1) 1-3-5-9-7-8-2-4-6

(2) 1-3-9-5-7-8-2-4-6

(3) 1-3-9-5-7-2-8-4-6

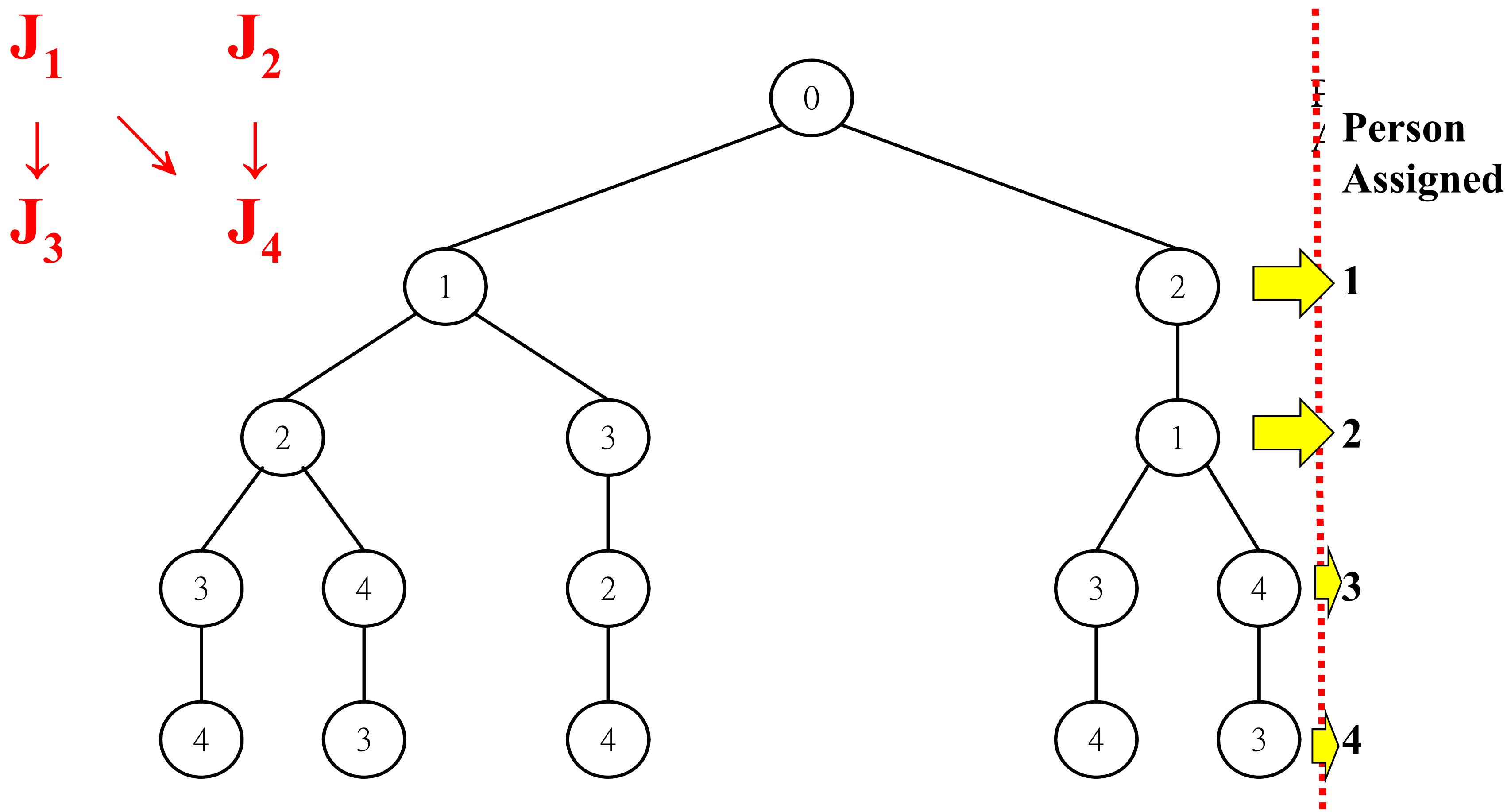
(4) 1-3-9-5-7-2-4-6-8.



