算法设计与分析

Lecture 13: Branch-and-Bound

卢杨
厦门大学信息学院计算机科学系
luyang@xmu.edu.cn

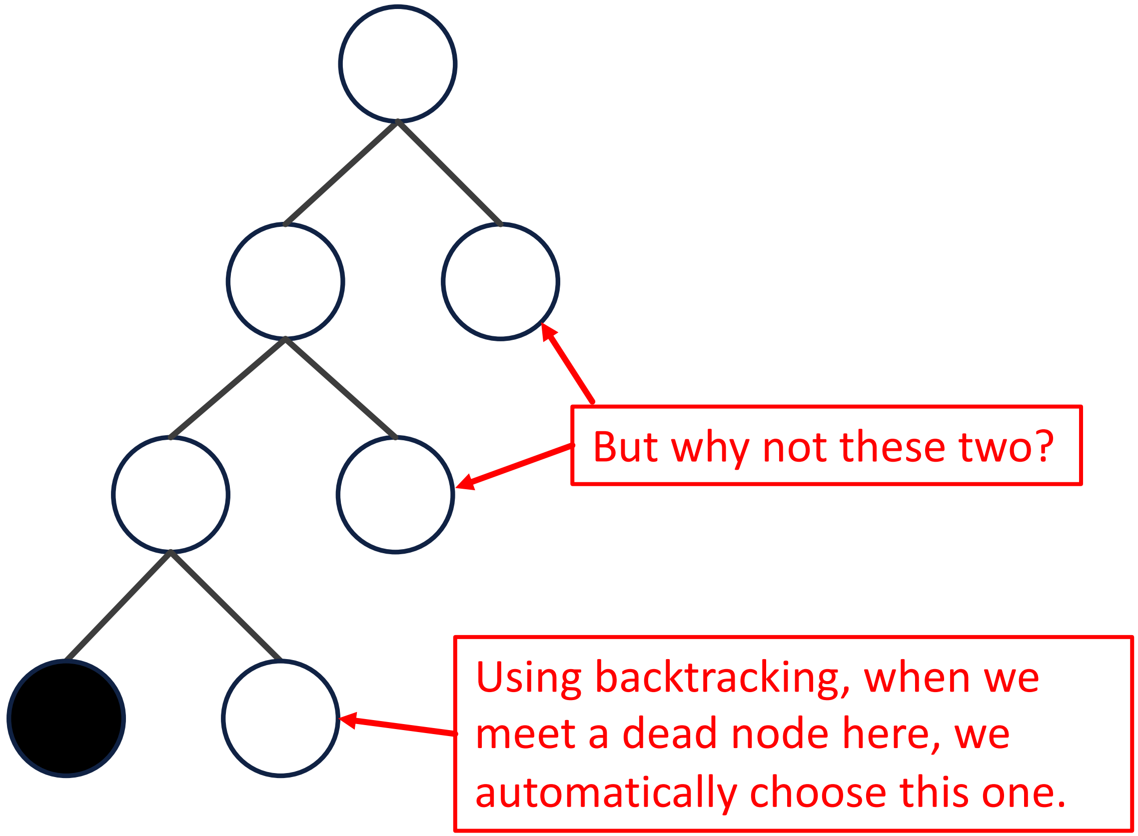
Limitation of Backtracking

- Backtracking works better if we can improve over the bounding function.
- However, there is still a mechanism that limits backtracking to be more efficient:

DFS

- No matter how you improve the bounding function, the traversal is still based on DFS.
  - Can we based on other methods to explore the solution space?

Limitation of Backtracking



Limitation of Backtracking

- Can we try BFS?
- No, it is very inefficient.
  - No solution is reached until level 1 to level $n-1$ of the tree is built.
  - No solution means bounding function is useless.
- Branch-and-bound (分支限界) is the techniques to improve BFS for solution space tree traversal.
  - FIFO branch-and-bound.
  - Max-profit branch-and-bound.

Branch-and-Bound

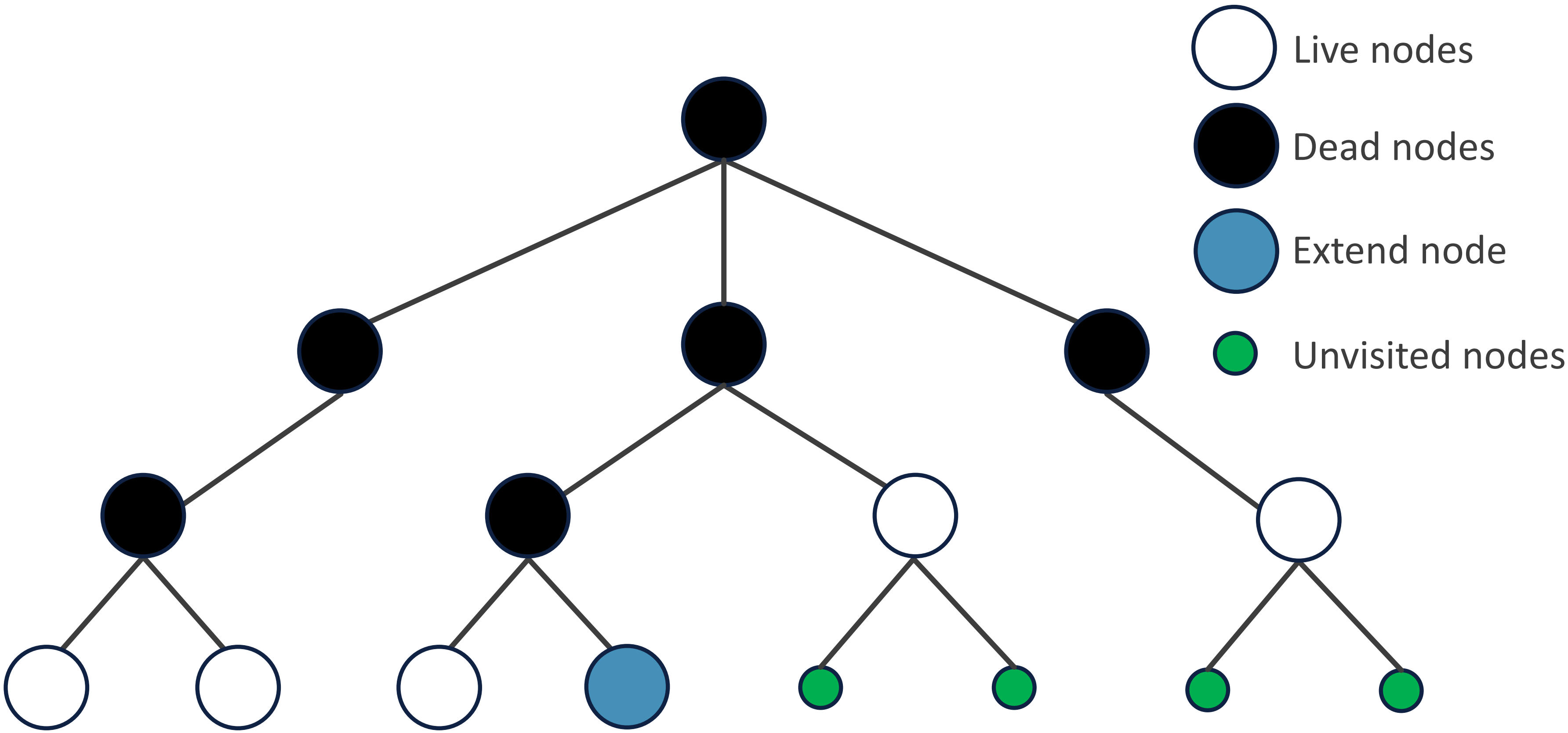
- Different from backtracking, the branch-and-bound method
  1. does not limit us to any particular way of traversing the tree;
  2. is used only for optimization problems.
- A branch-and-bound algorithm computes a upper bound and lower bound at a node.
- For maximization problem:
  - Upper bound is calculated by the bounding function.
  - Lower bound is recorded by the best solution so far.
- We increase the lower bound and decrease the upper bound until they are equal.

Branch-and-Bound

- Branch-and-bound is based on BFS, but does not exactly follow its FIFO machanism.
- We select the next node to branch based on some rule. Namely, we branch a node with the highest hope.
- All nodes can be separated into:
  - Live nodes: Visited but waiting for branching.
  - Dead nodes: Visited.
  - Extend node: Selected to branch in the next step.
  - Unvisited nodes: Unvisited.

Definition of live and dead nodes are slightly different from backtracking.

Branch-and-Bound



Extend node is selected among all live nodes based on designed rules.

CONTAINER LOADING PROBLEM

Container Loading Problem

- Given n containers (集装箱), container i has weight w_i. The ship can hold containers of total weight up to W.
- Container Loading problem is to load as many containers as is possible without sinking the ship.
- Assuming that the solutions are represented by vectors $(x_1, x_2, \dots, x_n)$, where $x_i \in \{0,1\}$. 1 denotes taking container i and 0 denotes not taking container i.
- The container loading problem can be formally stated as follows:

$$\max \sum_{i=1}^{n} w_i x_i \qquad s.t. \sum_{i=1}^{n} w_i x_i \le W$$

Container Loading Problem

In this example, we go though three versions of branch-and-bound.

- FIFO branch-and-bound with only constraint function.
- FIFO branch-and-bound.
- Max-profit branch-and-bound.

Container Loading Problem

- The constraint function is same as backtracking.
- Let $cw(i)$ denote the current weight up to level i, namely

$$cw(i) = \sum_{j=1}^{i} w_j x_j$$

then the constraint function is

$$C(i) = cw(i-1) + w_i$$

- The pruning condition is $C(i) > W$, which means there is no capacity to take container i.

FIFO with Only Constraint Function

Level i is used to check solution.

```
FIFOMaxLoading(w, W, n)
  i ← 1
  Enqueue(Q, −1)
  cw ← 0;  bestw ← 0
  while Q ≠ Ø do
    if C(i) ≤ W then
        SaveQueue(Q, C(i), bestw, i)
    SaveQueue(Q, cw, bestw, i)
    cw ← Dequeue(Q)
    if cw = −1 then
      if Q ≠ Ø then return bestw
        Enqueue(Q, −1)
        cw ← Dequeue(Q)
        i ← i + 1
 return bestw
```

```
SaveQueue(Q, wt, bestw, i)
 if i = n then
       if wt > bestw then
           bestw ← wt
 else
   Enqueue(Q, wt)
```

We insert -1 in the queue to show the separation between different levels.

Enqueue left and right child.

The current level is fully explored.

No live node to branch, terminate.

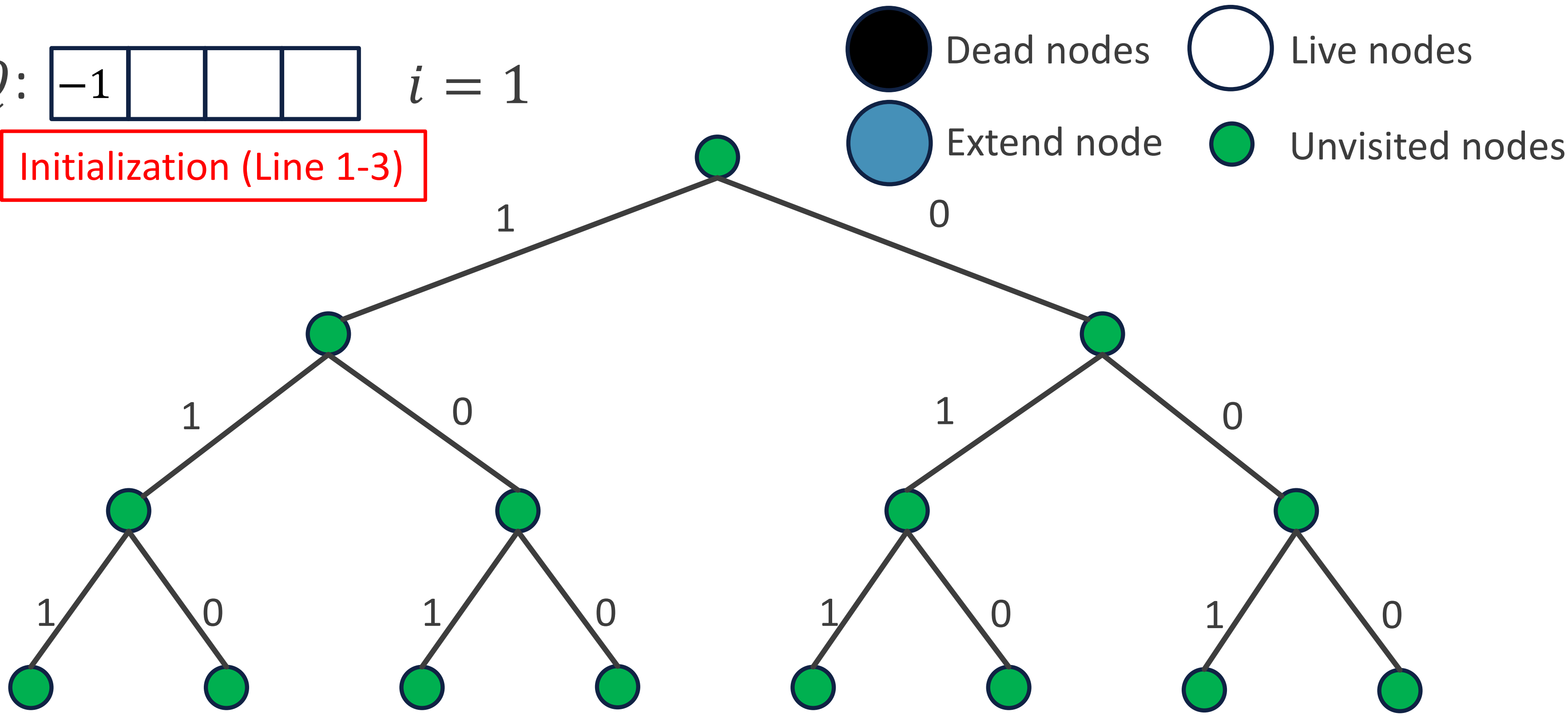
Continue to explore the next level.

Example

Q: | -1 | | | |

$i = 1$

Initialization (Line 1-3)



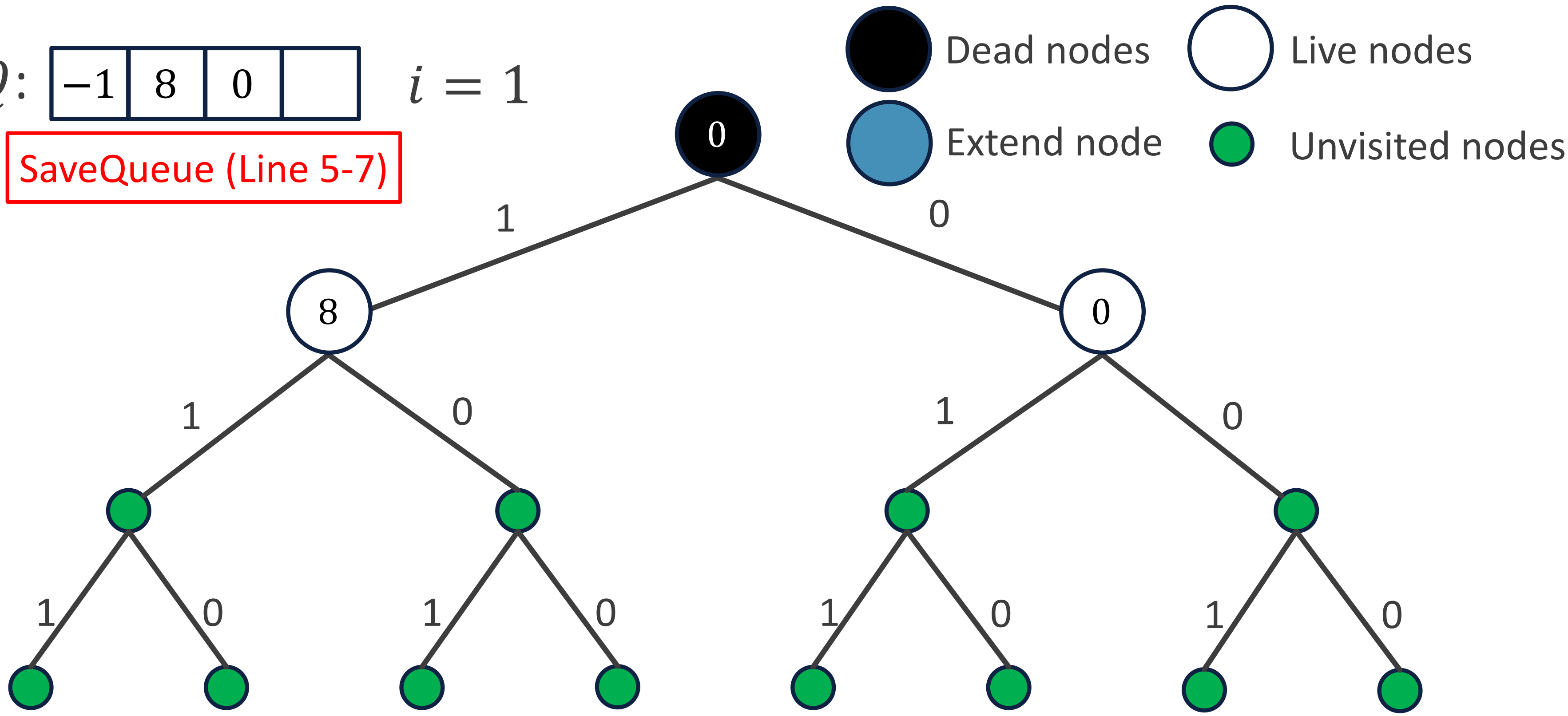
FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: | -1 | 8 | 0 | |

$i = 1$

SaveQueue (Line 5-7)

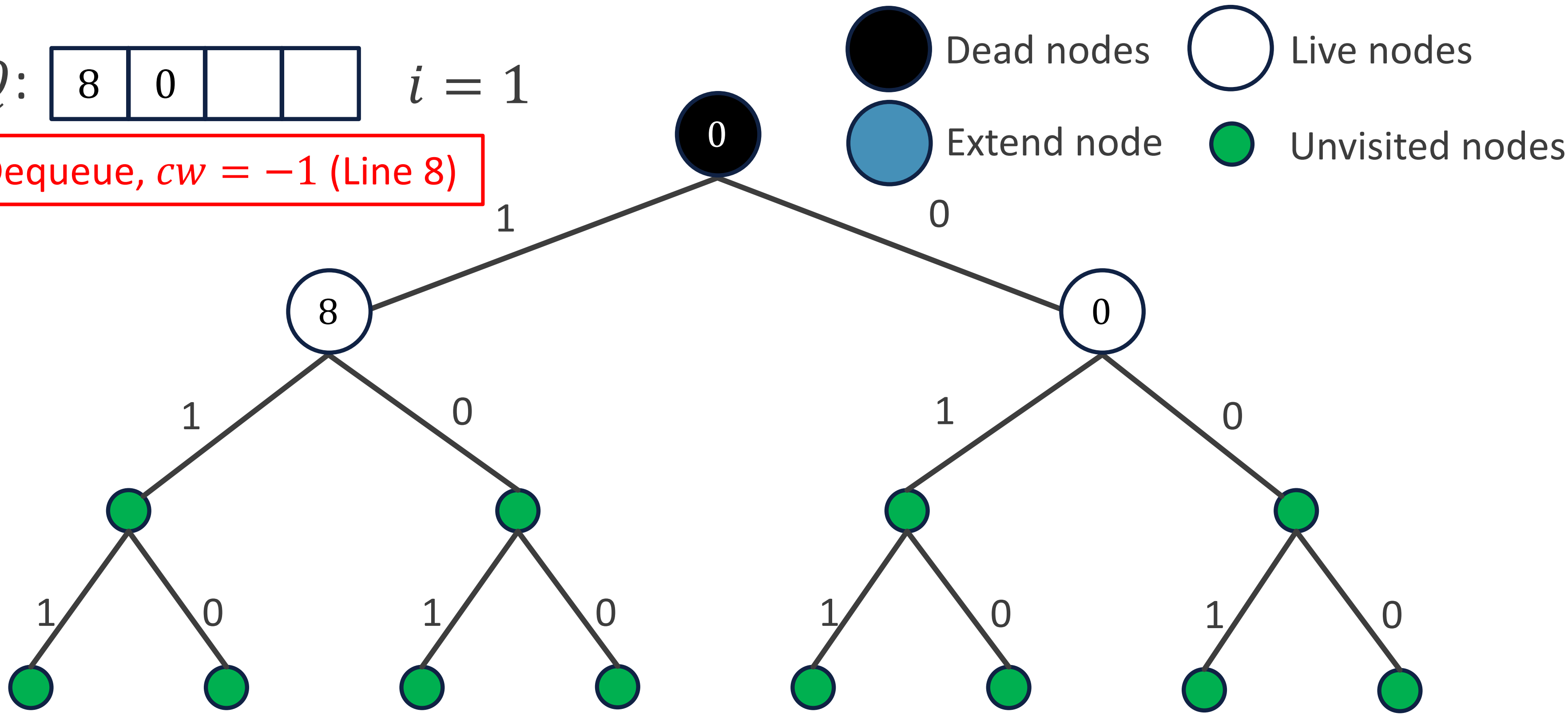


FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: | 8 | 0 | | |, $i = 1$

Dequeue, $cw = -1$ (Line 8)



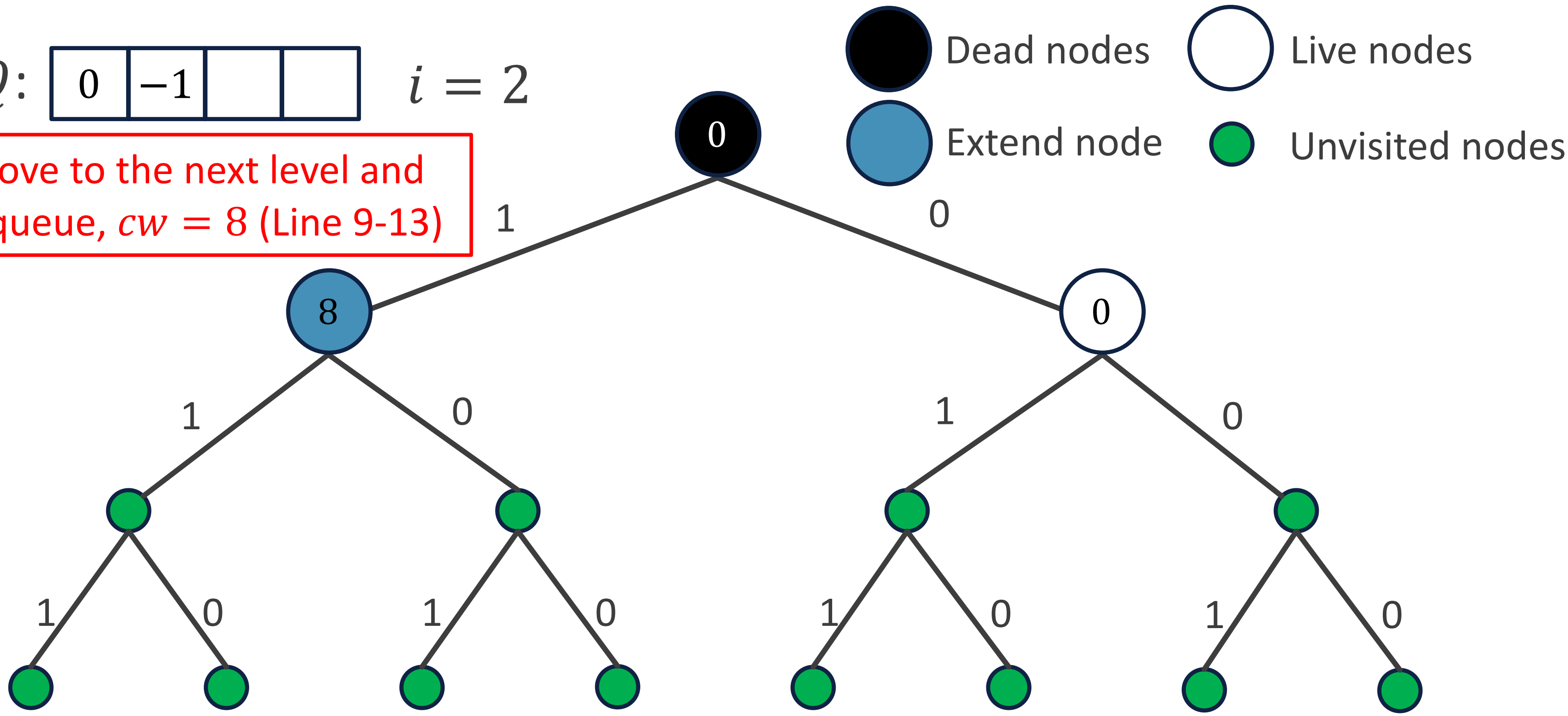
FIFO for $n = 3, w = [8,6,2], W = 12$

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Example

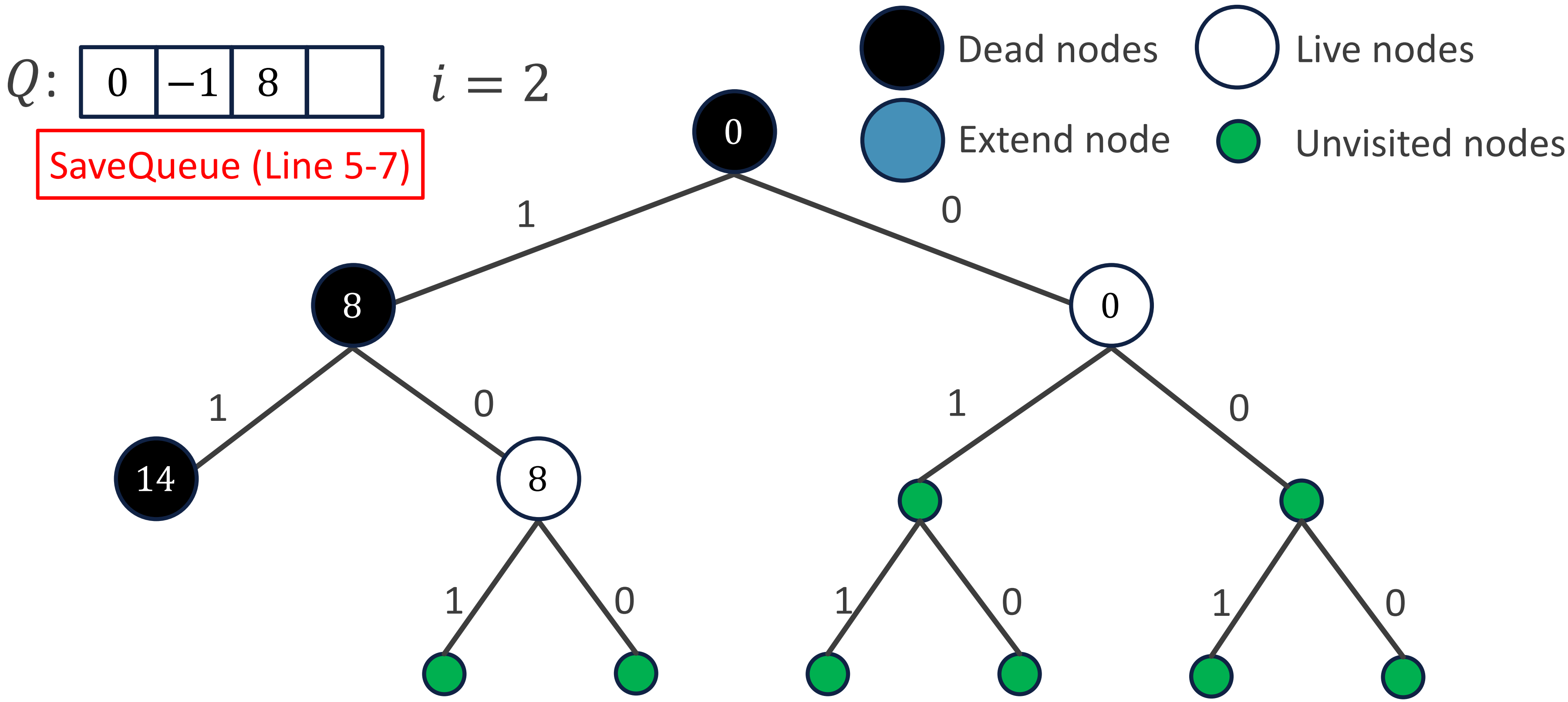
Q: | 0 | −1 | | | $i = 2$

Move to the next level and dequeue, $cw = 8$ (Line 9-13)



FIFO for $n = 3, w = [8,6,2], W = 12$

Example



FIFO for $n = 3, w = [8,6,2], W = 12$

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

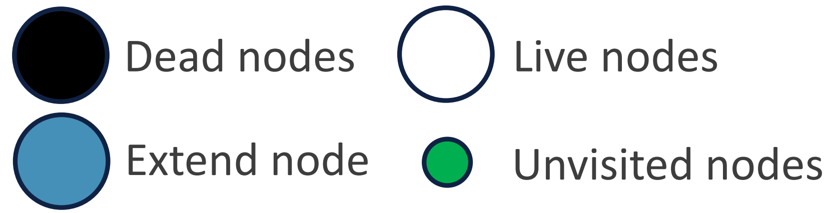
Example

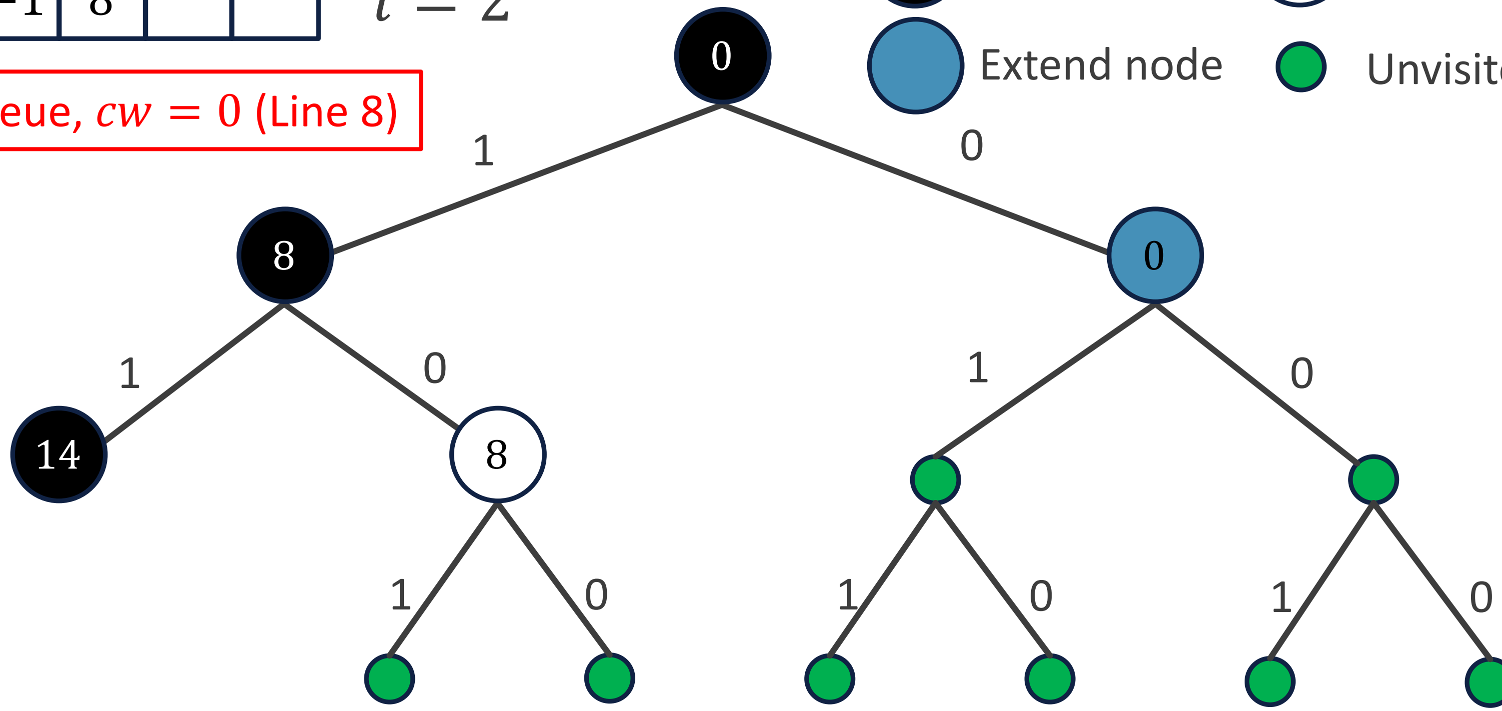
Q:

-1	8		

$i = 2$

Dequeue, $cw = 0$ (Line 8)





