Design and Analysis of Algorithms

UNIT-IV

Branch-and-Bound

Dr G. Kalyani

Topics

- General Method
- Travelling Sales Person Problem
- 0/1 Knapsack Problem

Branch-and-Bound

- The term branch-and-bound refers to all state space search methods in which all children of the E-node are generated before any other live node can become the E-node.

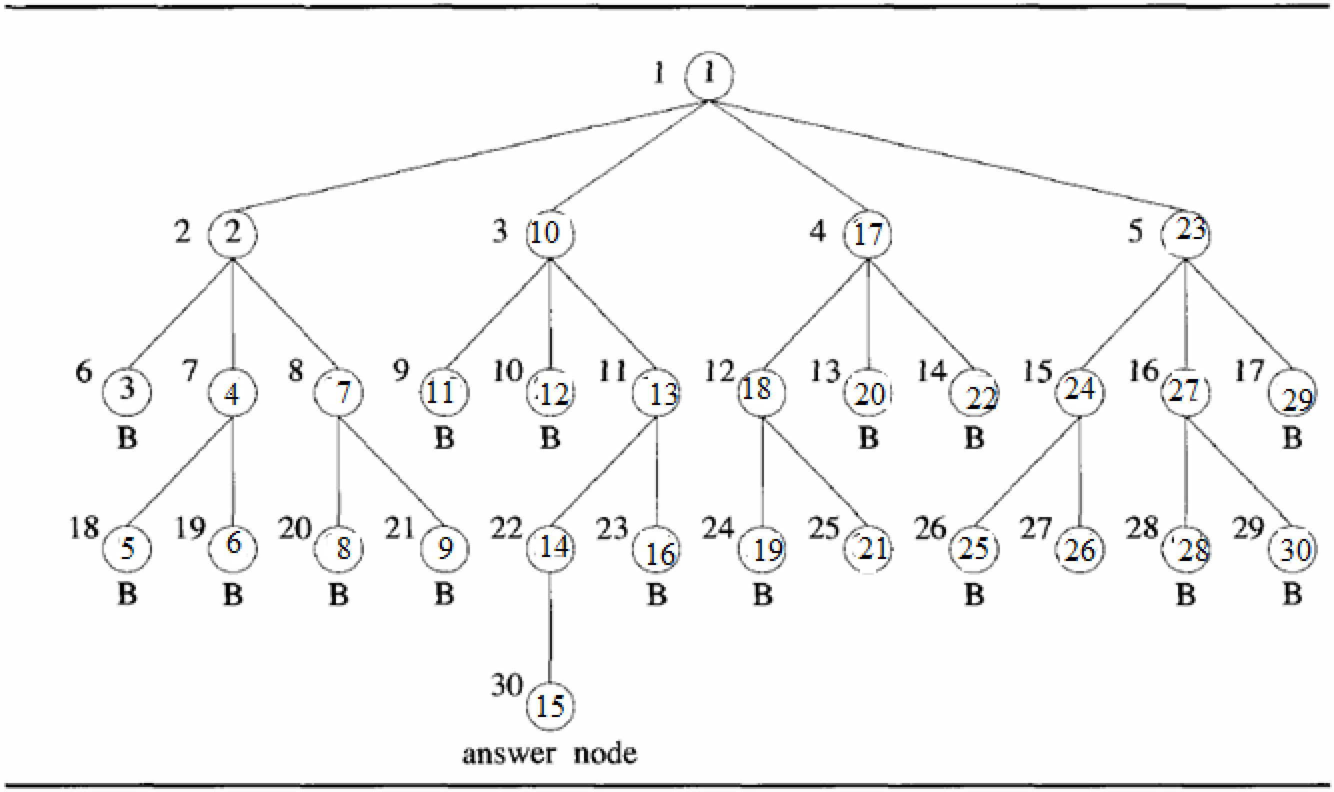
- We have already seen a graph search strategies, BFS in which the exploration of a new node can not begin until the node currently being explored is fully explored.

- In branch-and-bound terminology, a BFS-like state space search will be called FIFO(First In First Out) branch-and-bound as the list of live nodes is a first-in-first-out list (or queue).

- A D-search-like state space search will be called LIFO(Last In First Out) branch-and bound as the list of live nodes is a last-in-first-out list (or stack).