Ans. 3

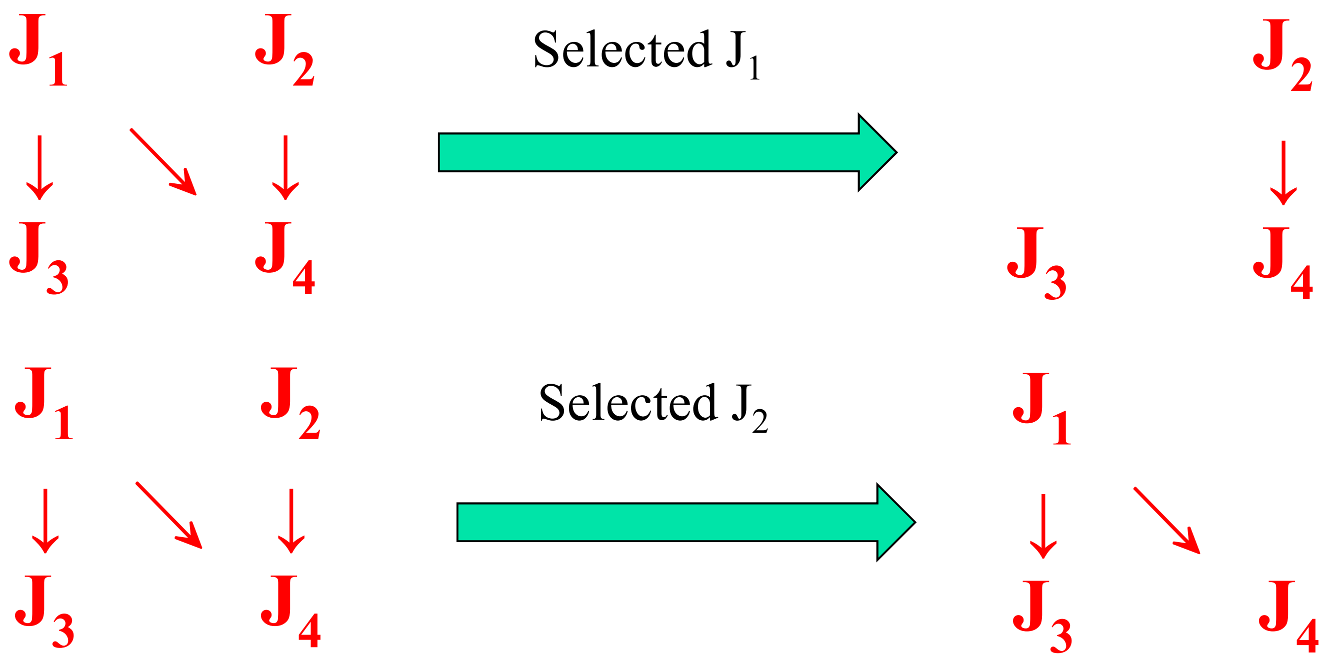
A solution tree

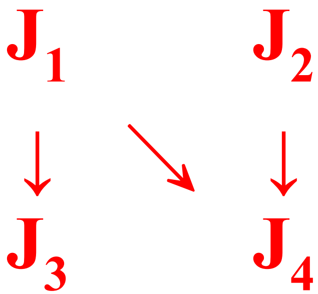
- All possible solutions can be represented by a solution tree.



Tree generated Steps (topology sorted order)

- Take an element which is not preceded by any other element in the partial ordering.
- Select this element as an element in a topologically sorted sequence.
- Remove this element just selected from the partial ordering set. The resulting set is still partially ordered.



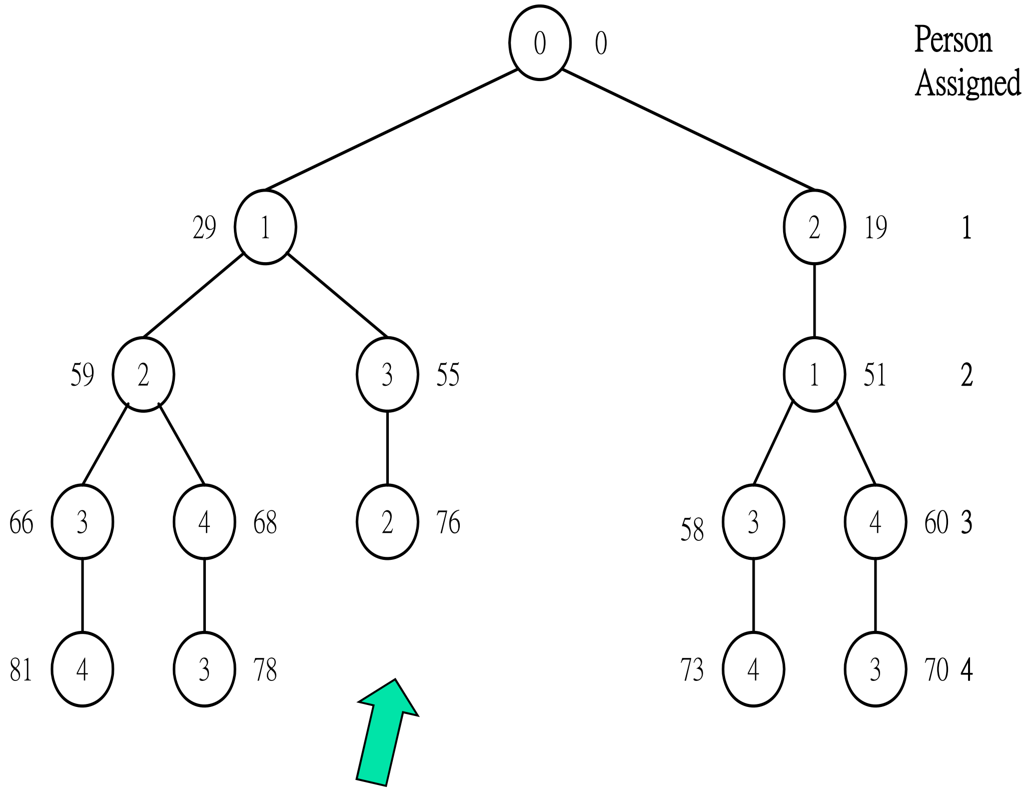


Cost matrix

- Cost matrix

Jobs / Persons	1	2	3	4
1	29	19	17	12
2	32	30	26	28
3	3	21	7	9
4	18	13	10	15

- **Apply the best-first search scheme:**



Only one node is pruned away.

Reduced cost matrix to find lower bound (LB)

- Cost matrix

Jobs / Persons	1	2	3	4
1	29	19	17	12
2	32	30	26	28
3	3	21	7	9
4	18	13	10	15

- Reduced cost matrix

Jobs / Persons	1	2	3	4	
1	17	4	5	0	(-12)
2	6	1	0	2	(-26)
3	0	15	4	6	(-3)
4	8	0	0	5	(-10)
		(-3)			

- Lower bound: least cost we need to find the solution.
- No solution can have a cost lower than LB.
- A higher LB will lead to an earlier termination.

Reduced cost matrix to find LB

- A reduced cost matrix can be obtained:

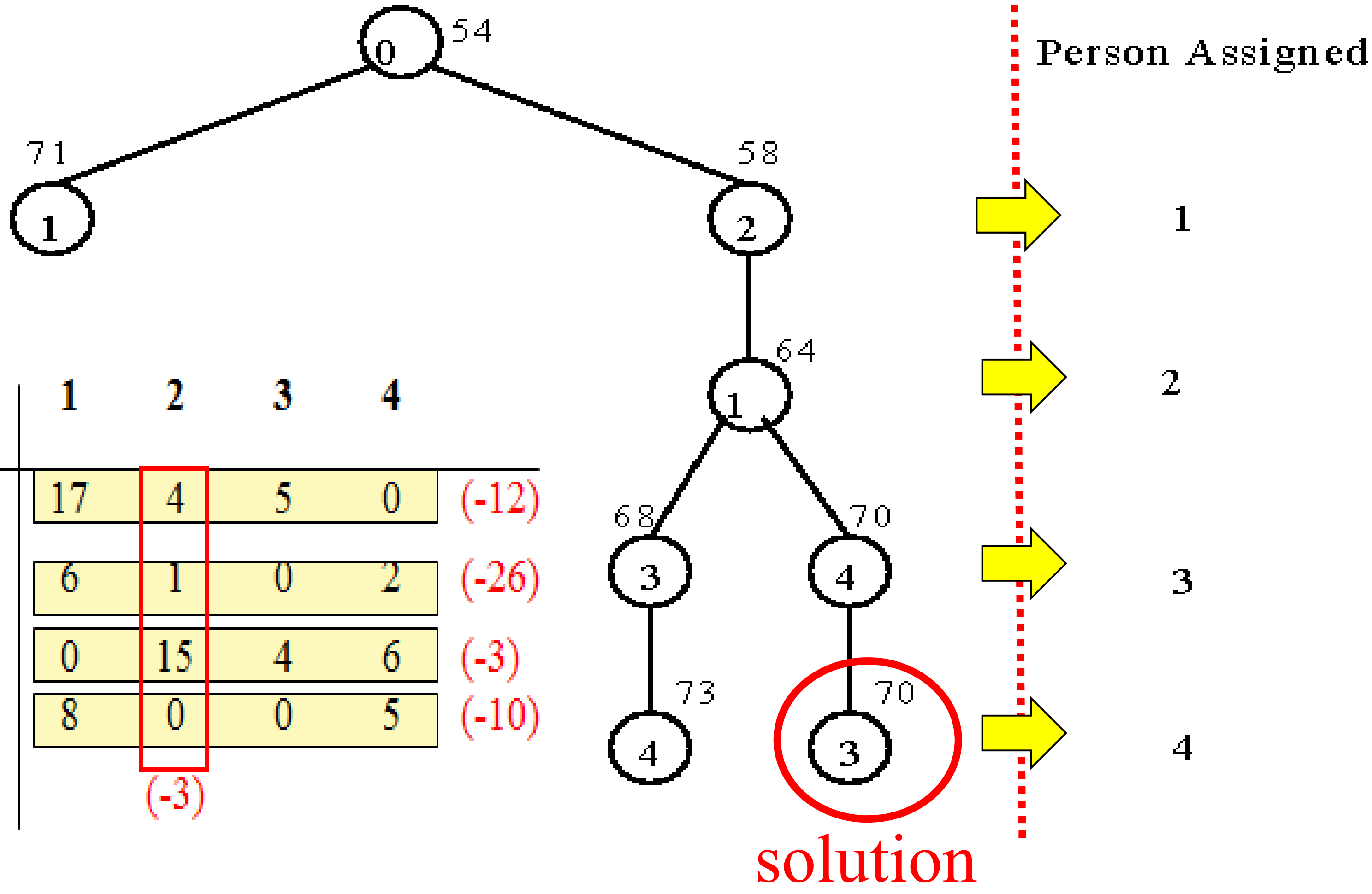
  subtract a constant from each row and each column respectively such that each row and each column contains at least one zero.
- Total cost subtracted: 12+26+3+10+3 = 54
- This is a lower bound of our solution.

Persons \ Jobs	1	2	3	4	
1	17	4	5	0	(-12)
2	6	1	0	2	(-26)
3	0	15	4	6	(-3)
4	8	0	0	5	(-10)
		(-3)			

Branch-and-bound for the personnel assignment problem

- Bounding of sub-solutions:

Jobs / Persons	1	2	3	4	
1	17	4	5	0	(-12)
2	6	1	0	2	(-26)
3	0	15	4	6	(-3)
4	8	0	0	5	(-10)
		(-3)			



Question:

- What is the lower bound of the cost matrix for the personnel assignment problem?

(1) 51

(2) 54

(3) 57

(4) 41.

Jobs / Persons	1	2	3	4
1	29	19	17	12
2	32	30	26	28
3	3	21	7	9
4	18	13	10	15

Ans. 2

The traveling salesperson problem (TSP)

學習目標

- **Traveling Salesperson Problem (TSP)問題定義**
- **Branch-and-bound strategy 演算法設計**

The traveling salesperson problem

- The basic principle of using the branch-and-bound strategy to solve the traveling salesperson optimization problem (TSP) consists of two parts.
  - There is a way to **split the solution space**.
  - There is a way to predict a lower bound for a class of solutions.
  - There is also a way to find an upper bound of an optimal solution.
  - If the lower bound of a solution exceeds this upper bound, this solution cannot be optimal.
  - Thus, we should terminate the branching associated with this solution.

The traveling salesperson problem

- It is **NP-complete**.
- A cost matrix (non-symmetric)

i \ j	1	2	3	4	5	6	7
1	∞	3	93	13	33	9	57
2	4	∞	77	42	21	16	34
3	45	17	∞	36	16	28	25
4	39	90	80	∞	56	7	91
5	28	46	88	33	∞	25	57
6	3	88	18	46	92	∞	7
7	44	26	33	27	84	39	∞

B&B for TSP

- Our branch-and-bound strategy splits a solution into two groups:
  - one group including a particular arc and
  - the other excluding this arc.

- Each splitting **incurs a lower bound** and we shall traverse the searching tree with the "lower" lower bound.
- If a constant subtracted from any row or any column of the cost matrix, an optimal solution does not change.

