Design and Analysis of Algorithms (UE18CS251)

Unit IV - Backtracking and Branch-and-Bound

Mr. Channa Bankapur
channabankapur@pes.edu

Travelling Salesman Problem:

1. Bengaluru
2. New Delhi
3. Mumbai
4. Chennai
5. Kolkata
6. Kochi
7. Hyderabad
8. Bhopal
9. Udaipur
10. Raipur

Travelling Salesman Problem:

1. Given *n* cities and distances between each pair of cities, find the shortest tour that passes through all other cities and returns to the origin city.
2. In case of a weighted complete graph, it's about finding the shortest *Hamiltonian circuit*.

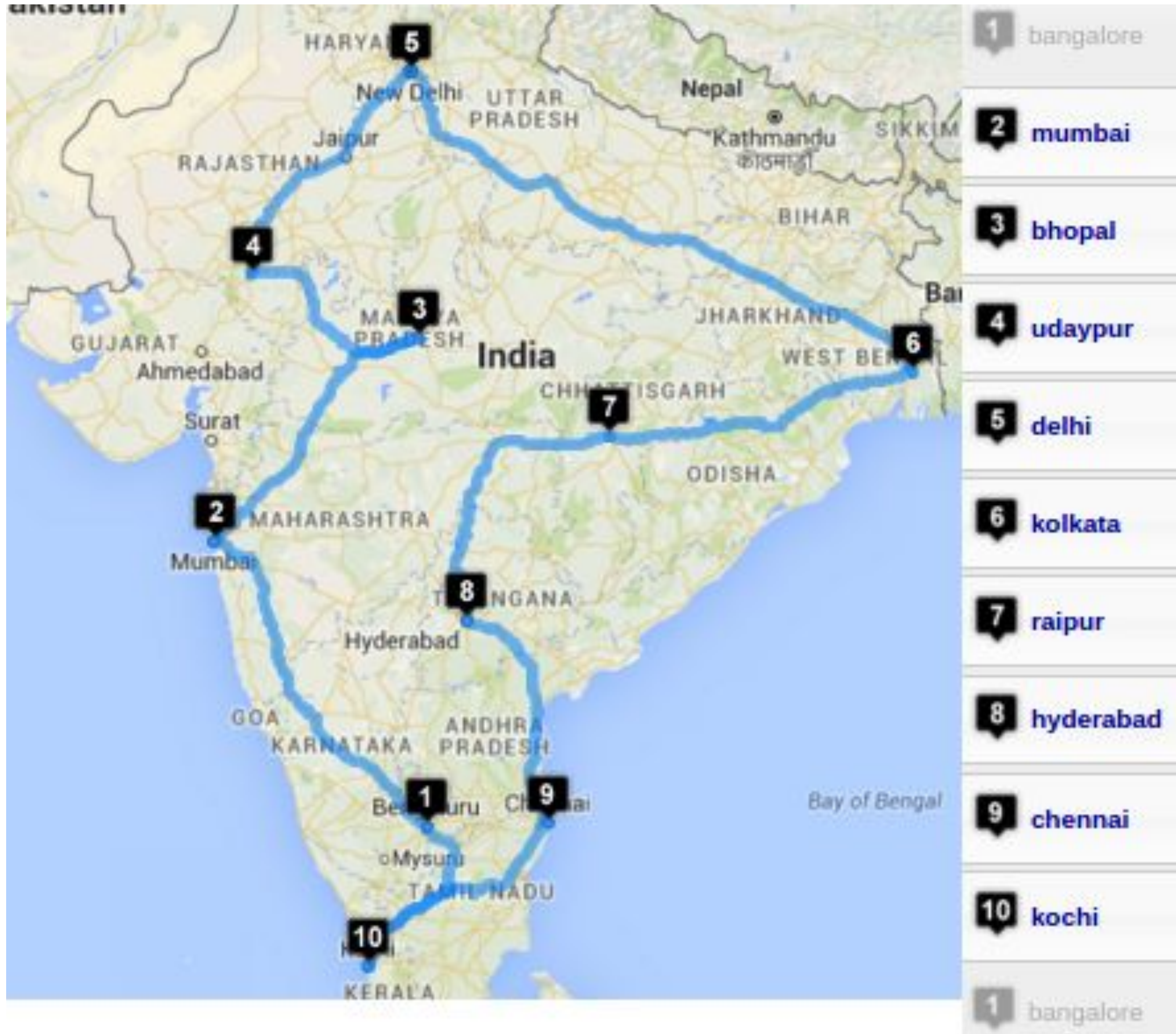
Eg: Driving time between some 10 cities of India (Cost Matrix).

000000	110189	050573	020948	109480	034435	028433	074836	091767	068406
109006	000000	079663	118195	079397	143304	083593	045792	037923	068146
051516	080265	000000	070149	121881	083636	044745	043763	042416	067450
021557	119539	069838	000000	095820	042397	037471	084186	111032	077756
110053	081231	121373	095977	000000	134475	085826	087690	100264	054016
034488	144238	082769	041728	134042	000000	062482	108885	123963	102455
028473	084770	045153	037117	085732	062772	000000	049417	078006	042987
075056	046162	044536	084245	086579	109354	049641	000000	031151	038399
092933	037994	042414	111566	099497	125053	078960	031010	000000	068113
068718	068844	068336	077907	055357	103016	043305	038648	068634	000000

Travelling Salesman Problem:

1. Bengaluru
2. Mumbai
3. Bhopal
4. Udaipur
5. New Delhi
6. Kolkata
7. Raipur
8. Hyderabad
9. Chennai
10. Kochi
11. Bengaluru

Shortest round trip takes **454201** sec.



```
Algorithm Travelling Salesperson Problem
   mincost ← Infinity
   for each permutation of (n - 1) cities
      cost ← 0
      for each edge in the Hamiltonian circuit
         cost ← cost + cost of the edge
      if (cost < mincost)
         mincost ← cost
   return mincost
```

Input Size: **n**

Basic Operation : **addition of cost of an edge**

C(n) **= n * (n - 1)! = n! ∈ Θ(n!)**

ALGORITHM **TravellingSalesmanProblem**

//Input: n x n adjacency matrix A. Assumed n > 1.