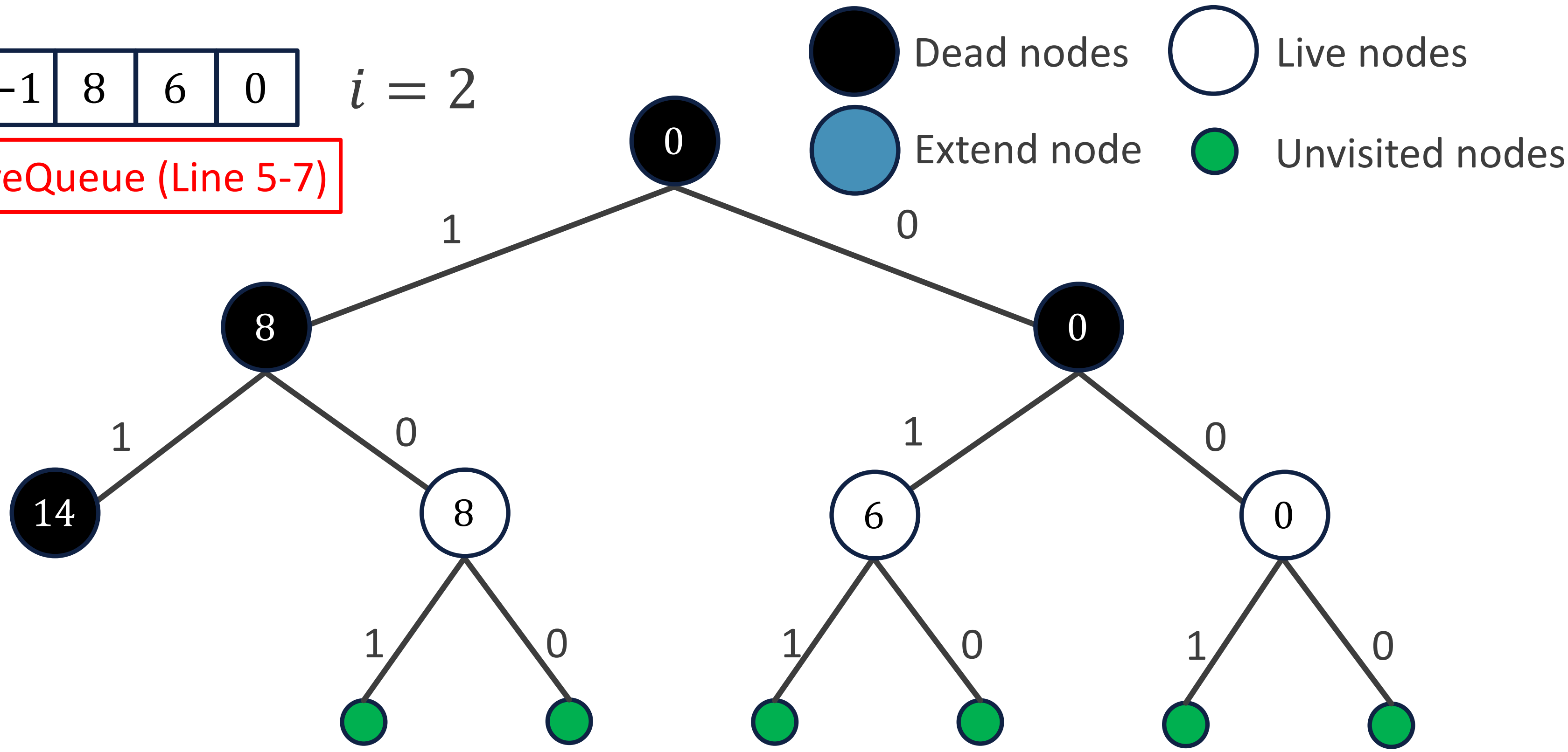
FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: | -1 | 8 | 6 | 0 |

$i = 2$

SaveQueue (Line 5-7)



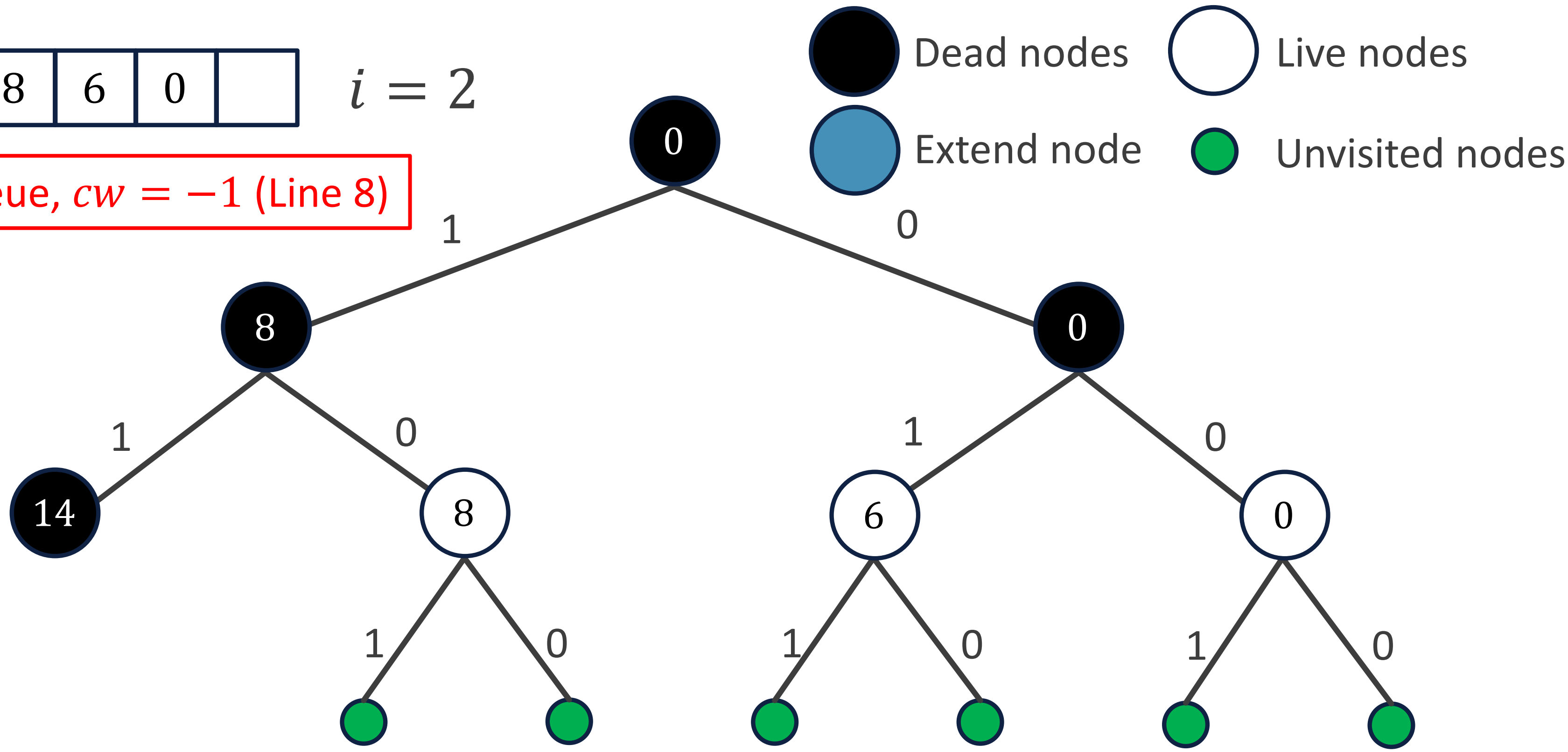
FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: | 8 | 6 | 0 | |

$i = 2$

Dequeue, $cw = -1$ (Line 8)



FIFO for $n = 3, w = [8,6,2], W = 12$

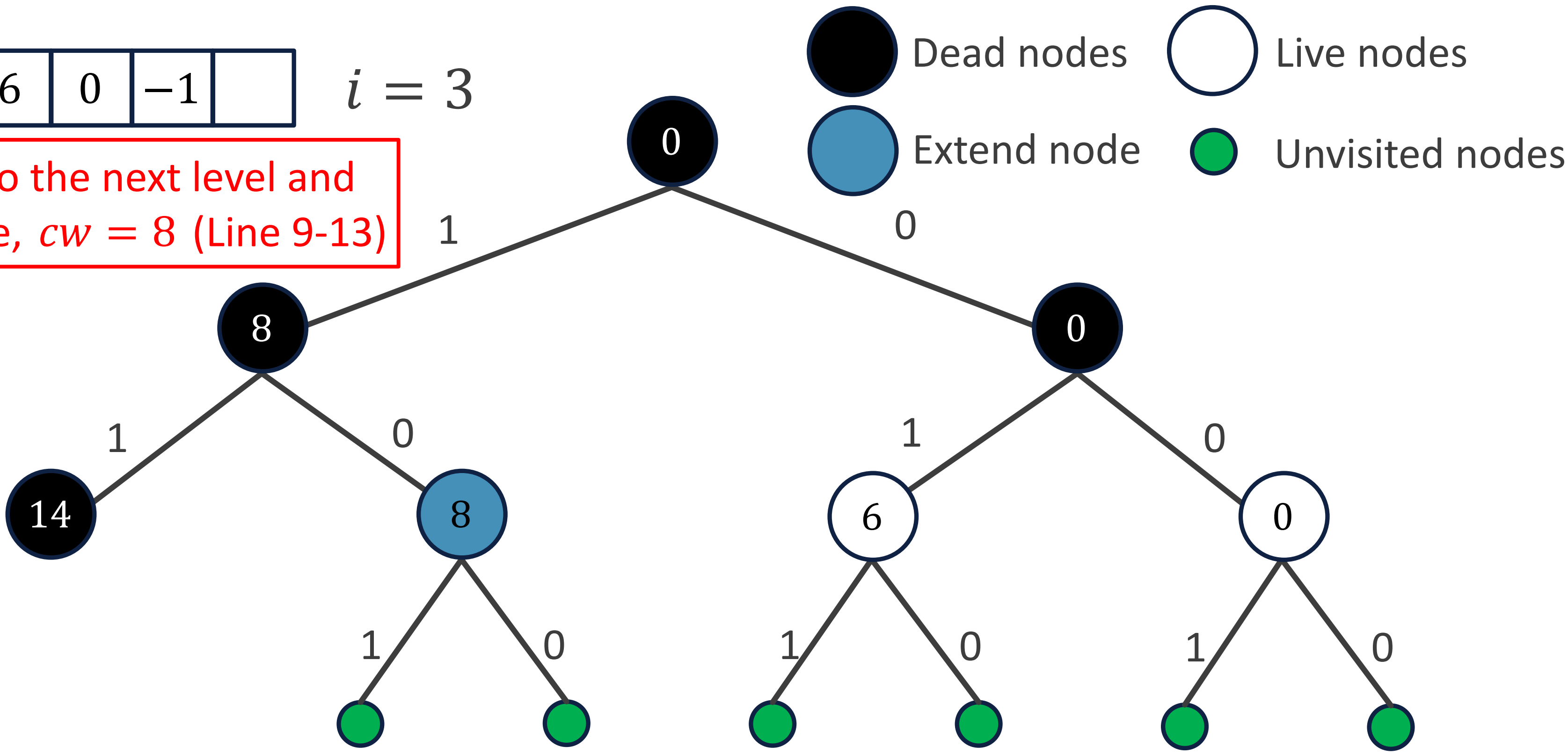
厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Example

Q: | 6 | 0 | −1 | |

$i = 3$

Move to the next level and dequeue, $cw = 8$ (Line 9-13)



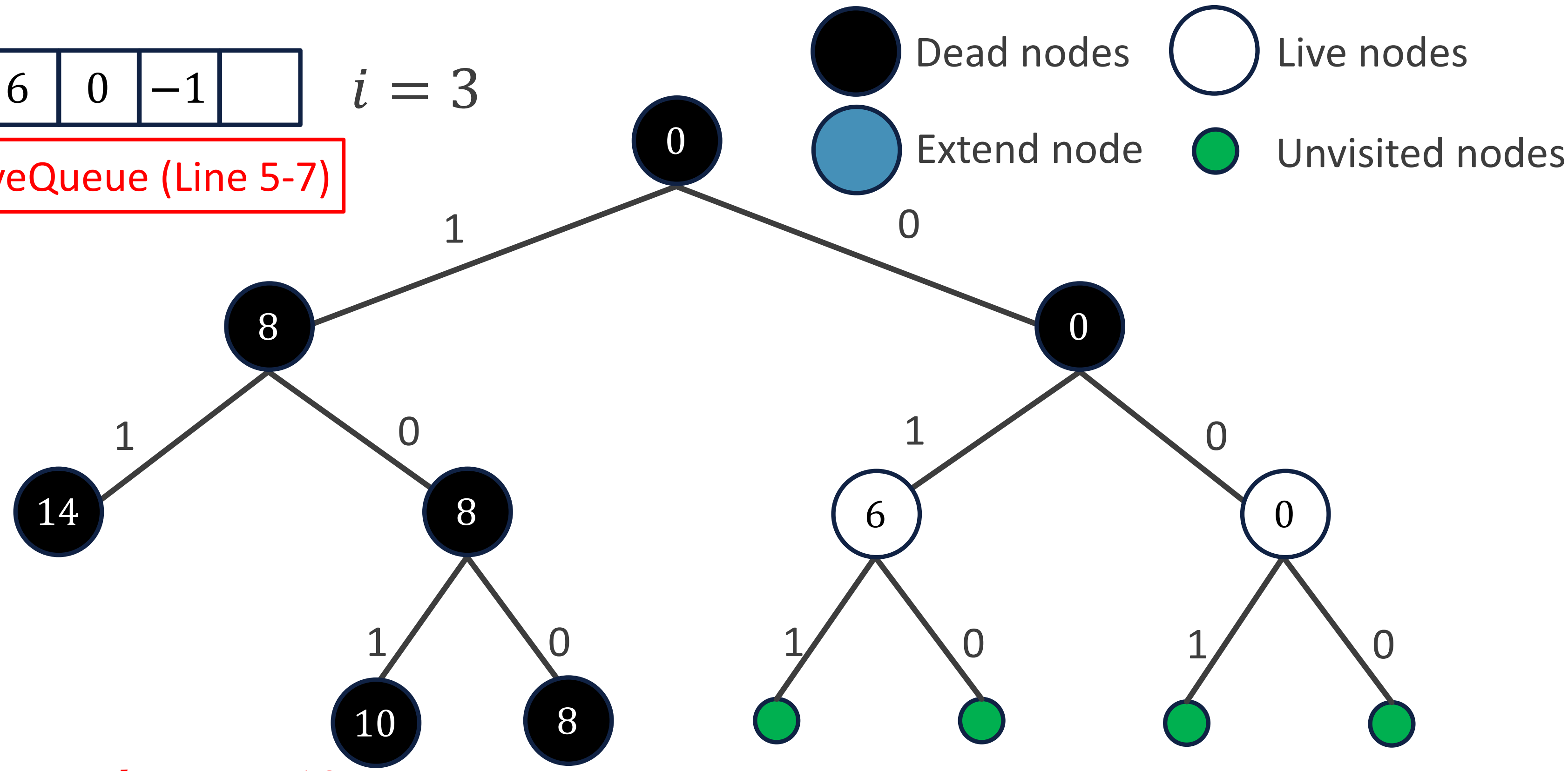
FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: | 6 | 0 | −1 | |

SaveQueue (Line 5-7)

$i = 3$



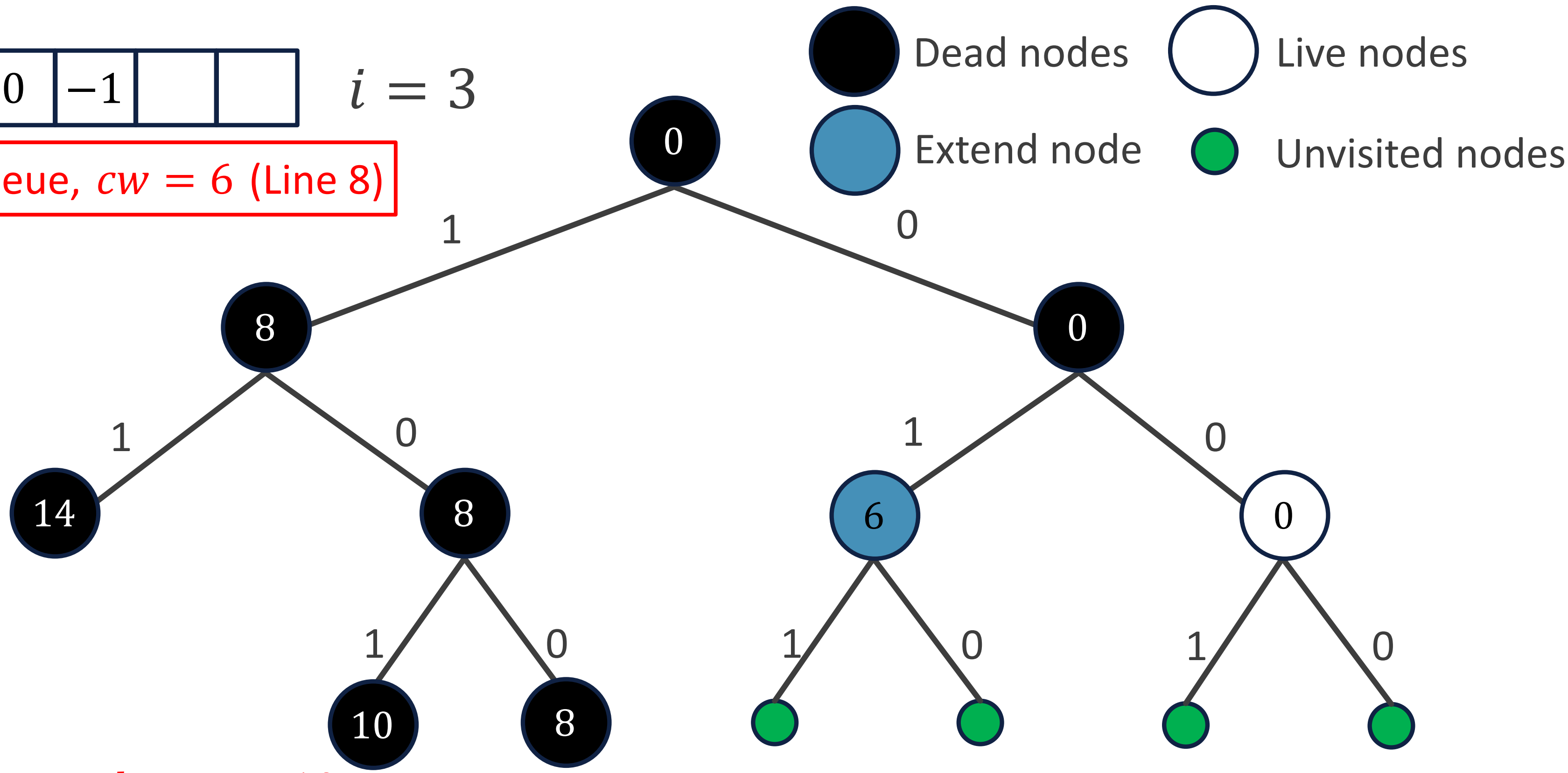
$bestw = 10$

FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: | 0 | −1 | | | $i = 3$

Dequeue, $cw = 6$ (Line 8)



$bestw = 10$

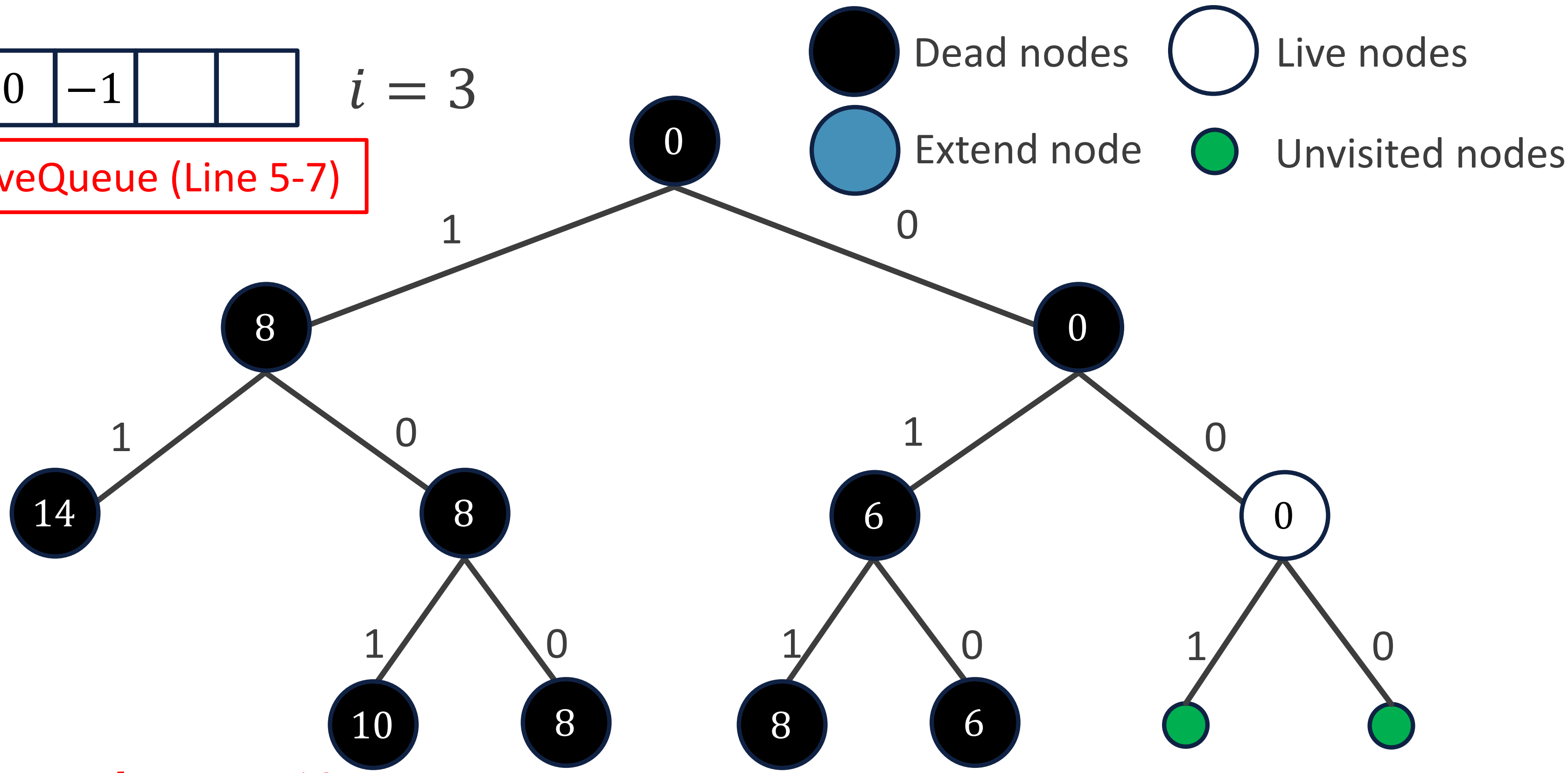
FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: | 0 | -1 | | |

$i = 3$

SaveQueue (Line 5-7)



$bestw = 10$

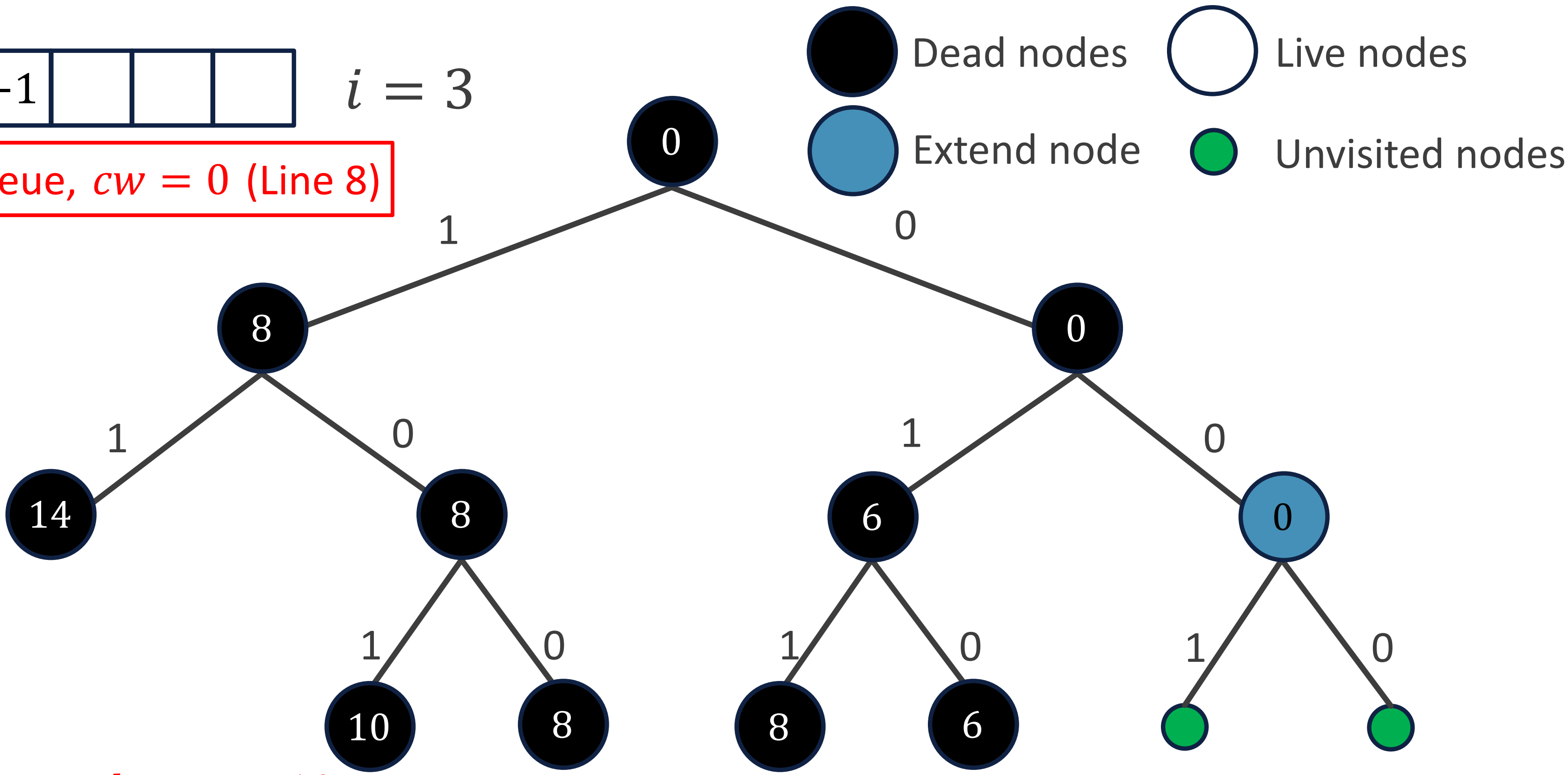
FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: -1 | | |

$i = 3$

Dequeue, $cw = 0$ (Line 8)



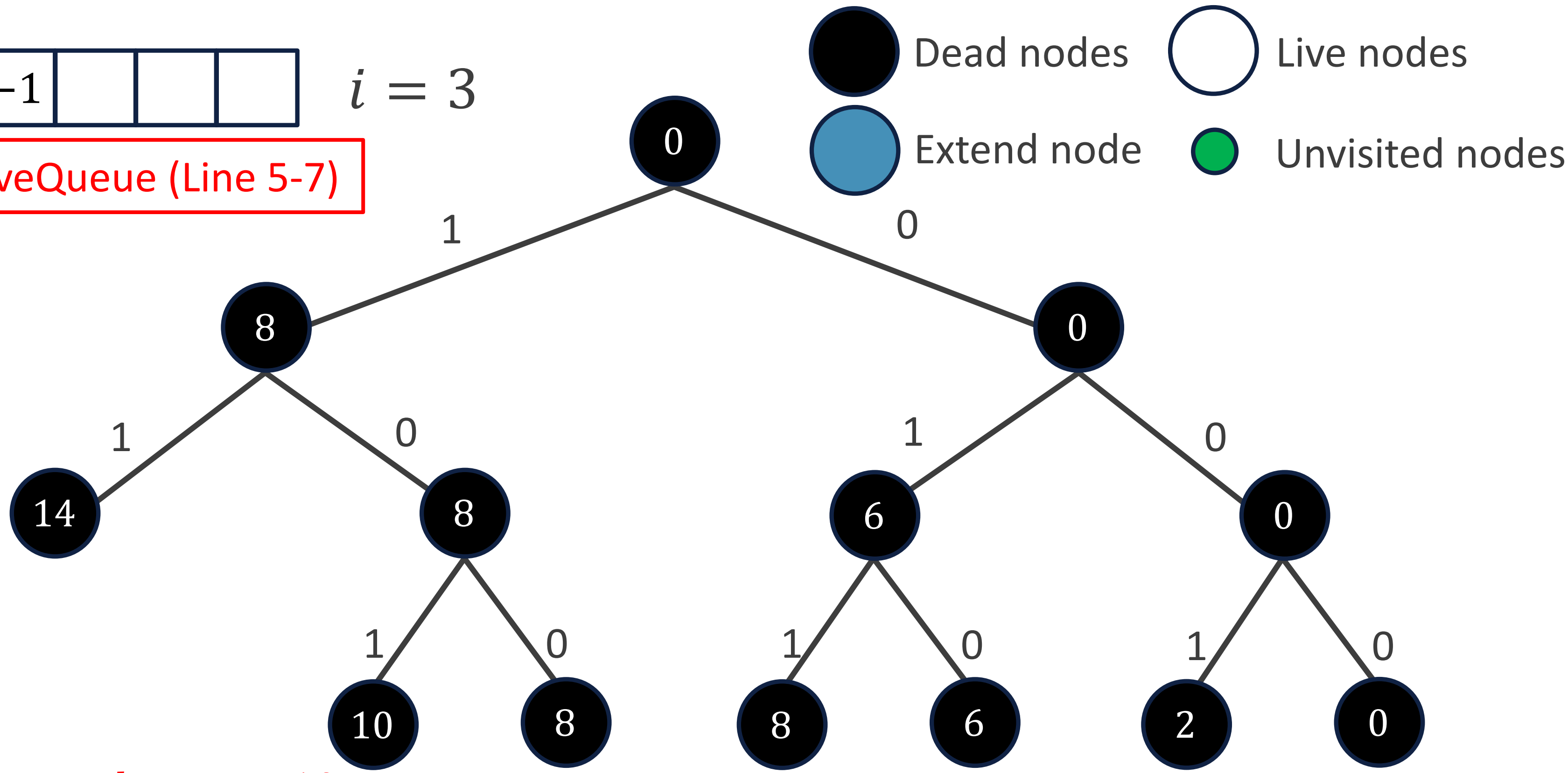
$bestw = 10$

FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: -1 | | | $\quad i = 3$

SaveQueue (Line 5-7)



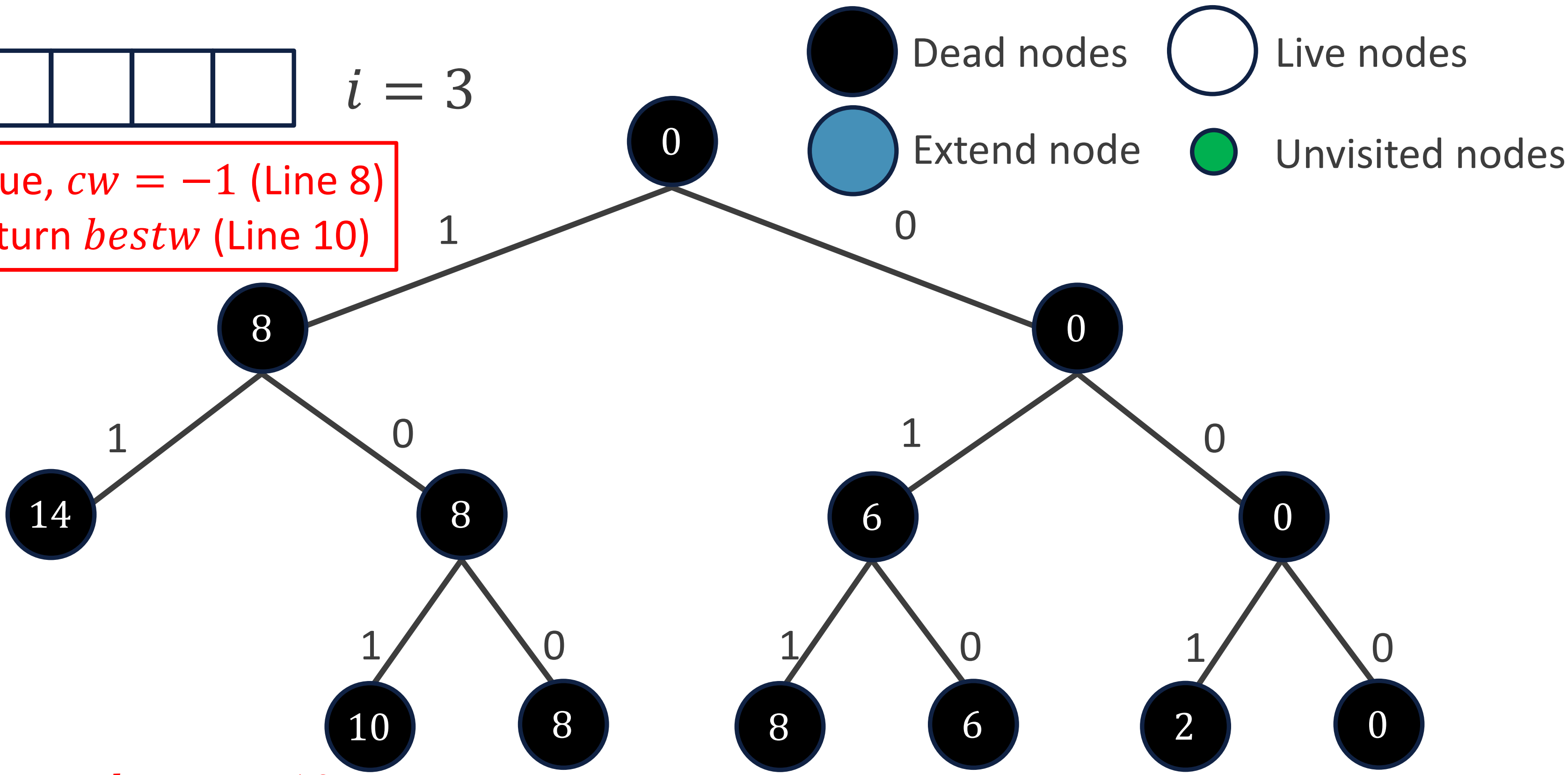
$bestw = 10$

FIFO for $n = 3, w = [8,6,2], W = 12$

Example

Q: [| | |] $i = 3$

Dequeue, $cw = -1$ (Line 8) and return $bestw$ (Line 10)



$bestw = 10$

FIFO for $n = 3, w = [8,6,2], W = 12$

FIFO Branch-and-Bound

- This version is obviously inefficient, because we didn't add the bounding function yet.
- We add the bounding function:

$$B(i) = C(i) + r(i)$$

where, $r(i)$ denotes the weight sum of the remaining containers, namely,

$$r(i) = \sum_{j=i+1}^{n} w_j$$

- The pruning condition is $B(i) \le bestw$

FIFO Branch-and-Bound

```
ImprovedFIFOMaxLoading(w, W, n)
i ← 1
Enqueue(Q, −1)
cw ← 0;  bestw ← 0;  r ← 0
for j ← 2 to n do r ← r + w[j]
while Q ≠ ∅ do
        if C(i) ≤ W then
                if C(i) > bestw then bestw ← C(i)
                if i < n then Enqueue(Q, C(i))
        if B(i) > bestw and i < n then Enqueue(Q, cw)
        cw ← Dequeue(Q)
        if cw = −1 then
                if Q = ∅ then return bestw
                Enqueue(Q, −1)
                cw ← Dequeue(Q)
                i ← i + 1
                r ← r − w[i]
return bestw
```

Upper bound is calculated from the second item, and reduced when level increased

We don’t enqueue leaf node

Enqueue live node with bounding condition

FIFO Branch-and-Bound

```
        if C(i) ≤ W then
            if C(i) > bestw then bestw ← C(i)
            if i < n then Enqueue(Q, C(i))
        if B(i) > bestw and i < n then Enqueue(Q, cw)
```

- Let's take a deep look into this part.
- Why can we update $bestw$ without checking it is a solution or not?
  - In backtracking, it is not necessary because the bounding function works only after a feasible solution is obtained.
  - However, using FIFO, we can update $bestw$ first to kill more nodes at the same level.
- This is the key factor that makes FIFO branch-and-bound efficient.