LB by using reduced cost matrix

- A reduced cost matrix

i \ j	1	2	3	4	5	6	7	
1	∞	0	90	10	30	6	54	(-3)
2	0	∞	73	38	17	12	30	(-4)
3	29	1	∞	20	0	12	9	(-16)
4	32	83	73	∞	49	0	84	(-7)
5	3	21	63	8	∞	0	32	(-25)
6	0	85	15	43	89	∞	4	(-3)
7	18	0	7	1	58	13	∞	(-26)

Reduced: 84

- Another reduced matrix

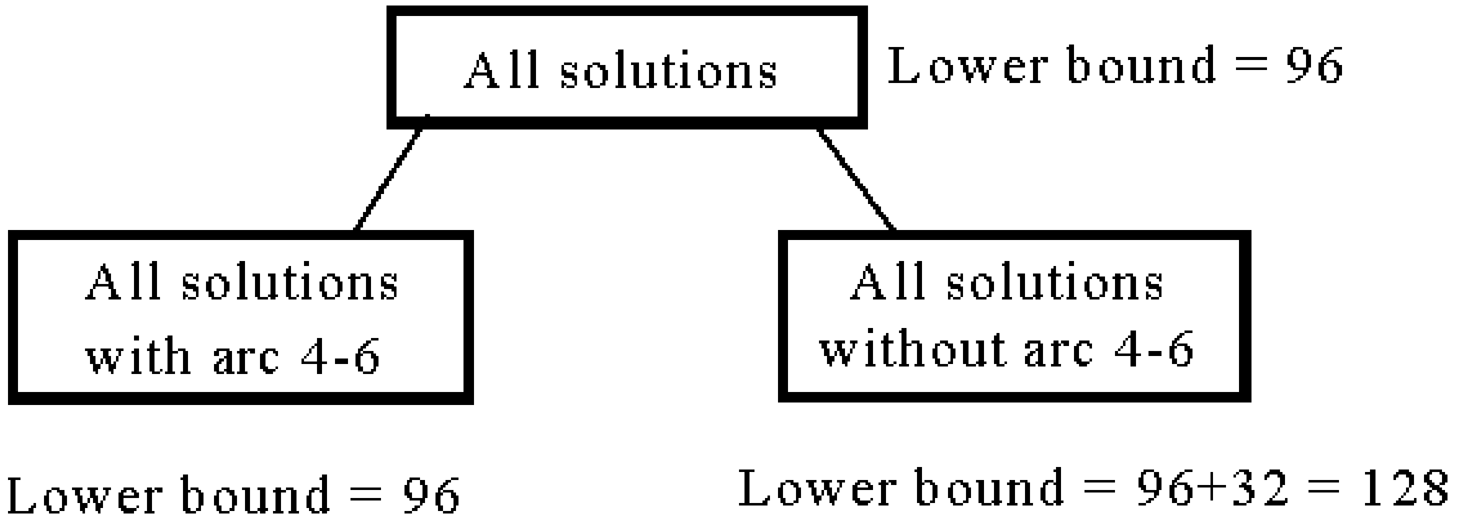
i \ j	1	2	3	4	5	6	7	
1	∞	0	83	9	30	6	50	6
2	0	∞	66	37	17	12	26	12
3	29	1	∞	19	0	12	5	1
4	32	83	66	∞	49	0	80	32
5	3	21	56	7	∞	0	28	3
6	0	85	8	42	89	∞	0	0
7	18	0	0	0	58	13	∞	0
			(-7)	(-1)			(-4)	

Minimal cost not use 4-6

- Total cost reduced: 84+7+1+4 = 96 (lower bound)
- **Arc 4-6 will cause the largest increase of lower bound.**
- **The larger LB the searching will terminate easier**

TREE for TSP

- The highest level of a decision tree:



- Why use 4-6?
- LB for include 4-6? LB for exclude 4-6?
- If we use arc 3-5 to split, the difference on the lower bounds is 17+1 = 18.

- A reduced cost matrix if arc (4,6) is included in the solution.

i \ j	1	2	3	4	5	7
1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5
5	3	21	56	7	∞	28
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

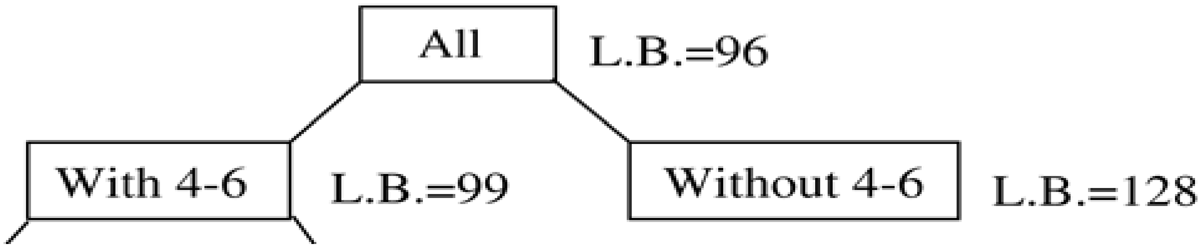
No zero can be reduced

- Arc (6,4) is changed to be infinity since it can not be included in the solution and set to ∞.

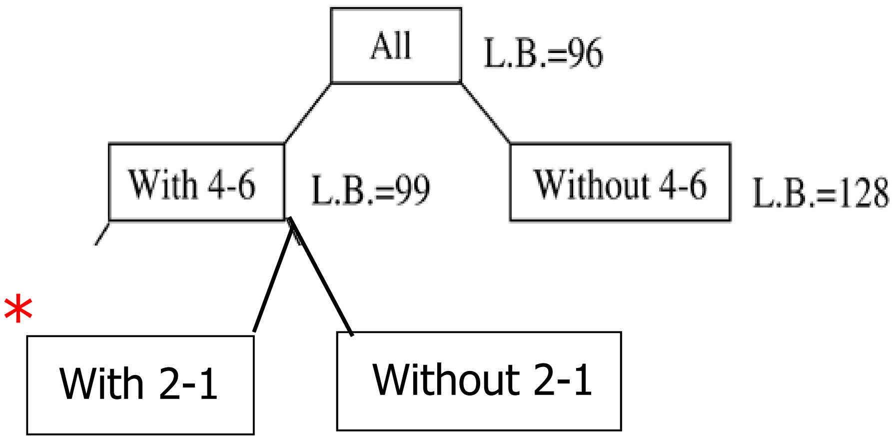
- The reduced cost matrix for all solutions with arc 4-6

i \ j	1	2	3	4	5	7	
1	∞	0	83	9	30	50	
2	0	∞	66	37	17	26	
3	29	1	∞	19	0	5	
5	0	18	53	4	∞	25	(-3)
6	0	85	8	∞	89	0	
7	18	0	0	0	58	∞	