//Output: Min Cost Hamiltonian circuit.

//getPermutation(P[]) returns true with next permutation in lexicographic

// order, if it exists. Returns false otherwise.

```
mincost ← INFINITY
Permutation[1..n-1] ← [1, 2, 3, …, n-1] //1st permutation
do
  cost ← A[0, Permutation[1]] //1st edge of the circuit
  for i ← 1 to n-2
    cost ← cost + A[Permutation[i], Permutation[i+1]]
  cost ← cost + A[Permutation[n-1], 0] //last edge
  if (cost < mincost) mincost ← cost
while(getNextPermutation(Permutation[1..n-1]))
return mincost
```

Travelling Salesman Problem:

Example:

Graph vertices: A, B, C, D

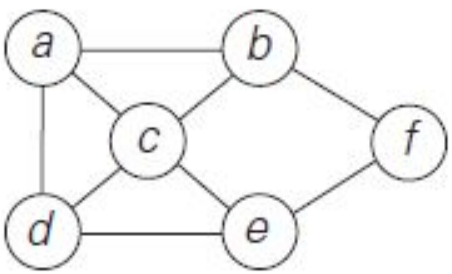
```
0  6  5  2

1  0  3  -

5  3  0  3

2  2  7  0
```

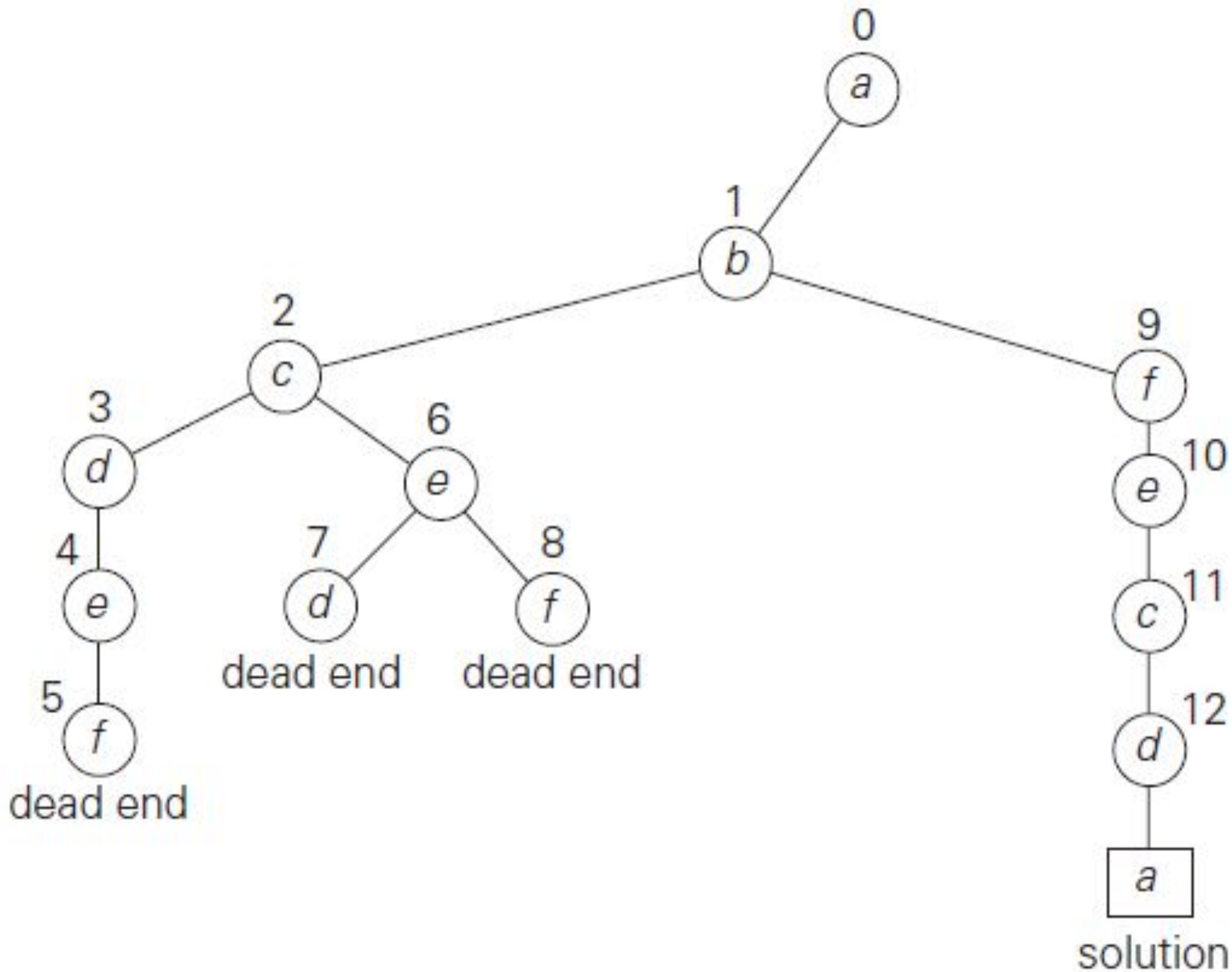
Backtracking

The exhaustive search technique suggests generating all candidate solutions and then identifying the one (or the ones) with a desired property.