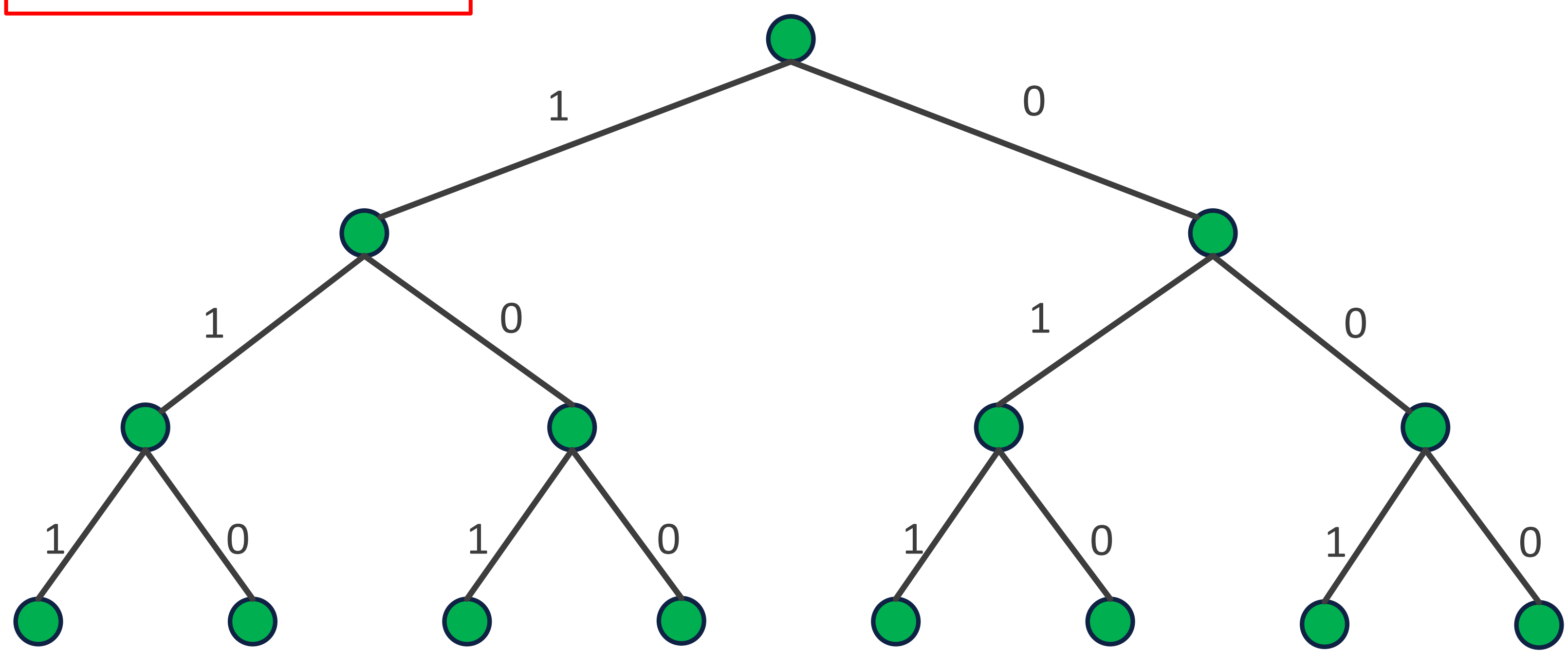
Example

Q: | -1 | | | |

$i = 1$

Initialization (Line 1-4)

$C(i)/B(i)$ Dead nodes
$C(i)/B(i)$ Live nodes
$C(i)/B(i)$ Extend node
Unvisited nodes



FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

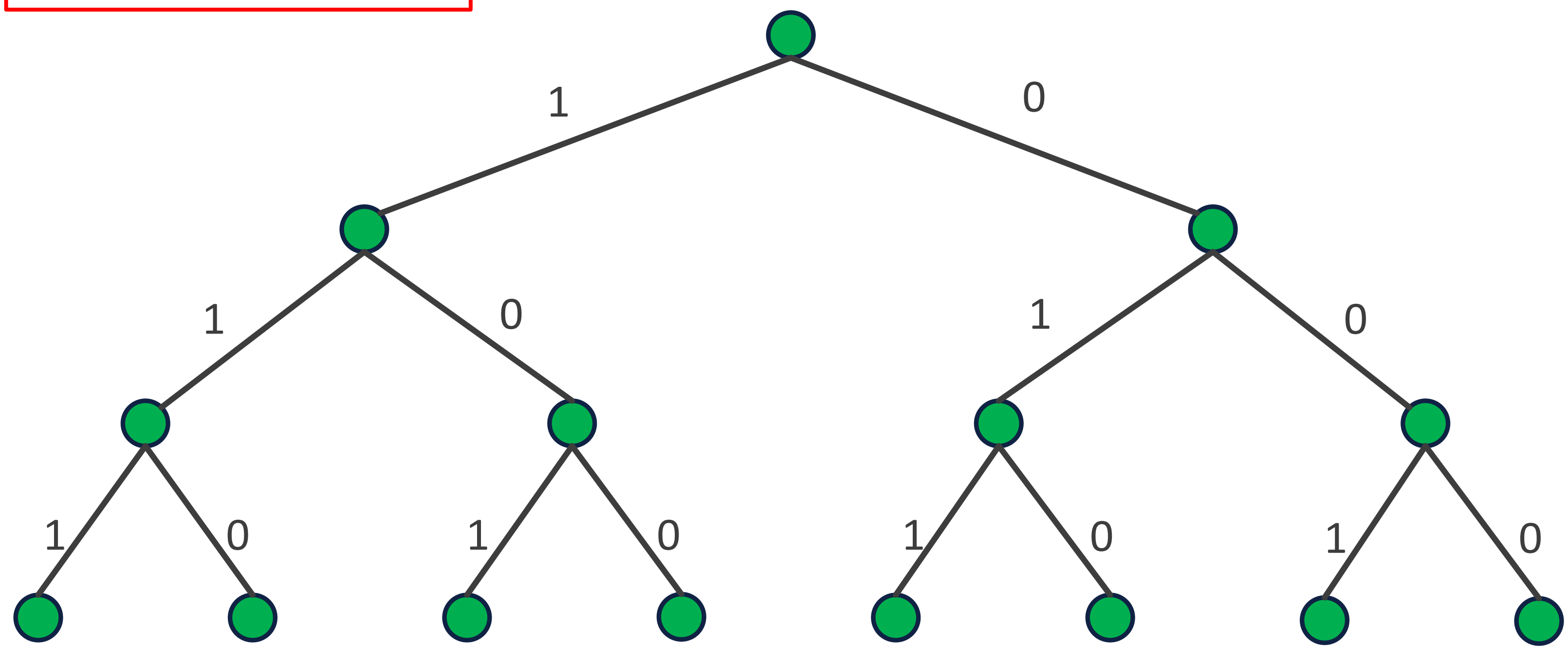
Example

Q: | -1 | | | | $i = 1$

Initialization (Line 1-4)

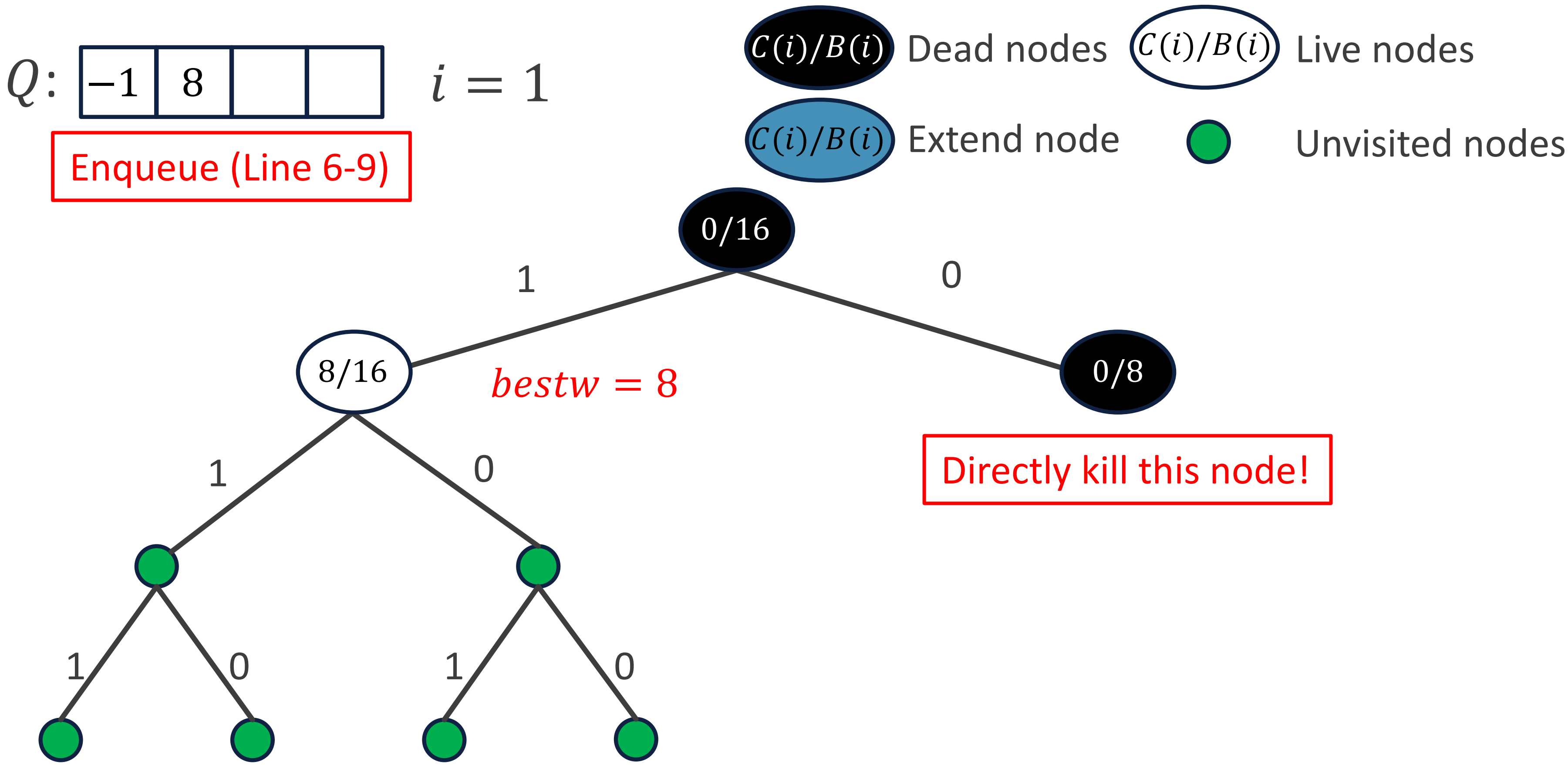
$C(i)/B(i)$ Dead nodes — $C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node — Unvisited nodes



FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

Example



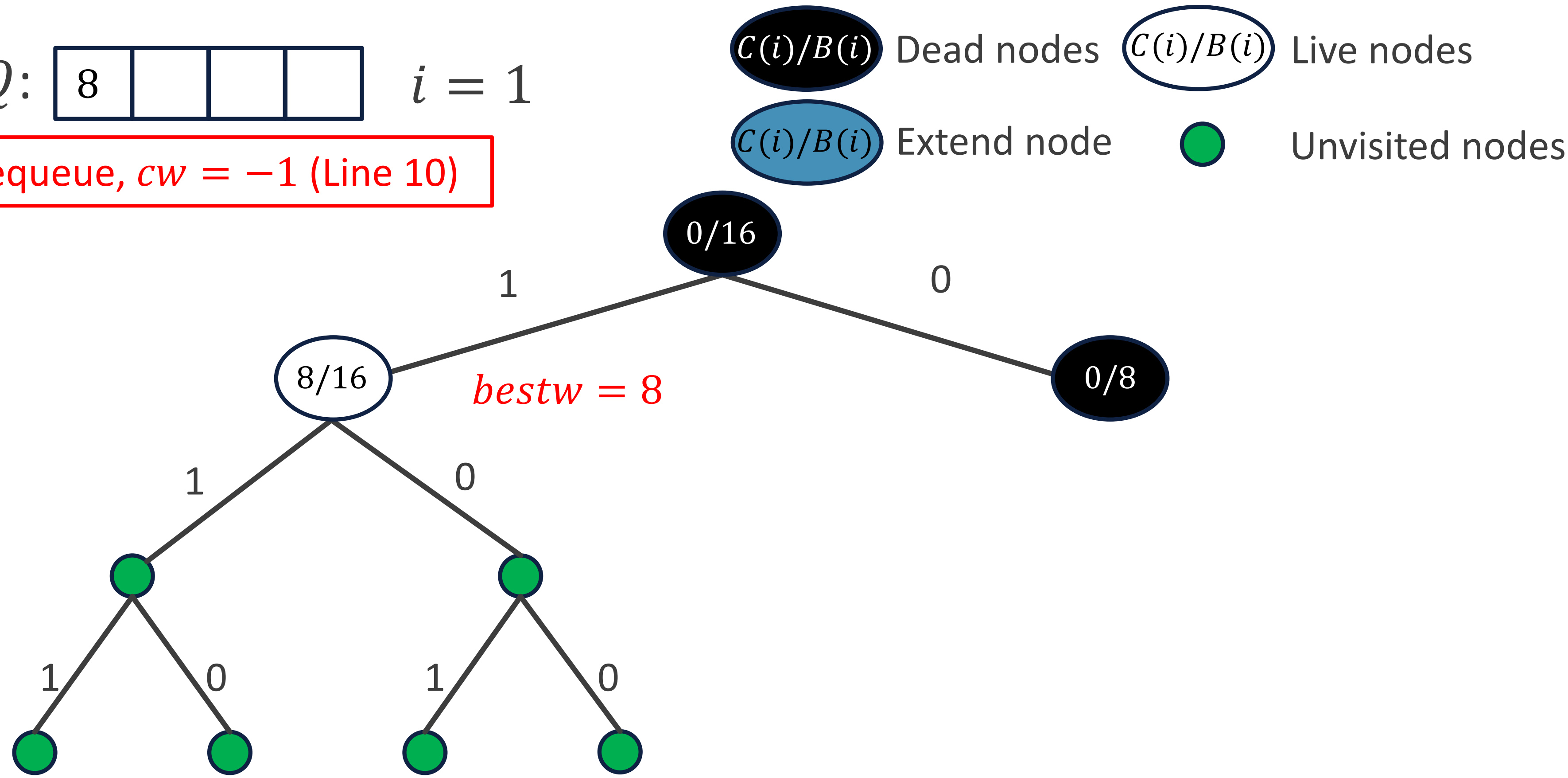
FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Example

Q: | 8 | | | | $i = 1$

Dequeue, $cw = -1$ (Line 10)



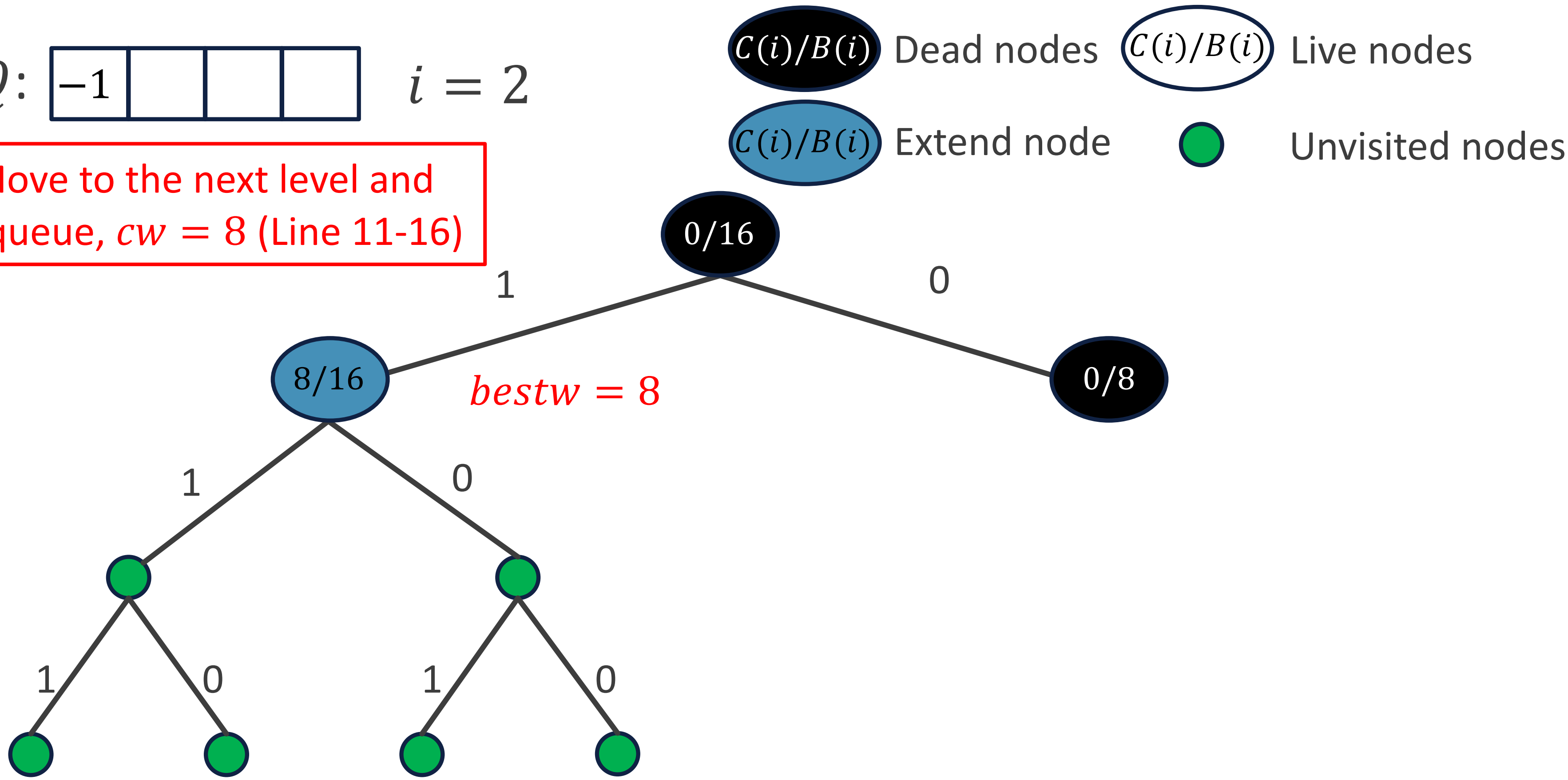
FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

Example

Q: | -1 | | | |

$i = 2$

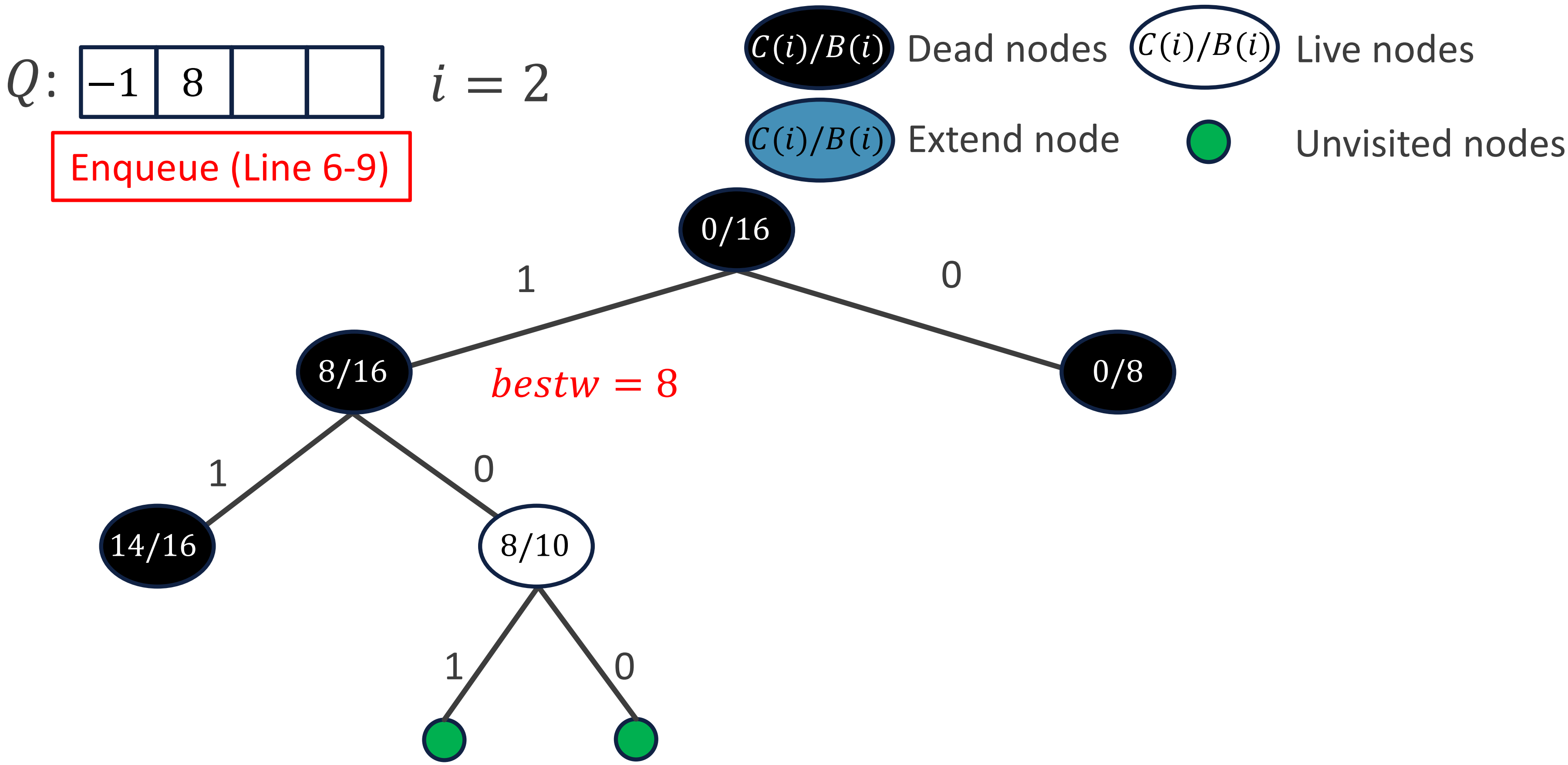
Move to the next level and dequeue, $cw = 8$ (Line 11-16)



FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Example



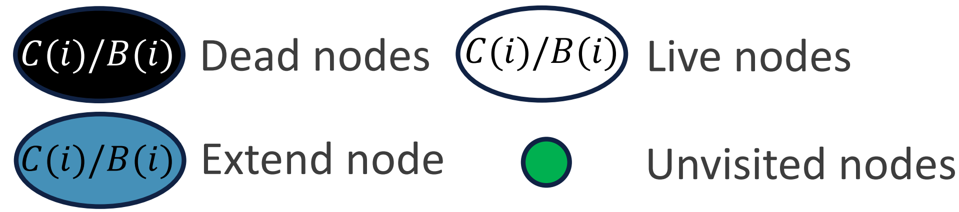
FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

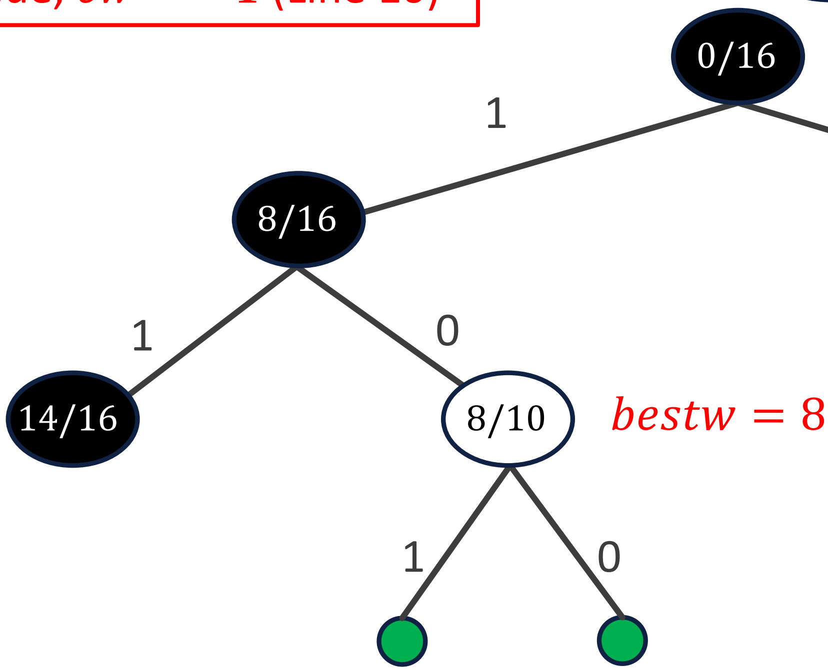
Example

Q: | 8 | | | |

$i = 2$

Dequeue, $cw = -1$ (Line 10)





FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

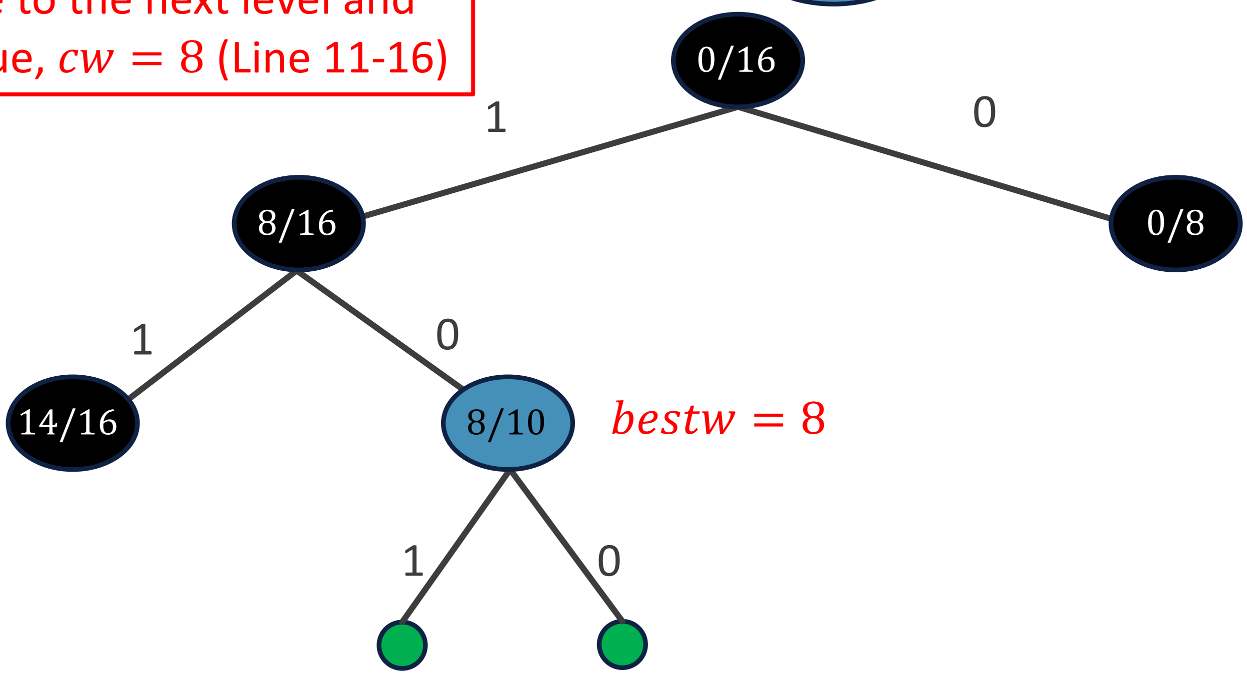
Example

Q: | −1 | | | | $i = 3$

Move to the next level and dequeue, $cw = 8$ (Line 11-16)

$C(i)/B(i)$ Dead nodes $C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node Unvisited nodes



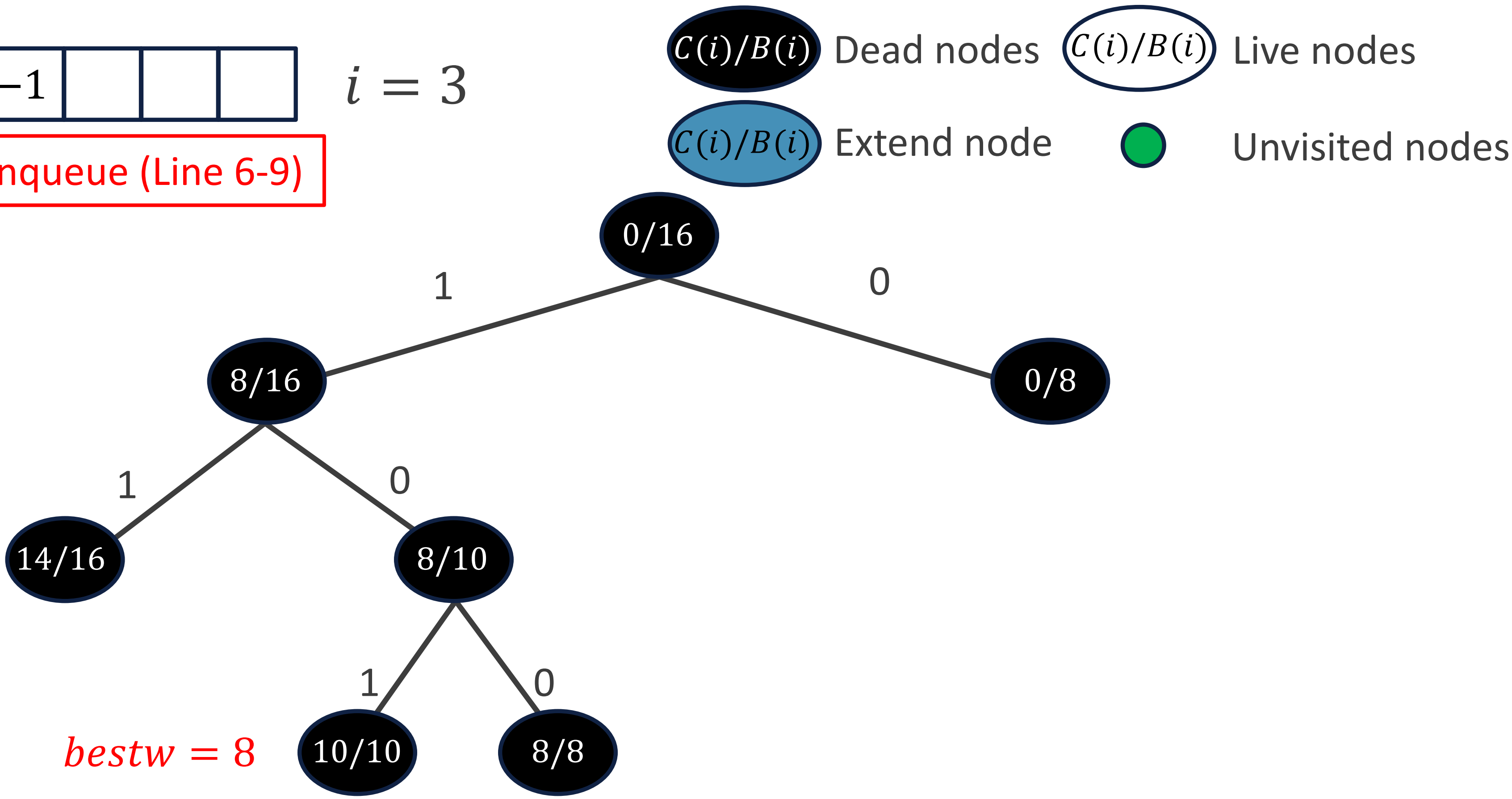
FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

Example

Q: | -1 | | | |

Enqueue (Line 6-9)

$i = 3$



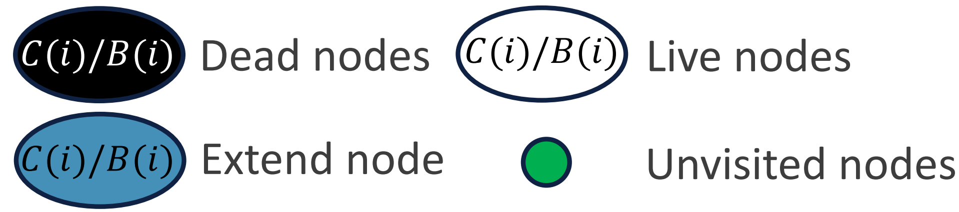
$bestw = 8$

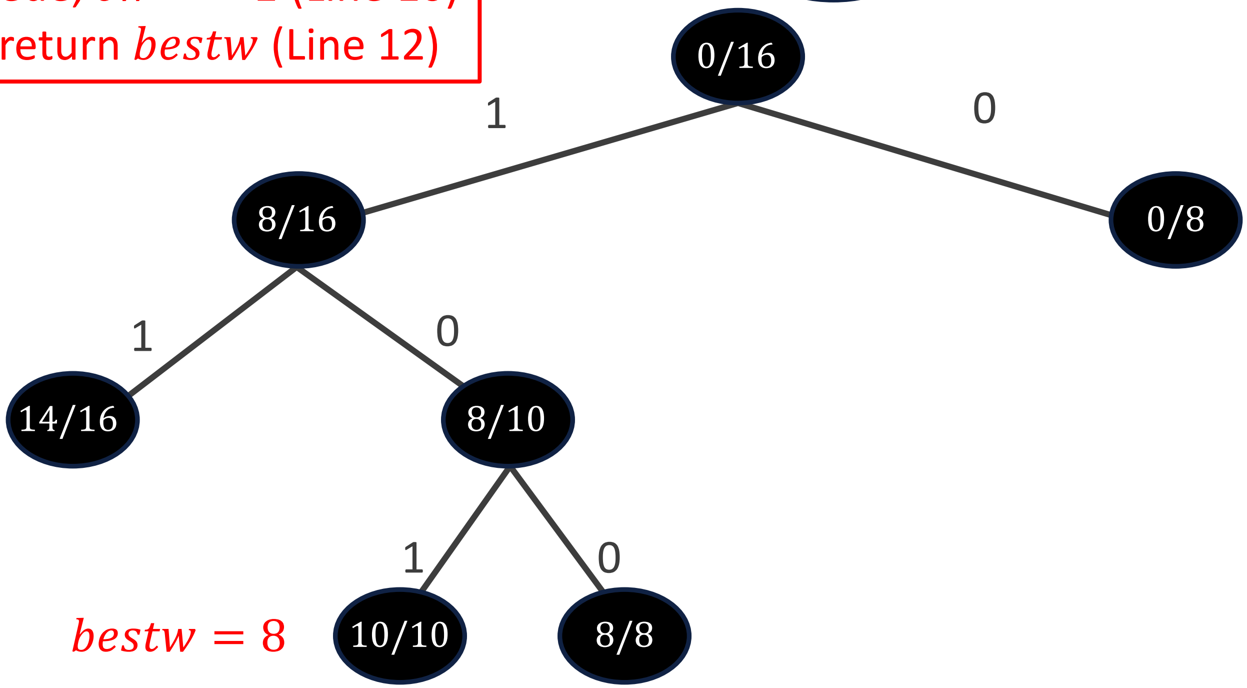
FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

Example

Q: $i = 3$

Dequeue, $cw = -1$ (Line 10) and return $bestw$ (Line 12)





$bestw = 8$

FIFO with bounding for $n = 3, w = [8,6,2], W = 12$

Record the Solution

```
SolutionFIFOMaxLoading( )
i ← 1
Enqueue(Q, −1)
cw ← 0; bestw ← 0; r ← 0
    for j ← 2 to n do r ← r + w[j]
    while Q ≠ Ø do
        if C(i) ≤ W then
            SaveQueue(Q, C(i), i, bestw, E, bestE, bestx, 1)
        if B(i) > bestw then
            SaveQueue(Q, cw, i, bestw, E, bestE, bestx, 0)
        E ← Dequeue(Q)
        if E = −1 then
            if Q = Ø then return bestw
            Enqueue(Q, −1); E ← Dequeue(Q); i ← i + 1;  r ← r − w[i]
        cw ← E.weight
for j ← n − 1 downto 1 do
    bestx[j] ← bestE.Lchild
    bestE ← bestE.parent
return bestw
```

$bestx[i]$ records the decision made on step i.