- Total cost reduced: 96+3 = 99 (**new lower bound**)



i \ j	1	2	3	4	5	7	
1	∞	0	83	9	30	50	9
2	0	∞	66	37	17	26	17*
3	29	1	∞	19	0	5	1
5	0	18	53	4	∞	25	4
6	0	85	8	∞	89	0	0
7	18	0	0	0	58	∞	0

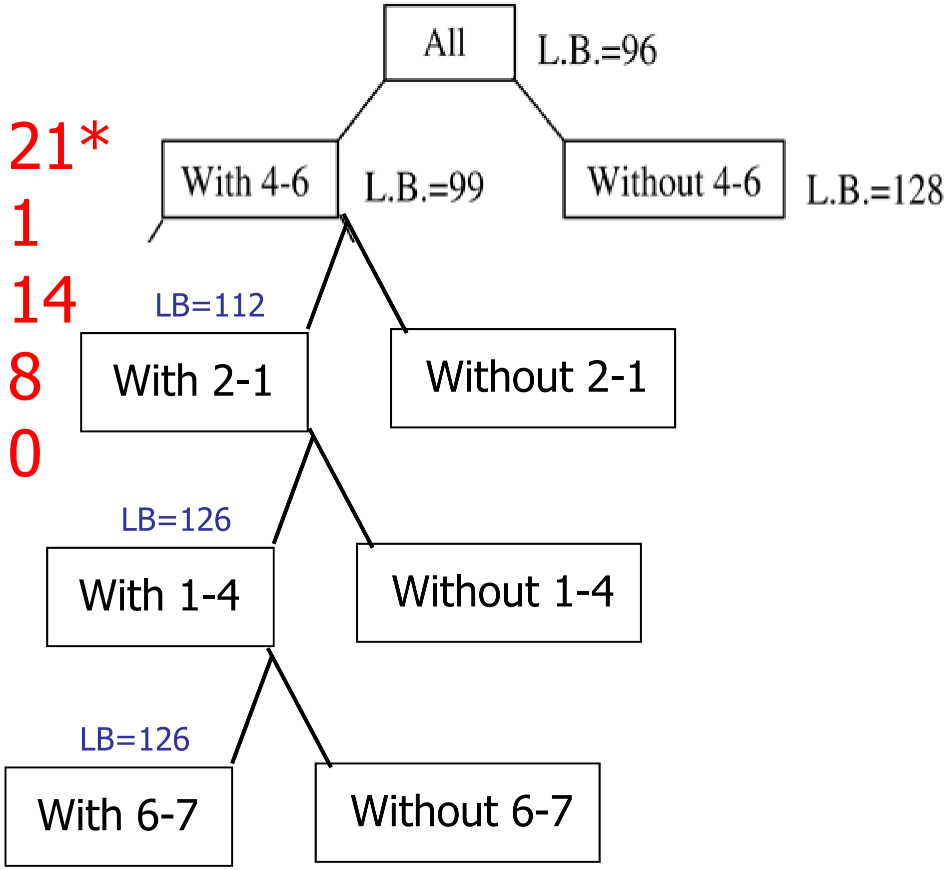


i \ j	2	3	4	5	7	
1	∞	83	9	30	50	-9
3	1	∞	19	0	5	
5	18	53	4	∞	25	-4
6	85	8	∞	89	0	
7	0	0	0	58	∞	

i\i	**2**	**3**	**4**	**5**	**7**
1	∞	74	0	21	41
3	1	∞	19	0	5
5	14	49	0	∞	21
6	85	8	∞	89	0
7	0	0	0	58	∞

i\i	2	3	4	5	7
1	∞	74	0	21	41
3	1	∞	19	0	5
5	14	49	0	∞	21
6	85	8	∞	89	0
7	0	0	0	58	∞