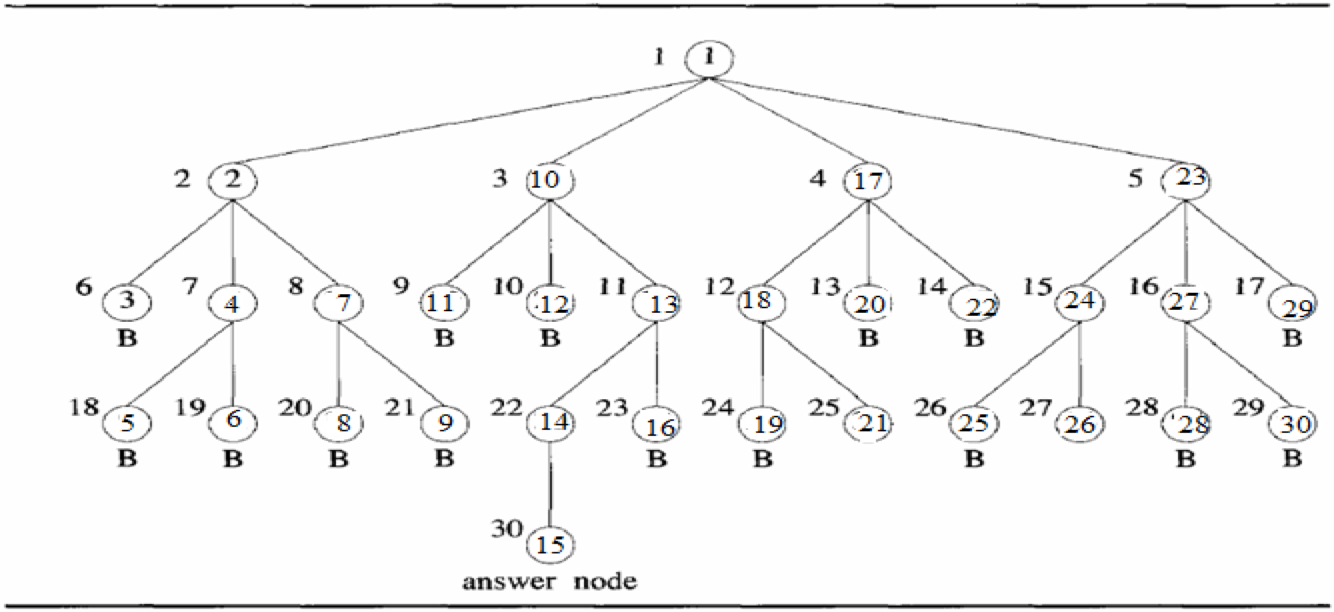
Branch-and-Bound

- In both FIFO and LIFO selection for next E-node is rigid and not useful to get answer node quickly.



LC Search

- The search for an answer node can often be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes.
- The next E-node is selected on the basis of this ranking function.
- If in the previous example we use a ranking function that assigns node 10 a better rank than all other live nodes, then node 10 will become the E-node following node 1.
- The remaining live nodes will never become E-nodes as the expansion of node 10 results in the generation of an answer node (node31).



LC Search

Let $\hat{g}(x)$ be an estimate of the additional effort needed to reach an answer node from x. Node x is assigned a rank using a function $\hat{c}(\cdot)$ such that $\hat{c}(x) = f(h(x)) + \hat{g}(x)$, where $h(x)$ is the cost of reaching x from the root and $f(\cdot)$ is any nondecreasing function.

A search strategy that uses a cost function c^(x)= f(h(x))+g^(x) select the next E-node with least c^(.). Hence, such a search strategy is called an LC-search (Least Cost search).

An LC-search coupled with bounding functions is called an LC branch-and-bound search.

Control Abstraction of LC Search

```
listnode = record {
    listnode *next, *parent; float cost;
}

Algorithm LCSearch(t)
// Search t for an answer node.
{
    if *t is an answer node then output *t and return;
    E := t; // E-node.
    Initialize the list of live nodes to be empty;
    repeat
    {
        for each child x of E do
        {
            if x is an answer node then output the path
                from x to t and return;
            Add(x); // x is a new live node.
            (x → parent) := E; // Pointer for path to root.
        }
        if there are no more live nodes then
        {
            write ("No answer node"); return;
        }
        E := Least();
    } until (false);
}
```

Topics

- **General Method**
- **Travelling Sales Person Problem**
- **0/1 Knapsack Problem**

Travelling Sales Person Problem

- Let G = (V, E) be a directed graph defining an instance of the traveling sales person problem.
- Let Cij equal the cost of edge(i,j), Cij = ∞ if (i,j) n$ot\ in$ E, and let |V|= n.
- Without loss of generality, we can assume that every tour starts and ends at vertex1.
- **Identify the tour path for TSP in such a way that every location should be visited only once.**

An Example State Space Tree for TSP

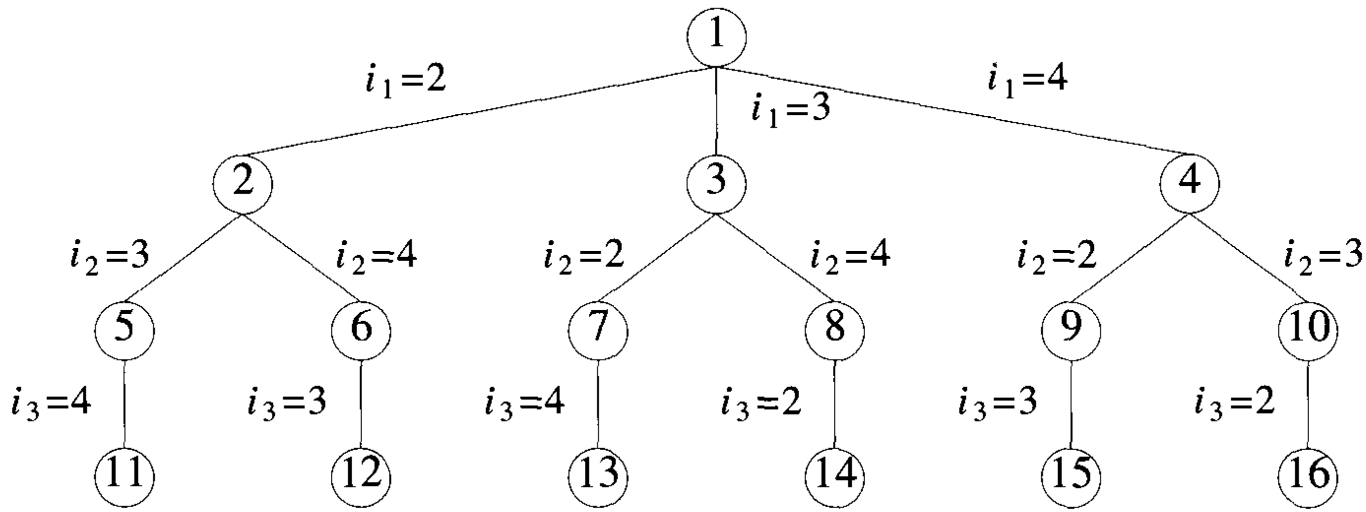


Figure 8.10 State space tree for the traveling salesperson problem with $n = 4$ and $i_0 = i_4 = 1$