Use data structure:

- $E.weight$: Current weight
- $E.parent$: Parent node
- $E.Lchild$: Decision (0/1)

```
SaveQueue(Q, wt, i, bestw, E, bestE, bestx, ch)
if i = n then
    if wt > bestw then
        bestE ← E
        bestw ← wt
        bestx[n] ← ch
else
    b.weight ← wt
    b.parent ← E
    b.Lchild ← ch
    Enqueue(Q, b)
```

FIFO Using Constraint Function and Bounding Function

- It seems as good as backtracking. Can we further improve?
- The current version of branch-and-bound uses FIFO, just like backtracking using FILO.
  - It is still limited by FIFO when we are branching.
- Can we choose the node to branch out of the order determined by FIFO or FILO?

Max-Profit Branch-and-Bound

- Instead, we use max-priority queue. Select node with maximum upper bound!
- Live nodes become E-nodes in decreasing order of $B(i)$.
  - Notice that if x is a node with an upper bound, then no node in its subtree has weight more than this upper bound.
- When do we stop? The node with maximum upper bound is a leaf, which means no remaining live node can lead to a leaf with more weight.

Max-Profit Branch-and-Bound

```
MaxCostLoading( )
  i ← 1
  r[n] ← 0
  for j ← n − 1 downto 1 do r[j] ← r[j + 1] + w[j + 1]
  while i ≠ n + 1 do
      if C(i) ≤ W then
          AddLiveNode(Q, E, C(i) + r[i], 1, i + 1)
      AddLiveNode(Q, E, cw + r[i], 0, i + 1)
      N ← ExtractMax(Q)
      i ← N.level
      E ← N.ptr
      cw ← N.weight − r[i − 1]
  for j ← n downto 1 do
      bestx[j] ← E.Lchild
      E ← E.parent
  return cw
```

In FIFO, the level is always increasing, so we don't need to store it. Now, we need to store level.

Stop if extracted node is a leaf.

No bounding condition here. You can also add it.

Use data structure in max-priority queue:

- $N.weight$: Node upper bound
- $N.level$: Node level
- $N.ptr$: Pointer to node E
- $E.parent$: Parent node
- $E.Lchild$: Decision (0/1)

```
AddLiveNode(Q, E, wt, ch, lev)
  b.parent ← E
  b.Lchild ← ch
  N.weight ← wt
  N.level ← lev
  N.ptr ← b
  Insert(Q, N)
```

Max-Profit Branch-and-Bound

We store upper bounds, rather than current weight

```
    if C(i) ≤ W then
        AddLiveNode(Q, E, [C(i) + r[i]], 1, i + 1)
    AddLiveNode(Q, E, [cw + r[i]], 0, i + 1)
    N ← ExtractMax(Q)
    i ← N.level
    E ← N.ptr
    cw ← N.weight − r[i − 1]
```

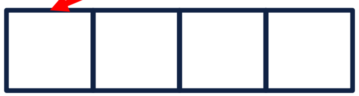
$$
\begin{array}{l}
\quad \textbf{if } C(i) \le W \textbf{ then} \\
\qquad \text{AddLiveNode}(Q, E, \boxed{C(i) + r[i]}, 1, i + 1) \\
\quad \text{AddLiveNode}(Q, E, \boxed{cw + r[i]}, 0, i + 1) \\
\quad N \leftarrow \text{ExtractMax}(Q) \\
\quad i \leftarrow N.level \\
\quad E \leftarrow N.ptr \\
\quad cw \leftarrow N.weight - r[i-1]
\end{array}
$$

The current weight is calculated by upper bound – remaining weight.

Example

Since we store level, we don't need -1 any more.

Q:



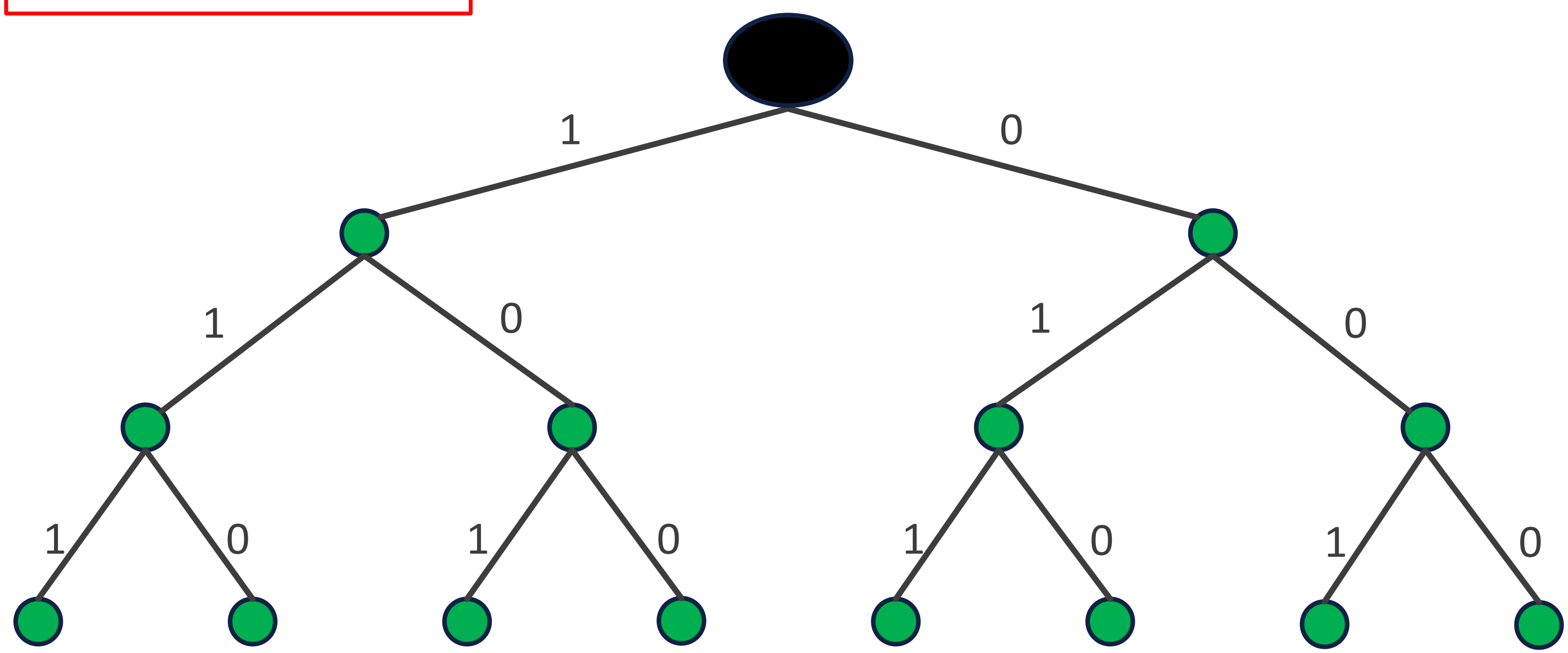
Initialization (Line 1-3)

$C(i)/B(i)$ Dead nodes

$C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node

Unvisited nodes

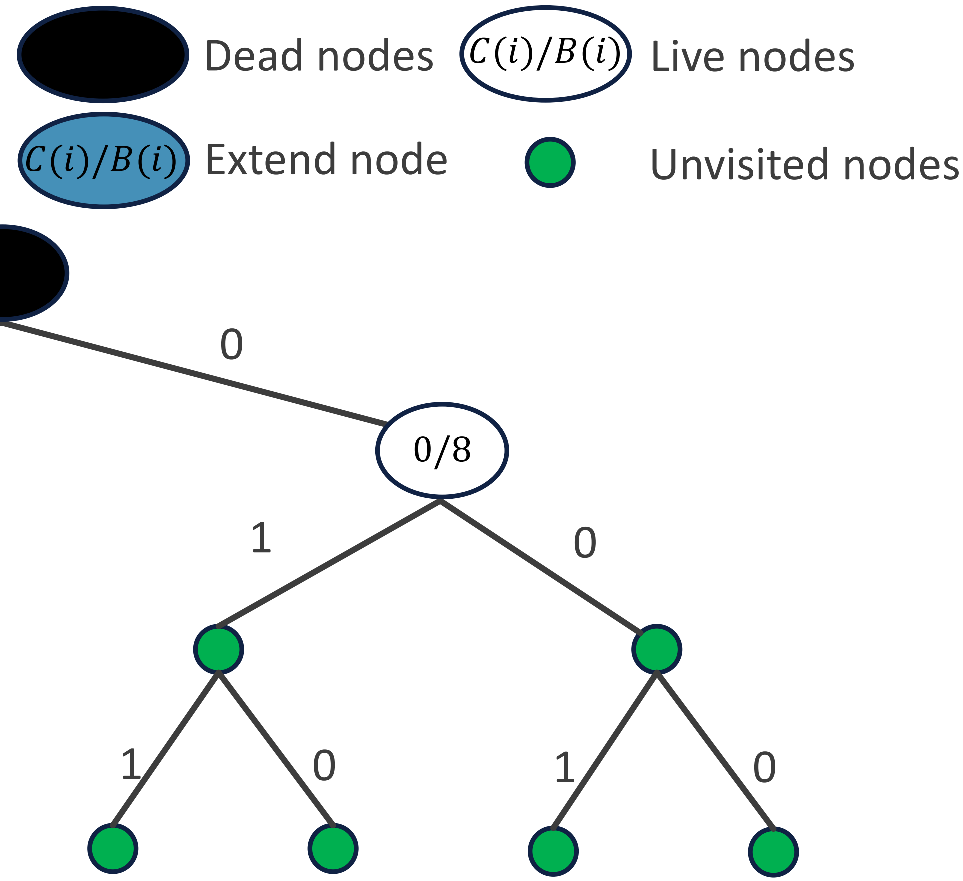


Max-profit branch-and-bound for $n = 3, w = [8,6,2], W = 12$

Example

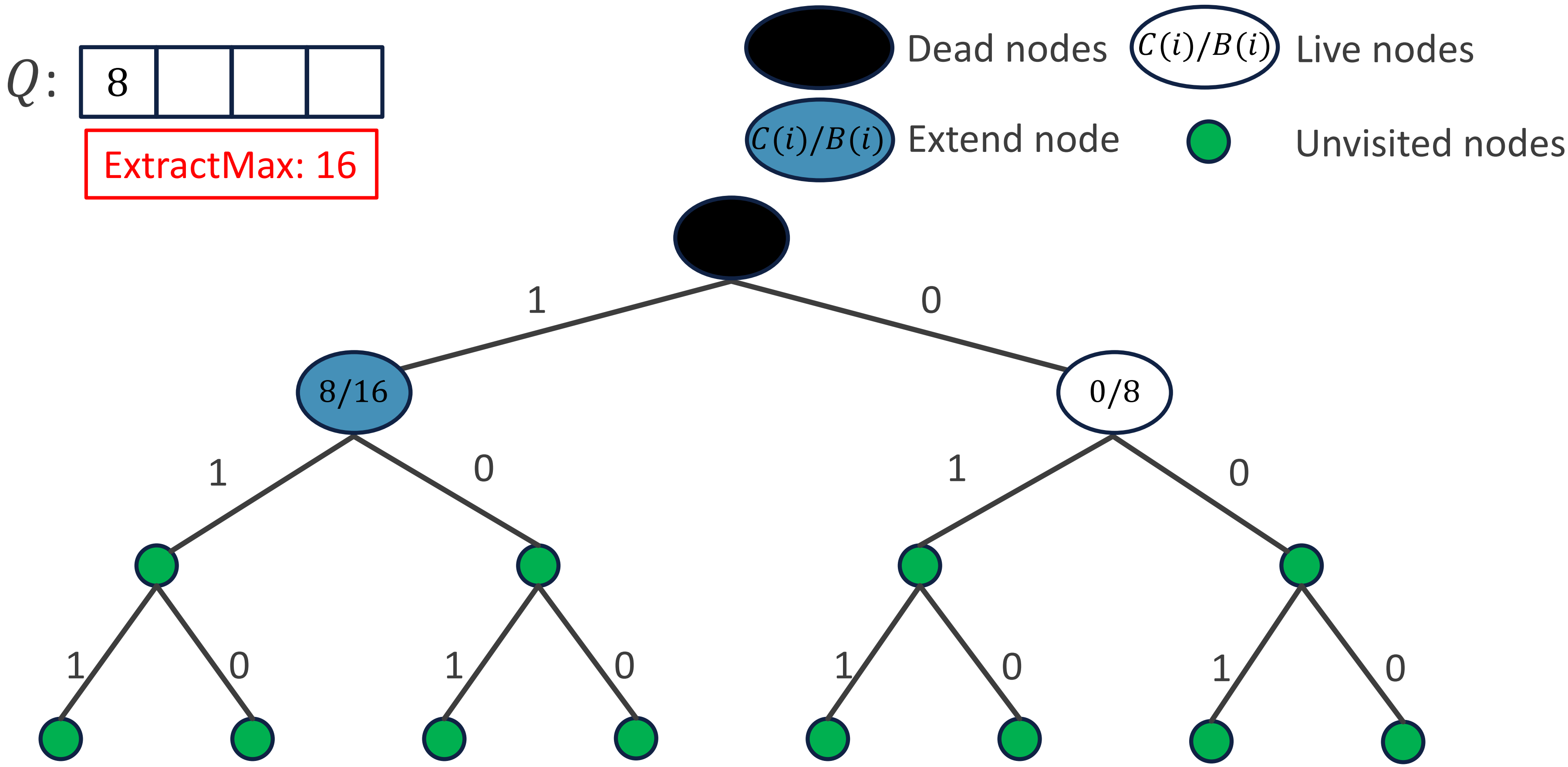
Q: | 16 | 8 | | |

AddLiveNode (Line 6-7)



Max-profit branch-and-bound for $n = 3, w = [8,6,2], W = 12$

Example

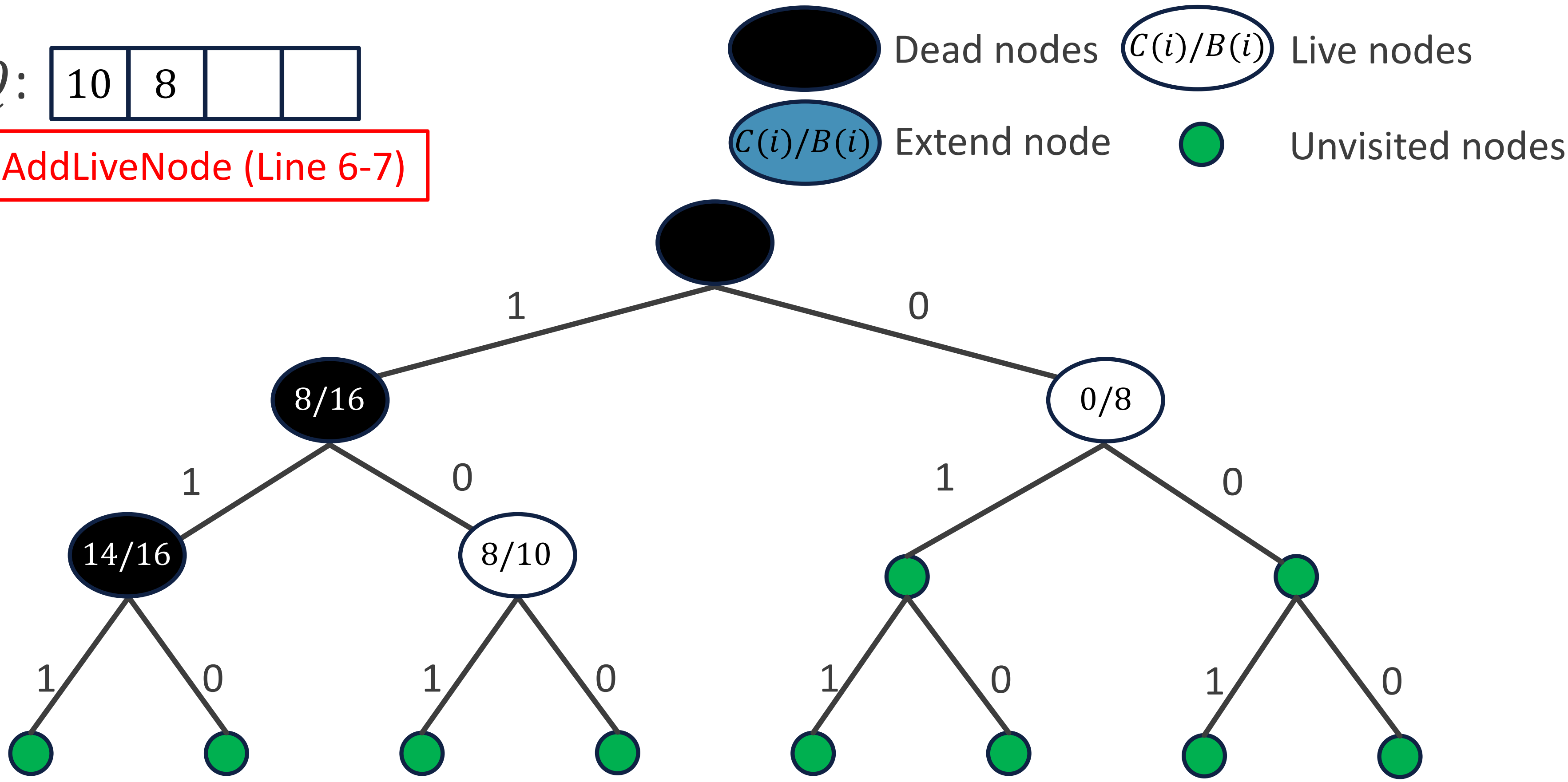


Max-profit branch-and-bound for $n = 3, w = [8,6,2], W = 12$

Example

Q: | 10 | 8 | | |

AddLiveNode (Line 6-7)

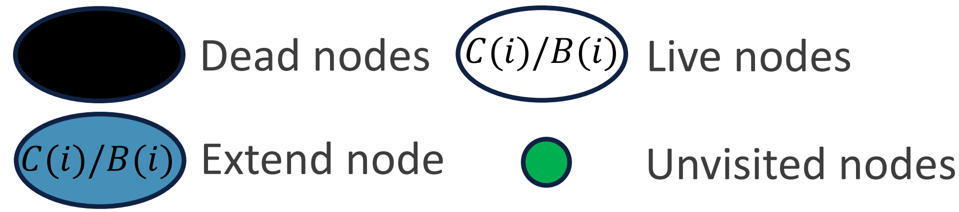


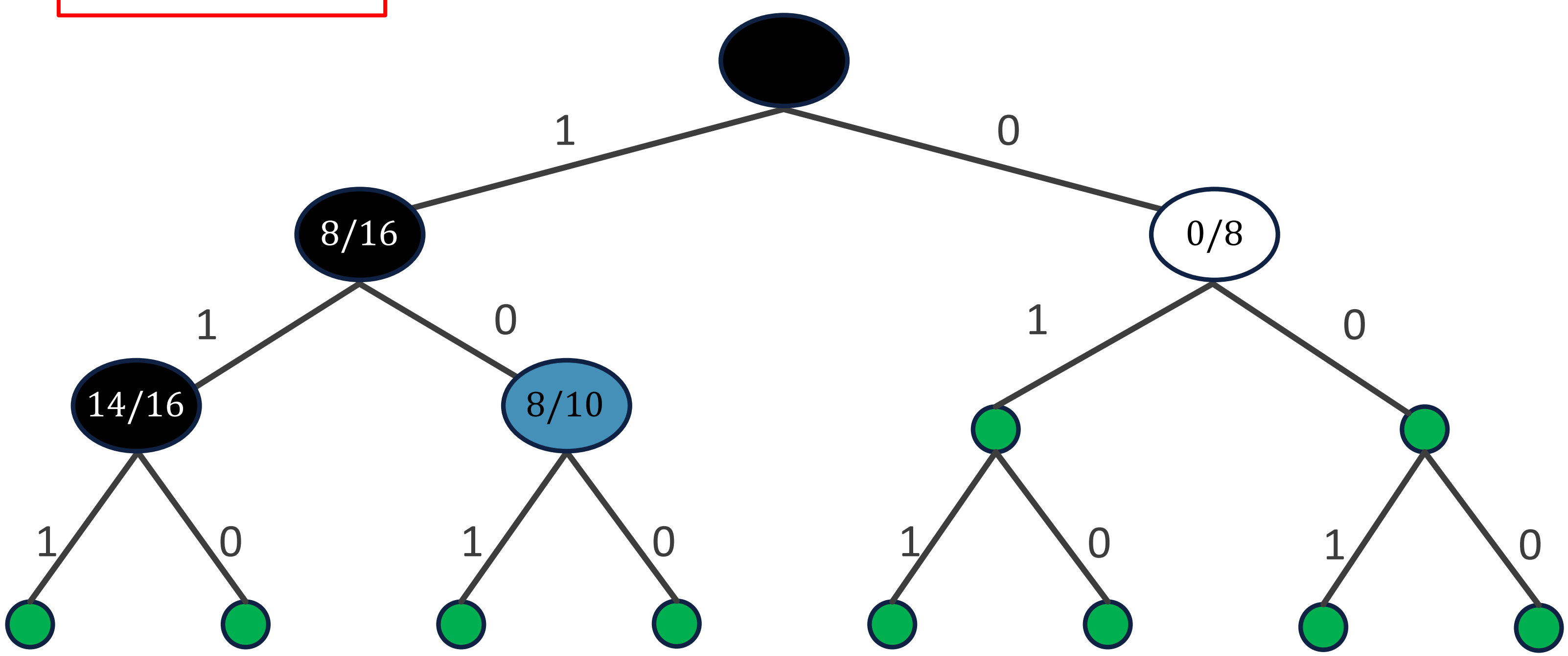
Max-profit branch-and-bound for $n = 3$, $w = [8,6,2]$, $W = 12$

Example

Q: | 8 | | | |

ExtractMax: 10



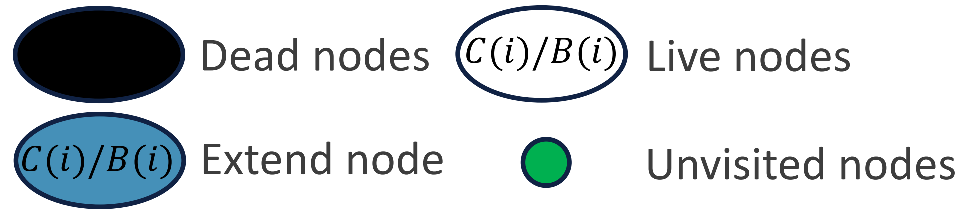


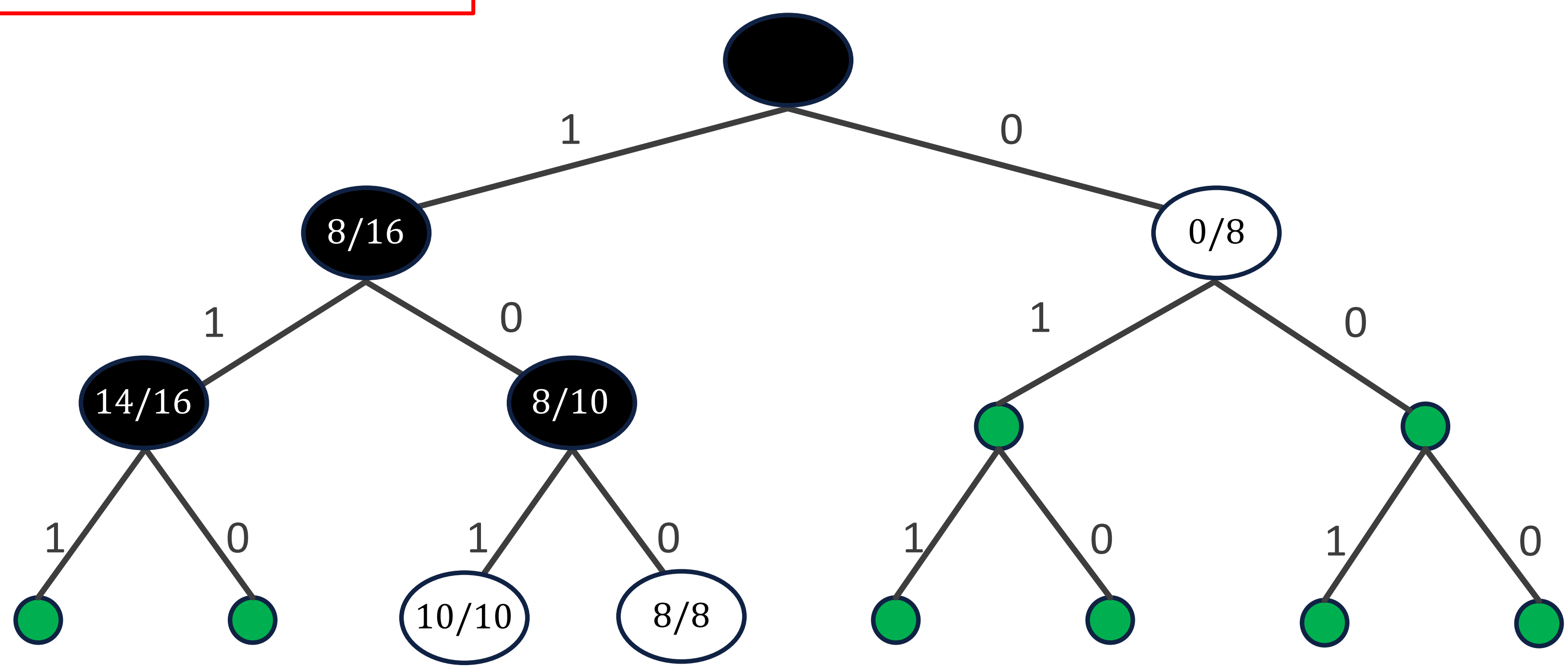
Max-profit branch-and-bound for $n = 3$, $w = [8,6,2]$, $W = 12$

Example

Q: | 10 | 8 | 8 | |

AddLiveNode (Line 6-7)





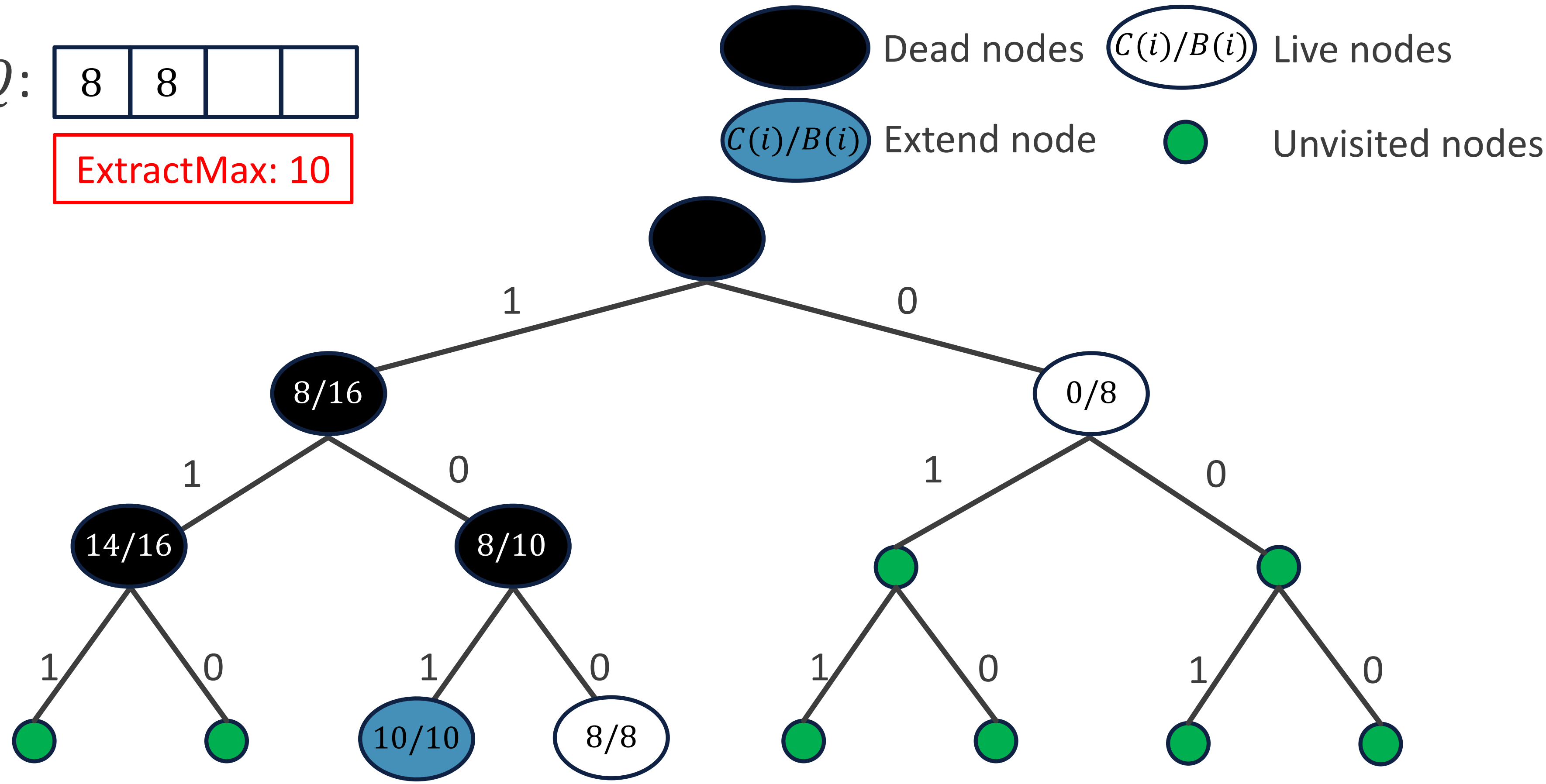
Max-profit branch-and-bound for $n = 3, w = [8,6,2], W = 12$

Example

Q:

8	8		

ExtractMax: 10



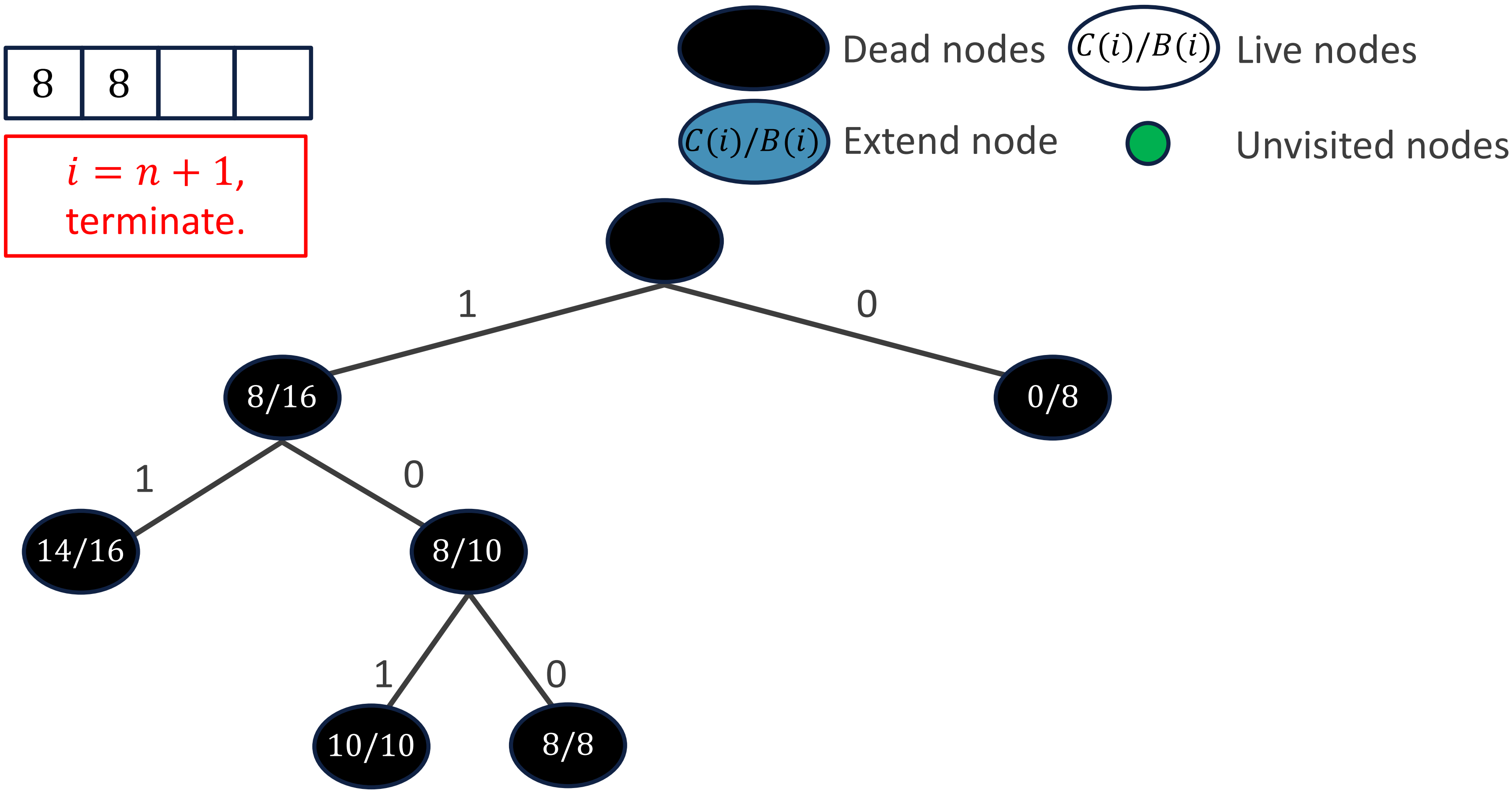
Max-profit branch-and-bound for $n = 3$, $w = [8,6,2]$, $W = 12$

厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

Example

Q: | 8 | 8 | | |

$i = n + 1$,
terminate.