- Construct solutions one component at a time and evaluate such partially constructed candidates.
- Construct the **state-space tree**
  - **non-leaf nodes**: promising nodes with partial solutions
  - **leaves**: non-promising nodes or solutions
  - **edges**: choices in extending partial solutions
- Explore the state space tree using **depth-first search**.
- **Backtrack** at non-promising nodes.

Backtracking - Hamiltonian Circuit Problem





Knapsack Problem:

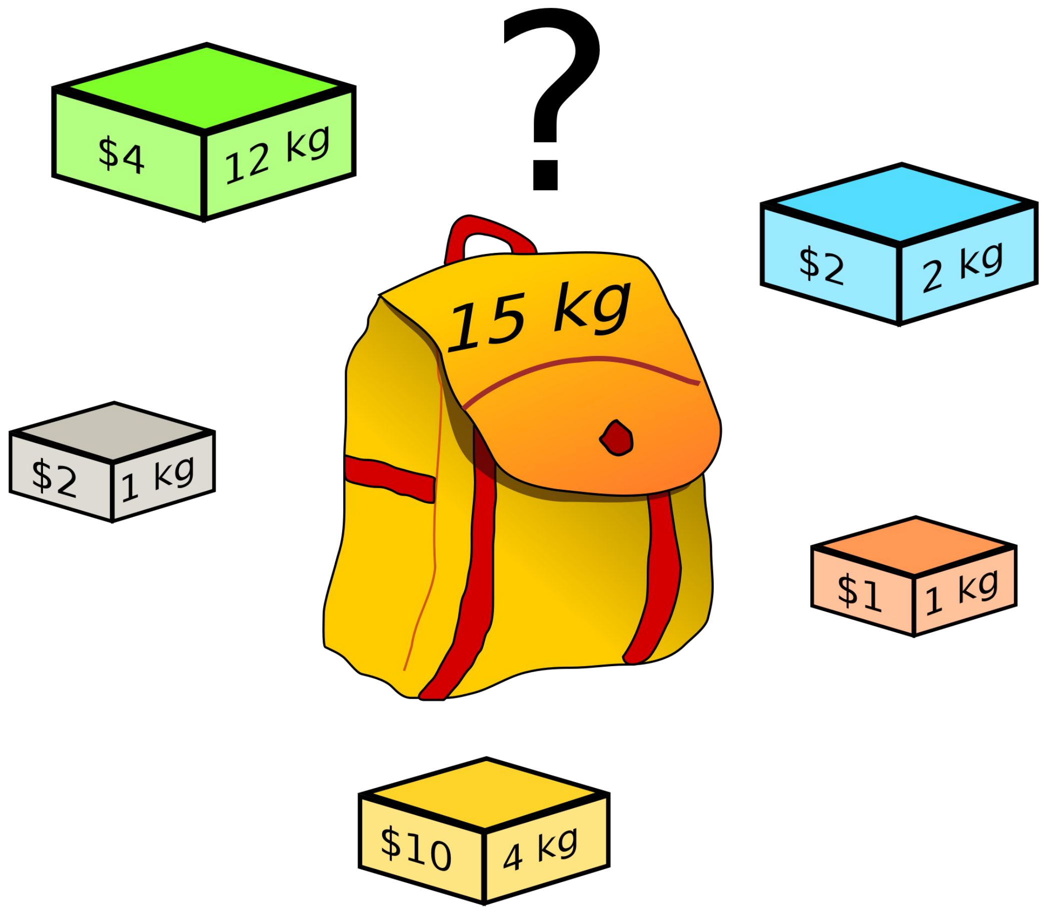
Given n items:

weights: $w_1 \quad w_2 \; \ldots \; w_n$

values: $v_1 \quad v_2 \; \ldots \; v_n$

a knapsack of capacity W

Find most valuable subset of the items that fit into the knapsack.



Knapsack Problem:

Given n items:

weights: $w_1 \quad w_2 \; \ldots \; w_n$

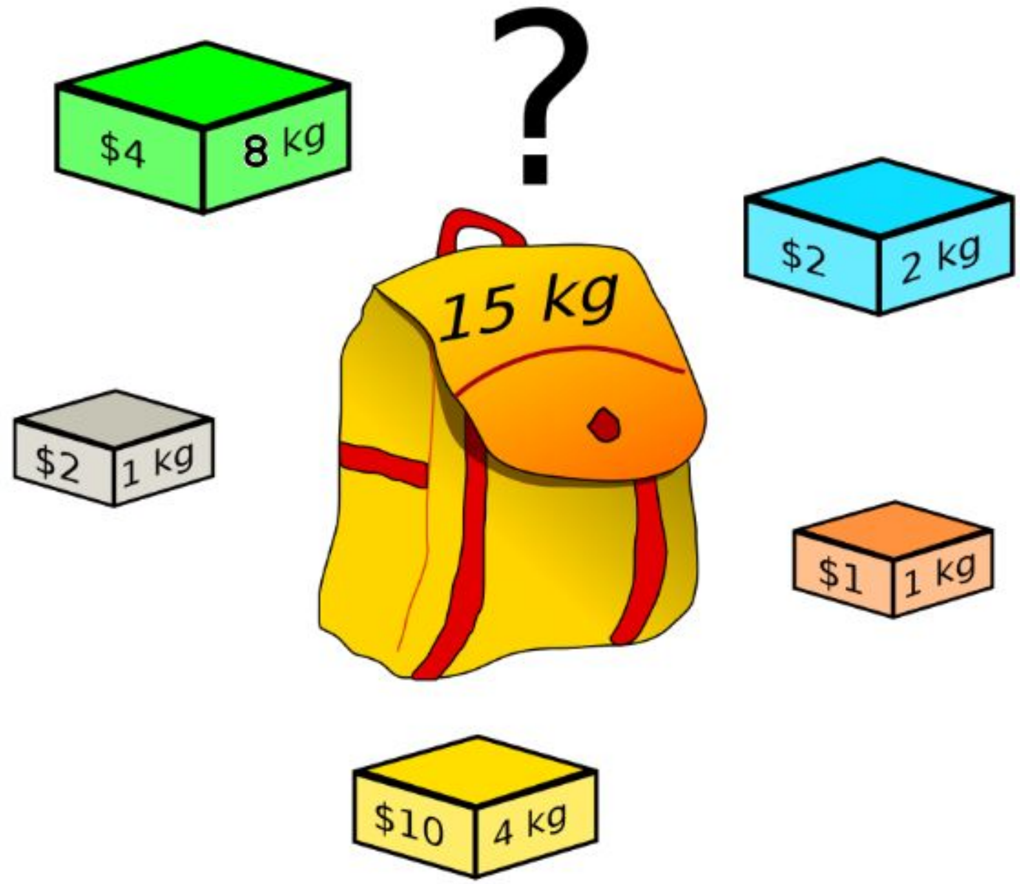
values: $v_1 \quad v_2 \; \ldots \; v_n$

a knapsack of capacity W

Find most valuable subset of the items that fit into the knapsack.

Ans: {B,O,Y,W} 8kg, $15

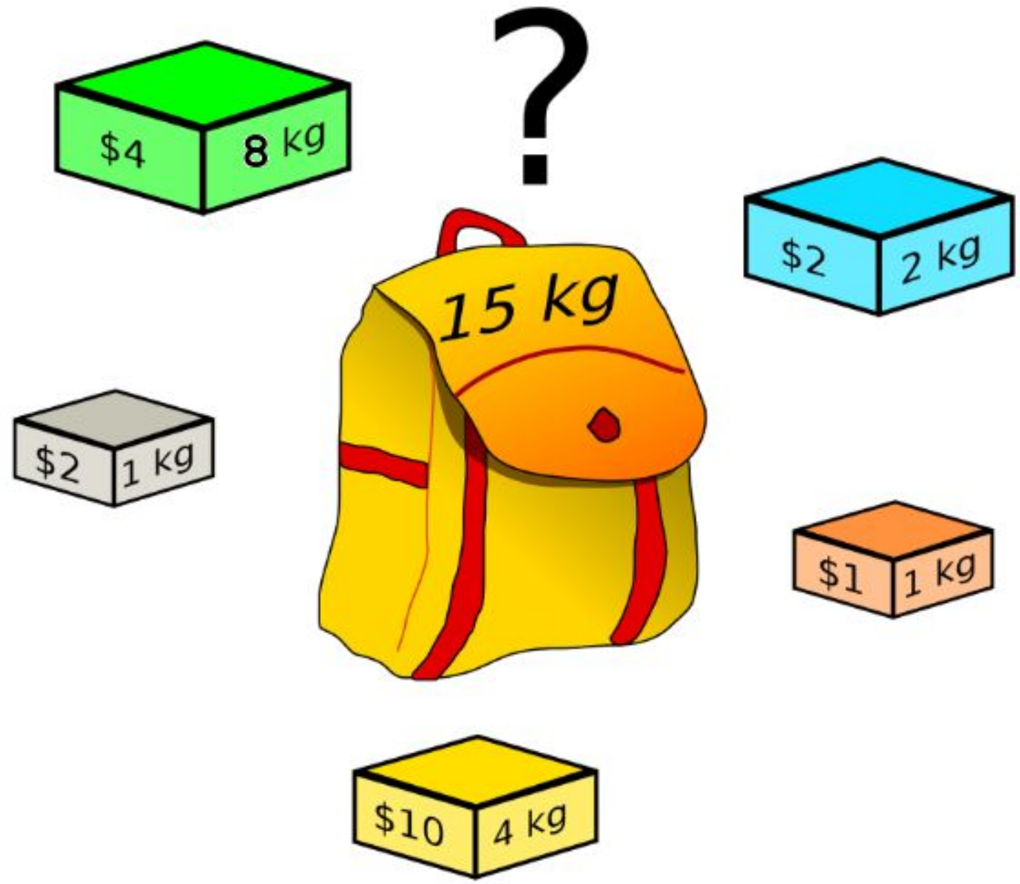
What if the green object weighs 8 kg instead of 12 kg?



Knapsack Problem:

What if the green object weighs 8 kg instead of 12 kg?