TSP with LCBB

To use LCBB to search the traveling salesperson state space tree, we need to define a cost function $c(\cdot)$ and two other functions $\hat{c}(\cdot)$ and $u(\cdot)$ such that $\hat{c}(r) \leq c(r) \leq u(r)$ for all nodes r. The cost $c(\cdot)$ is such that the solution node with least $c(\cdot)$ corresponds to a shortest tour in G. One choice for $c(\cdot)$ is

$$c(A) = \begin{cases} \text{length of tour defined by the path from the root to } A, \text{ if } A \text{ is a leaf} \\ \text{cost of a minimum-cost leaf in the subtree } A, \text{ if } A \text{ is not a leaf} \end{cases}$$

- A better c^(.) can be obtained by using the reduced cost matrix corresponding to G.
- A matrix is reduced iff every row and column is reduced.
- A row (column)is said to be reduced iff it contains at least one zero and all remaining entries are non-negative.
- The total amount subtracted from the columns and rows is a lower bound on the length of a minimum-cost tour and can be used as the c^(.) value for the root of the state space tree.

An Example for TSP with LCBB

- Find the shortest path tour for the following TSP problem.

Ex:

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

An Example for TSP with LCBB

- **Reduce the matrix to get c^(.)**
- **Row reduction**

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix} \Rightarrow$$

$R_1 : 10$
$R_2 : 2$
$R_3 : 2$
$R_4 : 3$
$R_5 : 4$

An Example for TSP with LCBB

- **Reduce the matrix to get c^(.)**
- **Column reduction:**

Ex:

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

R_1: 10
R_2: 2
R_3: 2
R_4: 3
R_5: 4

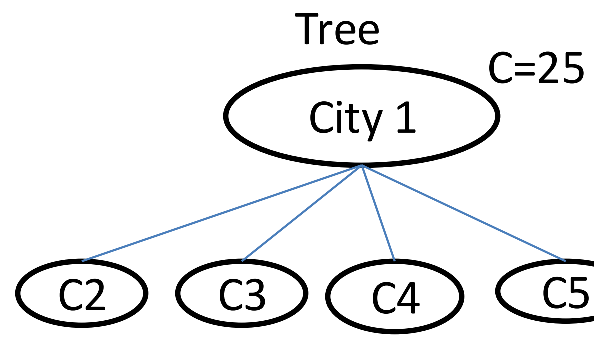
C_1: 1
C_2: 0
C_3: 3
C_4: 0
C_5: 0

$$\begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

$\Rightarrow$ reduced cost $= 10 + 2 + 2 + 3 + 4 + 1 + 3$

An Example for TSP with LCBB

City 1 to City 2:
1→2



$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & \underline{3} & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

An Example for TSP with LCBB

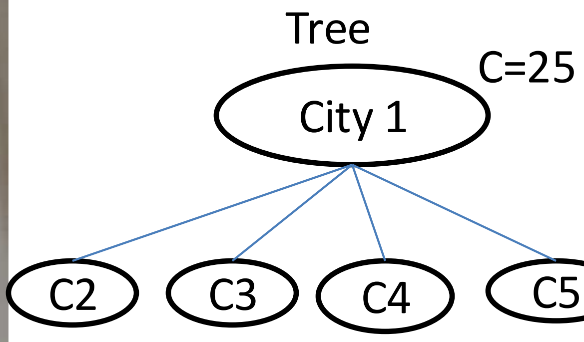
city 1 ⟶ city 2. $\Rightarrow 25 + c(1,2) = 25 + 10 = 35$

$$2 \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & 11 & 2 & 0 \\ \infty & \infty & 0 & 2 \\ \infty & 12 & \infty & 0 \\ \infty & 0 & 12 & \infty \end{bmatrix} \Rightarrow \text{reduced}$$

city 1 → city 3 $\Rightarrow 25 + c(1,3) = 25 + 17 = 42$

$$3 \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & 2 & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

Total cost at 3 = 42 + 11
= 53.



An Example for TSP with LCBB

for 4 ⇒ city 1 → city 4 ⇒ 25+0 = 25

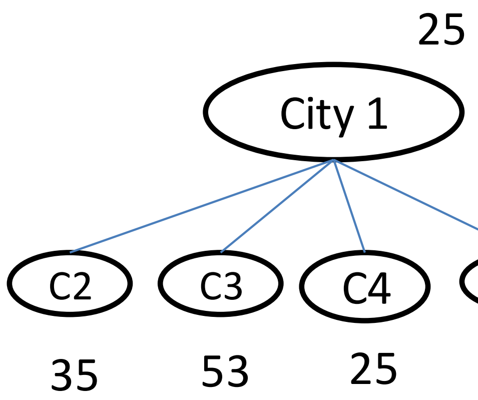
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \Rightarrow \text{reduced.}$$

for 5 ⇒ city 1 → city 5 ⇒ 25+1 = 26.