Max-profit branch-and-bound for $n = 3, w = [8,6,2], W = 12$

Branch-and-Bound

- Now, we look back the name of branch-and-bound:
- Branch: We explore all of candidate branches.
  - That’s why branch-and-bound is based on BFS.
- Bound: We select a branch based on its bound.
  - Bound represents the degree of hope.

Classroom Exercise

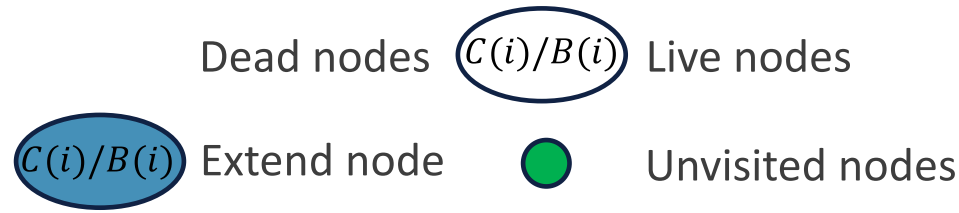
- Draw the pruned solution space tree for the following container loading problem instance by FIFO branch-and-bound and max-profit branch-and-bound.

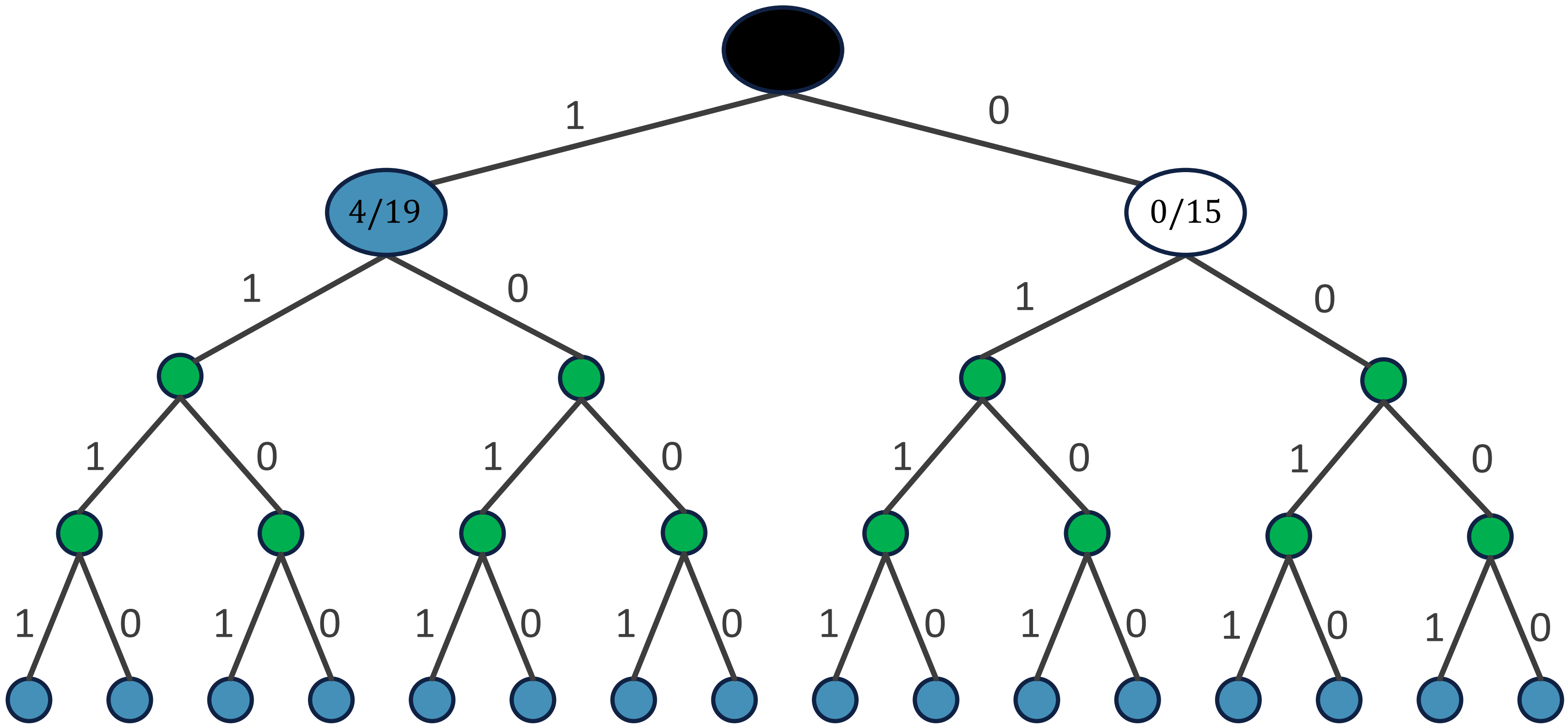
$$n = 4, w = [4,7,5,3], W = 15$$

- Compare these two results with the one solved by backtracking.

Q:

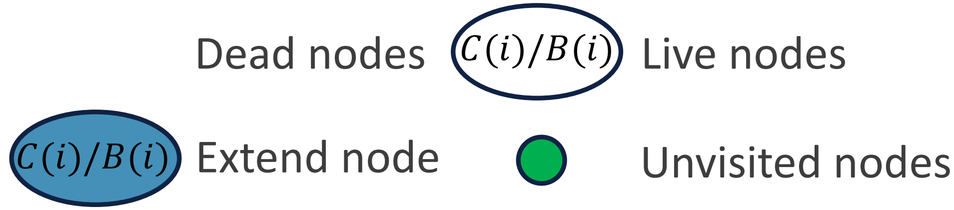
19	15		

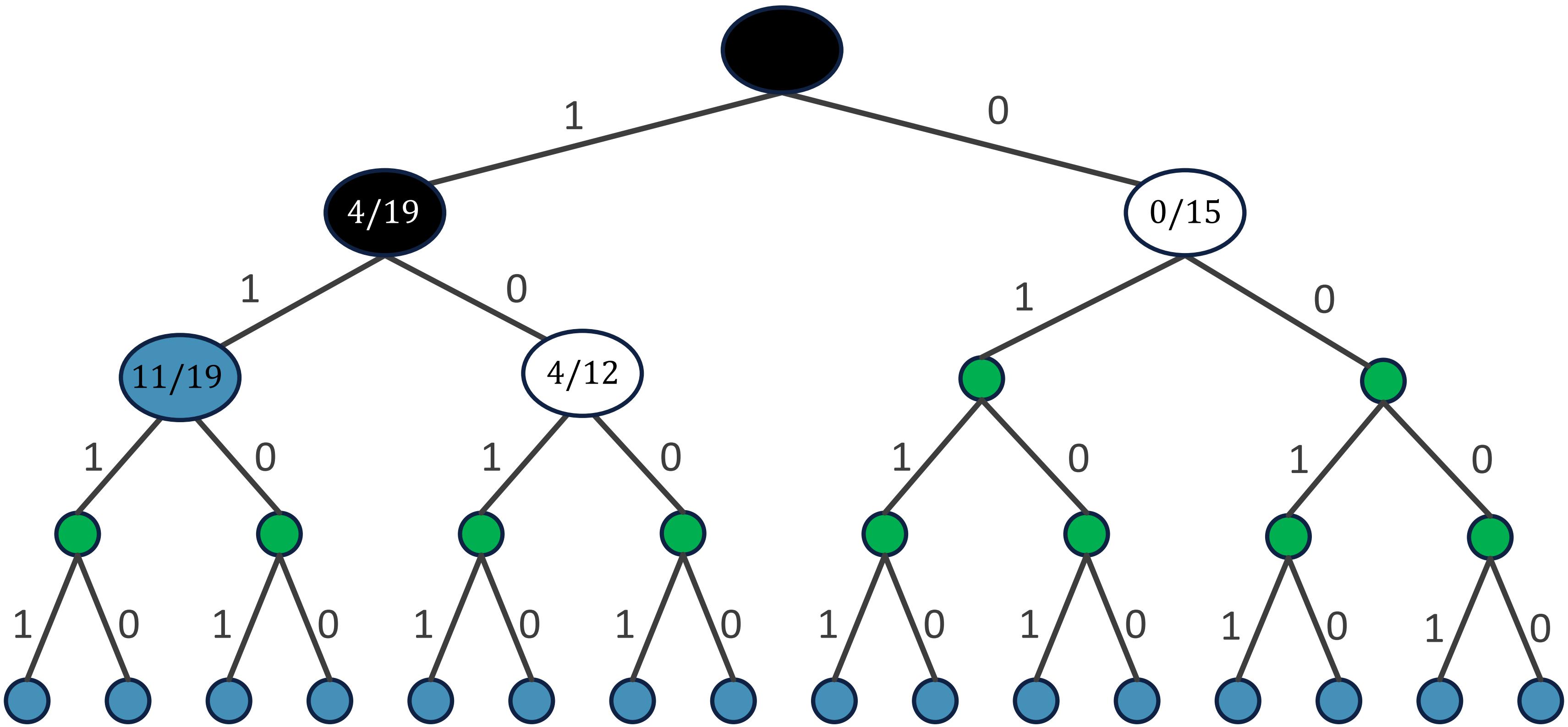




Max-profit branch-and-bound for $n = 4, w = [4,7,5,3], W = 15$

Q: | 19 | 15 | 12 | |
|---|---|---|---|

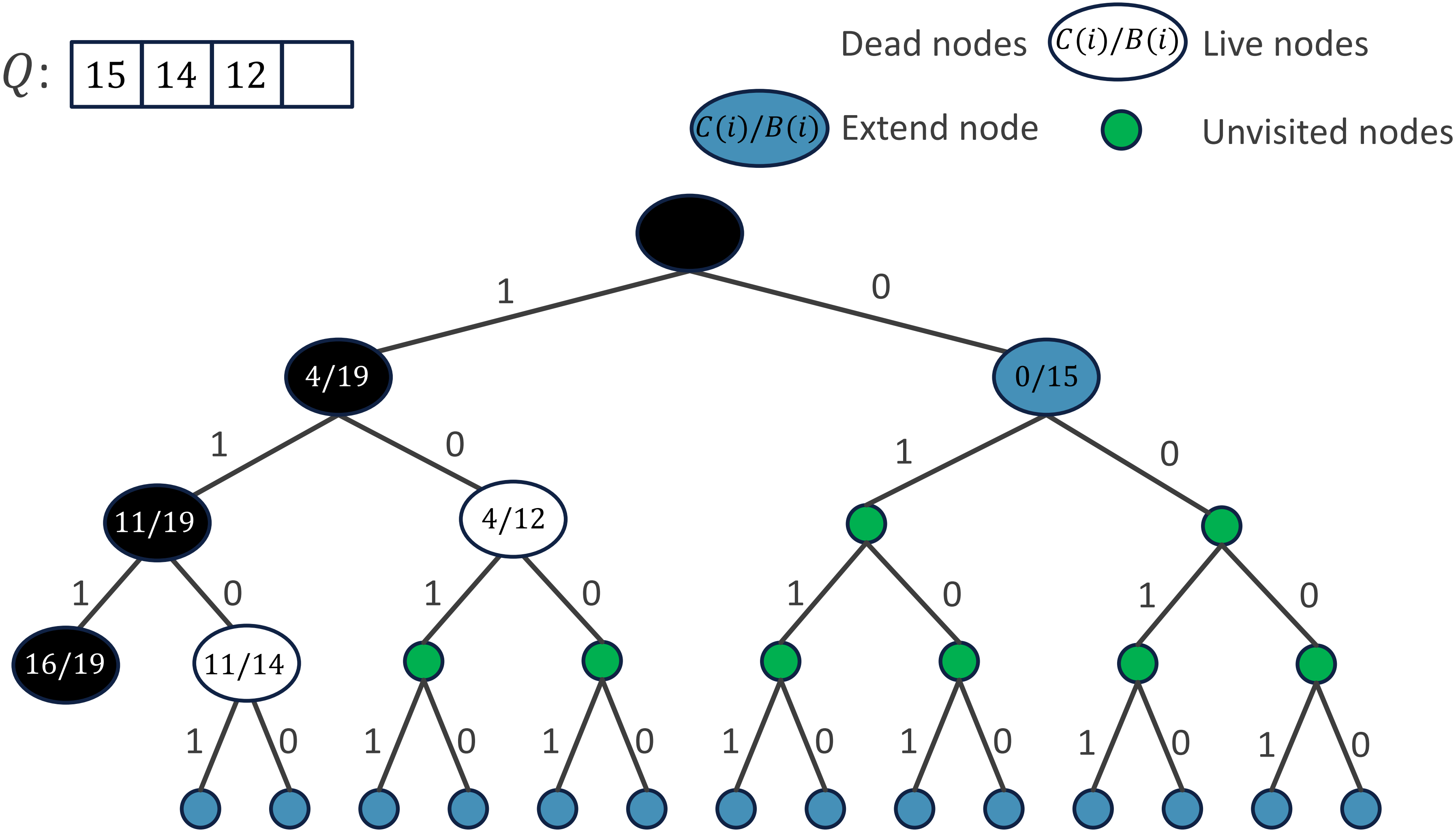




Max-profit branch-and-bound for $n = 4, w = [4,7,5,3], W = 15$

Q:

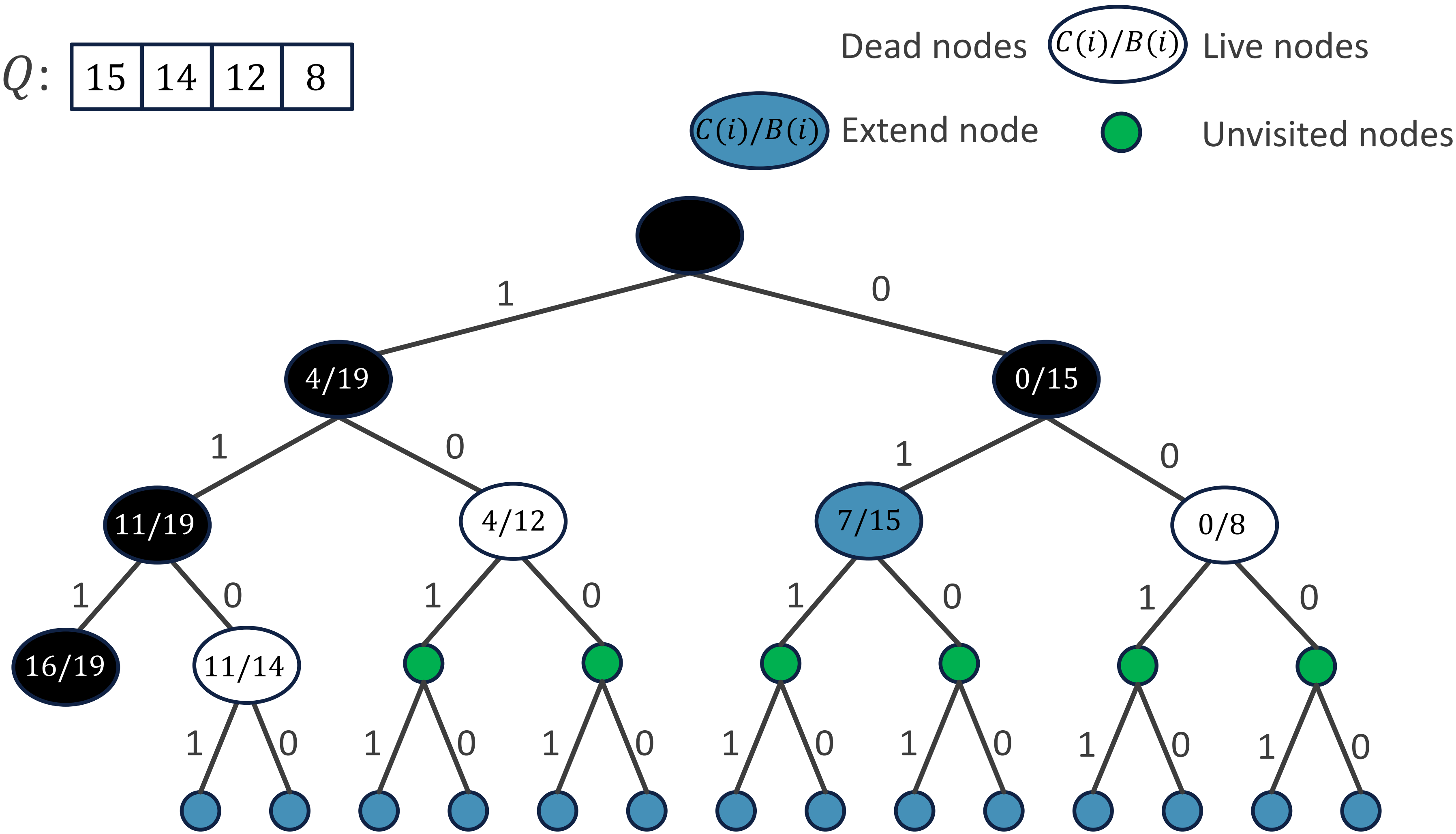
15	14	12	



Max-profit branch-and-bound for $n = 4, w = [4,7,5,3], W = 15$

Q:

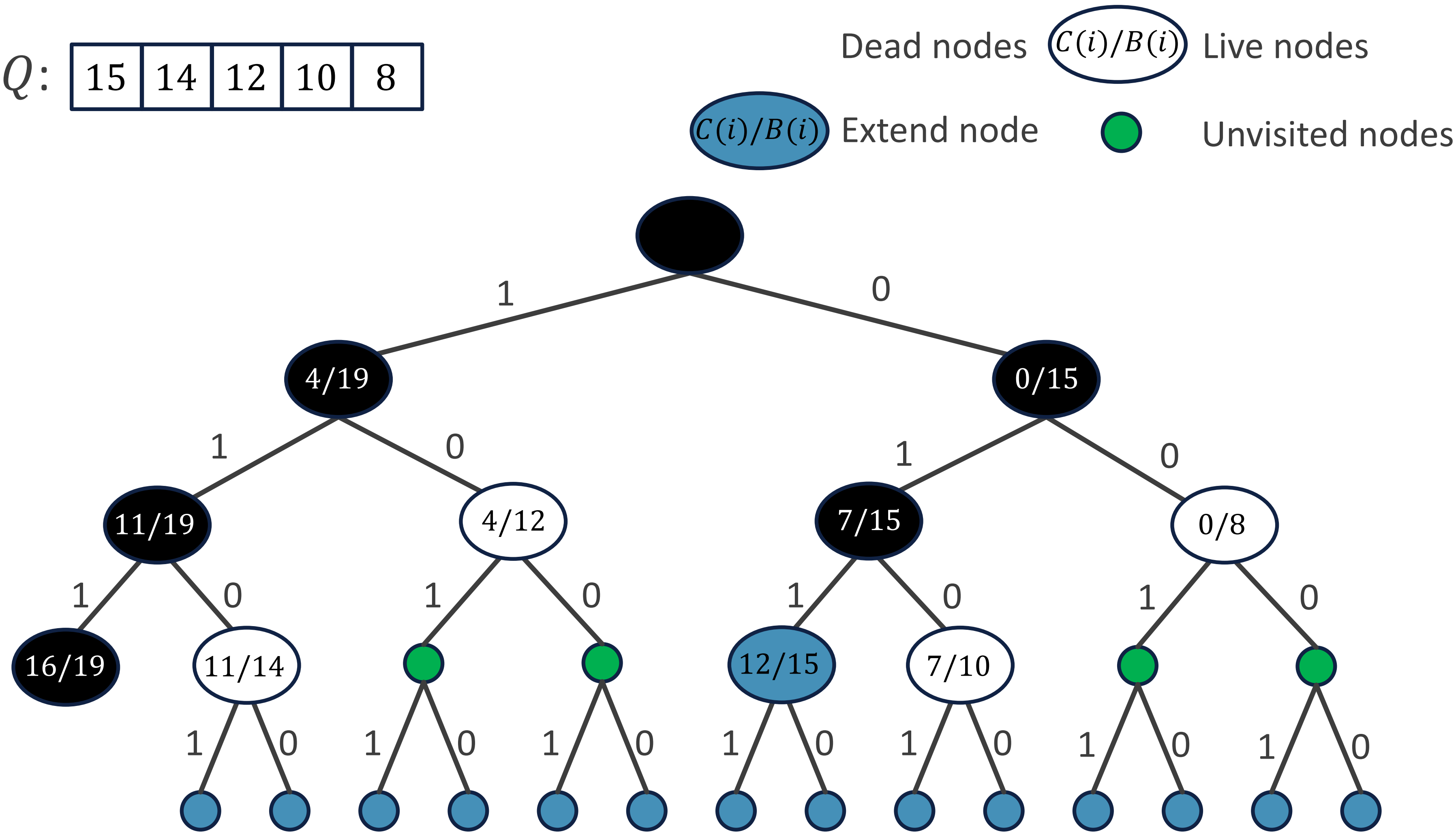
15	14	12	8



Max-profit branch-and-bound for $n = 4, w = [4,7,5,3], W = 15$

Q:

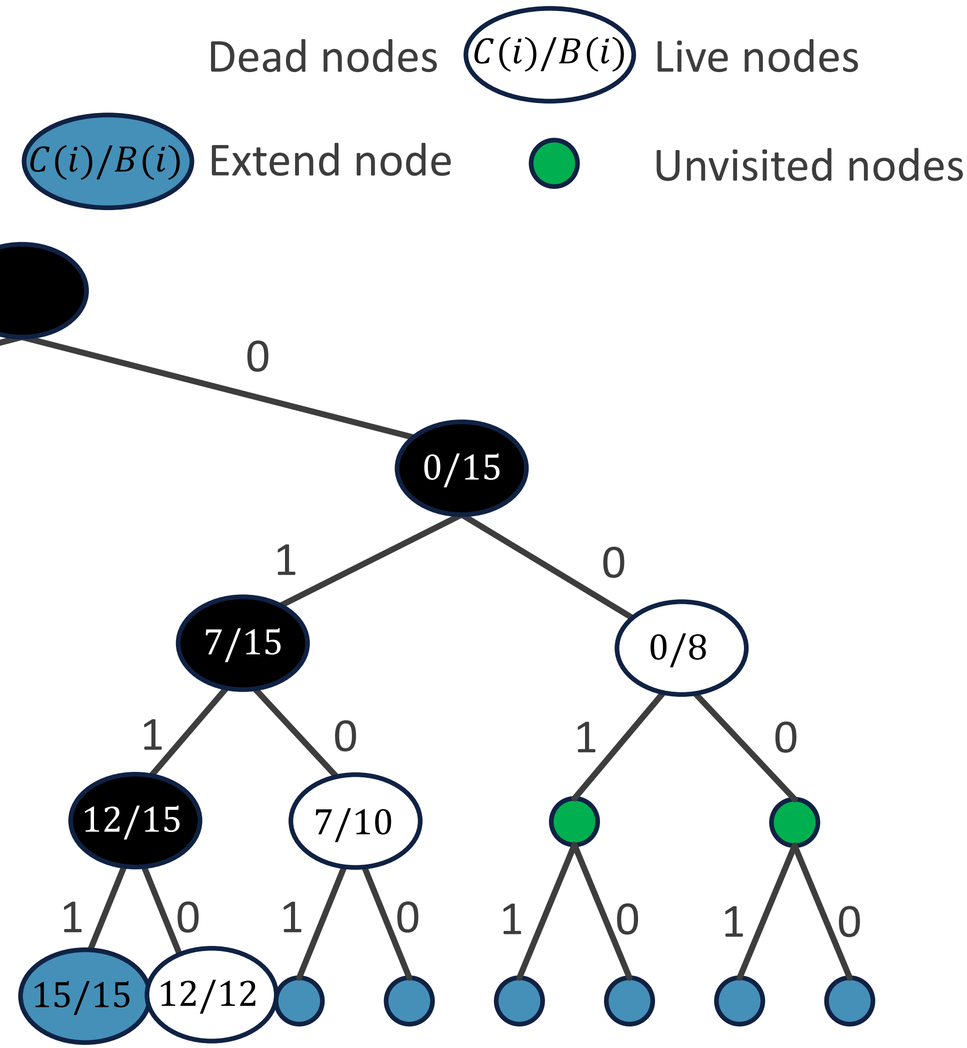
15	14	12	10	8



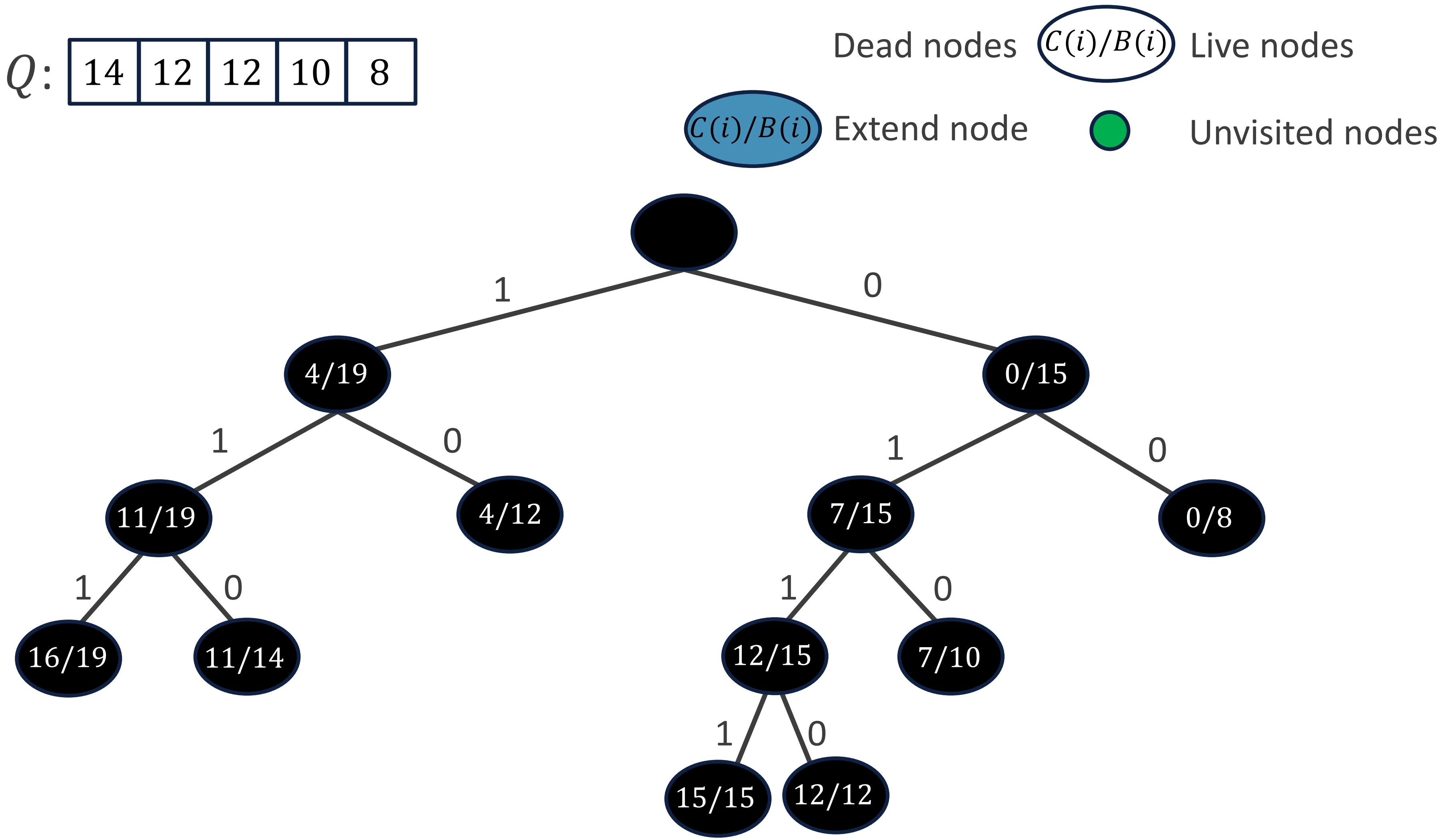
Max-profit branch-and-bound for $n = 4, w = [4,7,5,3], W = 15$

Q:

15	14	12	12	10	8

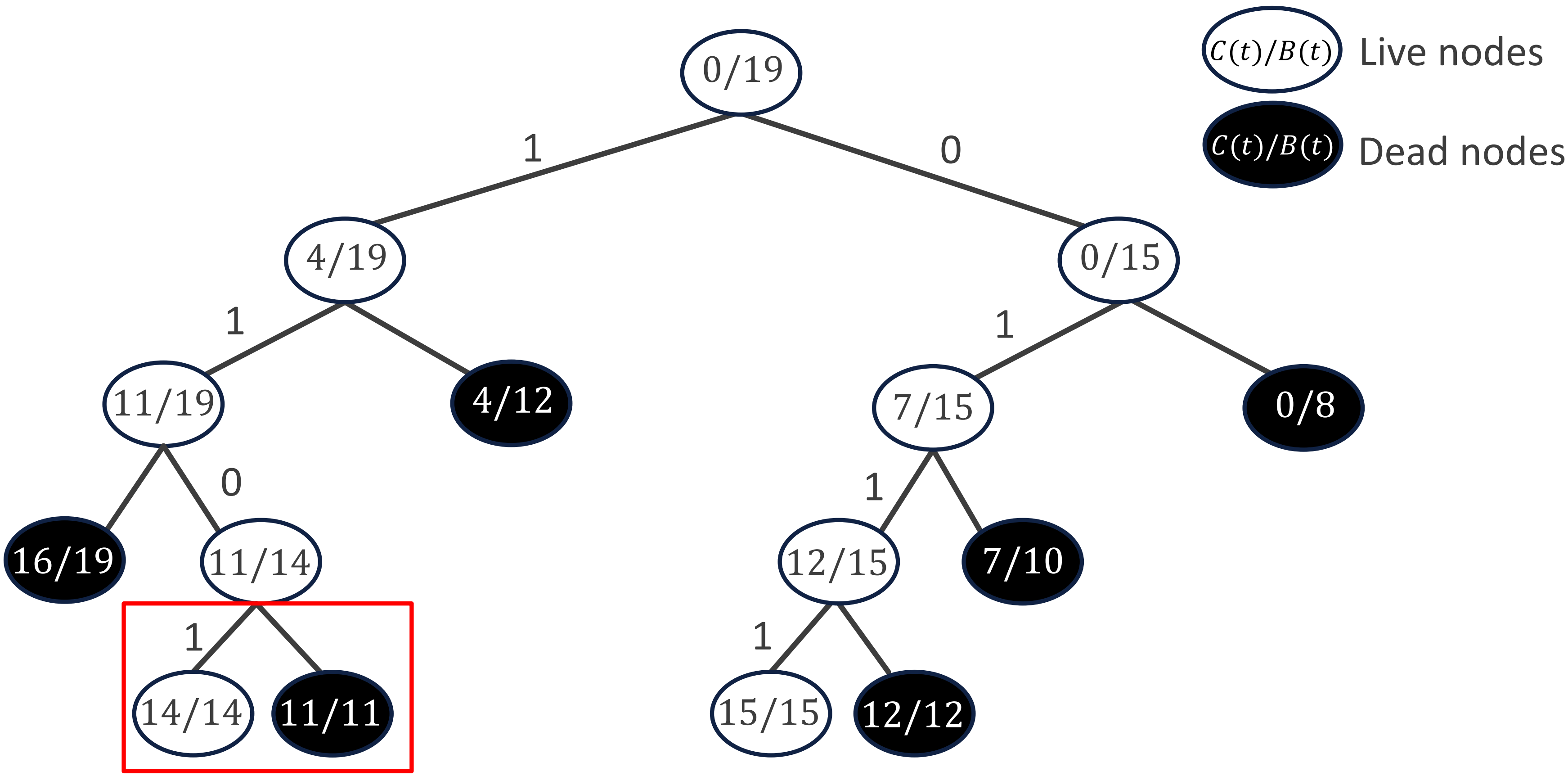


Max-profit branch-and-bound for $n = 4, w = [4,7,5,3], W = 15$



Max-profit branch-and-bound for $n = 4, w = [4,7,5,3], W = 15$

Classroom Exercise



Saved

Backtracking for $n = 4, w = [4,7,5,3], W = 15$

0/1 KNAPSACK PROBLEM

0/1 Knapsack Problem

- Constraint function and bounding function are same as the ones used in backtracking.
- Now, we use max-profit branch-and-bound to solve.

Pseudocode

```
MaxProfitKnapsack()
    i ← 1
    uv ← B(1);  bestv ← 0
    while i ≠ n + 1 do
        if C(i) ≤ W then
            if cv + v[i] > bestv then bestv ← cv + v[i]
            AddLiveNode(uv, cv + v[i], C(i), 1, i + 1)
            uv ← B(i)
        if B(i) ≥ bestv then
            AddLiveNode(B(i), cv, cw, 0, i + 1)
        N ← ExtractMax(Q); E ← N.ptr;  cw ← N.weight
        cv ← N.value; uv ← N.upvalue;  i ← N.level
    for j ← n to 1 do
        bestx[j] ← E.LChild;  E ← E.parent
    return bestv
```

uv: upper value calculated by $B(i)$

Add bounding condition. It is also ok it we don't add it.