21*
1
14
8
0



i\i	2	3	5	7
3	1	∞	0	5
5	14	49	∞	21
6	85	8	89	0
7	0	0	58	∞

-14

i\i	2	3	5	7
3	1	∞	0	5
5	0	35	∞	7
6	85	8	89	0
7	0	0	58	∞

1
7
8*
0

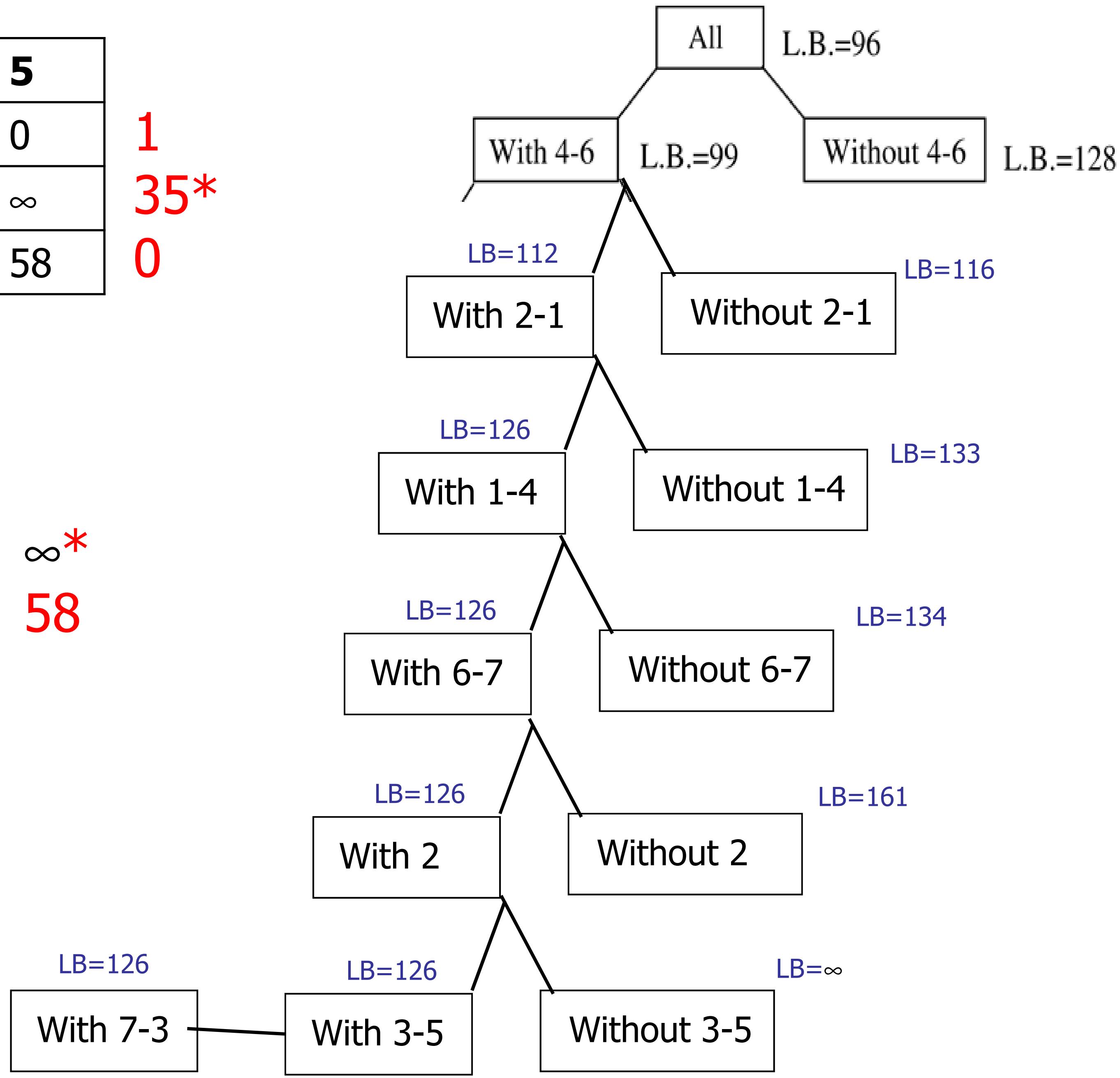
i\i	2	3	5
3	1	∞	0
5	0	35	∞
7	0	0	58

1
35*
0

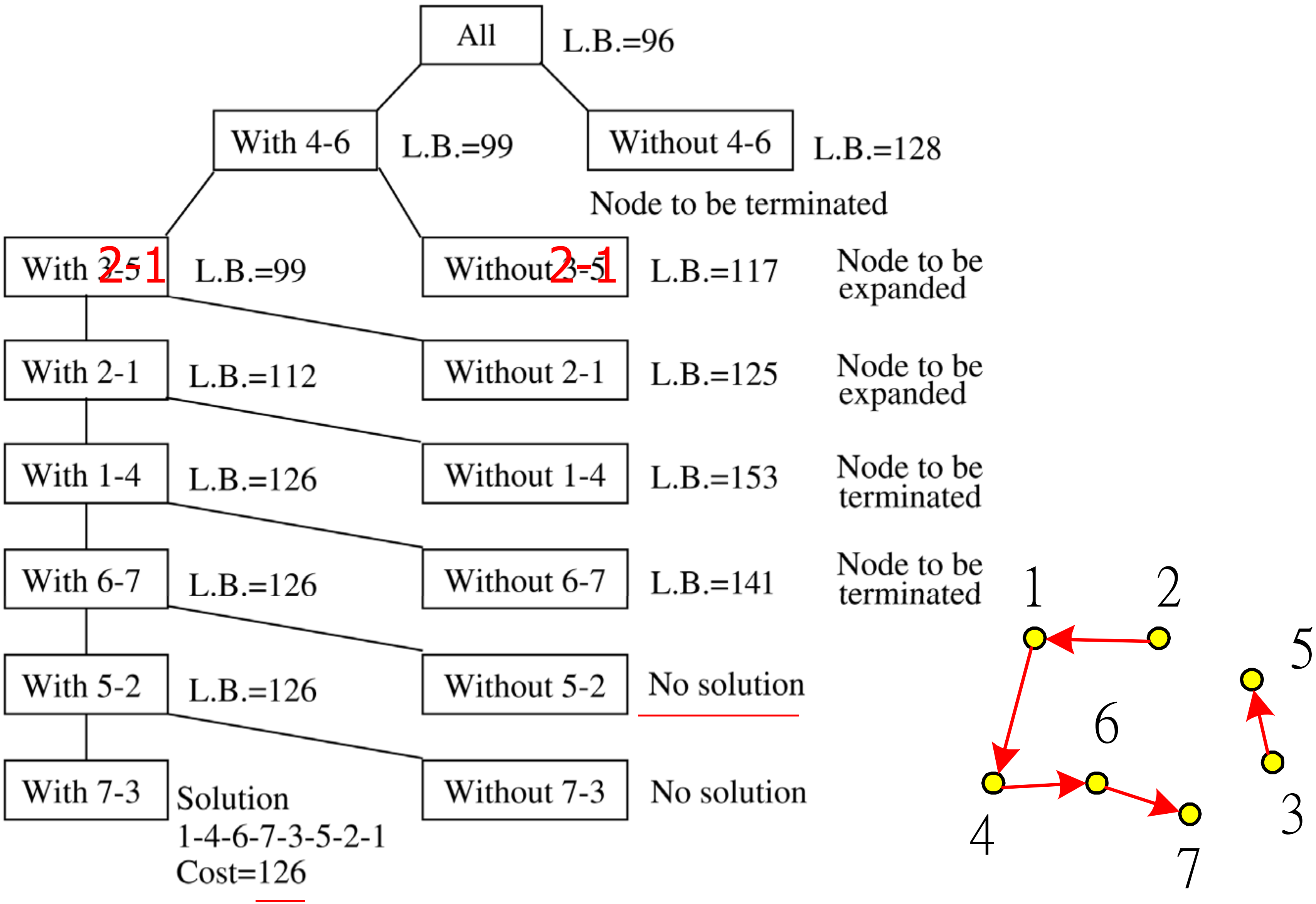
i\i	2	3	5	
3	1	∞	0	1
5	0	35	∞	35*
7	0	0	58	0

i\i	3	5	
3	∞	0	∞*
7	0	58	58

i\i	3
7	0



$4 \to 6 \to 7 \to 3 \to 5 \to 2 \to 1 \to 4$



A branch-and-bound solution of a traveling salesperson problem.

