Design and Analysis of Algorithms

L37: LC Branch and Bound

0-1 Knapsack Problem

Dr. Ram P Rustagi
Sem IV (2020-Even)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Resources

- Text book 2: Horowitz
  – Sec 8.2
- Text book 1: Levitin
  – Sec 12.1, 12.2
- R1: Introduction to Algorithms
  - Cormen et al.
- Youtube link for lecture recording
  – https://www.youtube.com/watch?v=j556E7LgvbI
- Youtube (other)
  – https://www.youtube.com/watch?v=yV1d-b_NeK8

BB Search: State Space Tree

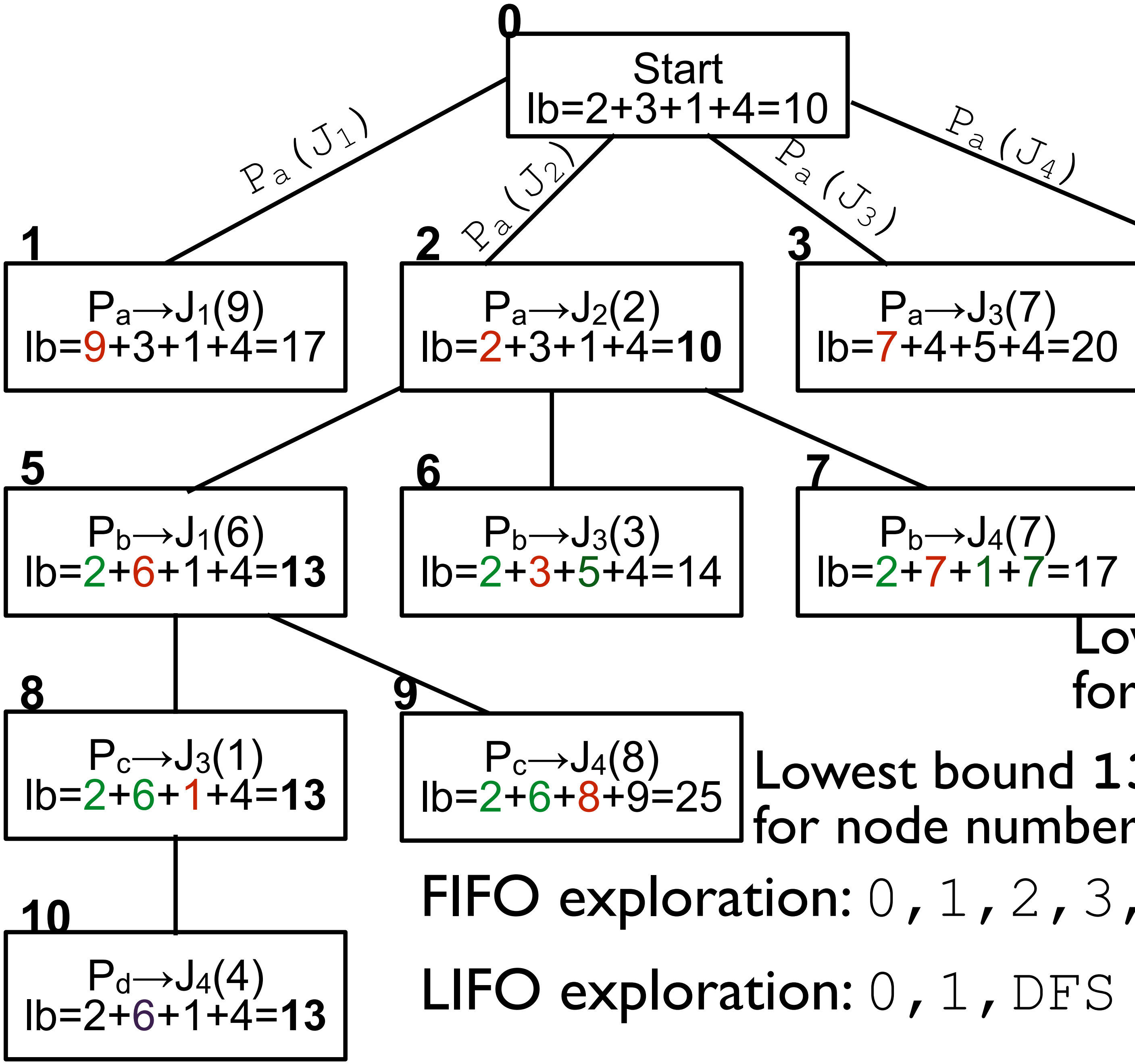
```
Algo BBSearch(node t) // search tree with root at t.
if t is an answer node
    output t and return
E←t // make t an E-node
Initialize the list L (of live nodes) to be empty.
do
    for each child x of E
        if x is an answer node
            output the path from x to t and return
        Add(x)  to list L of live nodes
        parent(x)←E
    if L is empty // there are no more live nodes
        output "No answer nodes" and return
    E←Next(L) // take the next live node from to search
while True
```

BB Search: State Space Tree

- Three possible implementation of search space
  - Depends upon how the list `L` is implemented
  - and how the `Next(L)` is taken out
- L is Queue i.e. FIFO (First In First Out)
  - E-nodes are removed in the order they are added
  - Also called BFS (Breadth First search)
- L is Stack i.e. LIFO (Last in First Out)
  - E-nodes are removed in the reverse order it is added
  - Also called D-search (Depth First search)
- L is Heap (can be min or max heap)
  - E-nodes are removed as min (or max) value
  - Called Least Cost (LC) Search

Ex: Assignment Problem

	J_1	J_2	J_3	J_4
P_a	9	**2**	7	8
P_b	**6**	4	3	7
P_c	5	8	**1**	8
P_d	7	6	9	**4**



Lowest bound 10 for node number 2.