Key of max-priority queue

Example

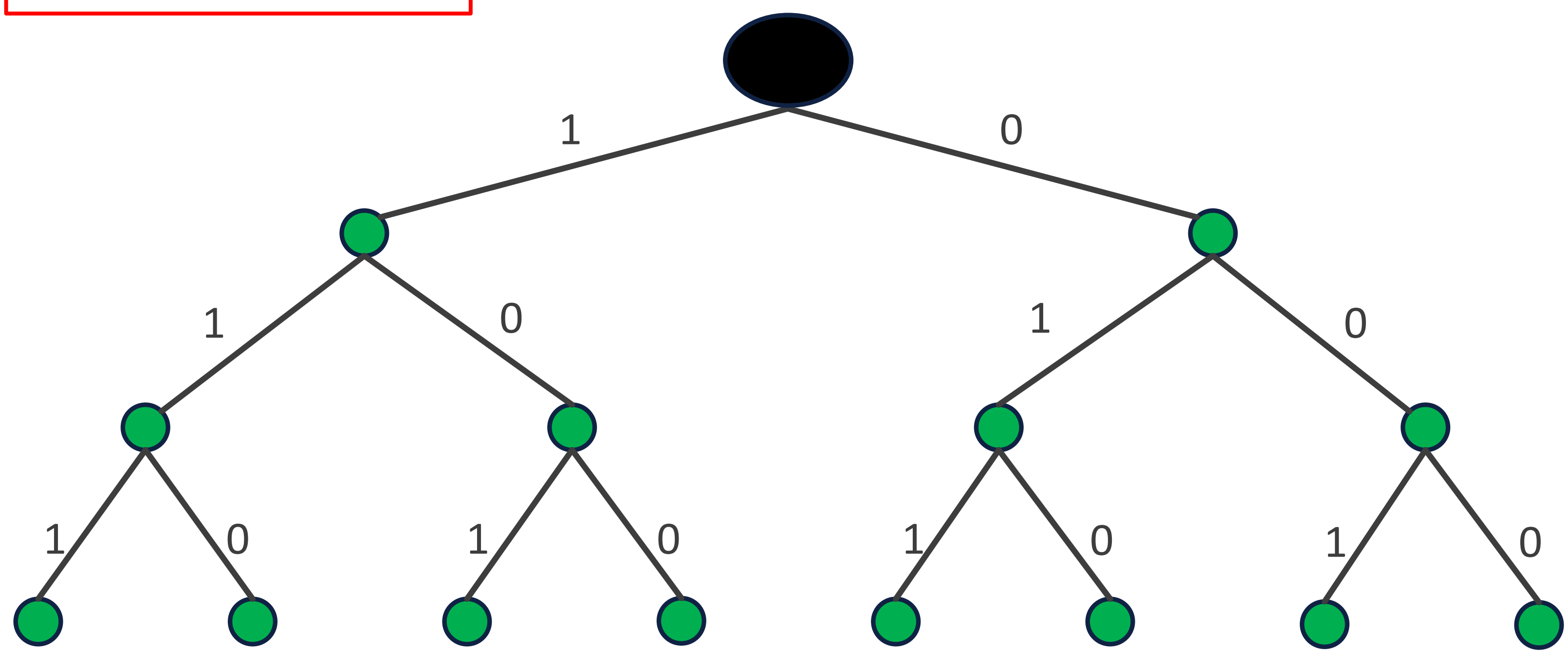
$Q:$

Initialization (Line 1-2)

$bestv = 0$

$C(i)/B(i)$ Dead nodes $C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node Unvisited nodes



$$n = 3, w = [20,15,15], v = [40,25,25], W = 30$$

Example

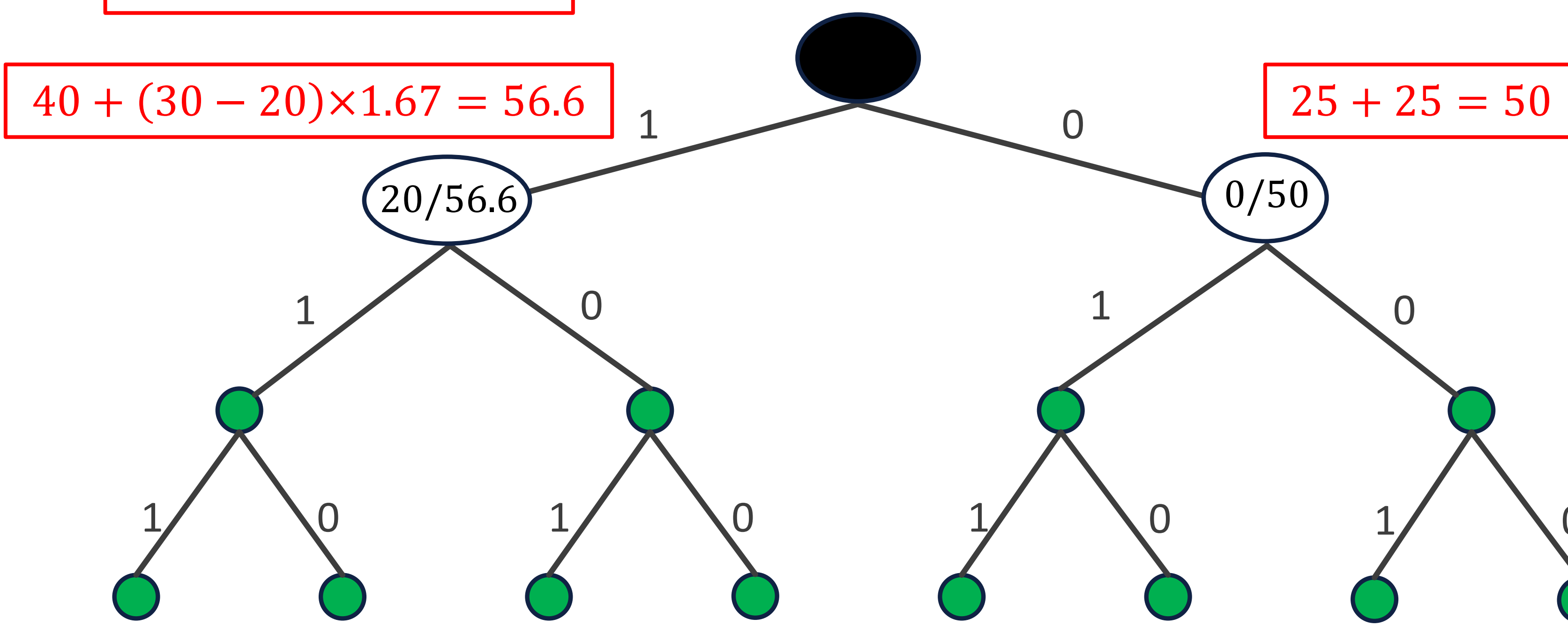
Q: | 56.6 | 50 | | |

$bestv = 40$

$C(i)/B(i)$ Dead nodes $C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node Unvisited nodes

AddLiveNode (Line 4-9)



$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$

Example

Q: | 50 | | | |

ExtractMax: 56.6

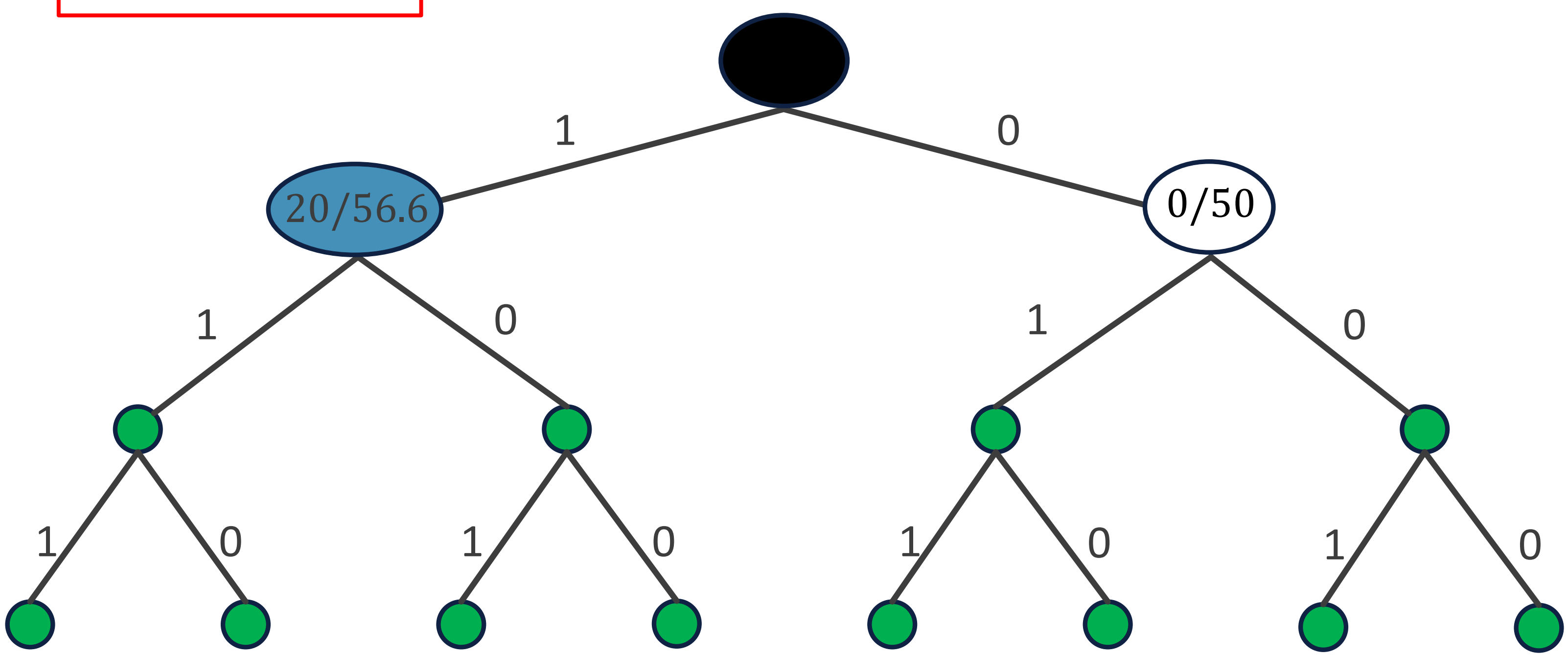
$bestv = 40$

$C(i)/B(i)$ Dead nodes

$C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node

Unvisited nodes



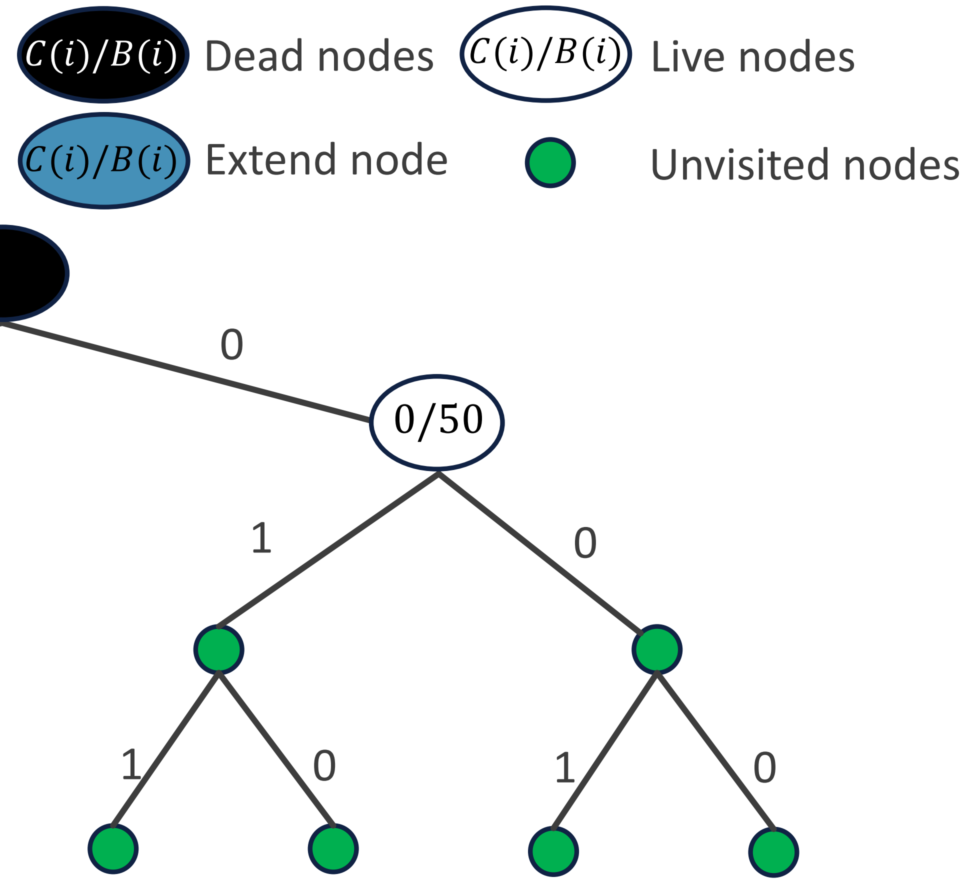
$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$

Example

Q: | 56.6 | 50 | | |

AddLiveNode (Line 4-9)

$bestv = 40$



$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$

Example

Q: | 50 | | | |

ExtractMax: 56.6

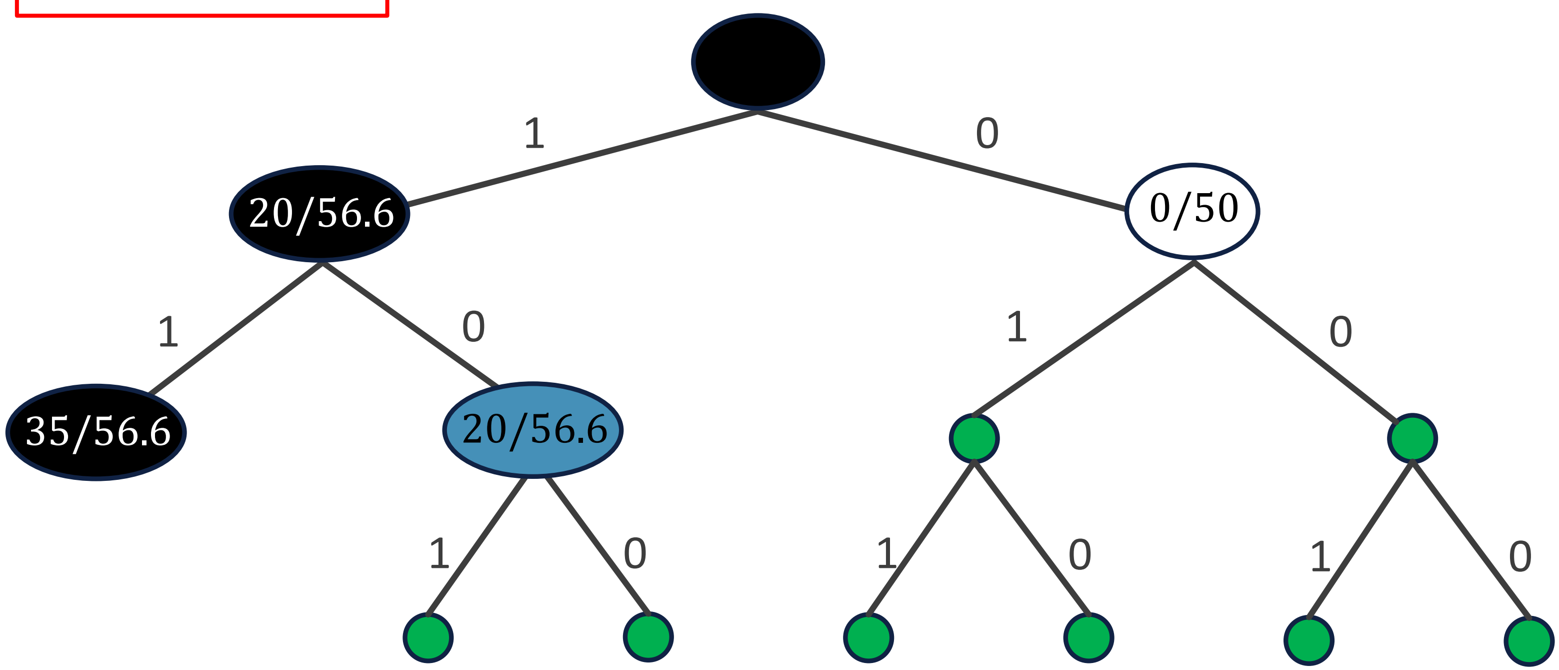
$bestv = 40$

$C(i)/B(i)$ Dead nodes

$C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node

Unvisited nodes



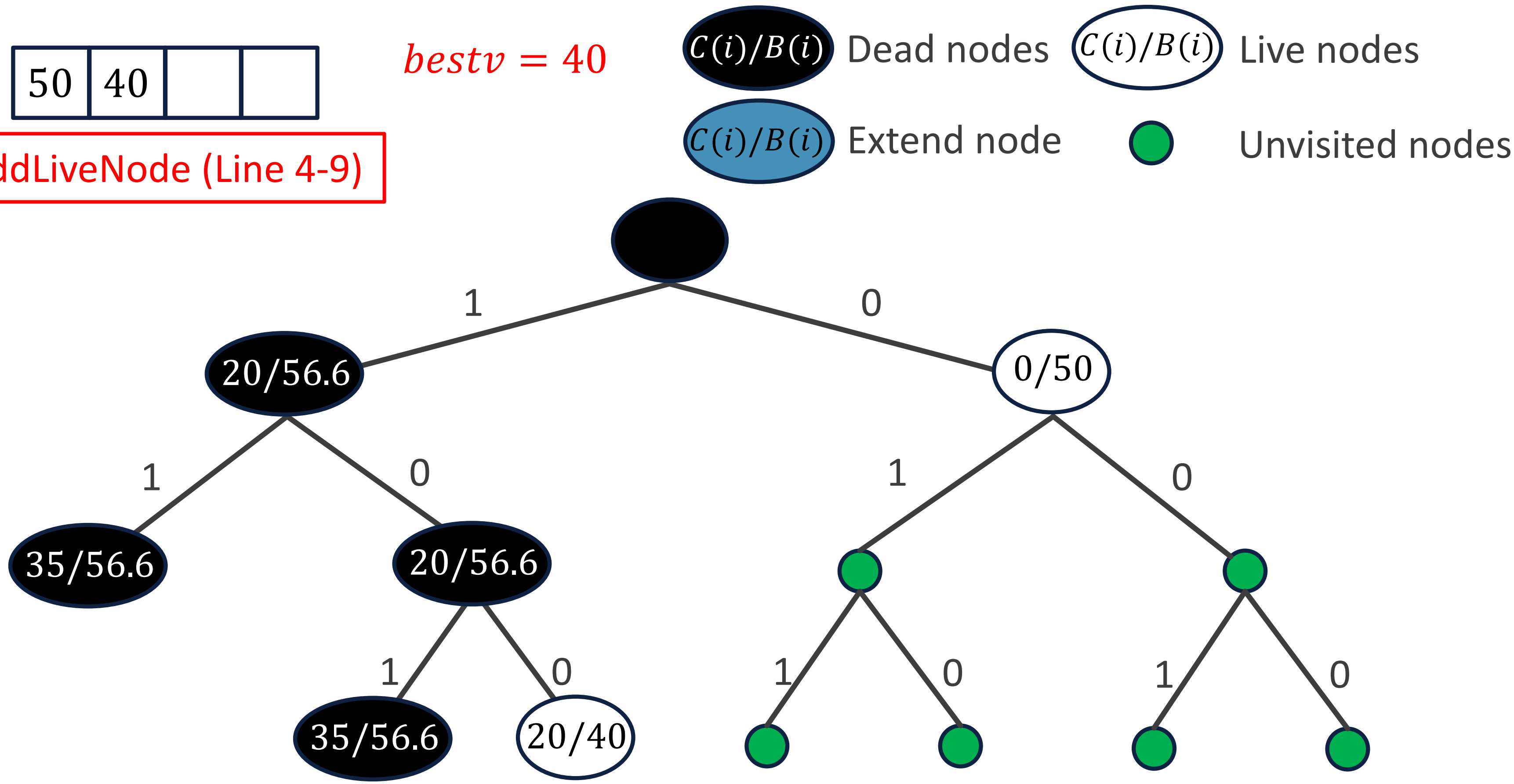
$$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$$

Example

Q: | 50 | 40 | | |

AddLiveNode (Line 4-9)

$bestv = 40$



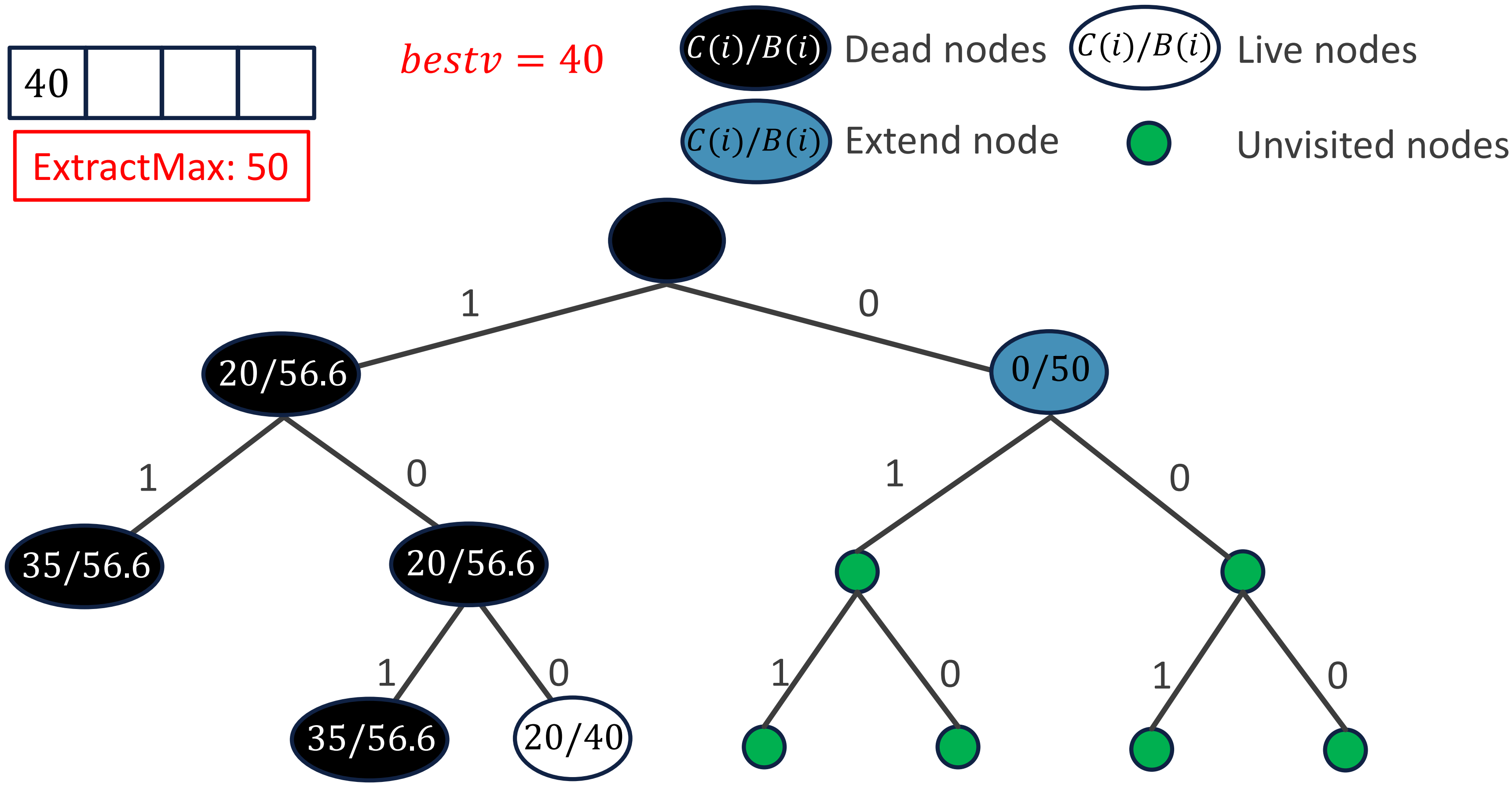
$$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$$

Example

Q: | 40 | | | |

ExtractMax: 50

$bestv = 40$



$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$

Example

Q: | 50 | 40 | | |

AddLiveNode (Line 4-9)

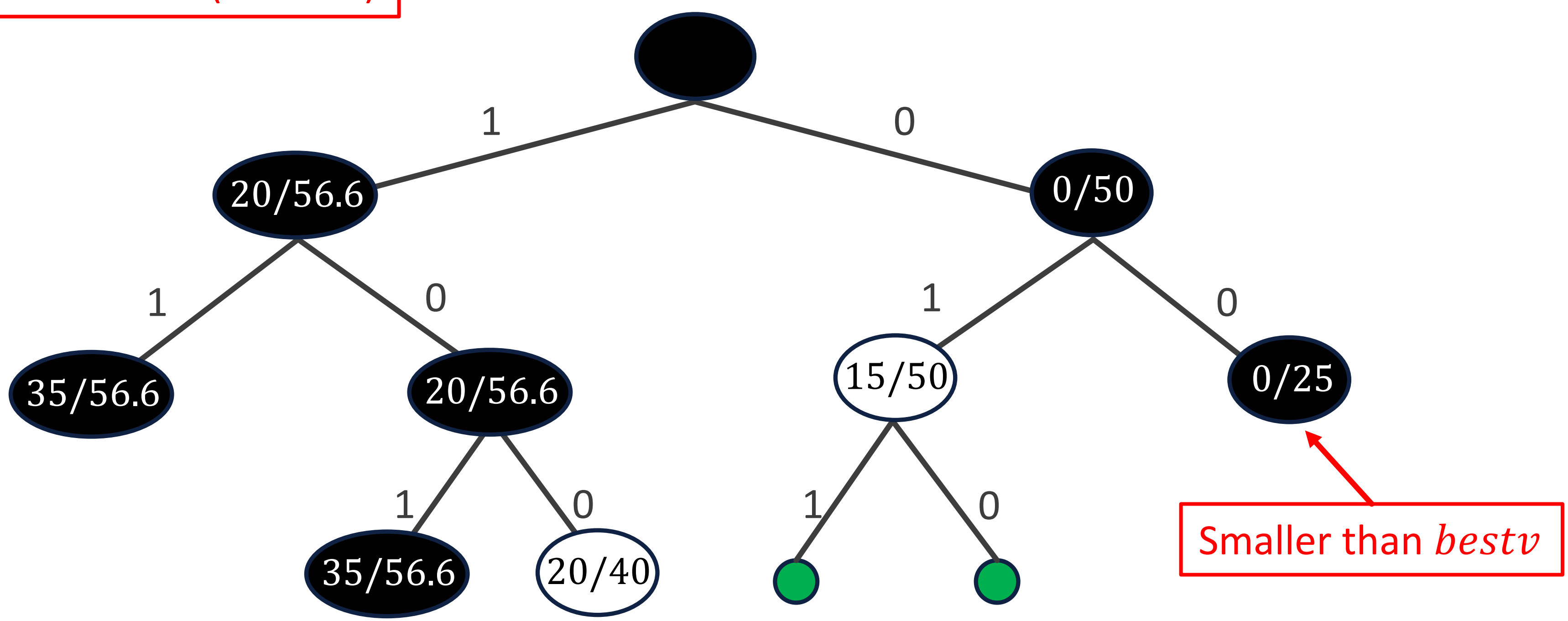
$bestv = 40$

$C(i)/B(i)$ Dead nodes

$C(i)/B(i)$ Live nodes

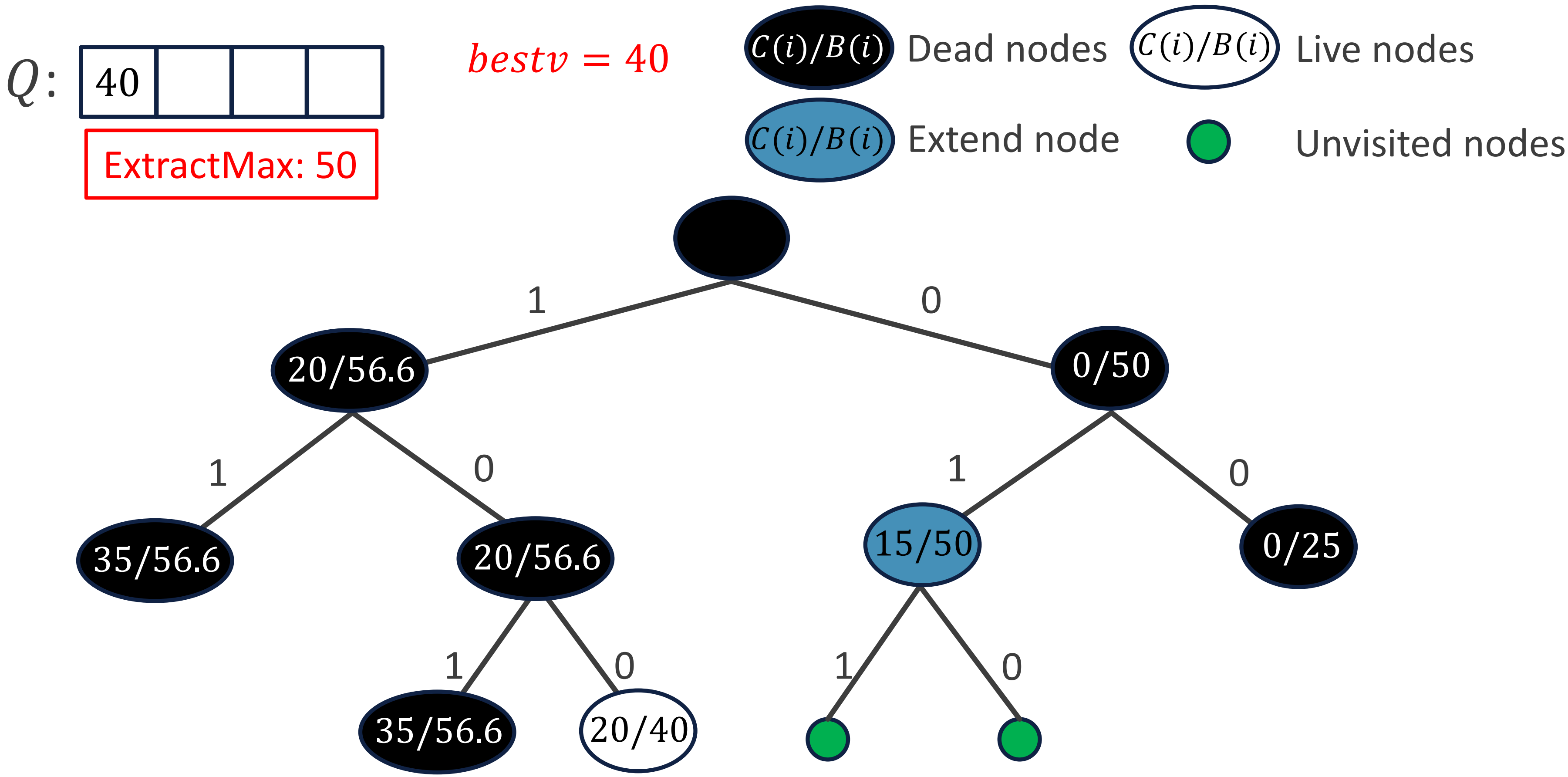
$C(i)/B(i)$ Extend node

Unvisited nodes



$$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$$

Example



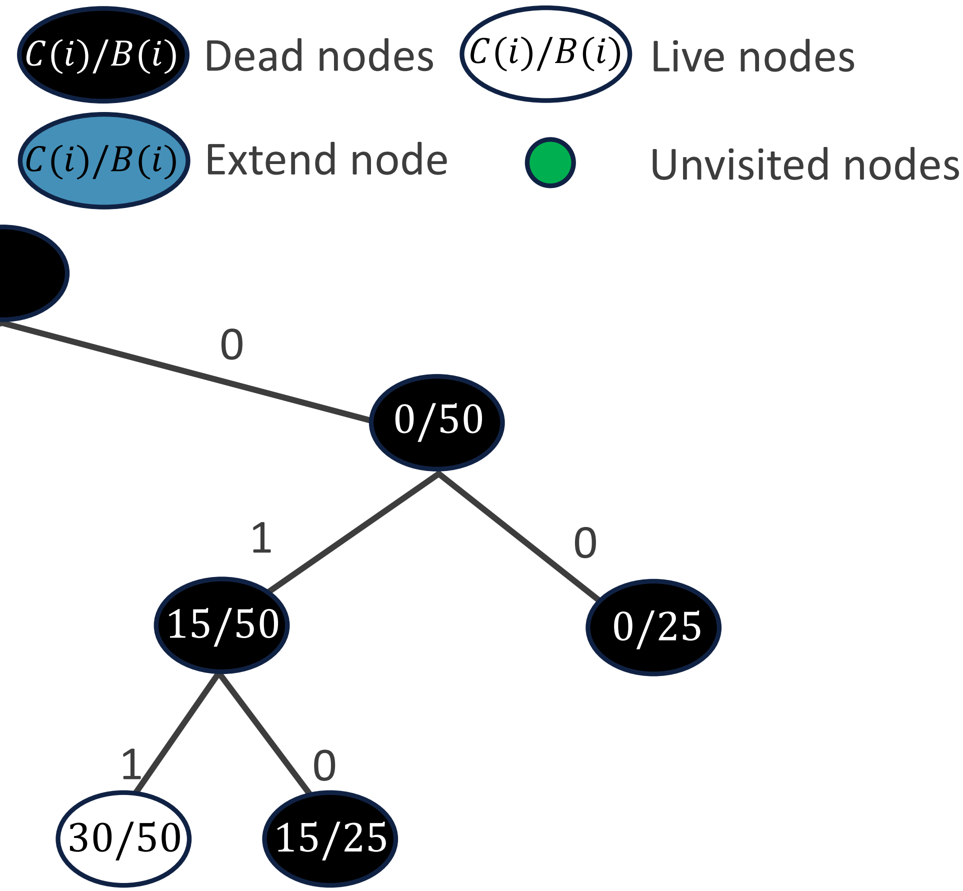
$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$

Example

Q: | 50 | 40 | | |

AddLiveNode (Line 4-9)