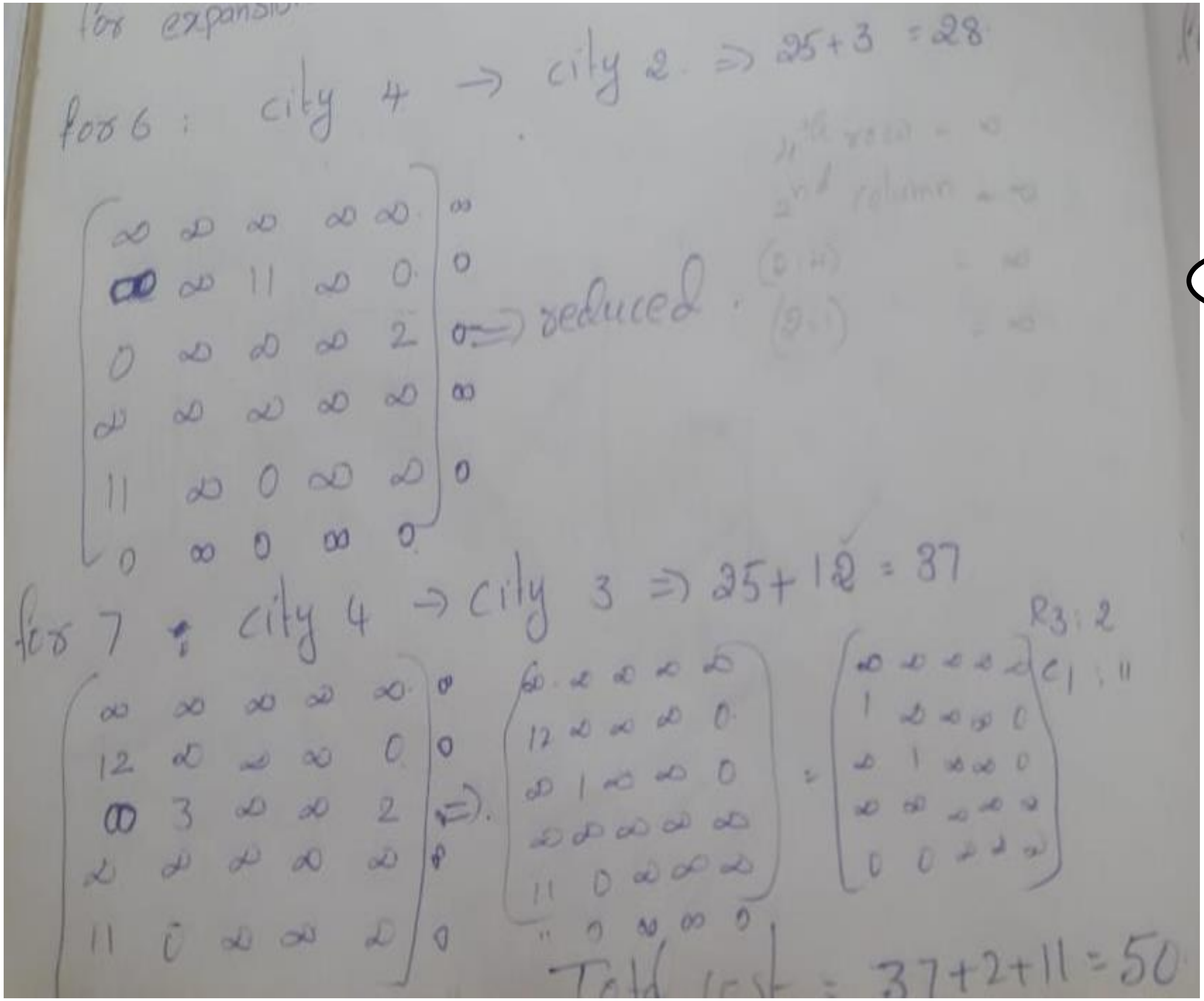
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

$R_2 : 2$
$R_4 : 3$

∴ Total cost = 26+2+3 = 31



An Example for TSP with LCBB



for expansion

for 6 : city 4 → city 2 ⇒ 25+3 = 28

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix} \begin{matrix} \infty \\ 0 \\ 0 \\ \infty \\ 0 \end{matrix} \Rightarrow \text{reduced}$$

$$0 \quad \infty \quad 0 \quad \infty \quad 0$$

for 7 : city 4 → city 3 ⇒ 25+12 = 37

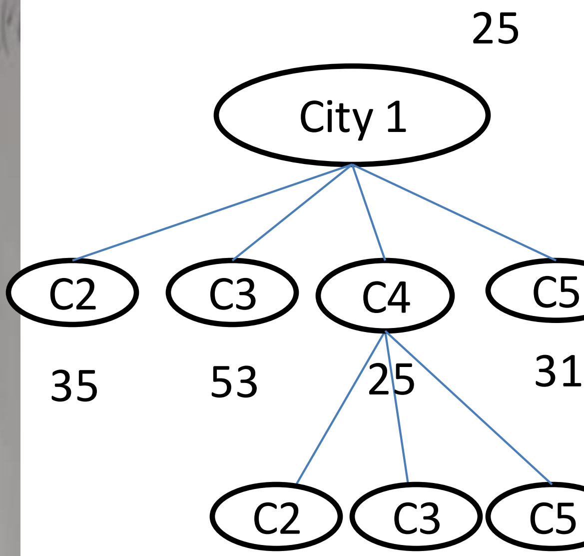
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 2 \\ \infty \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