Lowest bound 13 for node number 5.

Lowest bound 13 for node number 8.

FIFO exploration: 0, 1, 2, 3, 4, 5, 6...

LIFO exploration: 0, 1, DFS of 1, 2, 5, 8, 10

LC Search: State Space Tree

```
Algo LCSearch(node t) // search tree with root at t.
if t is an answer node
    output t and return
E←t // make t an E-node
Initialize the list L (of live nodes) to be empty.
do
    for each child x of E
        if x is an answer node
            output the path from x to t and return
        Add(x)  to list L of live nodes
        parent(x)←E
    if L is empty // there are no more live nodes
        output "No answer nodes" and return
    E←Least(L) // take the next live node from to search
while True
```

Bound Implementation

- Consider a tree `t` rooted at node `x`.
- Let `c(x)` denotes the cost of minimum cost answer node
  - Typically, this cost is equal to objective function value, e.g.
    - Min cost in job assignment problem, TSP
  - Computing `c(x)` is the main task to find the answer node
- When exploring state space Tree using LCBB
  - We pick the node with least cost.
  - Finding (computing) it requires exploring the entire tree
    - Leads to exponential cost
  - Thus, we would try to estimate c(x)
- Define $\hat{c}(x)$: a heuristic (estimate) value for $c(x)$
  - Use this $\hat{c}(x)$ in picking the next least cost E-node.
  - Thus it must follow $\hat{c}(x) \leq c(x)$
  - Otherwise, we are exploring a path with cost $> c(x)$
    - Reach an answer node which is not min cost

Bound Implementation

- Defining $\hat{c}(x)$: a heuristic value for $c(x)$
  - Provides a lower bound on solutions obtainable
    $\hat{c}(x) \leq c(x)$
- Let `upper`: upper bound on cost of min cost solution
  - Then, consider all live nodes with $\hat{c}(x) > upper$
  - These live nodes can be killed, as answer node reachable from these live nodes will have
    $c(x) \geq \hat{c}(x) > upper$
  - Initial value of upper should be more than $c(x)$
    - May even be set as ∞
- Thus, as long as value of `upper` is $\geq c(x)$
  - Killing of live nodes will not cause killing a node that can reach min cost answer node.

0-1 Knapsack Problem

- Knapsack problem:
  - Given n items of known weights $w_1, \ldots, w_n$, and
  - Values $v_1, \ldots, v_n$ and knapsack capacity m
  - Find the most valuable subset of items that fit into the knapsack.
    - i.e.maximize the value of knapsack
    - An item has to be included in full (0-1 knapsack problem)
  - Note: All the weights w_i's and knapsack capacity m are integers, but values v_i's can be real numbers.
- 0-1 knapsack is a maximization problem
  - Branch and Bound solves minimization problem.
  - So convert knapsack to minimization problem

$0-1$ Knapsack Problem

- $0-1$ Knapsack problem (maximization problem)
  - Maximize $\Sigma_{1\leq i\leq n}\ v_i x_i$,
  - Subject to $\Sigma_{1\leq i\leq n}\ w_i x_i \leq m$
  - x_i is 0 or 1, and $1\leq i\leq n$
- Problems of TSP and Job Assignment were minimization problem solved using Branch-n-Bound
- Convert knapsack maximization to minimization

  Minimize $-\Sigma_{1\leq i\leq n}\ v_i x_i$, (call it cost)
    - It maximizes $\Sigma_{1\leq i\leq n}\ v_i x_i$ (values)

  subject to $\Sigma_{1\leq i\leq n}\ w_i x_i \leq m$ (knapsack constraint)
- State space tree formation
  - Using fixed tuple size, one variable for each weight
  - Using variable tuple size, uses the index of weight

$0-1$ Knapsack Problem

- State space tree formation
  - Using fixed tuple size, one variable for each weight
    - Each variable has two values 0 or 1
    - Thus, Each node has two children
  - Using variable tuple size, uses the index of weight
    - Can be easily built from fixed tuple size case
- Implementation: define two terms:
    - Cost per node (what can be reached theoretically)
    - Upper bound per node (what can be achieved)
  - Define $c(x)=-\sum_{1\le i\le n} v_i x_i$ for each answer node x
    - $c(x)=\infty$ for infeasible leaf nodes
  - For non-leaf nodes, define $c(x)$ recursively as
    - $\min\{c(Lchild(x)),\ c(Rchild(x))\}$
    - Thus, computation recursively becomes exponential

Example: LCBB $0-1$ Knapsack

- Consider knapsack instance with $n=4, m=15$, and
  - Values $(v_1, v_2, v_3, v_4)=(10, 10, 12, 18)$, and
  - Weights $(w_1, w_2, w_3, w_4)=(2, 4, 6, 9)$
  - v_i/w_i ratios are $5>2.5>2\geq 2$
- Using fixed tuple implementation, trace LCBB
  - Fixed implementation implies 4 tuple variables
    - x_1, x_2, x_3, x_4 and each can take value 0 or 1.
- We need to compute following values for each node
  $\hat{c}(x)$, $u(x)$, upper
- Consider root node i.e. start node at level 1.
  - Least Cost (LC) approach
    - Among all live nodes, choose the node with lowest cost to explore (i.e. it becomes E-node)
    - List L of live nodes is implemented as Heap