$bestv = 50$



$$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$$

Example

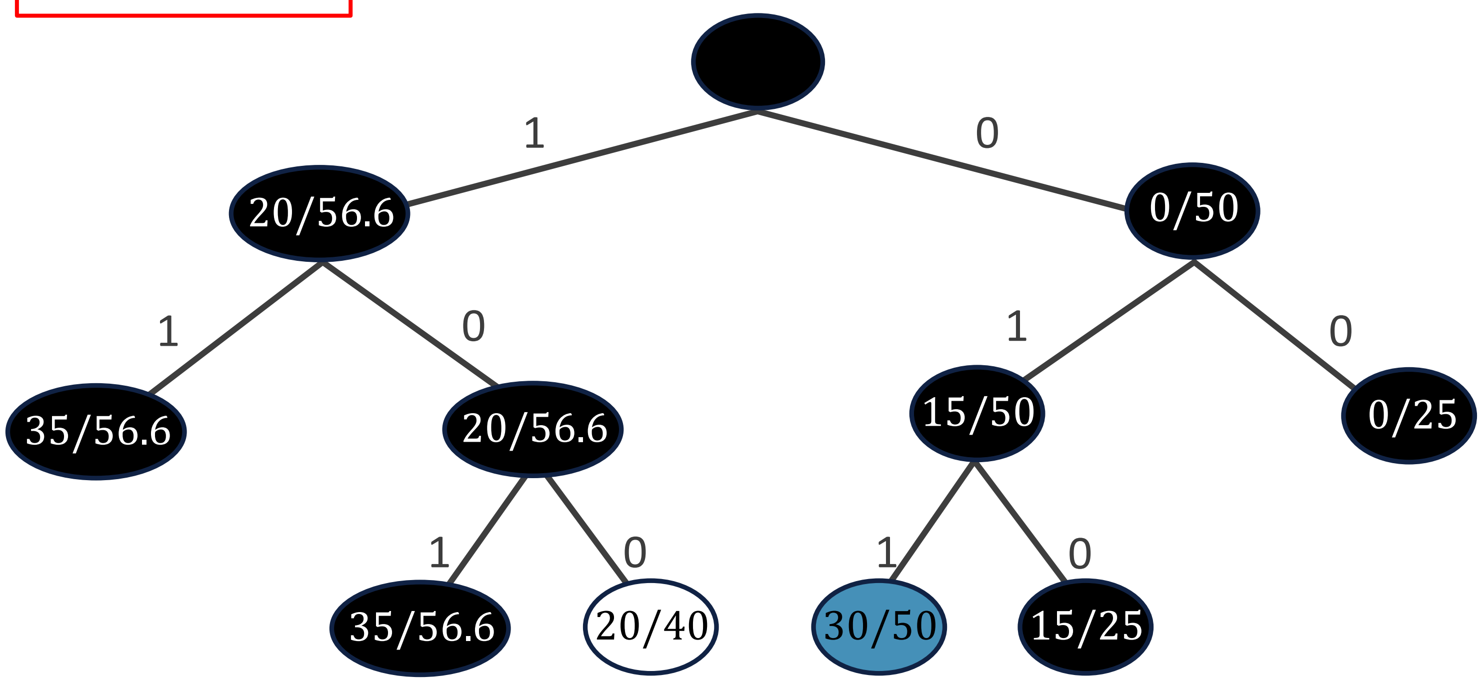
Q: | 40 | | | |

ExtractMax: 50

$bestv = 50$

$C(i)/B(i)$ Dead nodes $C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node Unvisited nodes



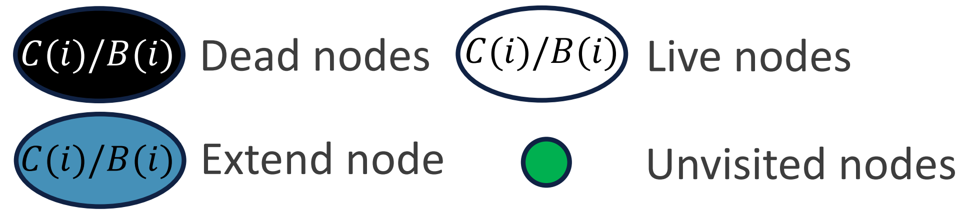
$$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$$

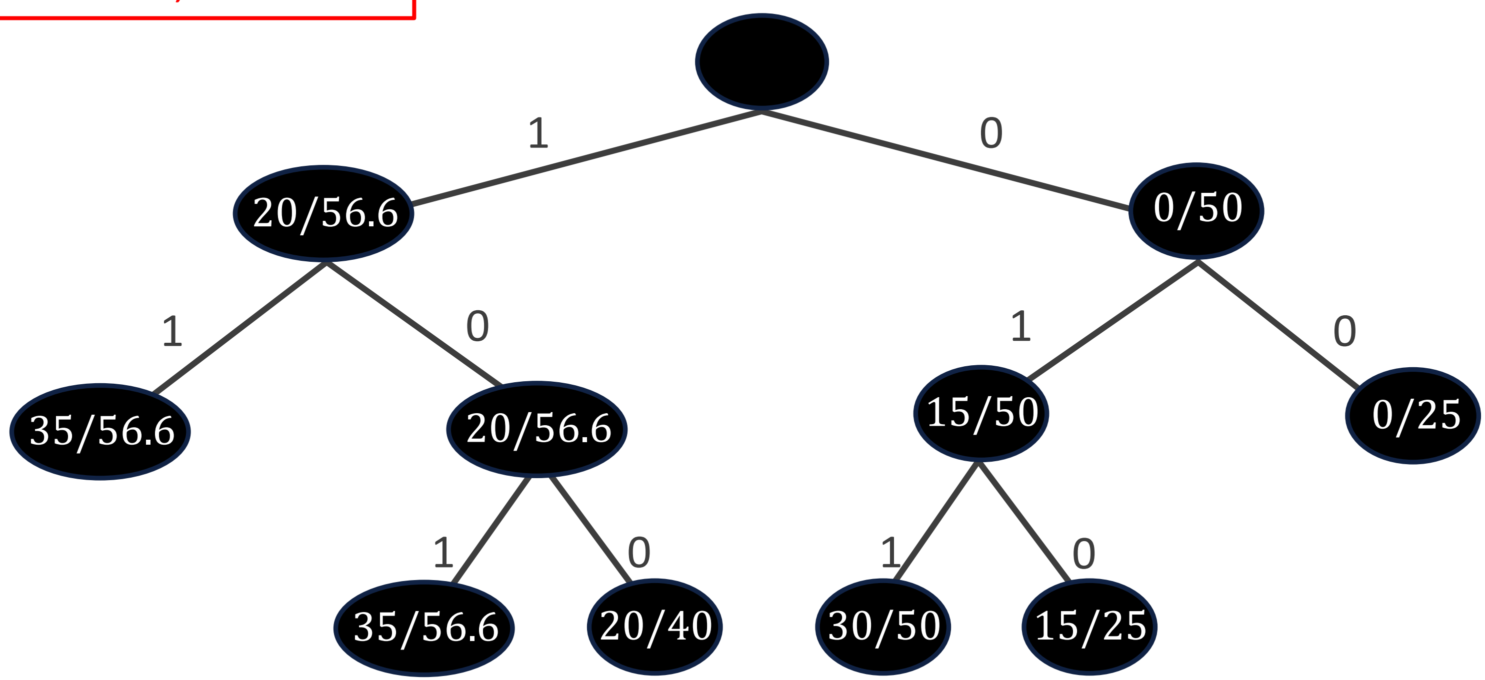
Example

Q: | 40 | | | |

$bestv = 50$

$i = n + 1$, terminate





$$n = 3, w = [20,15,15], v = [40,25,25], v/w = [2,1.67,1.67], W = 30$$

Classroom Exercise

- Draw the pruned solution space tree for the following container loading problem instance by max-profit branch-and-bound.

$$n = 3, v = [20,40,20], w = [2,5,4], W = 5$$

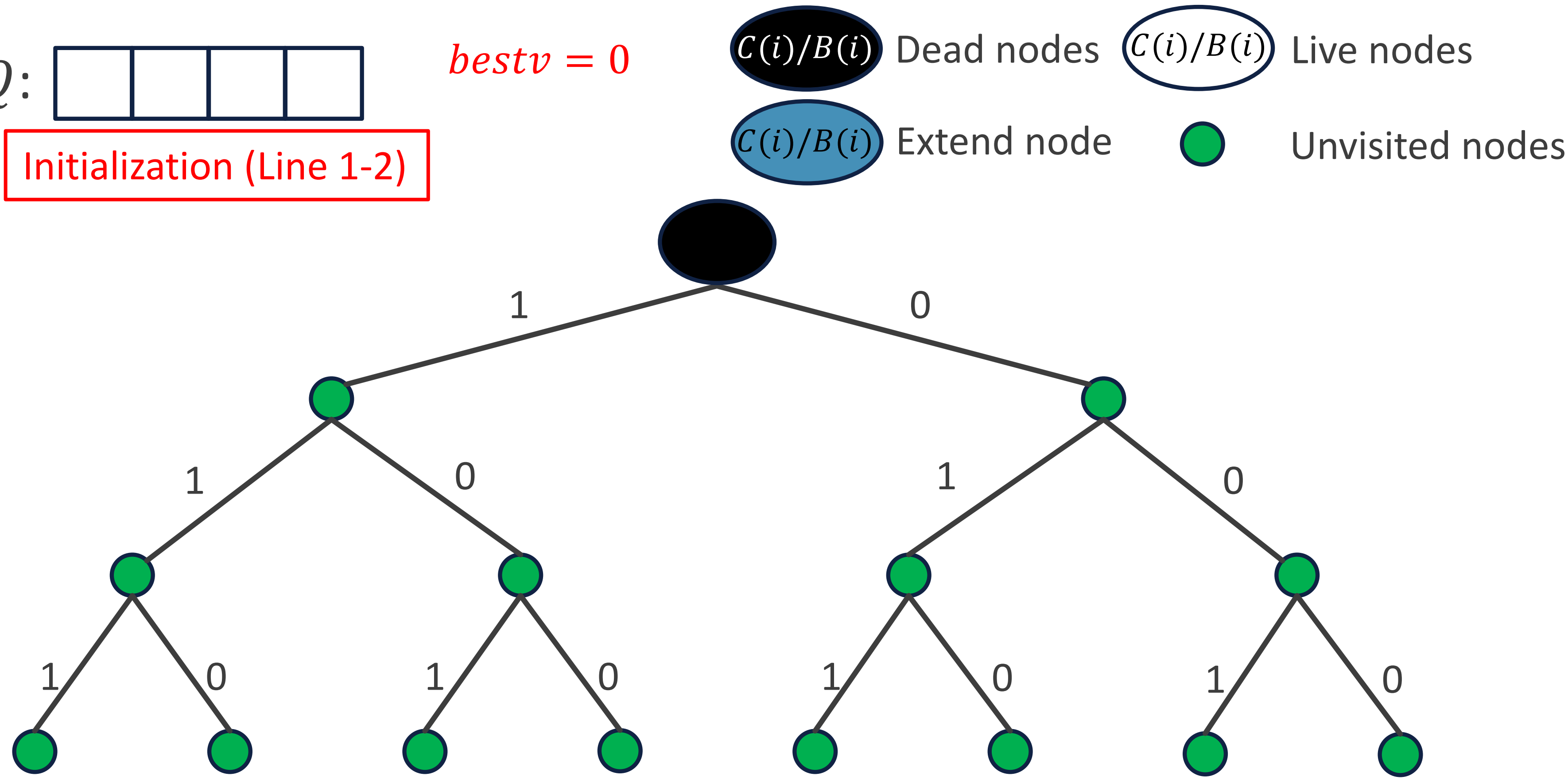
- Compare the result with the one solved by backtracking.

Classroom Exercise

Q:

Initialization (Line 1-2)

$bestv = 0$



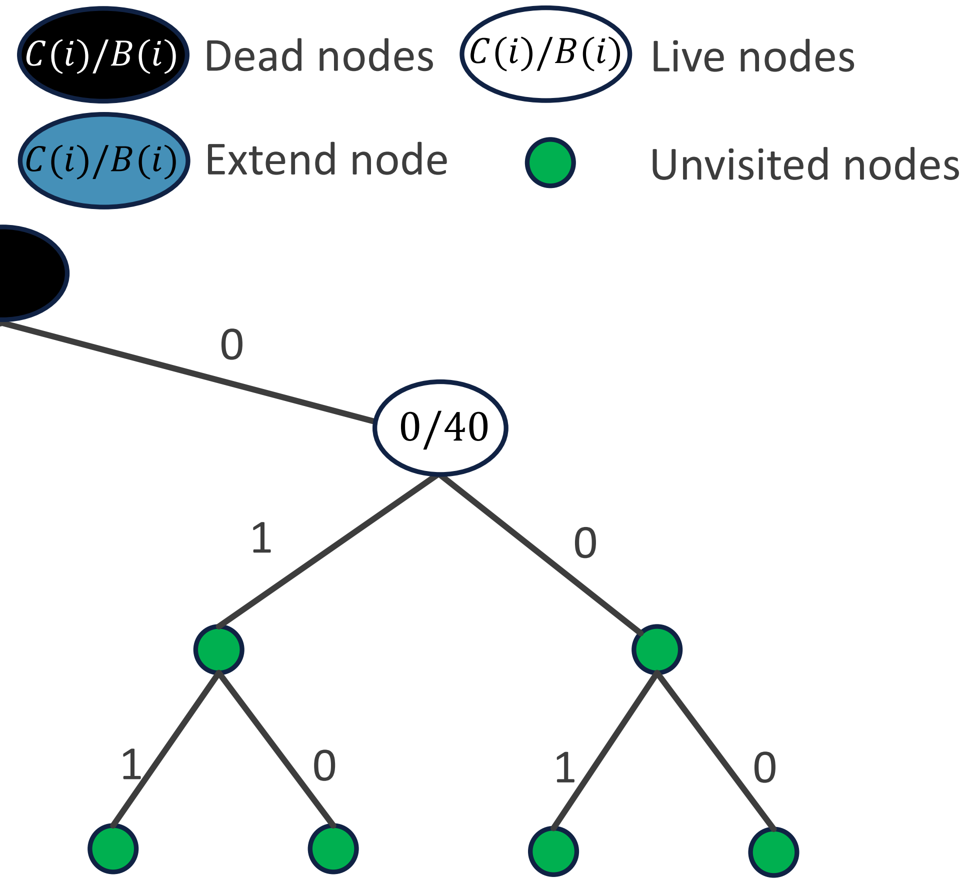
$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

Classroom Exercise

Q: | 44 | 40 | | |

$bestv = 20$

AddLiveNode (Line 4-9)



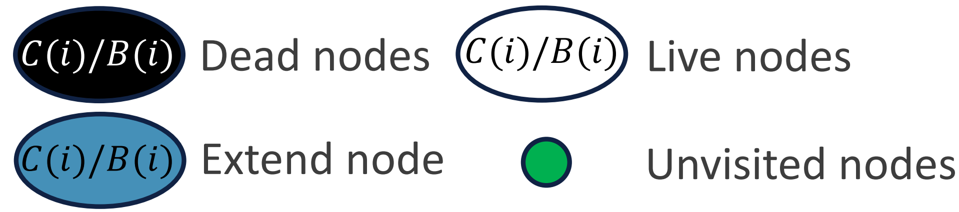
$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

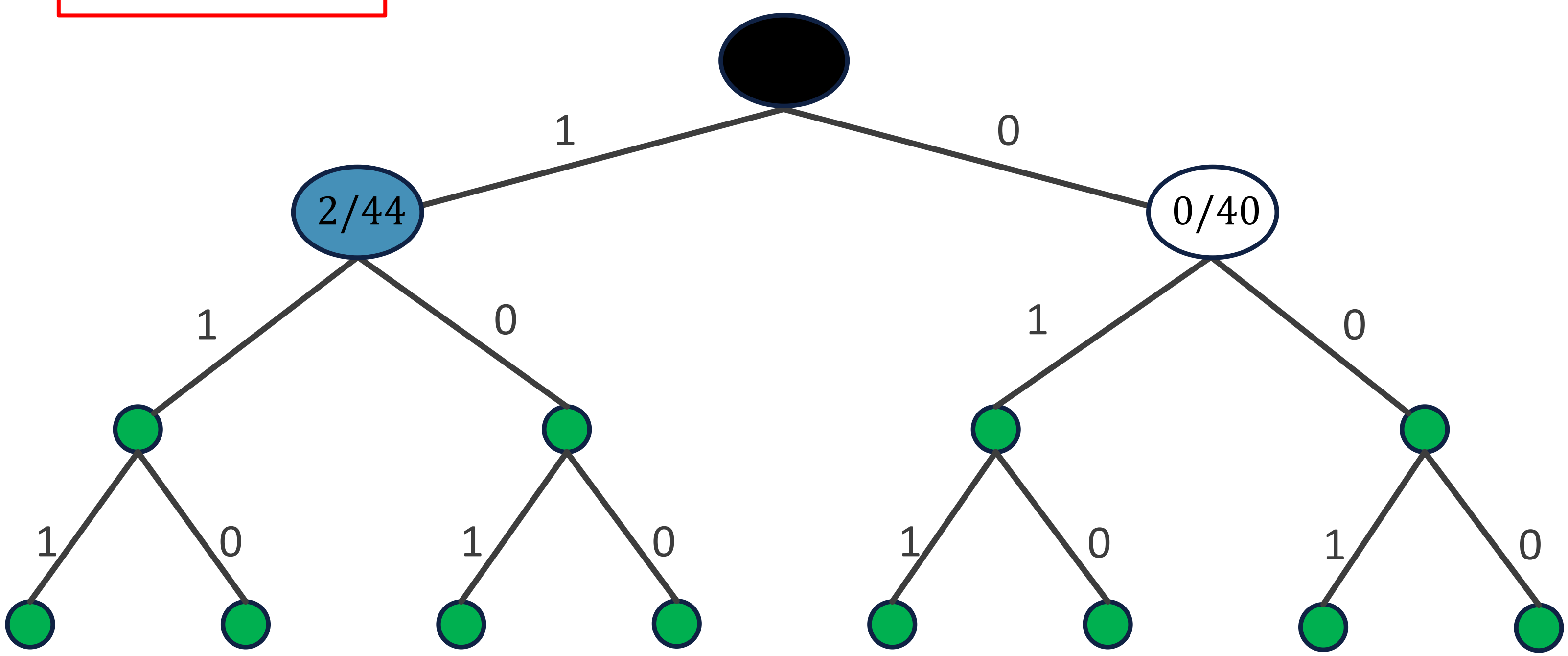
Classroom Exercise

Q: | 40 | | | |

ExtractMax: 44

$bestv = 20$





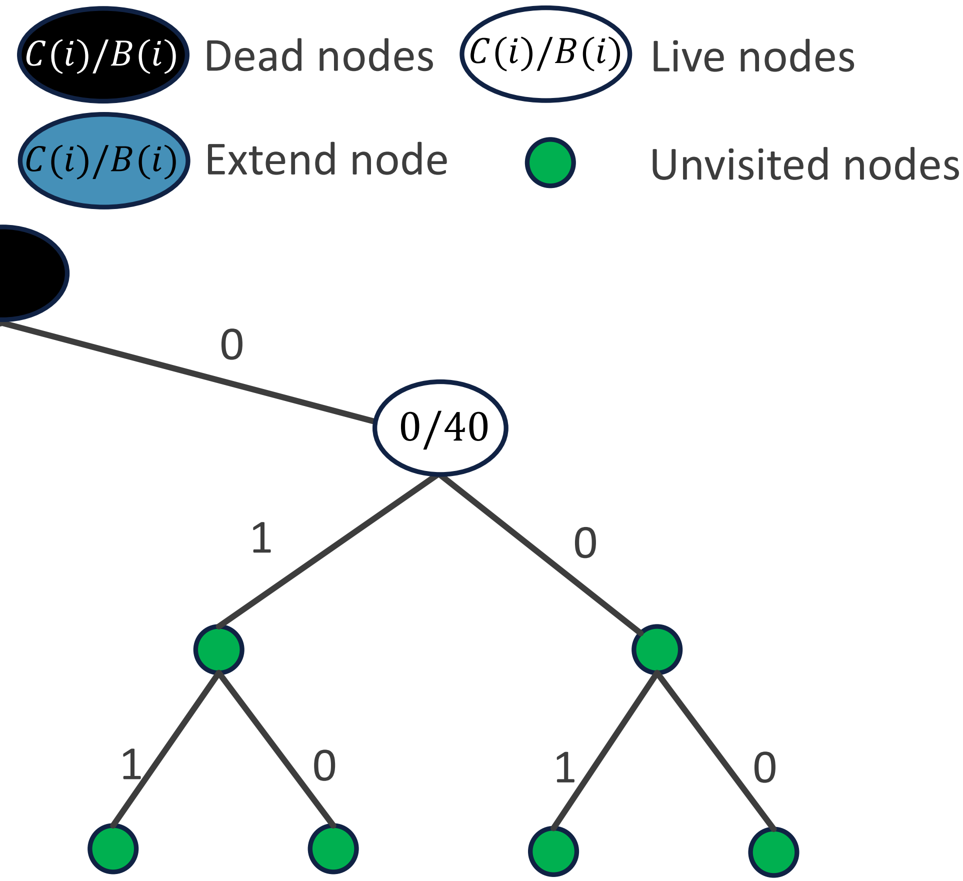
$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

Classroom Exercise

Q: | 40 | 35 | | |

AddLiveNode (Line 4-9)

$bestv = 20$



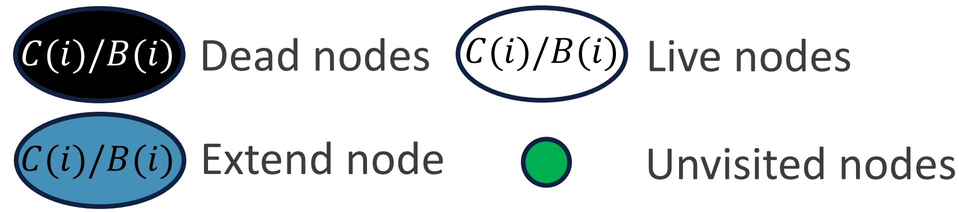
$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

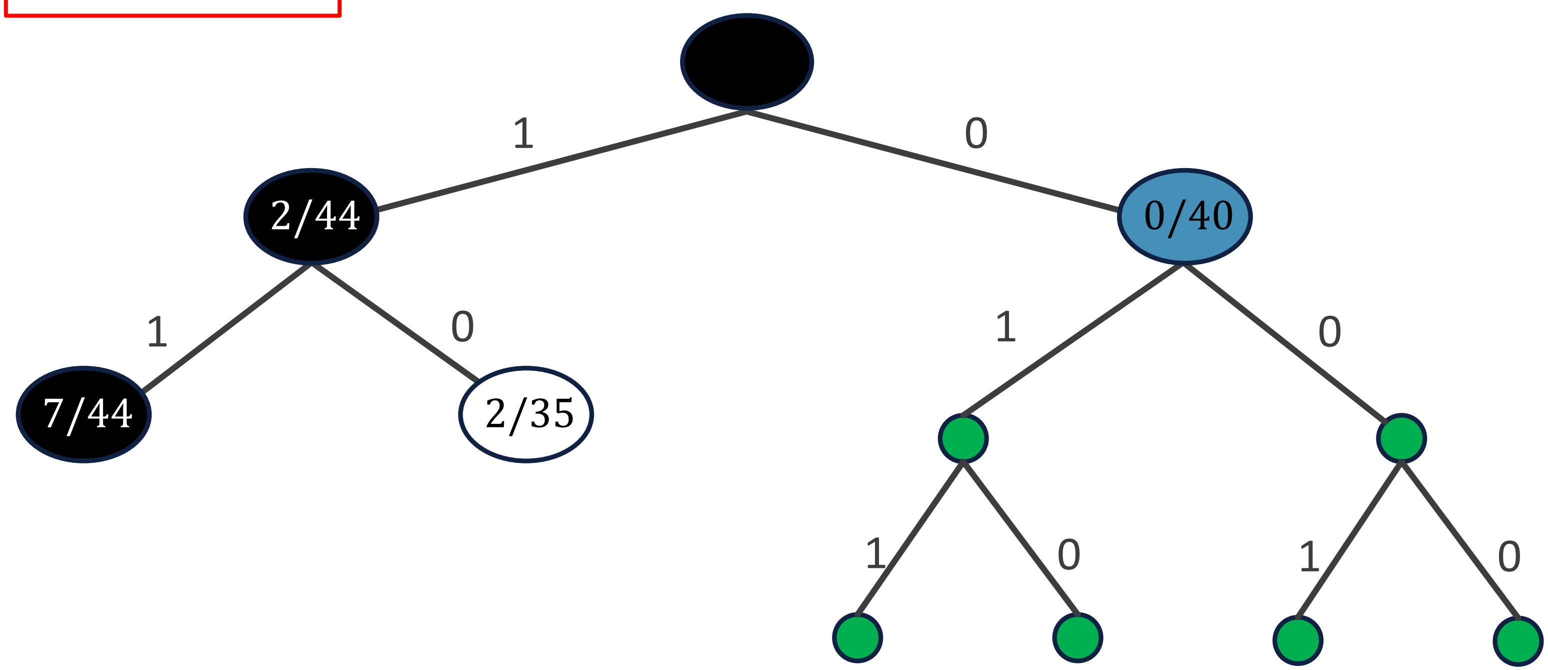
Classroom Exercise

Q: | 35 | | | |

ExtractMax: 40

$bestv = 20$





$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

Classroom Exercise

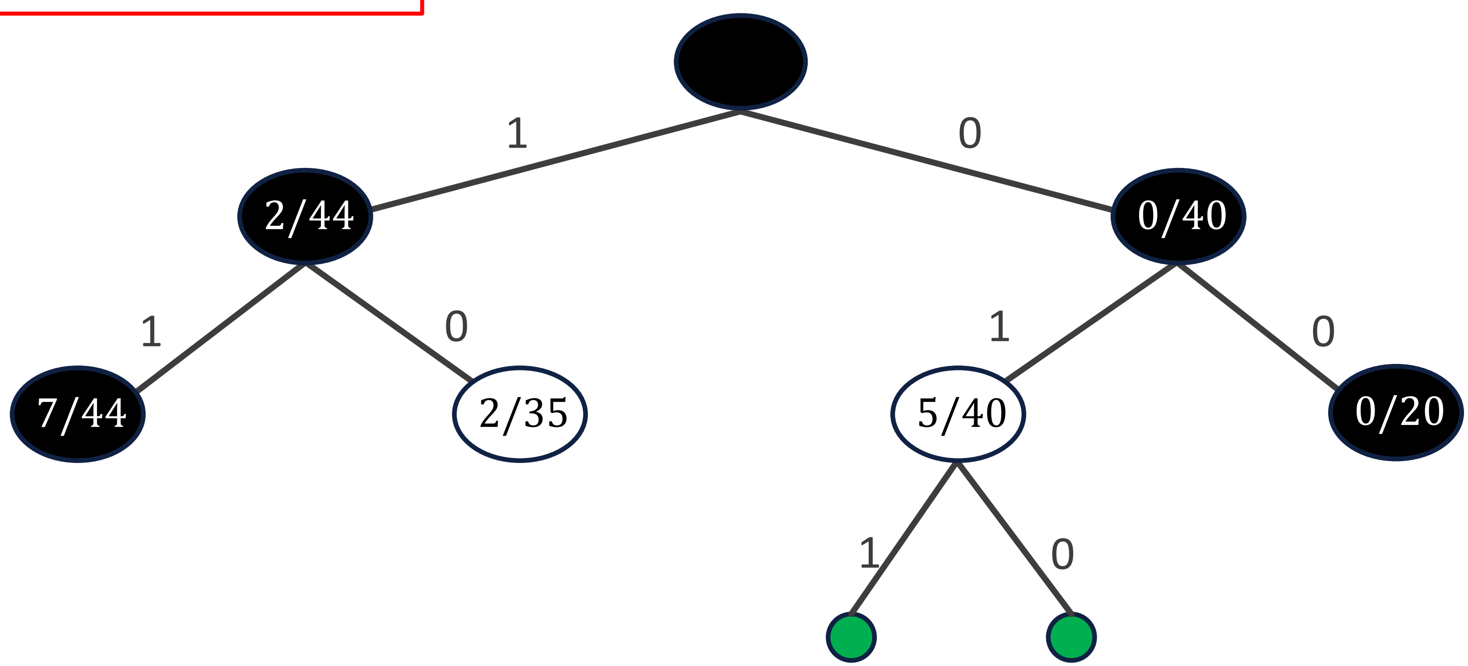
Q: | 40 | 35 | 20 | |

$bestv = 40$

AddLiveNode (Line 4-9)

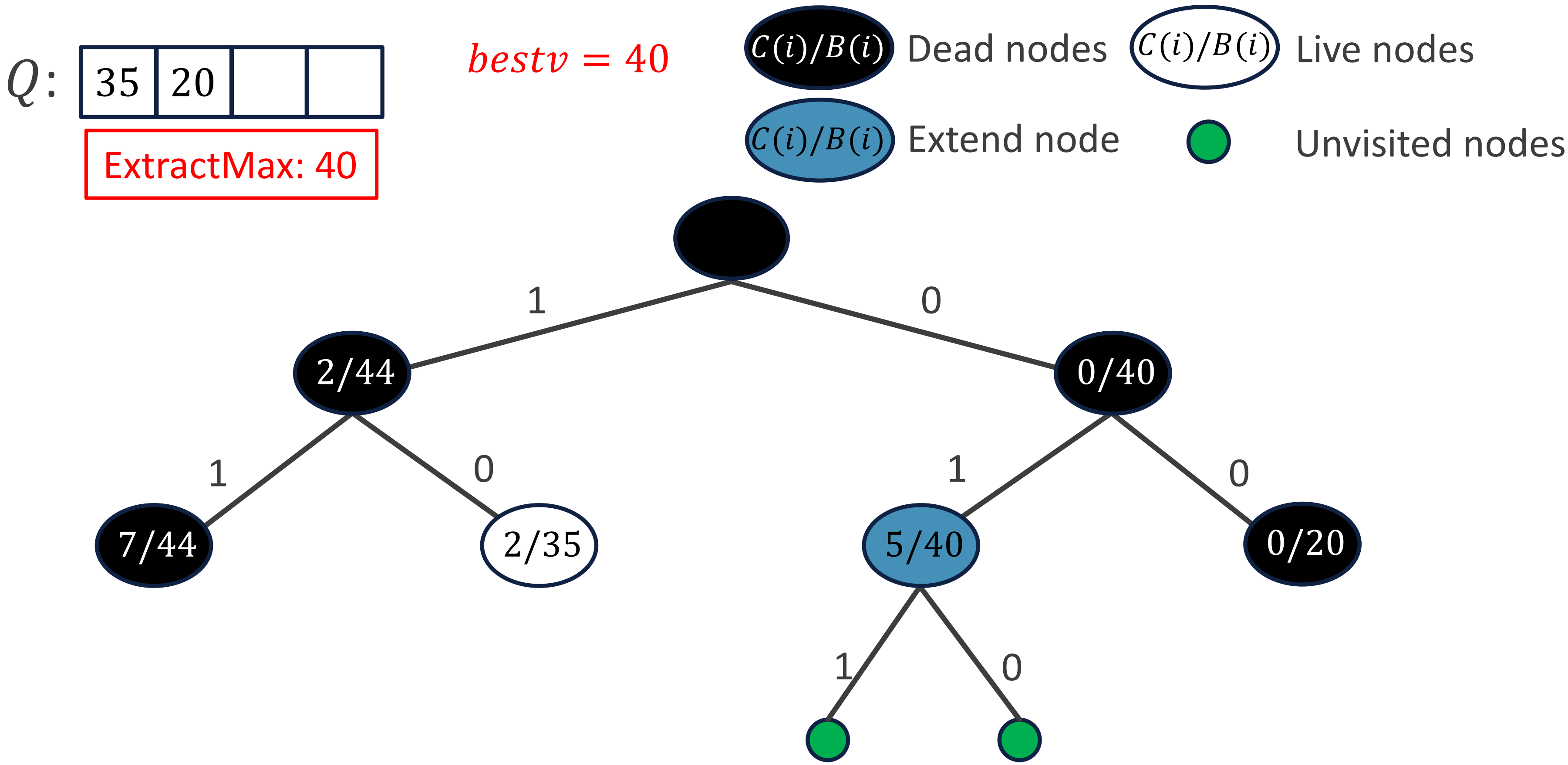
$C(i)/B(i)$ Dead nodes $C(i)/B(i)$ Live nodes

$C(i)/B(i)$ Extend node Unvisited nodes



$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

Classroom Exercise



$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

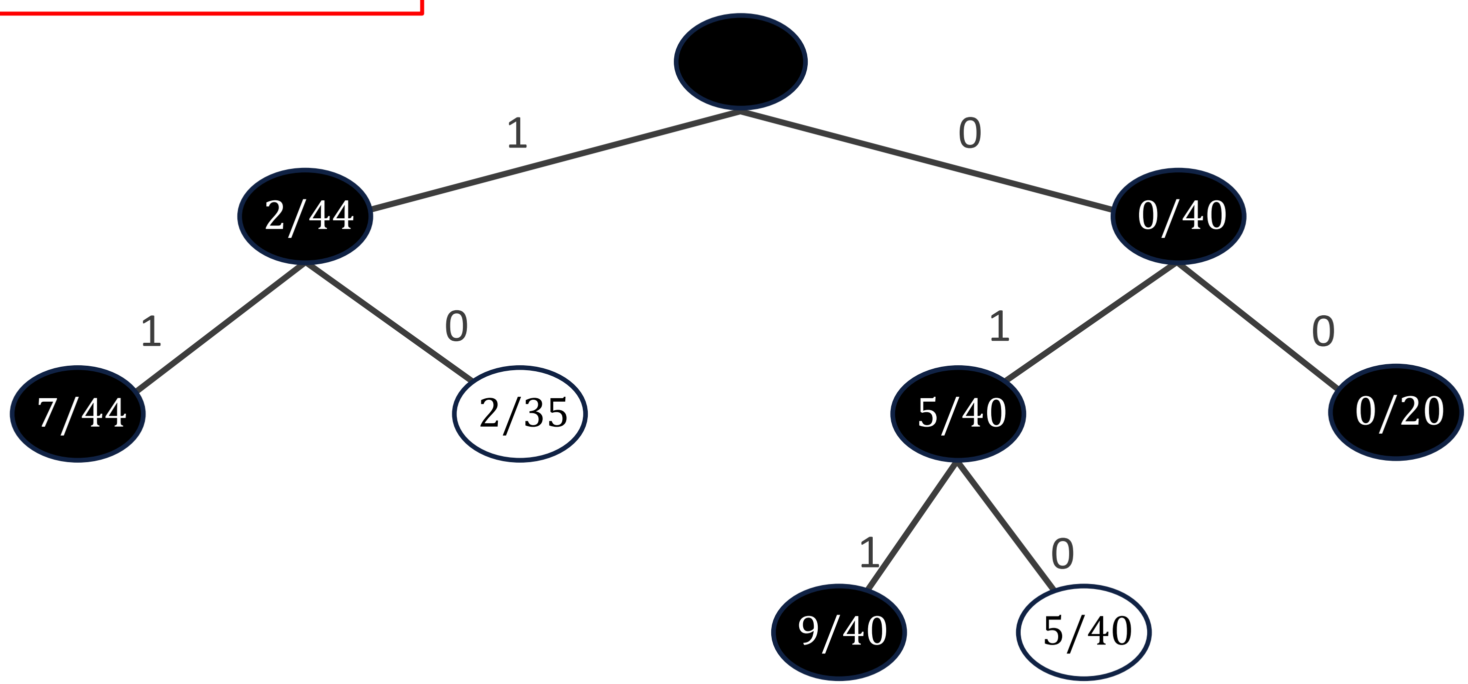
Classroom Exercise

Q: | 40 | 35 | 20 | |

$bestv = 40$

AddLiveNode (Line 4-9)