$$11 \quad 0 \quad \infty \quad \infty \quad 0$$

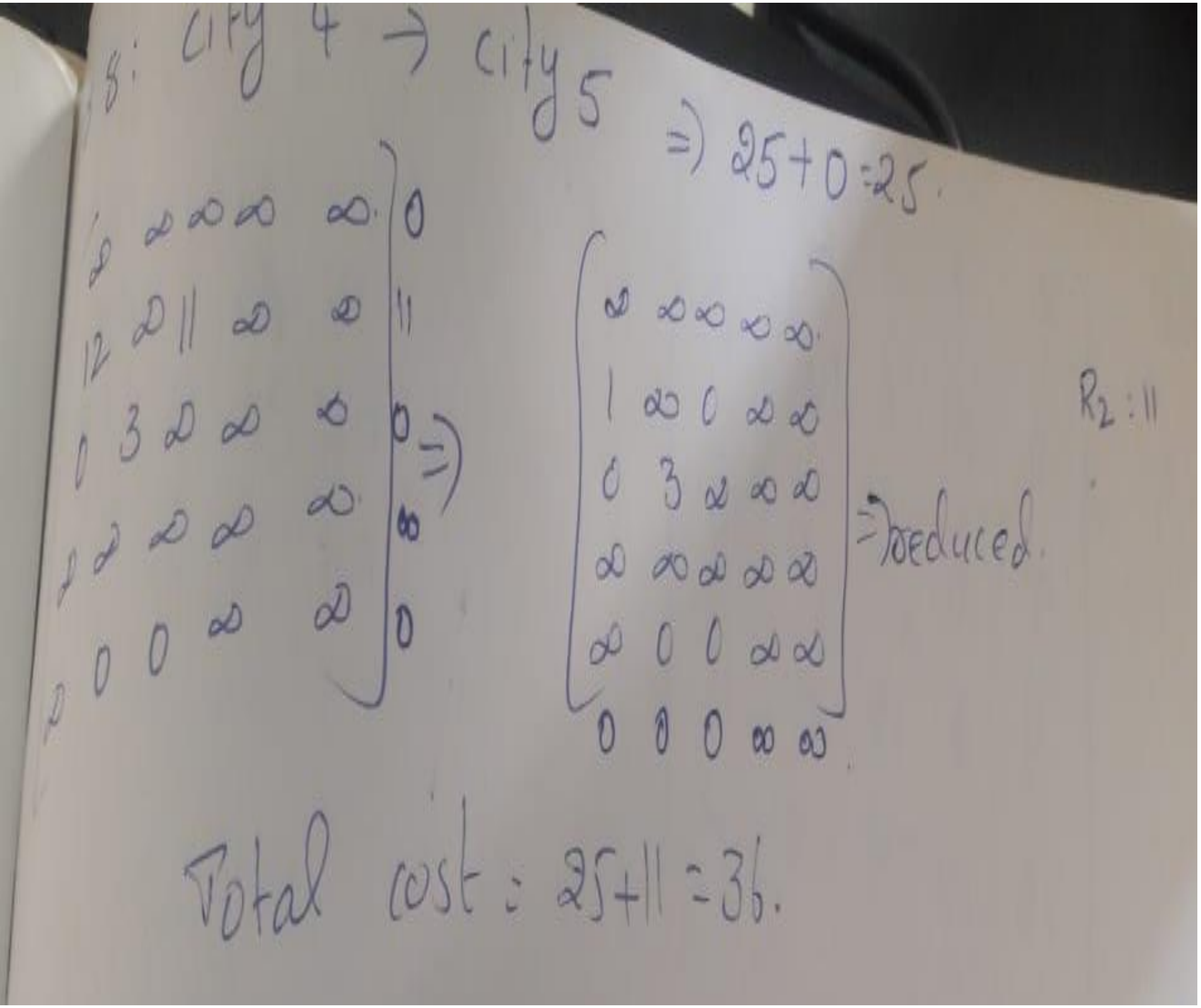
$R_3 : 2$

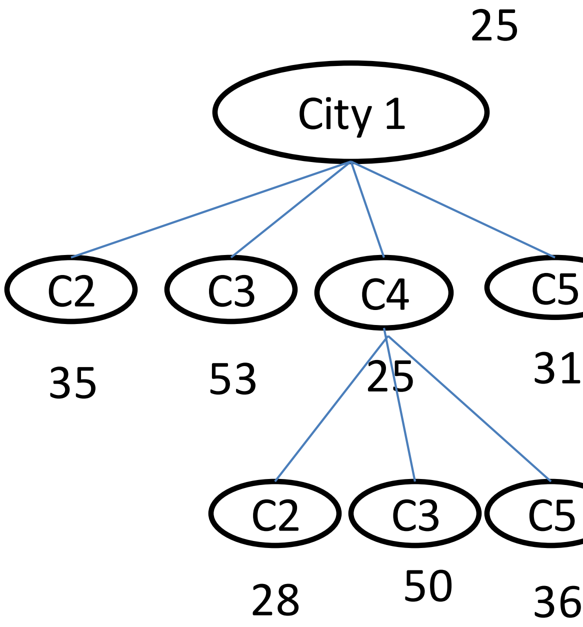
$C_1 : 11$

Total cost = 37+2+11 = 50

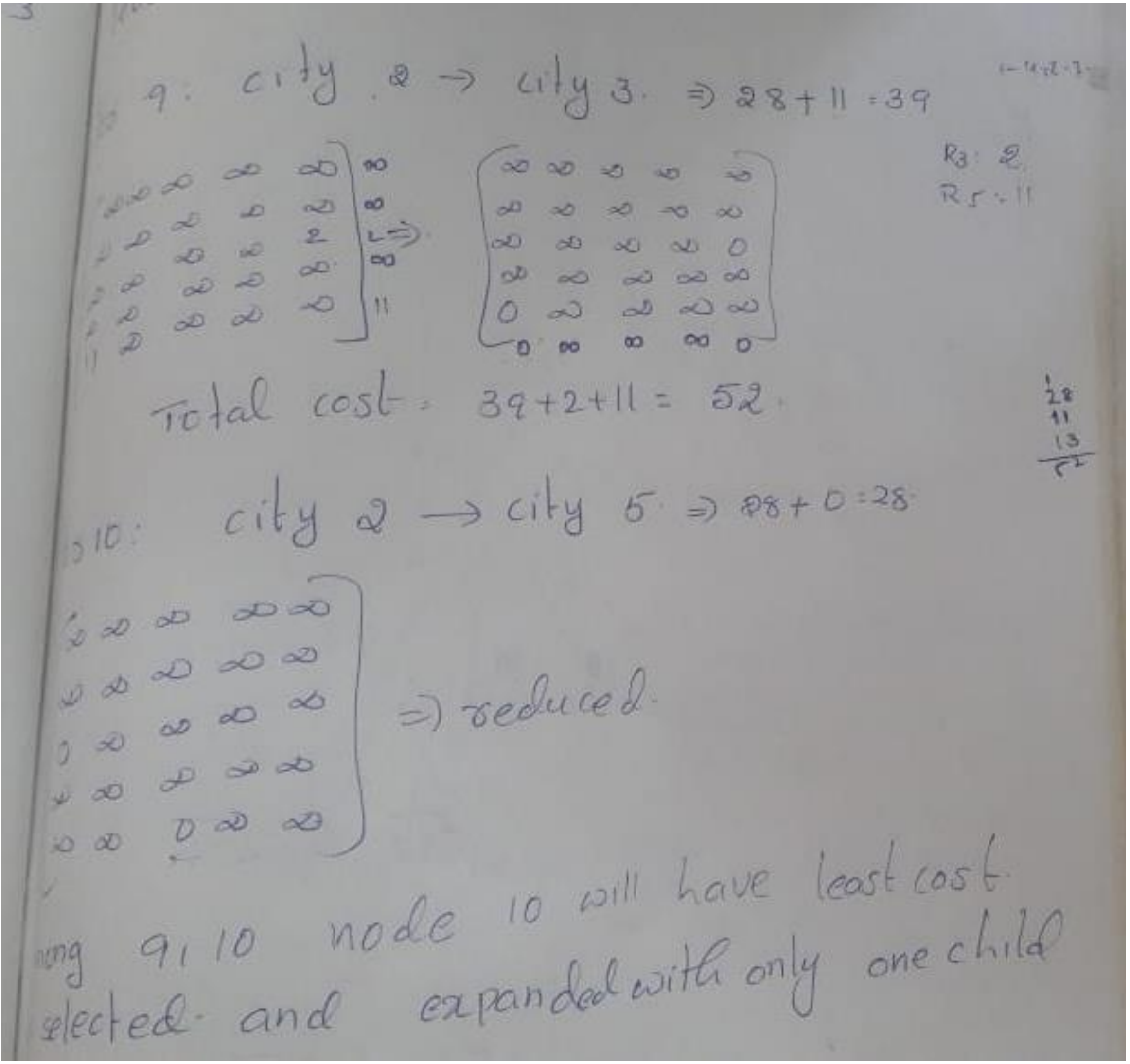


An Example for TSP with LCBB





An Example for TSP with LCBB



9: city 2 → city 3 ⇒ 28 + 11 = 39

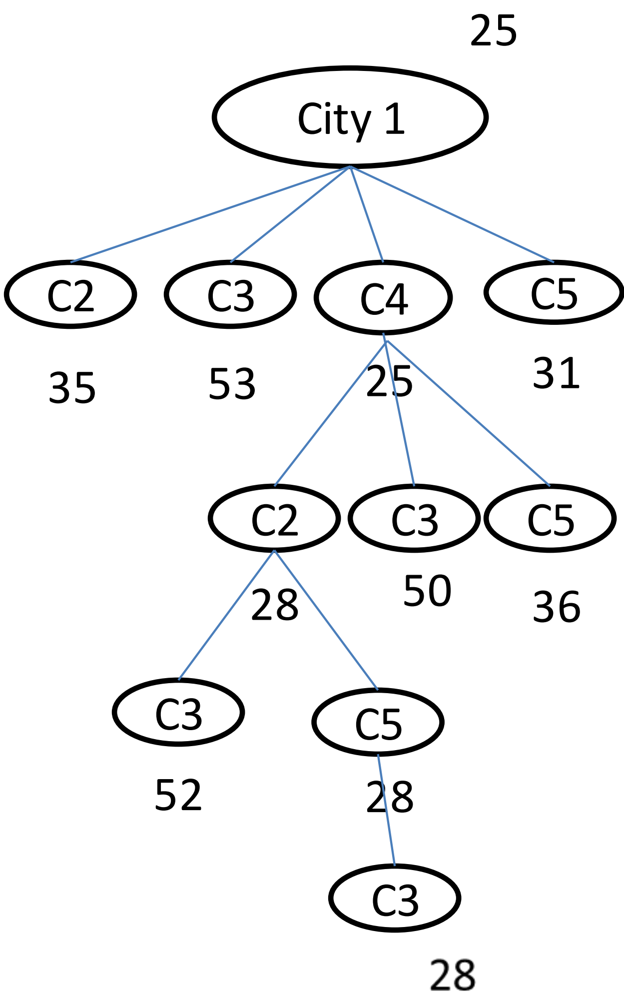
R3: 2
R5: 11

Total cost: 39 + 2 + 11 = 52

10: city 2 → city 5 ⇒ 28 + 0 = 28

⇒ reduced

among 9, 10 node 10 will have least cost
selected and expanded with only one child

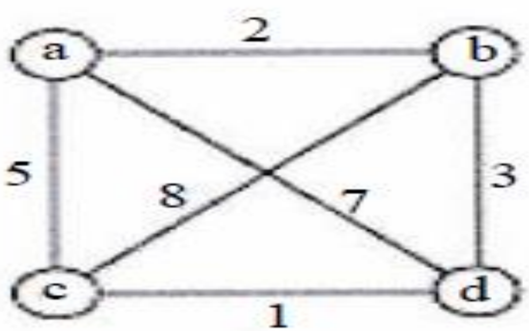


Problems for Practice

Find optimal tour of travelling salesperson for the following cost matrix using LCBB. **15M**

∞	7	3	12	8
3	∞	6	14	9
5	8	∞	6	18
9	3	5	∞	11
18	1	9	8	∞

Solve the following Travelling Sales Person Problem using branch and bound technique and draw the solution state space tree. **15M**



Topics

- **General Method**
- **Travelling Sales Person Problem**
- **0/1 Knapsack Problem**