LCBB: $0-1$ Knapsack

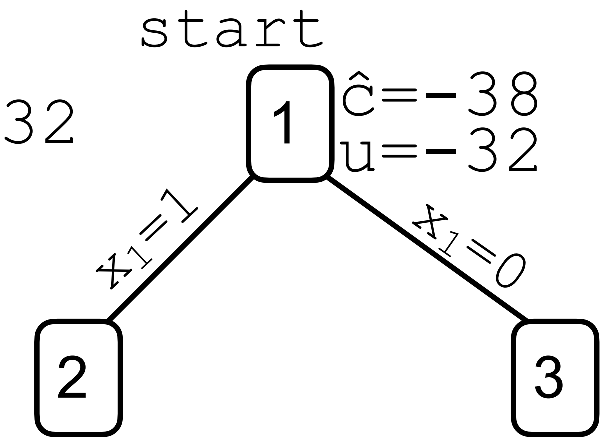
- Assume items are sorted in the non-increasing order of max profits per unit of weight i.e.
  $p[i]/w[i] \geq p[i+1]/w[i+1]$, $1 \leq i < n$
- Defining $\hat{c}(x)$ for $0-1$ knapsack
  - Cost till the first node which doesn't fit the knapsack
    - But including its partial value to max the knapsack
- Define $u(x)$: an upper bound for node x.
  - Cost till the first node which doesn't fit the knapsack,
    - But excluding the partial value.
- Thus, for node x, we have $\hat{c}(x) \leq c(x) \leq u(x)$
  - #note cost is $-ve$, removing partial value makes it more
- Maintain single `upper` variable.
  - This indicates minimum cost solution achieved so far.
- Thus, for any node when $\hat{c}(x) > upper$
  - Discard that path (i.e. kill that node), prune the tree

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4, m=15

upper=-32



- **Start with node 1:**

  $\hat{c}(x)$: w_1, w_2, and w_3 contributes fully, w_4 exceeds knapsack

  $\hat{c}=-(10+10+12+((15-12)/9)*18))=-38$

  $u(x)$: w_1, w_2, and w_3 contributes fully, w_4 exceeds knapsack

  $u=-(10+10+12+0)=-32]$

  upper=-32

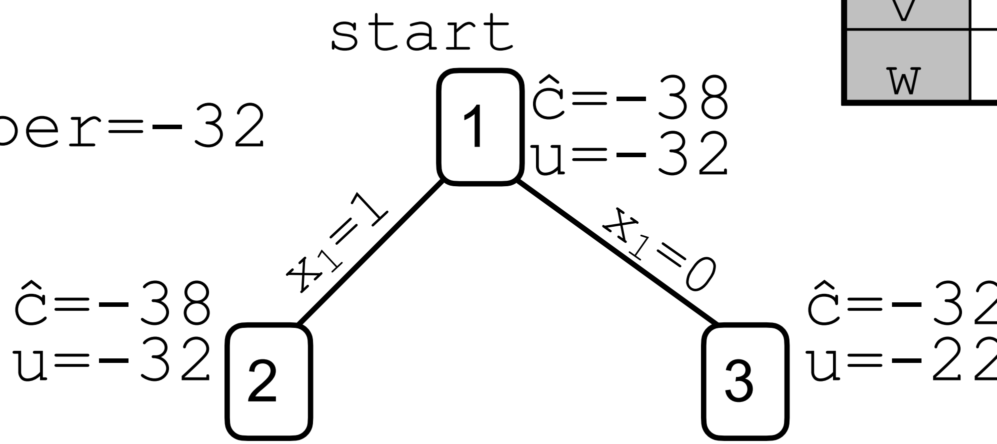
  - This node is live node ($\hat{c}\leq$upper) and only node so far,
  - Explore this node, two children
    - $x_1=1$ (include w_1), $x_1=0$ (exclude w_1)