Ans:
{G,B,Y,W}
15kg, $18



Example:

Knapsack capacity W=16

item	weight	value
1	2	$20
2	5	$30
3	10	$50
4	5	$10

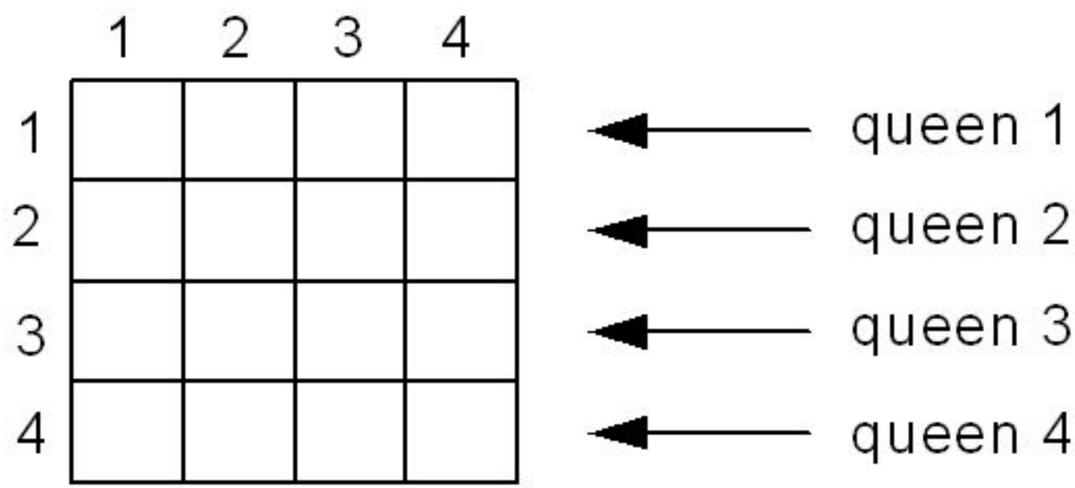
{2,3} with value **$80** is optimal.

C(n) $\in \mathbf{\Omega(2^n)}$

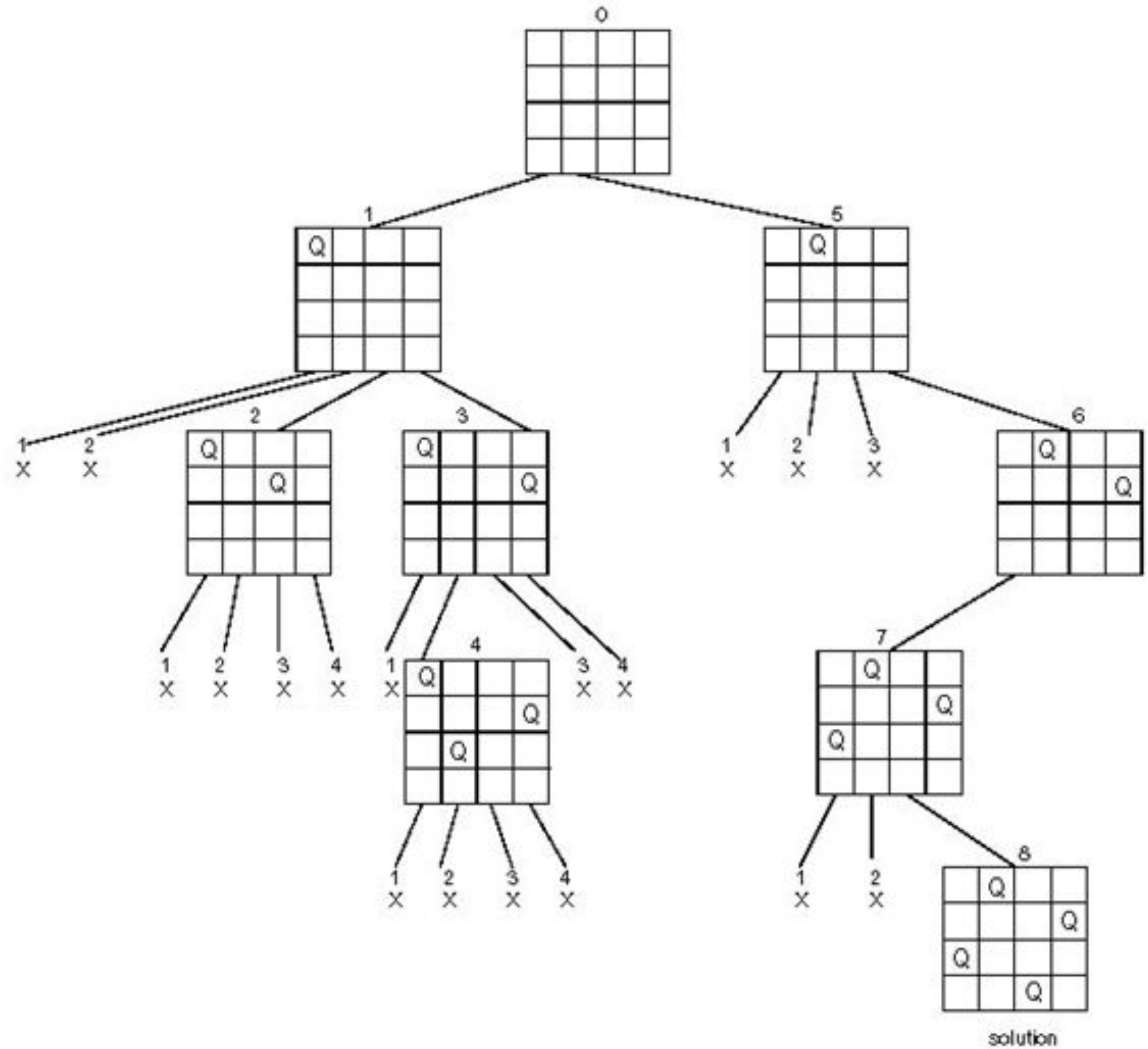
Subset	Total weight	Total value
{}	0	$0
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	-
{1,2,4}	12	$60
{1,3,4}	17	-
{2,3,4}	20	-
{1,2,3,4}	22	-

Backtracking - n-Queens Problem

Place n queens on an n-by-n chessboard so that no two of them are in the same row, column, or diagonal.