0/1 Knapsack Problem

we cannot directly apply the techniques of LCBB since these were discussed with respect to minimization problems whereas the knapsack problem is a maximization problem. This difficulty is easily overcome by replacing the objective function $\sum p_i x_i$ by the function $-\sum p_i x_i$. Clearly, $\sum p_i x_i$ is maximized iff $-\sum p_i x_i$ is minimized. This modified knapsack problem is stated as:

$$\text{minimize} \quad -\sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq m$$

$$x_i = 0 \text{ or } 1, \quad 1 \leq i \leq n$$

Upper bound function

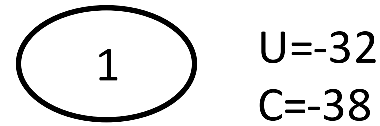
```
Algorithm UBound(cp, cw, k, m)
// cp, cw, k, and m have the same meanings as in
// Algorithm 7.11. w[i] and p[i] are respectively
// the weight and profit of the ith object.
{
    b := cp; c := cw;
    for i := k + 1 to n do
    {
        if (c + w[i] ≤ m) then
        {
            c := c + w[i]; b := b − p[i];
        }
    }
    return b;
}
```

An example for 0/1 Knapsack Problem with LCBB

- Consider the knapsack instance n = 4, (P1,P2, P3,P4) = (10,10,12,18),(W1,W2,W3,W4) = (2, 4, 6,9), and m = 15.

An example for 0/1 Knapsack Problem with LCBB

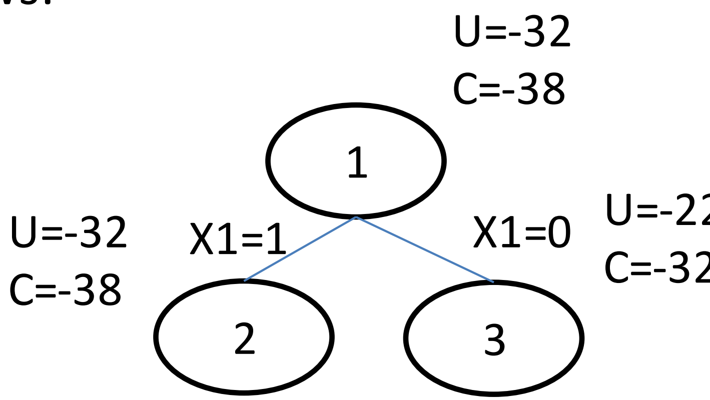
- The computationof u(1) and c(1)is done as follows: The bound u(1) has a value Ubound (0,0,0,15).
- Ubound(0,0,0,15) returns -32.



Function Bound is similar to UBound, except that it also considers a fraction of the first object that doesn't fit the knapsack. For example, in computing $\hat{c}(1)$, the first object that doesn't fit is 4 whose weight is 9. The total weight of the objects 1, 2, and 3 is 12. So, Bound considers a fraction $\frac{3}{9}$ of the object 4 and hence returns $-32 - \frac{3}{9} * 18 = -38$.

An example for 0/1 Knapsack Problem with LCBB

- The computation of U(2) and C(2)is done as follows:
  The bound U(2) has a value Ubound (-10,2,1,15).
- Ubound(-10,2,1,15) returns -32.
- C(2) = -32-18*$\frac{3}{9}$ = -38

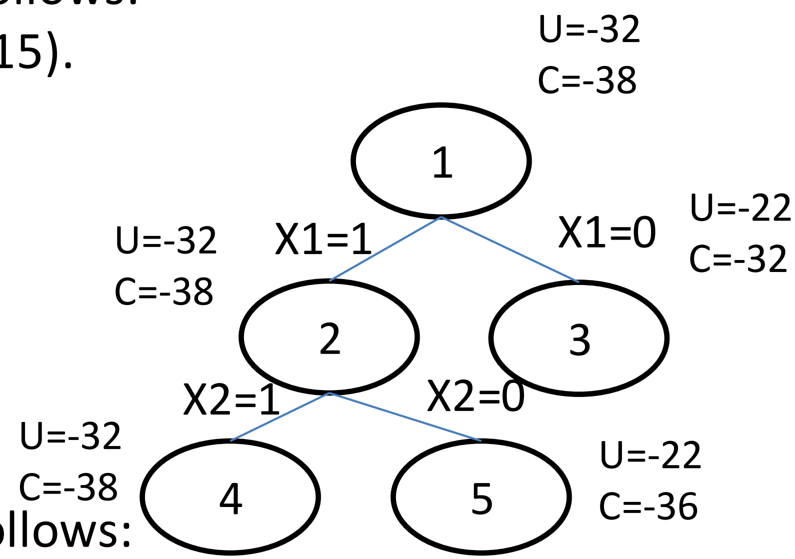


- The computation of U(3) and c(3)is done as follows:
  The bound U(3) has a value Ubound (0,0,1,15).
- Ubound(0,0,1,15) returns -22.
- C(3) = -22-18*$\frac{5}{9}$ = -32

- Because of node 2 is having lease cost expand it to the next level.

An example for 0/1 Knapsack Problem with LCBB

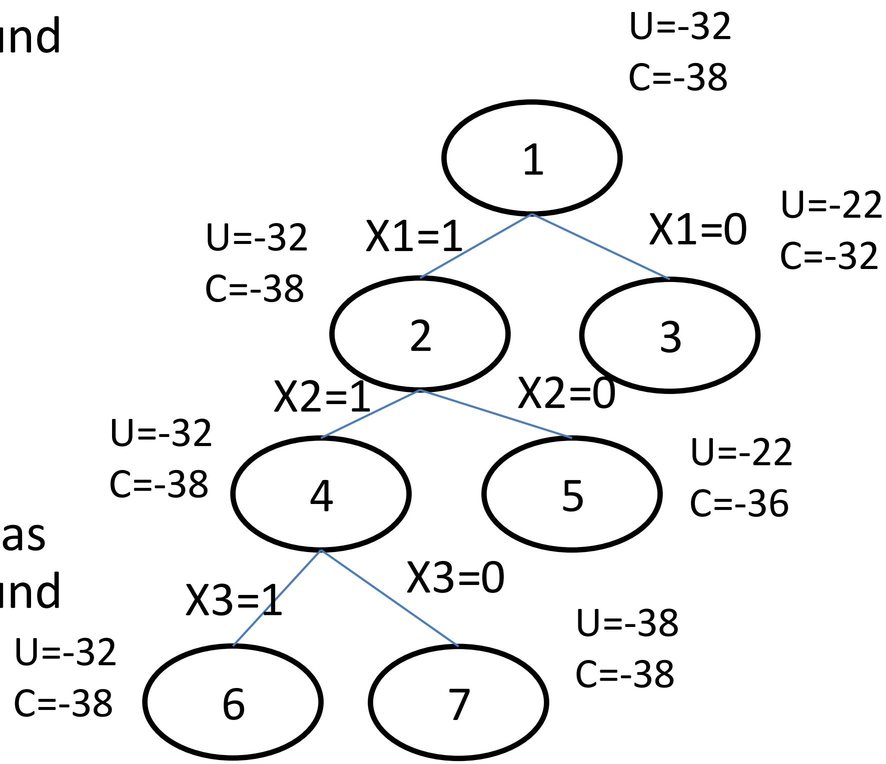
- The computation of U(4) and C(4)is done as follows: The bound U(4) has a value Ubound (-20,6,2,15).
- Ubound(-20,6,2,15) returns -32.
- $C(4) = -32-18*\frac{3}{9} = -38$