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4, m=15

upper=-32

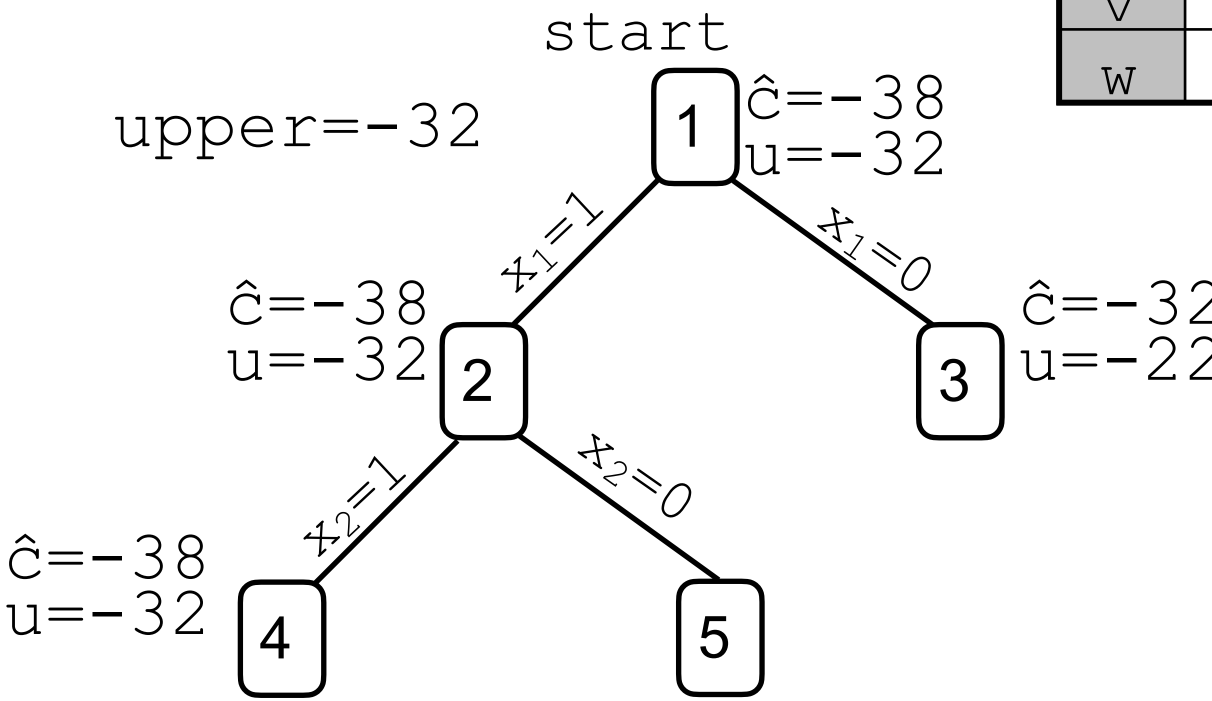


- **node** 2: $x_1=1$
  - $\hat{c}(x)$: w_1, w_2, and w_3 contributes fully, w_4 exceeds knapsack
    - $\hat{c}=-(10+10+12+((15-12)/9)*18))=-38$
  - $u(x)$: w_1, w_2, and w_3 contributes fully, w_4 exceeds knapsack
    - $u=-(10+10+12+0)=-32$
    - $\hat{c}(x)$, $u(x)$, upper don't change
- **node** 3: $x_1=0$ (partial weight of w_4 becomes 5)
  - $\hat{c}=-(0+10+12+((15-10)/9)*18))=-32$
  - $u=-(0+10+12+0)=-22$
  - upper remains −32 (doesn't change)
- Alives nodes are: 2 and 3 ($\hat{c}(x) \leq$ upper)

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4,m=15



- Least Cost ĉ among live nodes(2, 3) is −38 for node 2.
- Explore node 2 (2 becomes E-node, 3 remains live node)
  - $x_2=1$ (node 4), and $x_2=0$ (node 5)
- Node 4:

  ĉ=-(10+10+12+((15-12)/9)*18))=-38

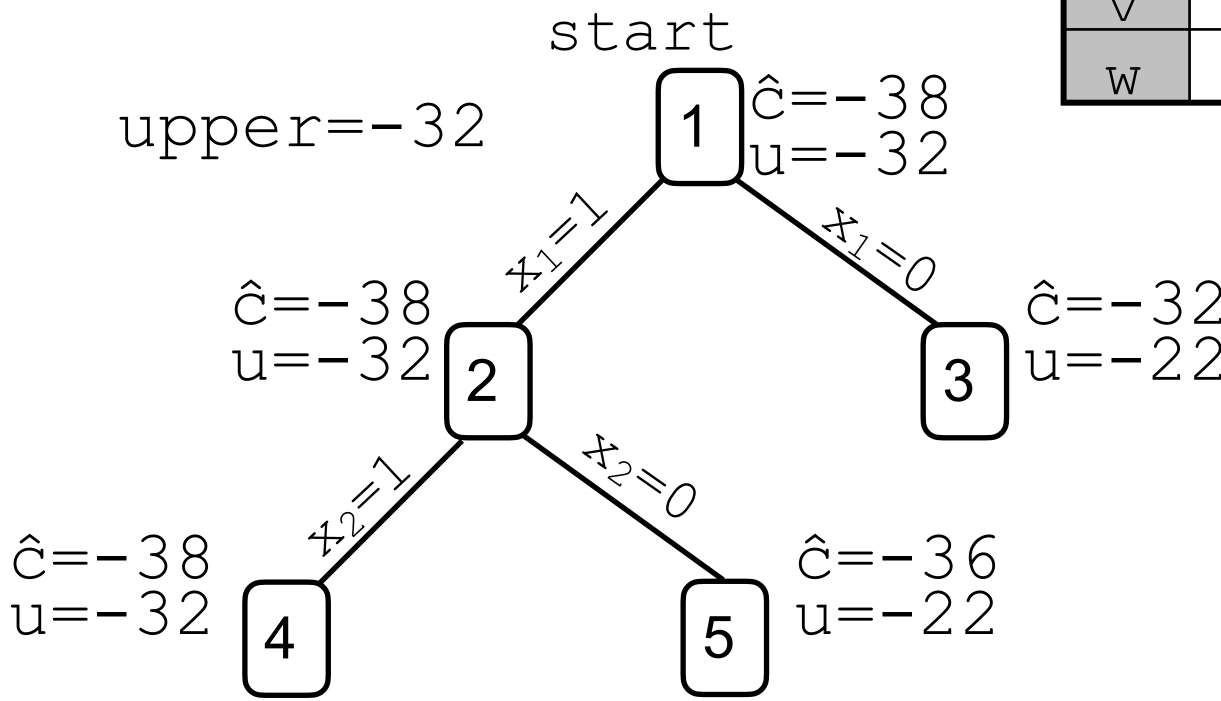
  u=-(10+10+12+0)=-32

  upper remains same and doesn't change

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4,m=15

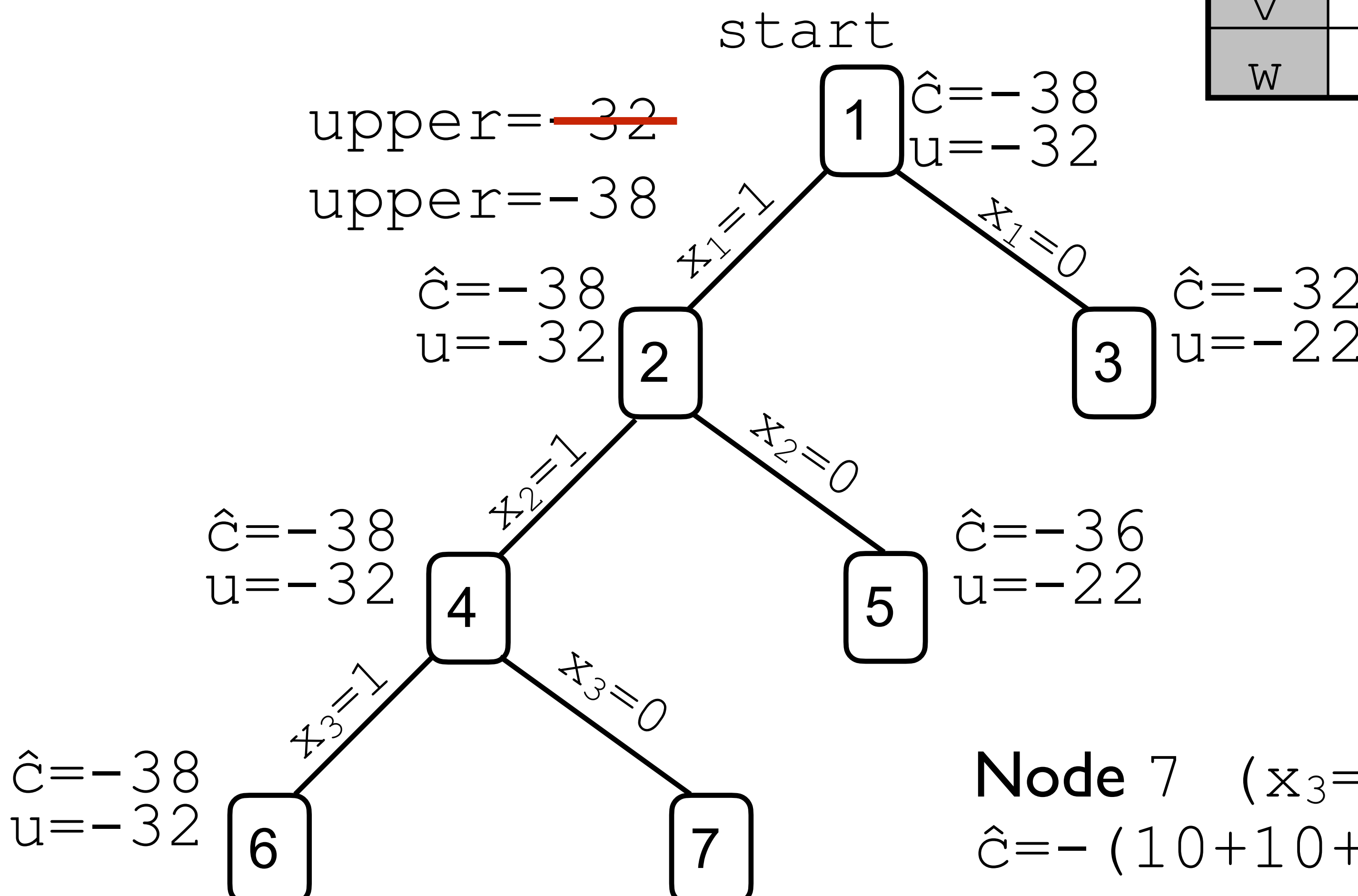


- Node 5 (partial weight of w_4 changes)
  ĉ=-(10+0+12+((15-8)/9)*18))=-36
  u=-(10+0+12+0)=-22
  upper remains same and doesn't change
- Lives nodes now: 3, 4, 5 (ĉ≤upper)
- Least Cost node is 4, ĉ=-38, explore it
  - x_3=1 (node 6), and x_3=0(node 7)

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4,m=15



Node 7 (x_3=0)

ĉ=-(10+10+0+18))=-38

u=-(10+10+0+18)=-38

upper **becomes less and hence changes to** −38

Node 6 (x_3=1)