Backtracking - n-Queens Problem



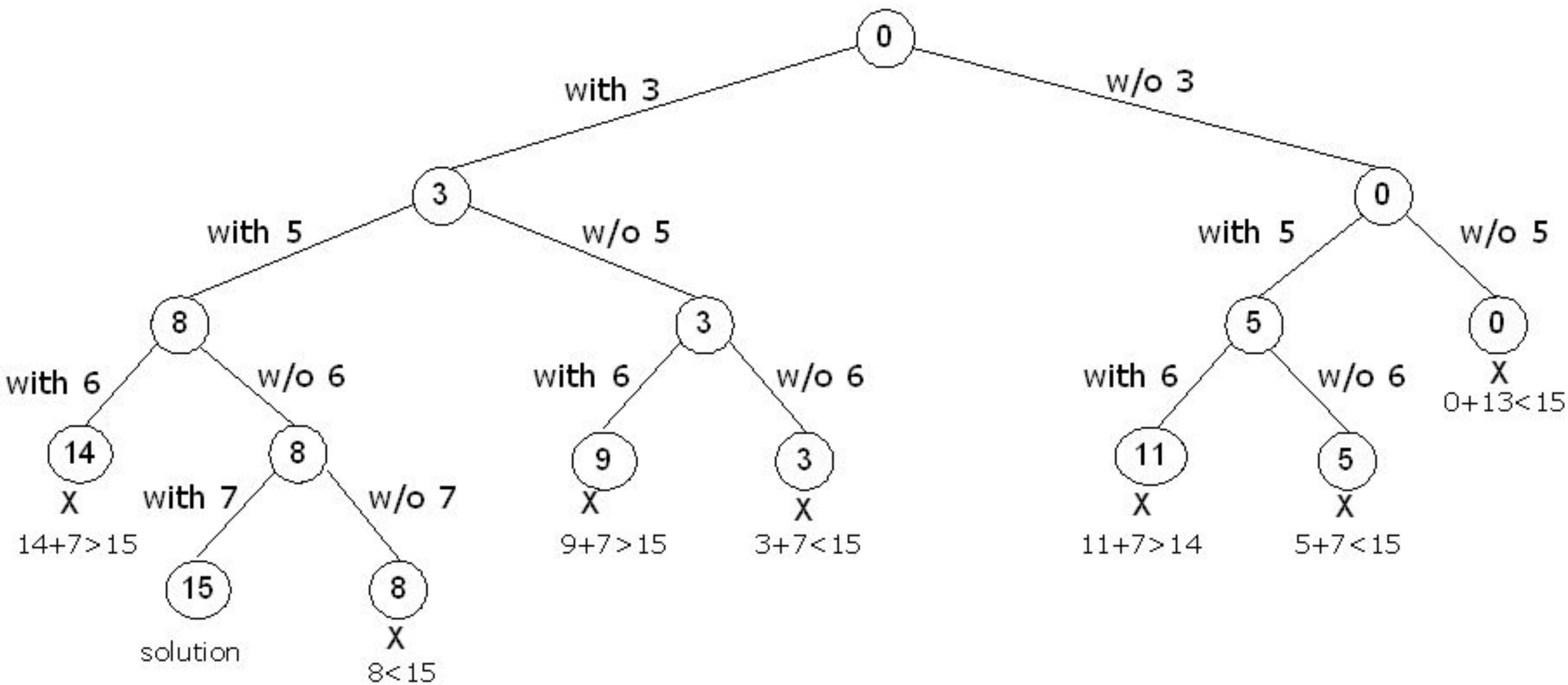
Backtracking - template algorithm

ALGORITHM *Backtrack*$(X[1..i])$

//Gives a template of a generic backtracking algorithm
//Input: $X[1..i]$ specifies first i promising components of a solution
//Output: All the tuples representing the problem's solutions
if $X[1..i]$ is a solution **write** $X[1..i]$
else
    for each element $x \in S_{i+1}$ consistent with $X[1..i]$ and the constraints **do**
        $X[i+1] \leftarrow x$
        Backtrack$(X[1..i+1])$

Backtracking - Subset-Sum Problem

$S = \{3, 5, 6, 7\}$ and $d = 15$



Branch-and-Bound

- An enhancement of backtracking
- Applicable to optimization problems
- Makes a note of the best solution seen so far
- For each node (partial solution) of a state-space tree, computes a **bound** on the value of the objective function for all descendants of the node (extensions of the partial solution)