- The computation of U(5) and c(5)is done as follows: The bound U(5) has a value Ubound (-10,2,2,15).
- Ubound(-10,2,2,15) returns -22.
- $C(5) = -22-18*\frac{7}{9} = -36$
- Because of node 4 is having lease cost expand it to the next level.

An example for 0/1 Knapsack Problem with LCBB

- The computation of U(6) and C(6)is done as follows: The bound U(6) has a value Ubound (-32,12,3,15).
- Ubound (-32,12,3,15) returns -32.
- C(6) = $-32-18*\frac{3}{9} = -38$



- The computation of U(7) and C(7)is done as follows: The bound U(7) has a value Ubound (-20,6,3,15).
- Ubound(-20,6,3,15) returns -38.
- C(3) = -38-0 = -38

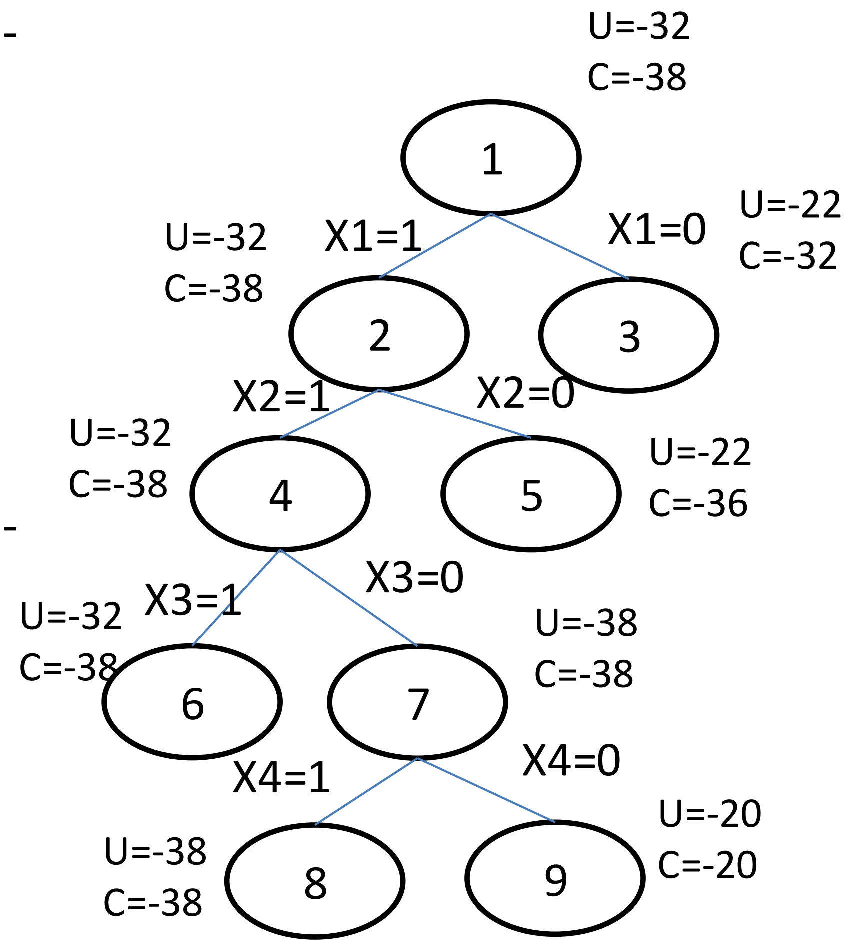
- Because of node 7 is having lease cost expand it to the next level.

An example for 0/1 Knapsack Problem with LCBB

- The computation of U(8) and C(8)is done as follows: The bound U(8) has a value Ubound (-38,15,4,15).
- Ubound (-38,15,4,15) returns -38.
- C(6) = -38-0=-38

- The computation of U(9) and C(9)is done as follows: The bound U(9) has a value Ubound (-20,6,4,15).
- Ubound(-20,6,4,15) returns -20.
- C(3) = -20-0 = -20

- Because of node 8 is having lease cost and all the object are over. Hence node 8 is the Answer Node.
- The solution is (1,1,0,1) with total profit of 38.



Problem for practice

- Consider the knapsack instance n = 5, (P1,P2, P3,P4,P5) = (10,15,6,8,4),(W1,W2,W3,W4,W5) = (4,6,3,4,2), and m = 12.

- Consider the knapsack instance n = 5, (P1,P2, P3,P4,P5) = (12,10,5,9,3),(W1,W2,W3,W4,W5) = (3,5,2,5,3), and m = 12.

Topics

- **General Method**
- **Travelling Sales Person Problem**
- **0/1 Knapsack Problem**