ĉ=-(10+10+12+((15-12)/9)*18))
=-38

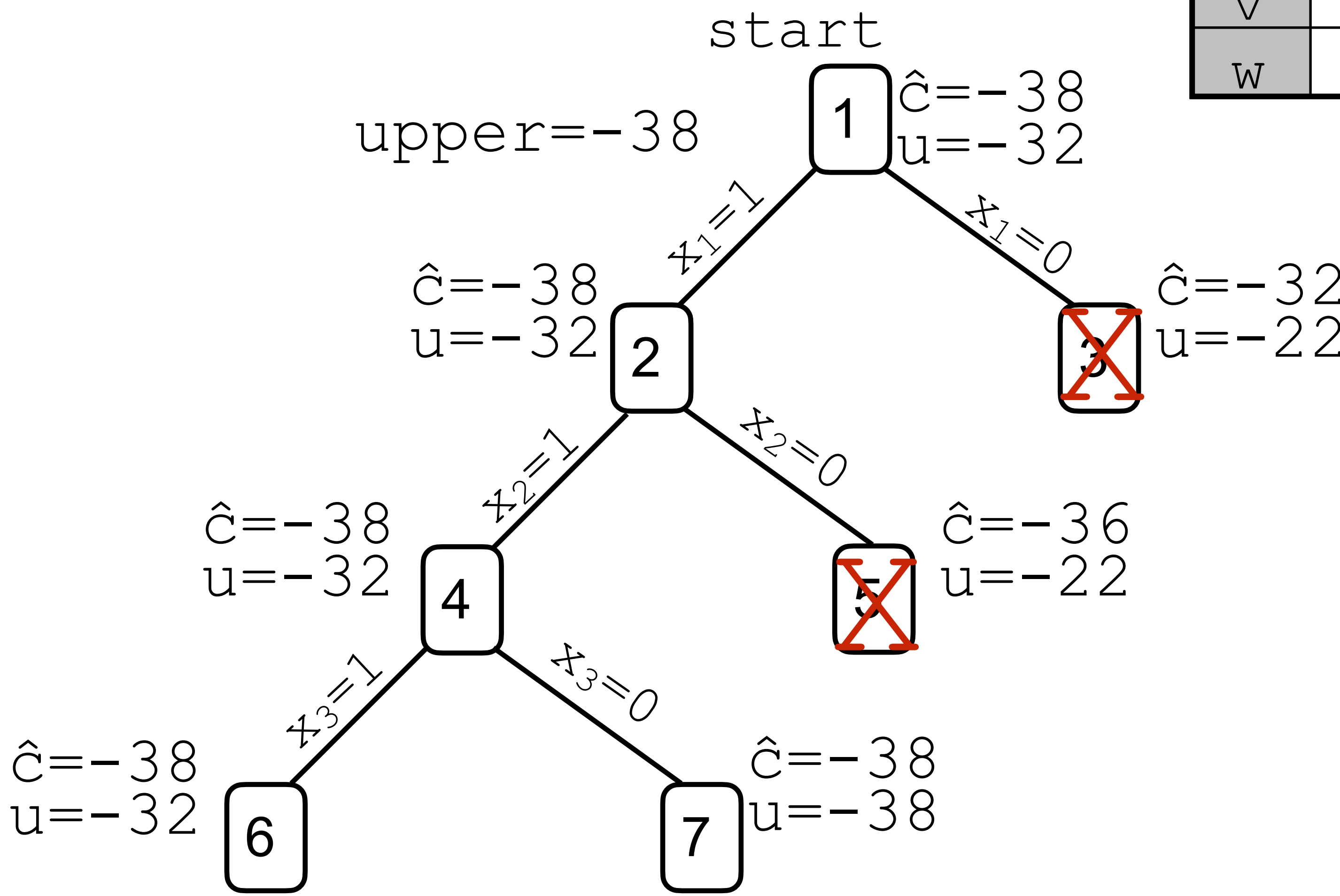
u=-(10+10+12+0)=-32

upper **remains same, doesn't change**

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4, m=15

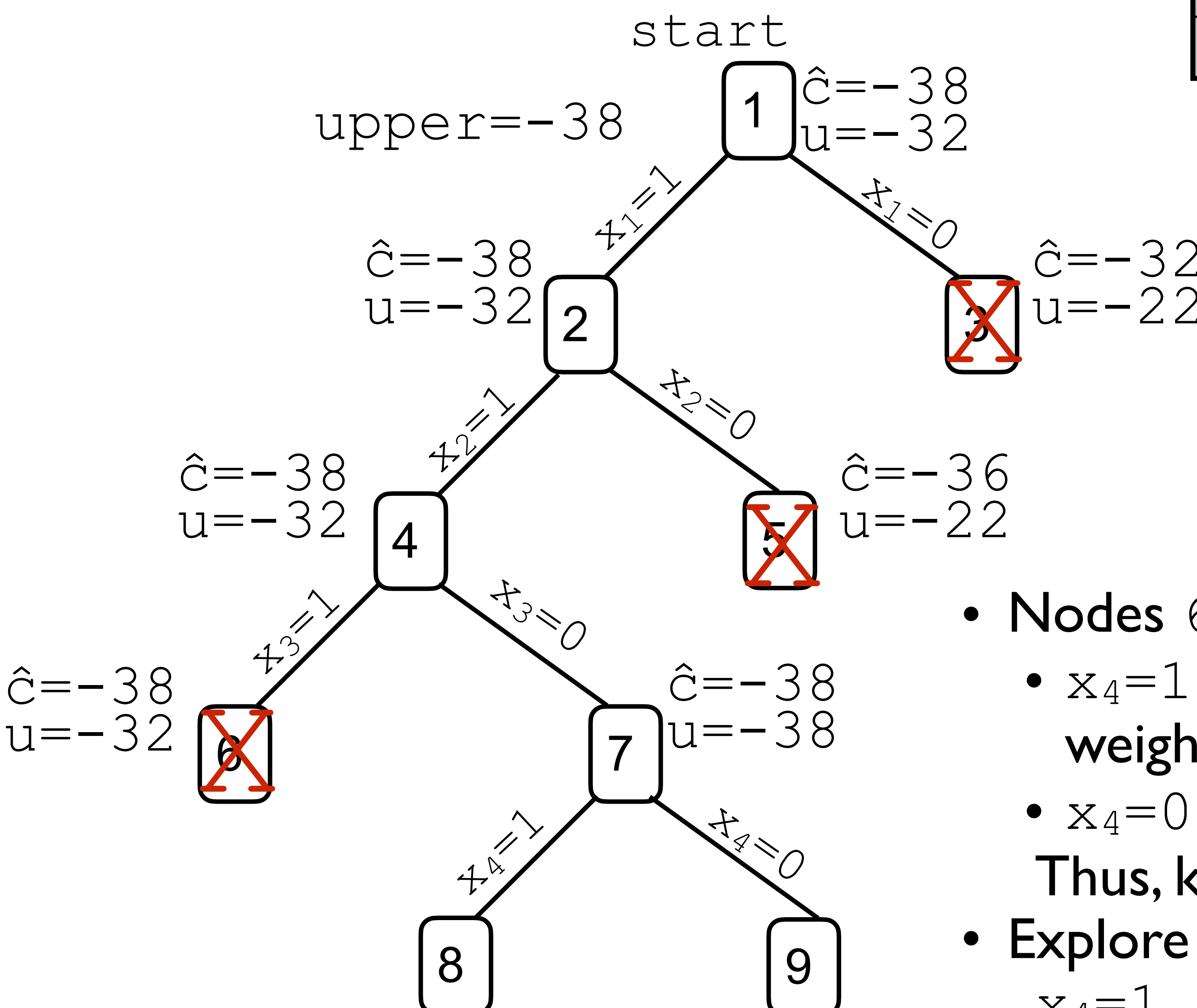


- Nodes 3 and 5 are killed, $\hat{c}(3)>upper, \hat{c}(5)>upper$
- Live nodes are 6 and 7. ($\hat{c}(6) \leq -38, \hat{c}(7) \leq -38$)
- Next Least Cost live node: can be taken either 6 or 7,
- Take 6 as least cost node. (both 6 & 7 are equal.)