Improvement

- In general, if paths i_1-i_2-…-i_m and j_1-j_2-…-j_n have already been included and a path from i_m, to j_1 is to be added, then path from j_n to i_1 must be prevented.

The 0/1 knapsack problem

學習目標

- **0/1 Knapsack problem 問題定義**
- **Branch-and-bound strategy 演算法設計**

The 0/1 knapsack problem

- Positive integer $P_1, P_2, \ldots, P_n$ (profit)
  $W_1, W_2, \ldots, W_n$ (weight)
  M (capacity)

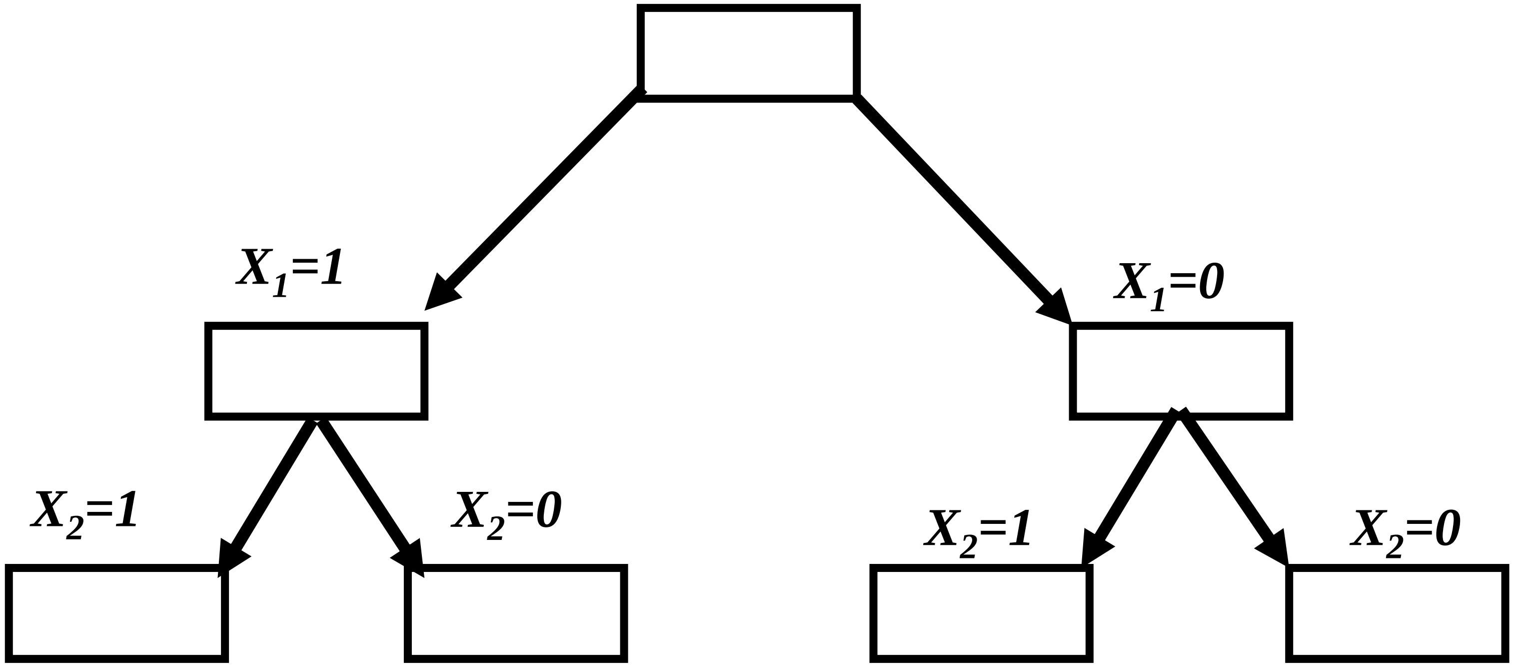
maximize $\sum_{i=1}^{n} P_i X_i$ Maximization problem

subject to $\sum_{i=1}^{n} W_i X_i \le M$ $X_i = 0$ or 1, $i = 1, \ldots, n$.

The problem is modified:

minimize $-\sum_{i=1}^{n} P_i X_i$ Minimization problem

Branching mechanism for 0/1 knapsack problem



B&B process

- We split solutions into two groups. For each group, a lower bound is found.
- At the same time, we try to search for a feasible solution. **Whenever a feasible solution is found, an upper bound is found**.
- Our branch-and-bound strategy terminates the expansion of a node if and only if one of the following conditions is satisfied:
  - **The node itself represents an infeasible solution**. Then no further expansion makes any sense.
  - The lower bound of this node is **higher than or equal to the presently found lowest upper bound**.

Improvement for 0/1-Knapsack

- *For each group, not only a lower bound is found, but also an upper hound is found by finding a feasible solution.*
- As we expand a node, we hope to find a solution with lower cost. This means that we wish to find a lower upper bound as we expand a node.
- **If we know that our upper bound cannot be lowered because it is already equal to its lower bound, then we should not expand this node any more.**
- In general, we terminate the branching **if and only if** one of the following conditions is satisfied:
  - The node itself represents an infeasible solution.
  - The lower bound of this node is higher than or equal to the presently found lowest upper bound.
  - The lower bound of this node is equal to the upper bound of this node.

How to find an upper bound and a lower bound?

- Lower bound can be considered as the value of best solution you can achieve.
- A lower bound of this node therefore corresponds to highest possible profit associated with this partial constructed solution.
- Upper bound : the cost of a feasible solution corresponding to this partially constructed solution.

Find upper bound

- e.g. $n = 6$, $\mathbf{M = 34}$

i	1	2	3	4	5	6
P_i	6	10	4	5	6	4
W_i	10	19	8	10	12	8

($P_i/W_i \geq P_{i+1}/W_{i+1}$ for i=1, 2, …, 6 sorting)

- A feasible solution can be found by staring from the smallest available *i*, scanning towards the larger i's until M is exceeded.
- A feasible solution: $X_1 = 1, X_2 = 1, X_3 = 0, X_4 = 0, X_5 = 0, X_6 = 0$

  $-(P_1+P_2) = -16$ (upper bound)

  Any solution higher than -16 can not be an optimal solution.