Branch-and-Bound - Knapsack Problem

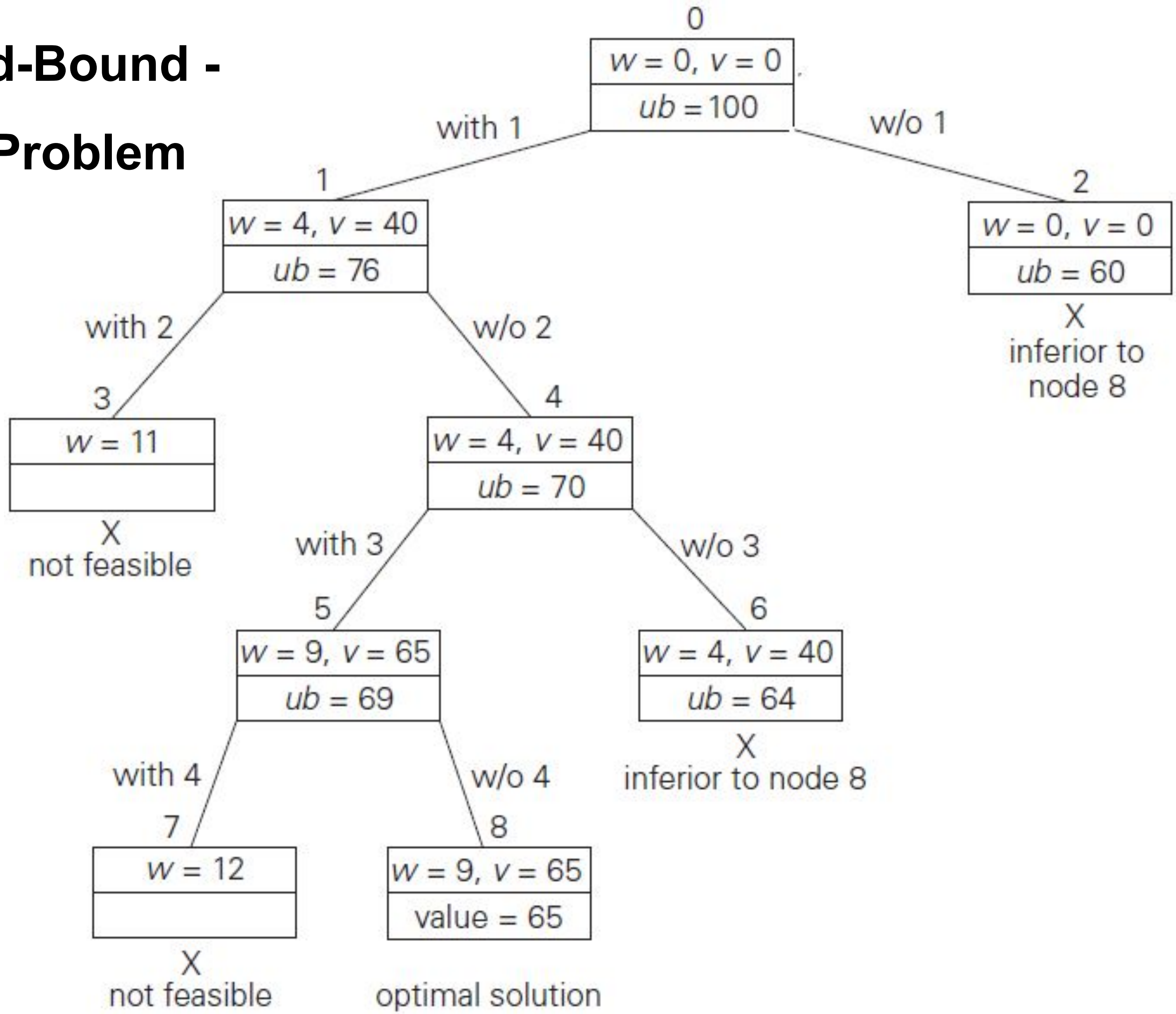
$$v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n.$$

$$ub = v + (W - w)(v_{i+1}/w_{i+1}).$$

item	weight	value	$\dfrac{\text{value}}{\text{weight}}$
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

The knapsack's capacity W is 10.

Branch-and-Bound - Knapsack Problem



Branch-and-Bound - Knapsack Problem

Example:

Knapsack capacity W=16

item	weight	value
1	2	$20
2	5	$30
3	10	$50
4	5	$10

Branch-and-Bound - Travelling Salesman Problem

Objective is to **minimize** the cost of the circuit. The effort of minimizing does not help if the lower bound is already higher than a solution found so far at an intermediate stage.

Example:

Graph vertices: A, B, C, D

```
0 6 5 2
1 0 3 -
5 3 0 3
2 2 7 0
```

Branch-and-Bound - Travelling Salesman Problem

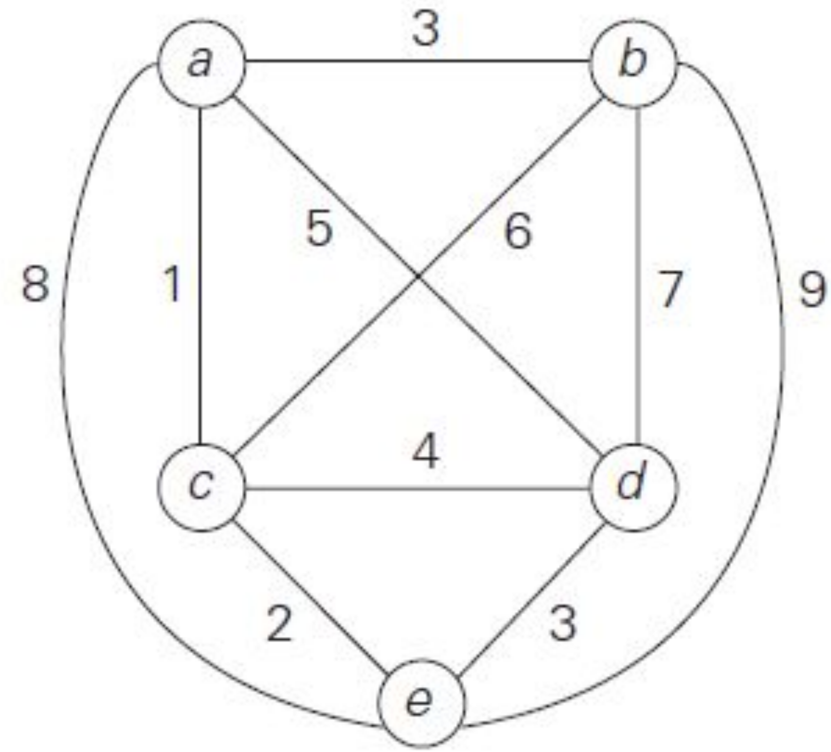
- One very simple lower bound can be obtained by finding the smallest element in the intercity distance matrix D and multiplying it by the number of cities n.
  - Lower bound = n * (cost of the least cost edge)
- A better method is for each city i, $1 \leq i \leq n$, find the sum s_i of the distances from city i to the two nearest cities; compute the sum s of these n numbers, divide the result by 2
  - Lower bound = ceil(s/2)

Branch-and-Bound - Travelling Salesman Problem

Initial lower bound =

ceil(**((1 + 3) + (3 + 6) + (1 + 2) +**

(3 + 4) + (2 + 3)) / 2) = 14.