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4, m=15



- Nodes 6
  - $x_4=1$ makes knapsack weight 21, can't consider
  - $x_4=0$ increases ĉ to −32

  Thus, kill the node 6.
- Explore node 7

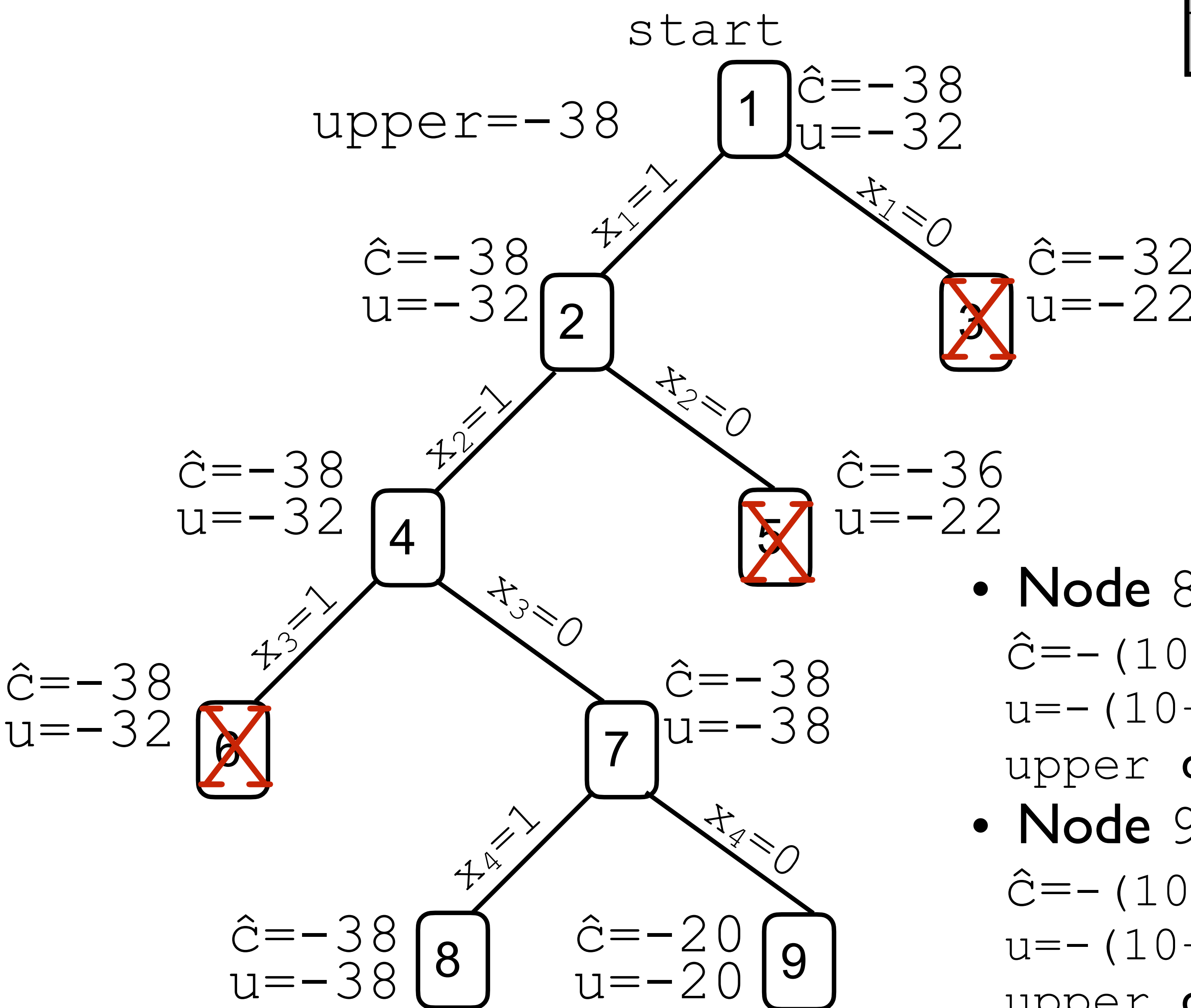
  $x_4=1$ (node 8),

  $x_4=0$ (node 9)