Find LB by relaxing the restriction

- **X_i is restricted to 0 an 1.**
- **Relax our restriction from $X_i = 0$ or 1 to $0 \le X_i \le 1$ (knapsack problem)**
- **0/1 knapsack problem -> knapsack problem (greedy method)**
- **Defined as :** Positive integer $P_1, P_2, \ldots, P_n$ (profit)

  $W_1, W_2, \ldots, W_n$ (weight)

  M (capacity)

$$\text{maximize} \quad \sum_{i=1}^{n} P_i X_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} W_i X_i \le M \quad 0 \leqq X_i \leqq 1,\ i = 1, \ldots, n.$$

The problem is modified:

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

Relax the restriction to find lower bound

- **X_i is restricted to 0 an 1.**
- **Relax our restriction from X_i = 0 or 1 to $0 \le X_i \le 1$ (knapsack problem)**

Let $-\sum_{i=1}^{n} P_i X_i$ be an optimal solution for 0/1 knapsack problem and $-\sum_{i=1}^{n} P_i X_i'$ be an optimal solution for knapsack problem. Let $Y = -\sum_{i=1}^{n} P_i X_i$,

$$Y' = -\sum_{i=1}^{n} P_i X_i'.$$

$$\Rightarrow Y' \le Y$$

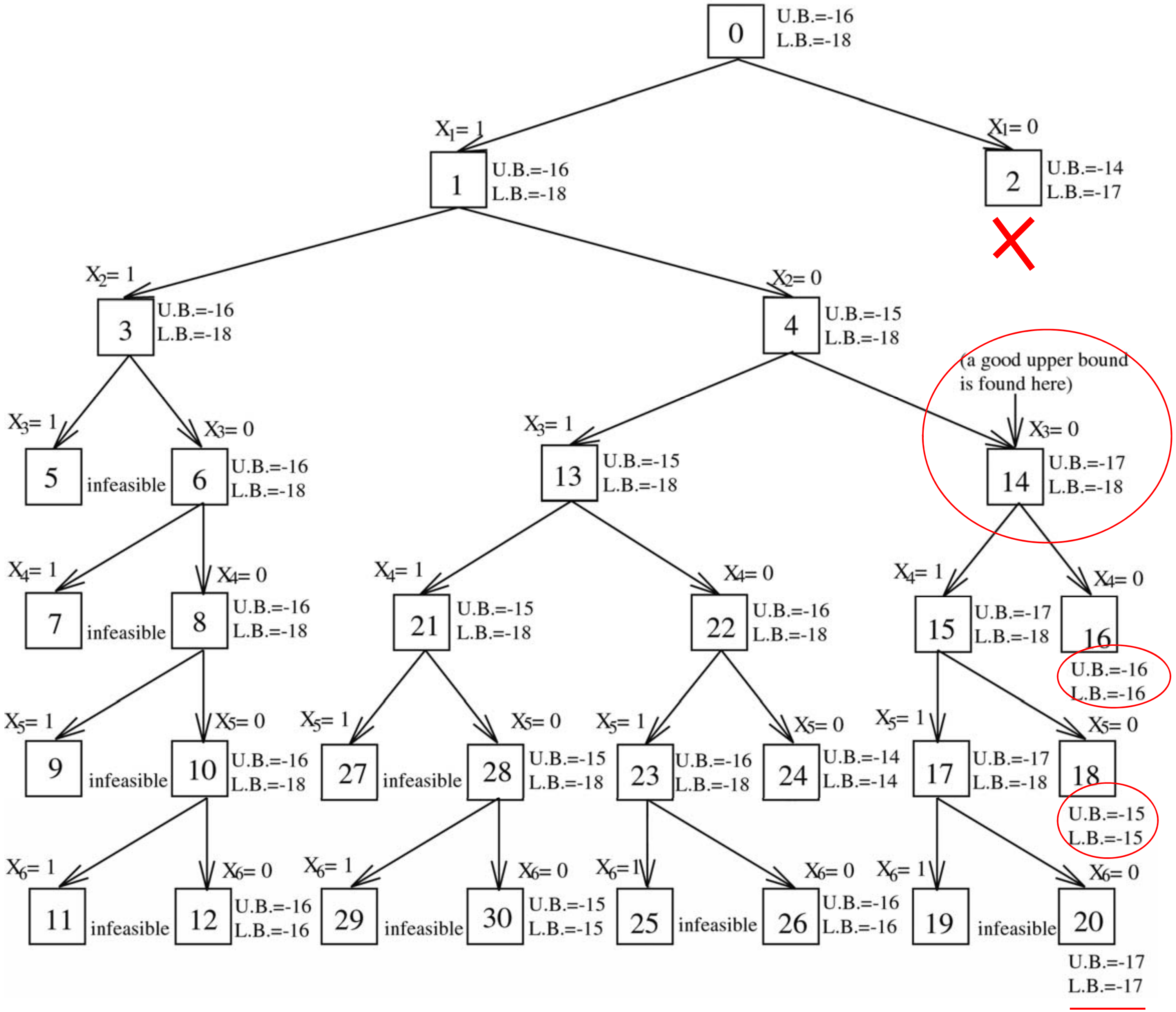
Upper bound and lower bound

- We can use the greedy method to find an optimal solution for knapsack problem:

  $X_1 = 1, X_2 = 1, X_3 = 5/8, X_4 = 0, X_5 = 0, X_6 = 0$
  $-(P_1+P_2+5/8P_3) = -18.5$ (lower bound)
  -18 is our lower bound. (only consider integers)

$\Rightarrow -18 \le$ optimal solution ≤ -16
optimal solution: $X_1 = 1, X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 1, X_6 = 0$
$-(P_1+P_4+P_5) = -17$

Expand the node with the best lower bound.



0/1 knapsack problem solved by branch-and-bound strategy.