Lower bound for all the

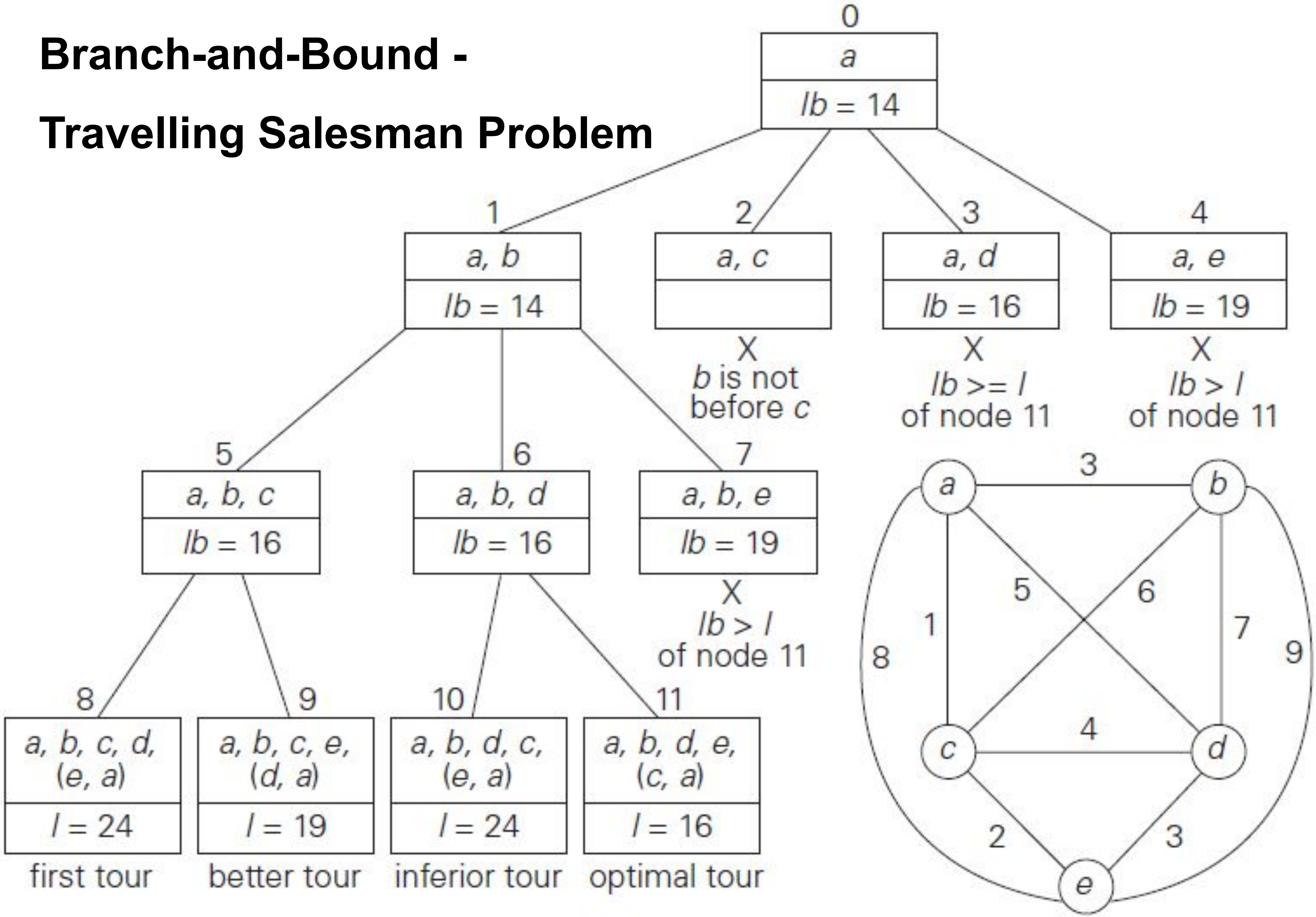
Hamiltonian circuits of the graph

that must include edge (a, d) =

ceil(((1 + **5**) + (3 + 6) + (1 + 2) +

(**5** + 3) + (2 + 3)) / 2) = **16**.

Branch-and-Bound - Travelling Salesman Problem



Branch-and-Bound - Assignment Problem

Objective is to **minimize** the cost of the assignment. The effort of minimizing does not help if the lower bound is already higher than a solution found so far at an intermediate stage.

$$
C = \begin{array}{c} \\ \\ \end{array}
\begin{array}{cccc}
\text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} \\
\end{array}
$$

$$
C = \left[\begin{array}{cccc}
9 & 2 & 7 & 8 \\
6 & 4 & 3 & 7 \\
5 & 8 & 1 & 8 \\
7 & 6 & 9 & 4
\end{array}\right]
\begin{array}{l}
\text{person } a \\
\text{person } b \\
\text{person } c \\
\text{person } d
\end{array}
$$

Deadend? Backtrack. Repeat!

**</
Backtracking
>**