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4, m=15



- **Node** 8

  ĉ=-(10+10+0+18)=-38

  u=-(10+10+0+18)=-38

  upper **does not change**
- **Node** 9

  ĉ=-(10+10+0+0)=-20

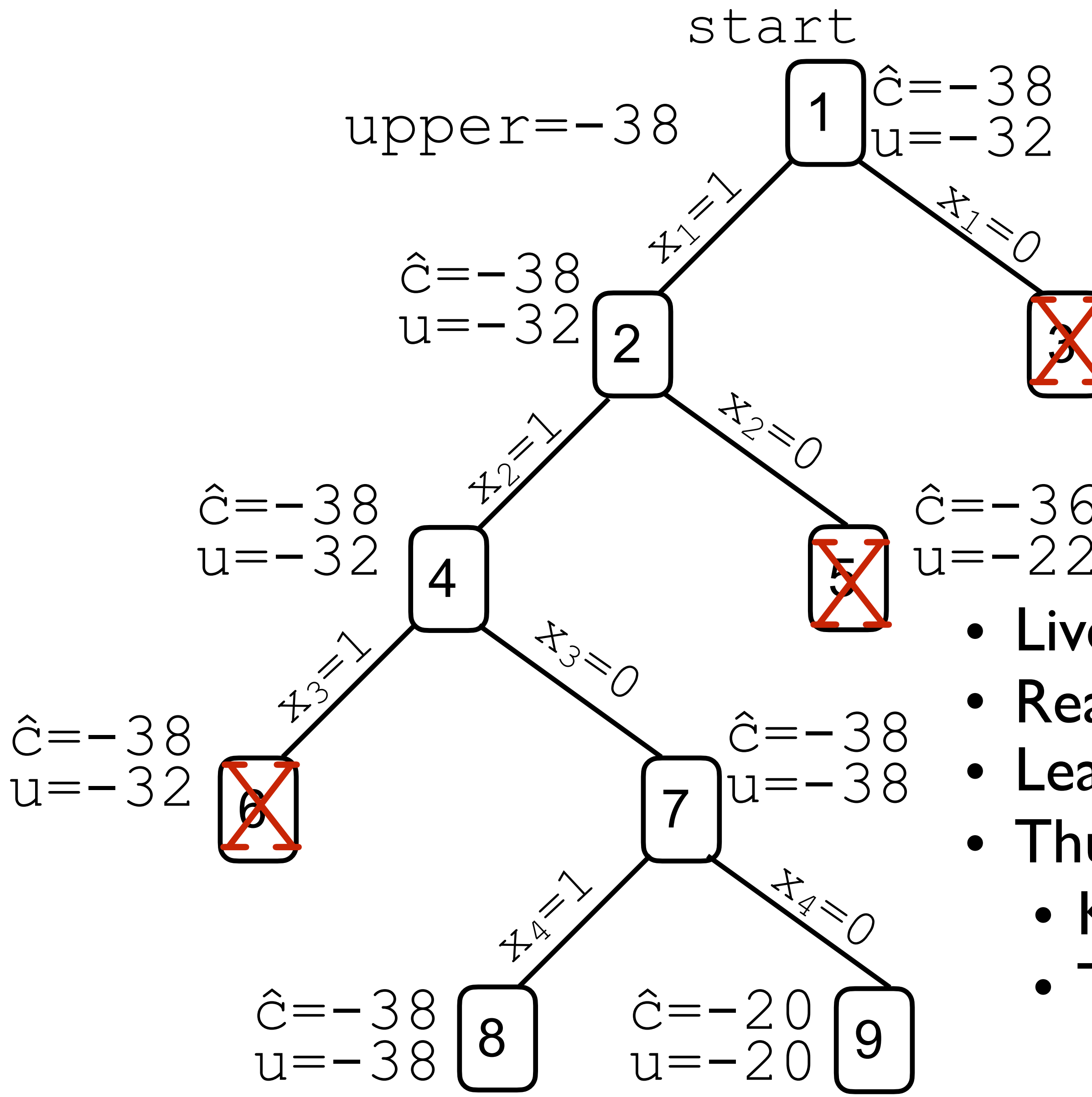
  u=-(10+10+0+0)=-20

  upper **does not change**

LCBB: 0-1 Knapsack

	1	2	3	4
v	10	10	12	18
w	2	4	6	9

n=4, m=15



- Live Nodes 8, 9
- Reached the leaf nodes
- Least Cost: node 8
- Thus, answer node is 8
  - Knapsack value=38
  - Tuple=(1,1,0,1)

`0-1` Knapsack Implementation

- State space tree is a binary tree with depth `n+1`
  - Define two functions
    - `Bound()` to compute $\hat{c}$
      - used to compute estimated cost
    - `UBound()` to compute `u`
      - used to compute upper value

$$u(x) = \mathrm{UBound}(-\Sigma_{1\le i<j} v_i x_i, \Sigma_{1\le i<j} w_i x_i, j-1, m)$$

$$c(x) \ge \mathrm{Bound}(\Sigma_{1\le i<j} v_i x_i, \Sigma_{1\le i<j} w_i x_i, j-1)$$

Bound()

```
Proc Bound(float cv, float cw, int k)
```

// provides an upper bound (partial knapsack) on best solution obtainable (by expanding any node `Z` at level `k+1`)

// includes the partial value of node which exceeds knapsack

//`cp`: current total value, `cw`: current total weight

// `k` is the index of last removed item of knapsack

```
    float b=cp;  float c=cw;
    for i←k+1 to n do ..............................................B1
        if (c+wi<m) then ...........................................B2
            c=c+wi ..................................................B2
            b = b - vi ..............................................B4
        else // include cost for partial weight
            return (b-(m-c)/wi)*vi ..................................B5
    return b ........................................................B6
```

UBound()

```
Proc UBound(float cv, float cw, int k,float m)
```

// provides an upper bound (0−1 knapsack) on best solution obtainable by expanding any node Z at level k+1

// does not include the cost last node that exceeds knapsack

//cp: current total value, cw: current total weight

// k is the index of last removed item of knapsack

```
float b=cp;
float c=cw;
for i←k+1 to n do .......................................U1
    if (c+w_i≤m) then ...................................U2
        c = c+w_i .......................................U3
        b = b-v_i .......................................U4
return b ................................................U5
```

Summary:

- Least Cost Branch and Bound for
  - `0-1` Knapsack problem
- Next to explore
  - FIFO Branch and Bound