$C(i)/B(i)$ Dead nodes
$C(i)/B(i)$ Live nodes
$C(i)/B(i)$ Extend node
Unvisited nodes



$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

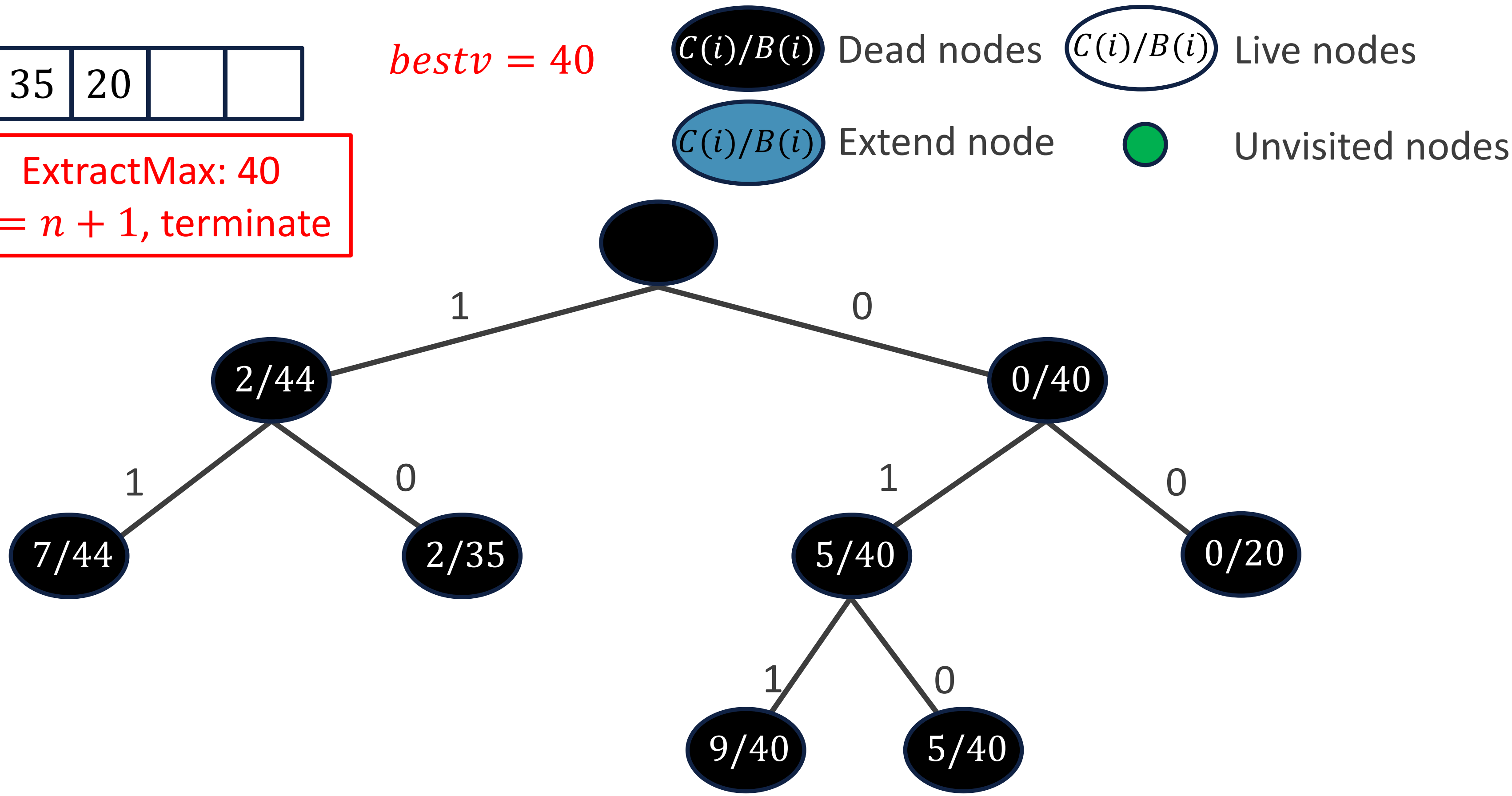
Classroom Exercise

Q:

35	20		

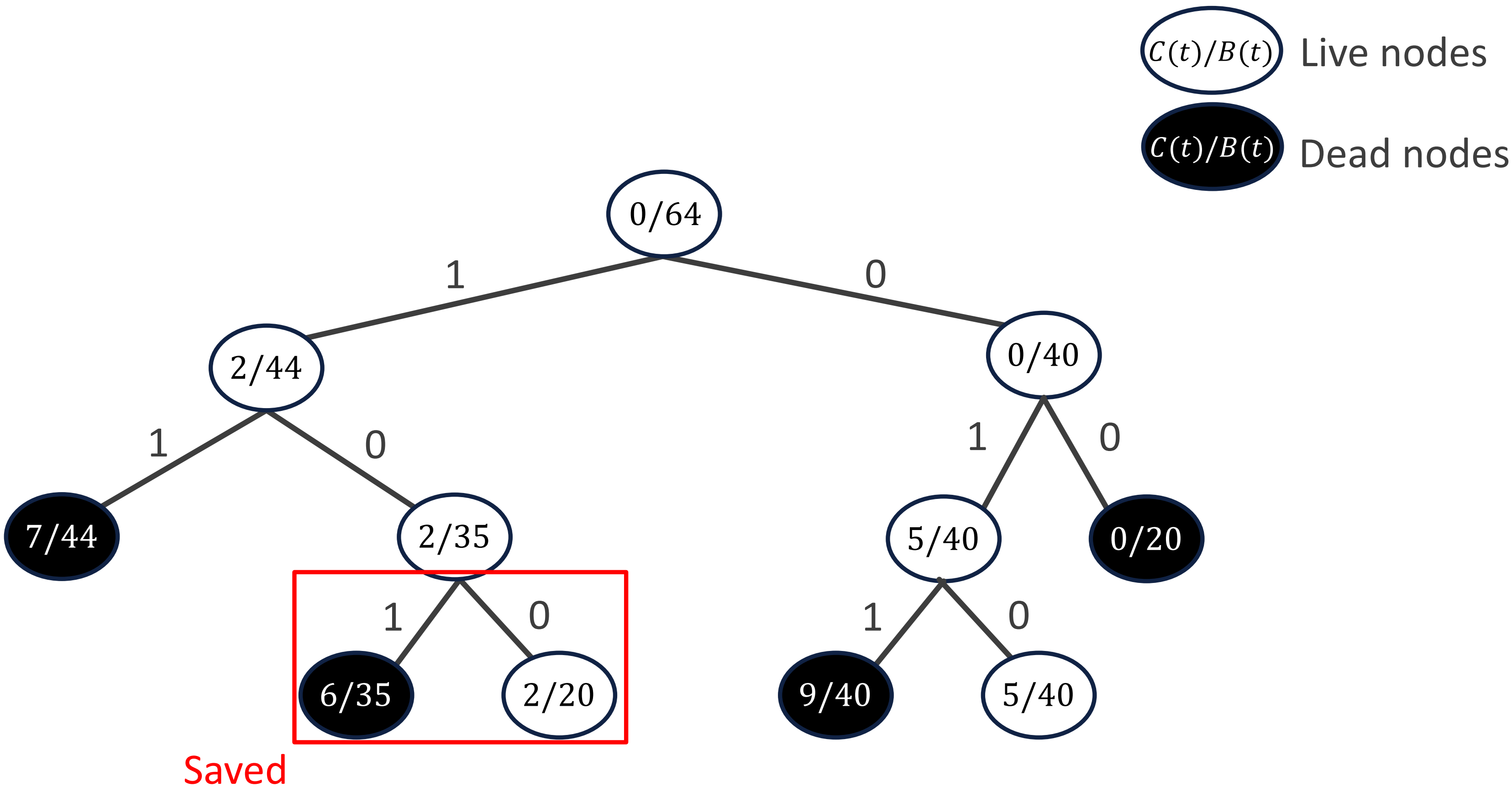
$bestv = 40$

ExtractMax: 40
$i = n + 1$, terminate



$$n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$$

Classroom Exercise



Backtracking for $n = 3, v = [20,40,20], w = [2,5,4], v/w = [10,8,5], W = 5$

SAT PROBLEM

SAT Problem

- We have seen the 3-CNF-SAT problem. Now we consider a more general k-CNF-SAT problem:

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

where each C_i has the following form:

$$C_i = l_{i1} \vee l_{i2} \vee \cdots \vee l_{ik}$$

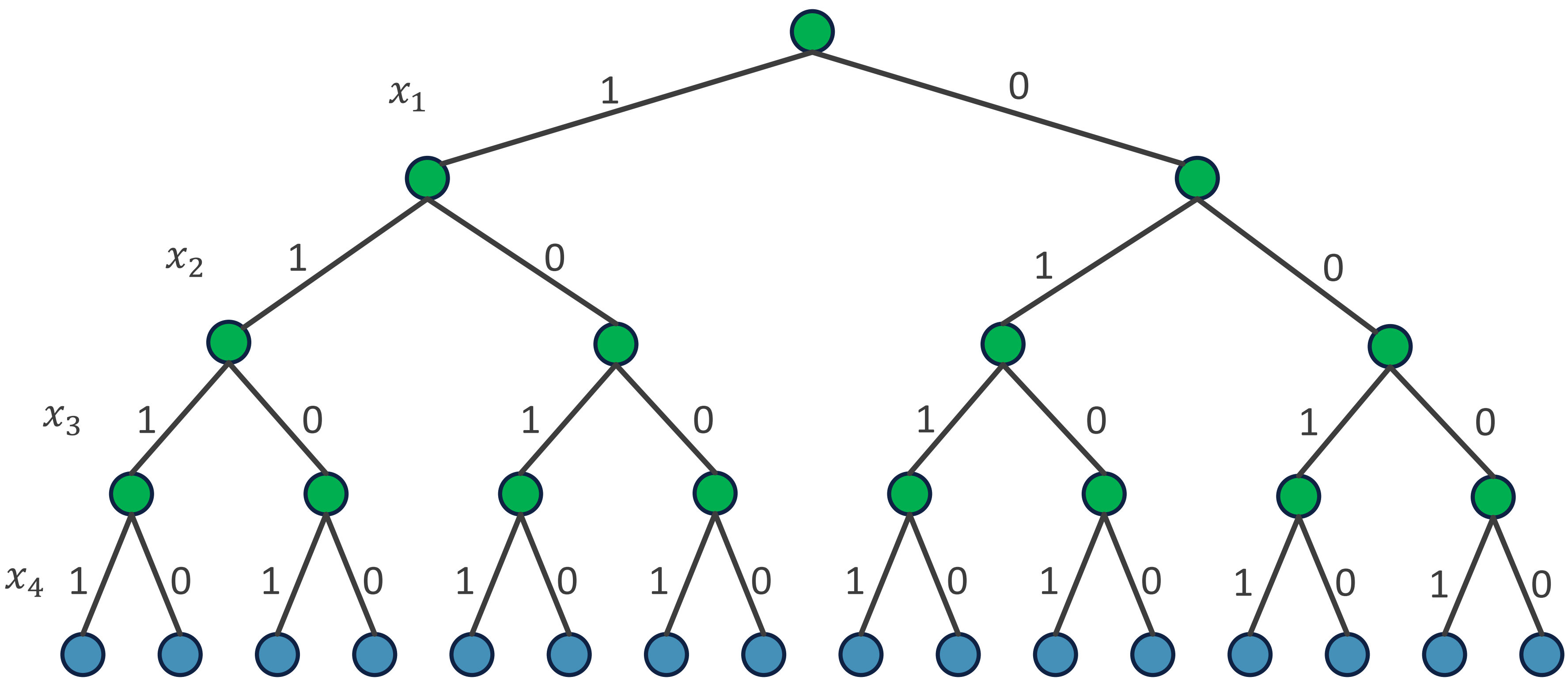
and the literal l_{ij} could be one of variables in $\{x_1, x_2, \ldots, x_m\}$ or its negation.

- For example, a 3-CNF-SAT with 4 variables could be:

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge$$
$$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$$

- Notice three different parameters: n, k and m.

Solution Space Tree for SAT Problem



A k-CNF-SAT problem with 4 variable.

SAT Problem

- This is a decision problem, rather than optimization problem.
- It seems that we don't have bounding function.
- What is the constraint function?

There's no constraint function. At any node, we still have hope before we assign the value to the last variable x_m.

SAT Problem

- Remember that any decision problem can also be converted to optimization problem.
- What is the optimization version for k-CNF-SAT?
- Find an assignment that satisfies the maximum number of clauses.

SAT Problem

- Now, we can design the bounding function.
- We can calculate the lower bound cv for each node, by counting the number of satisfied clauses.
- For example, $x_1 = 1$:

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$$

  - We get $cv = 2$.
  - No matter how we assign values to x_2, x_3 and x_4, the final solution will not be lower than 2.

SAT Problem

- Again, by the idea of branch-and-bound, we put cv in a max-priority queue.
- However, the different is that cv is the lower bound, rather than upper bound in 0/1 knapsack problem.
- It still works. Higher lower bound also means higher hope.

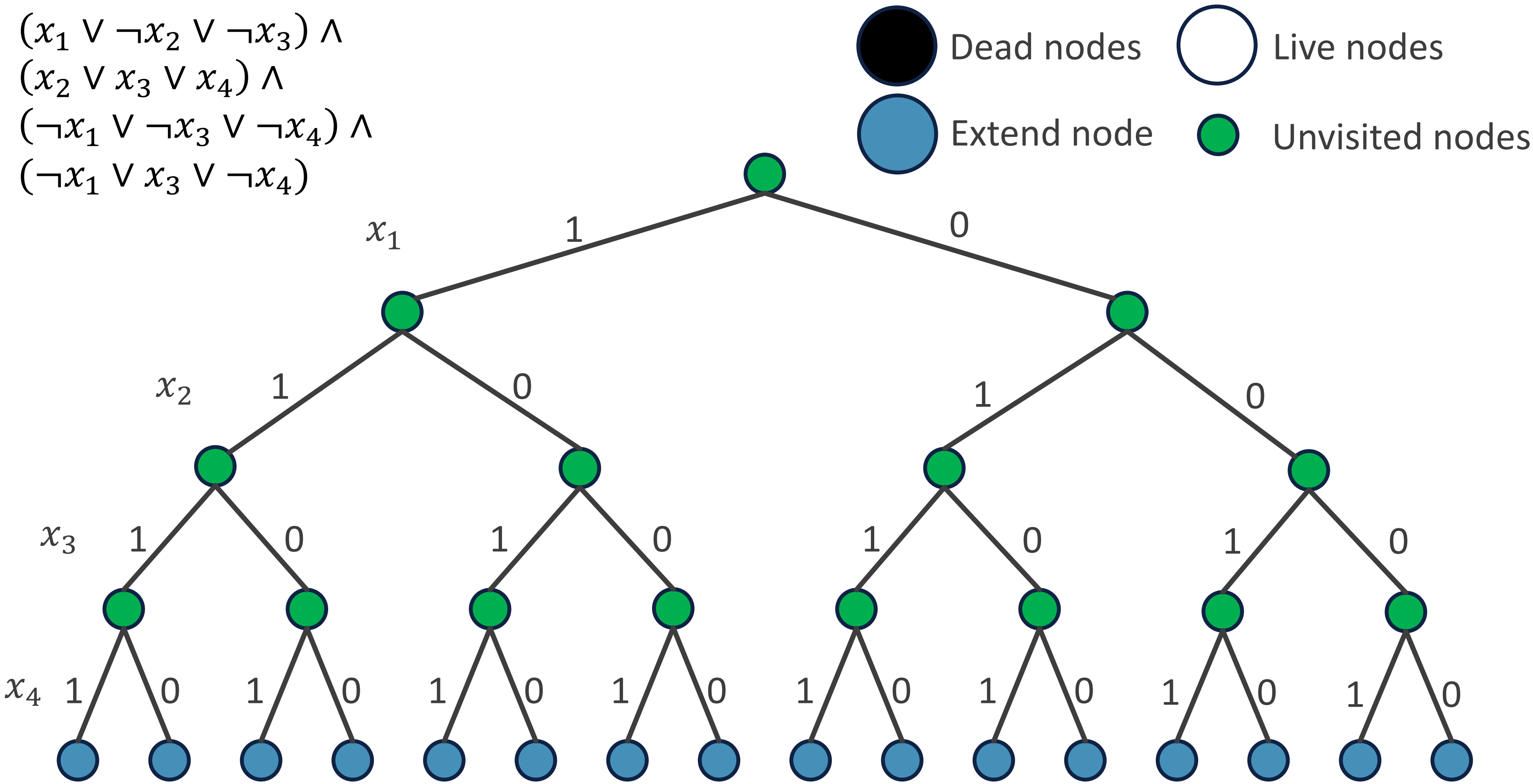
Pseudocode

```
MaxProfitSAT()
i ← 1
while i ≠ m + 1 do
    cv ← ok(i, N, 1)
    if cv > 0 then
        AddLiveNode(cv, 1, i + 1)
    cv ← ok(i, N, 0)
    if cv > 0 then
        AddLiveNode(cv, 0, i + 1)
    N ← ExtractMax(Q); i ← N.level
for j ← m downto 1 do
    bestx[j] ← E.LChild; E ← E.parent
```

```
ok(i, N, ch)
cn ← 0
for j ← 1 to n do
    if check(C_j, N, ch) = 0 then
        return 0
    else if check(C_j, N, ch) = 1 then
        cn ← cn + 1
return cn
```

Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge$
$(x_2 \vee x_3 \vee x_4) \wedge$
$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4)$

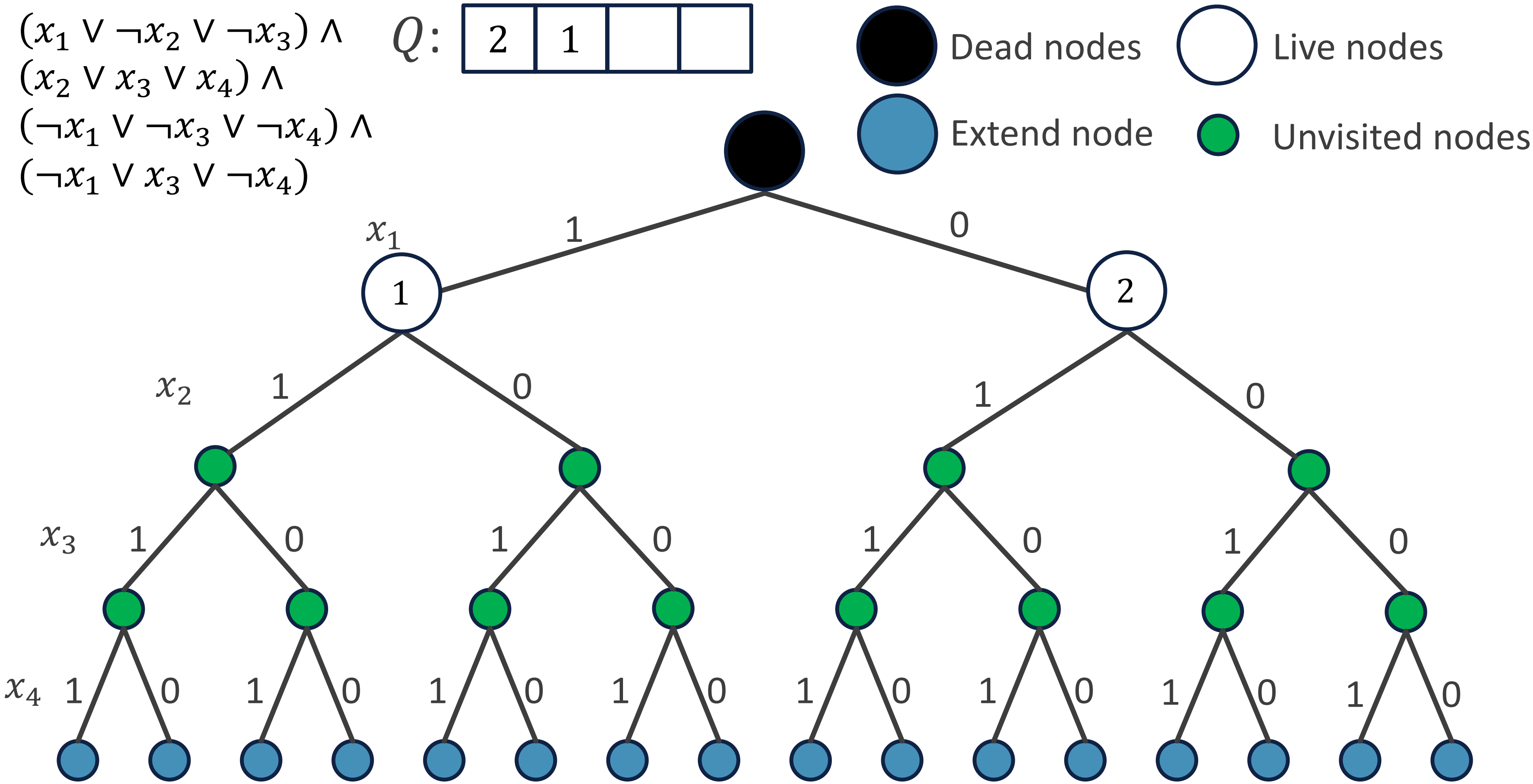


Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge$
$(x_2 \vee x_3 \vee x_4) \wedge$
$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4)$

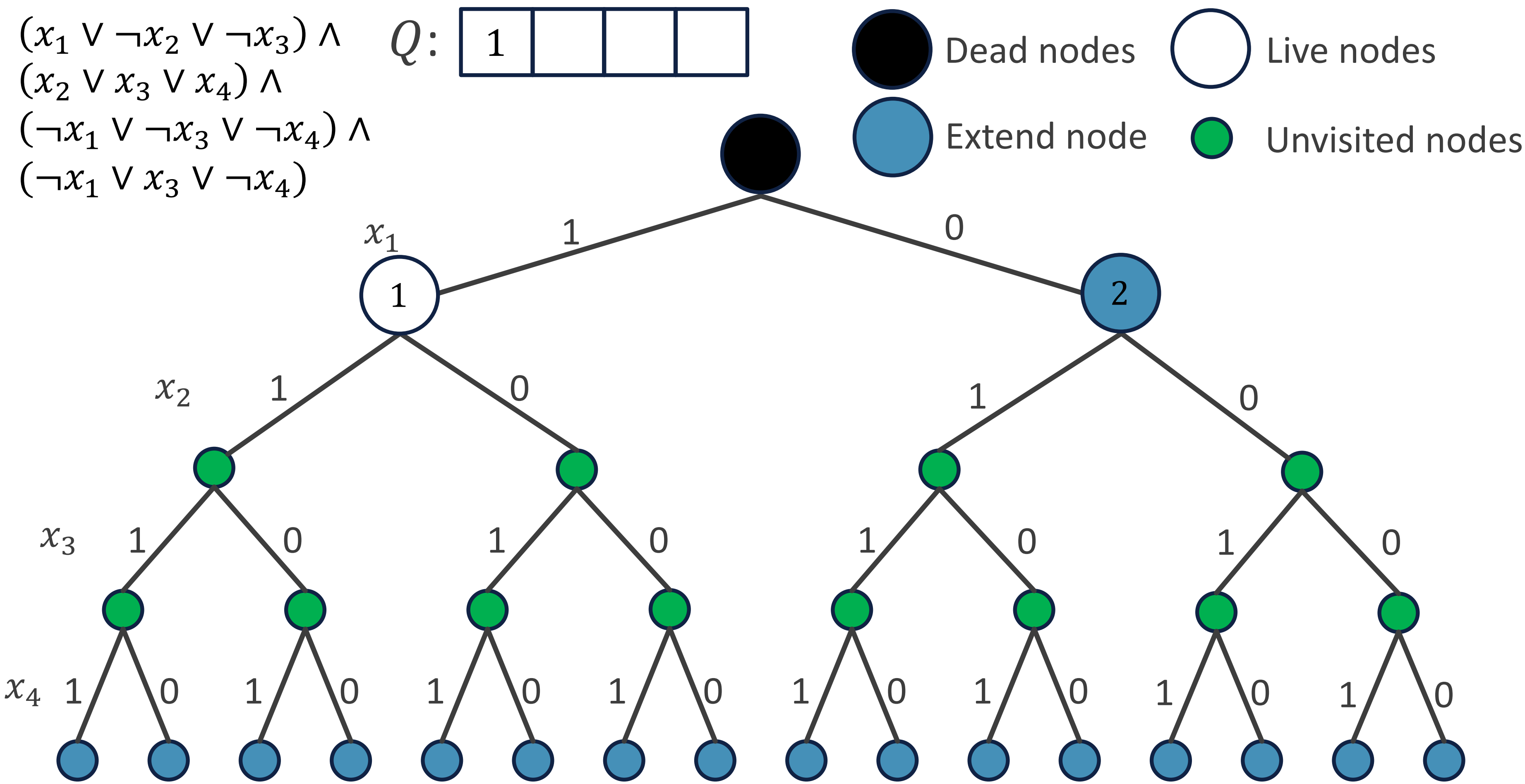
Q:

2	1		



Example

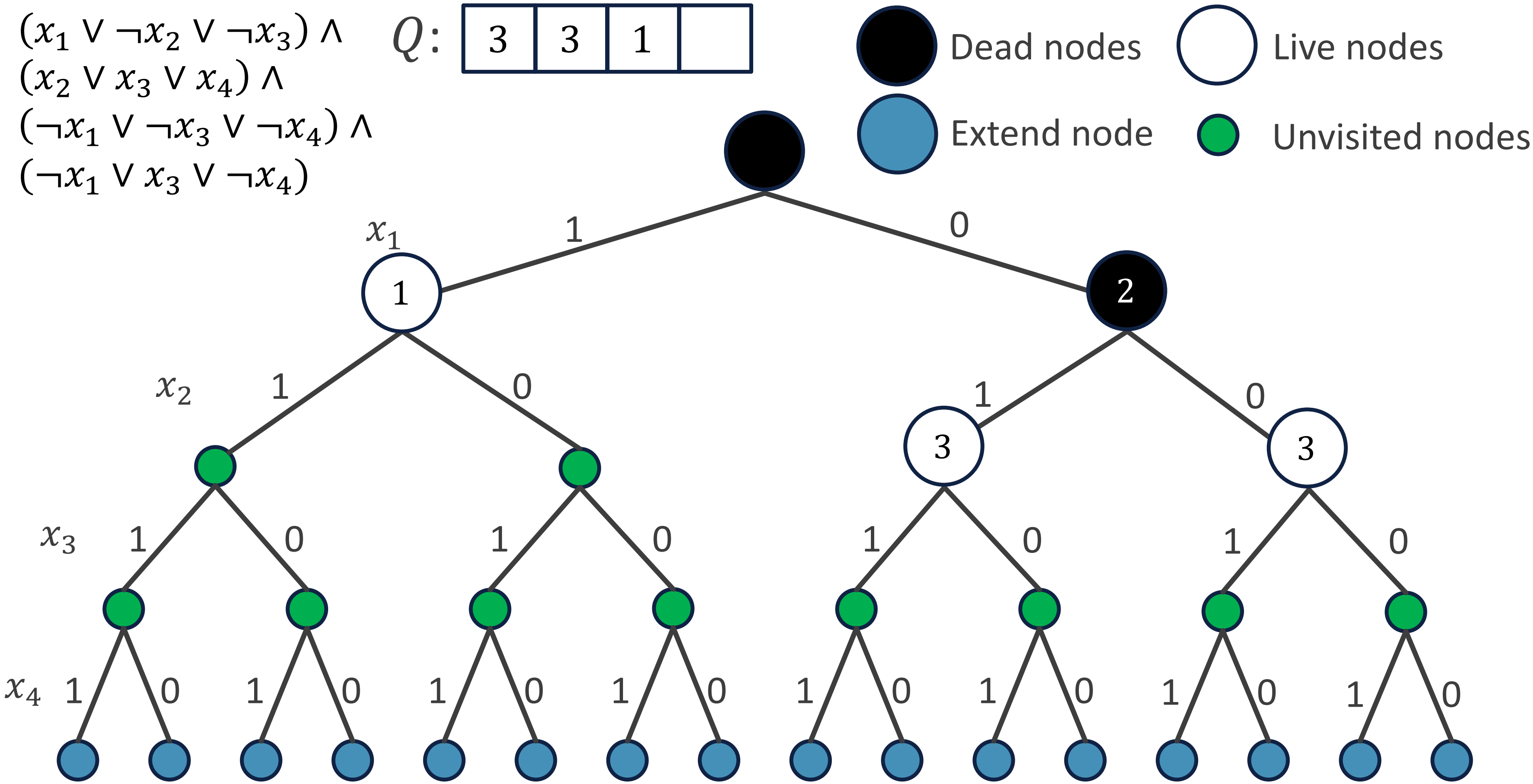
$(x_1 \lor \neg x_2 \lor \neg x_3) \land$
$(x_2 \lor x_3 \lor x_4) \land$
$(\neg x_1 \lor \neg x_3 \lor \neg x_4) \land$
$(\neg x_1 \lor x_3 \lor \neg x_4)$



Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge$
$(x_2 \vee x_3 \vee x_4) \wedge$
$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4)$

Q:	3	3	1	

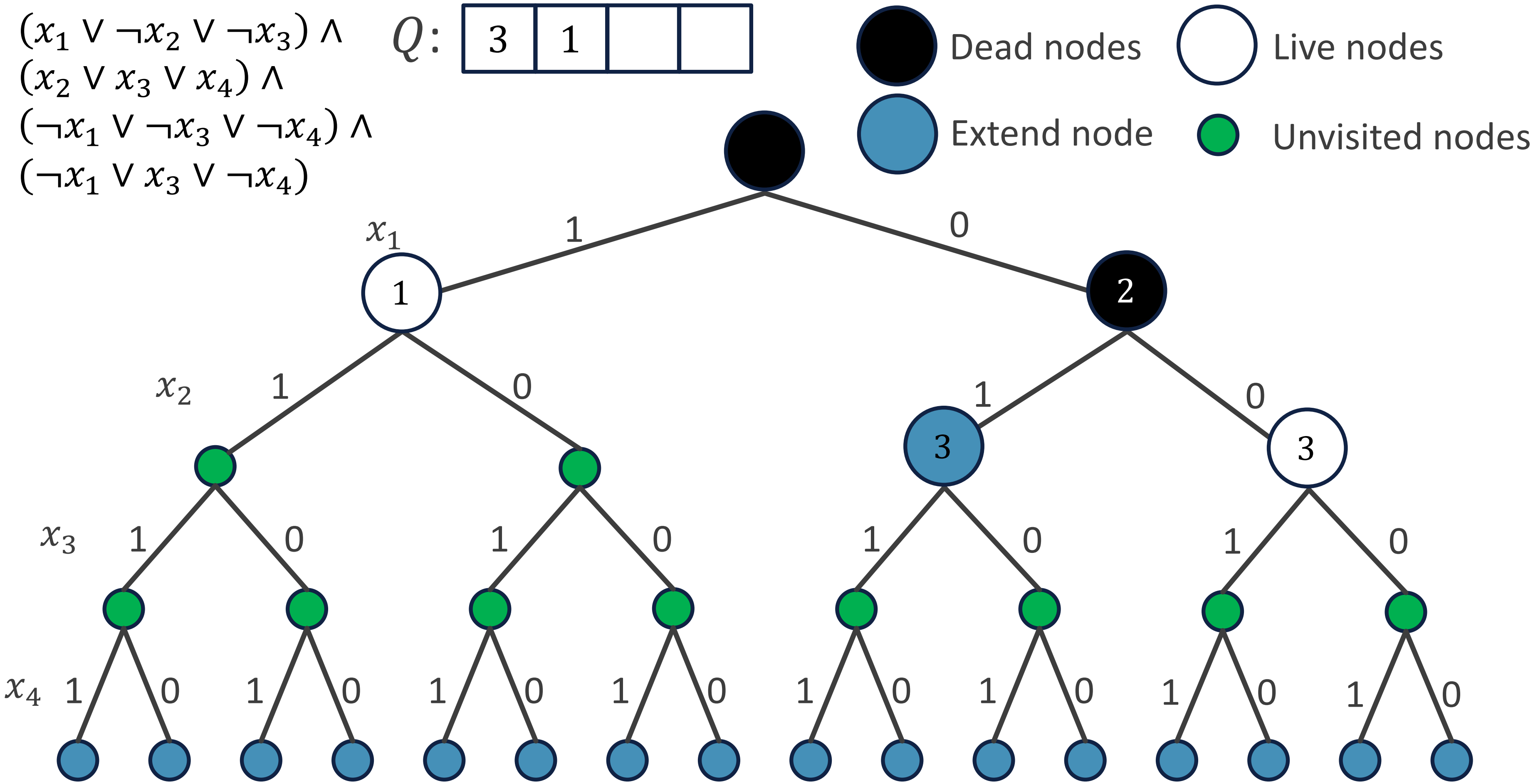


Example

$(x_1 \lor \neg x_2 \lor \neg x_3) \land$
$(x_2 \lor x_3 \lor x_4) \land$
$(\neg x_1 \lor \neg x_3 \lor \neg x_4) \land$
$(\neg x_1 \lor x_3 \lor \neg x_4)$

Q:

3	1		

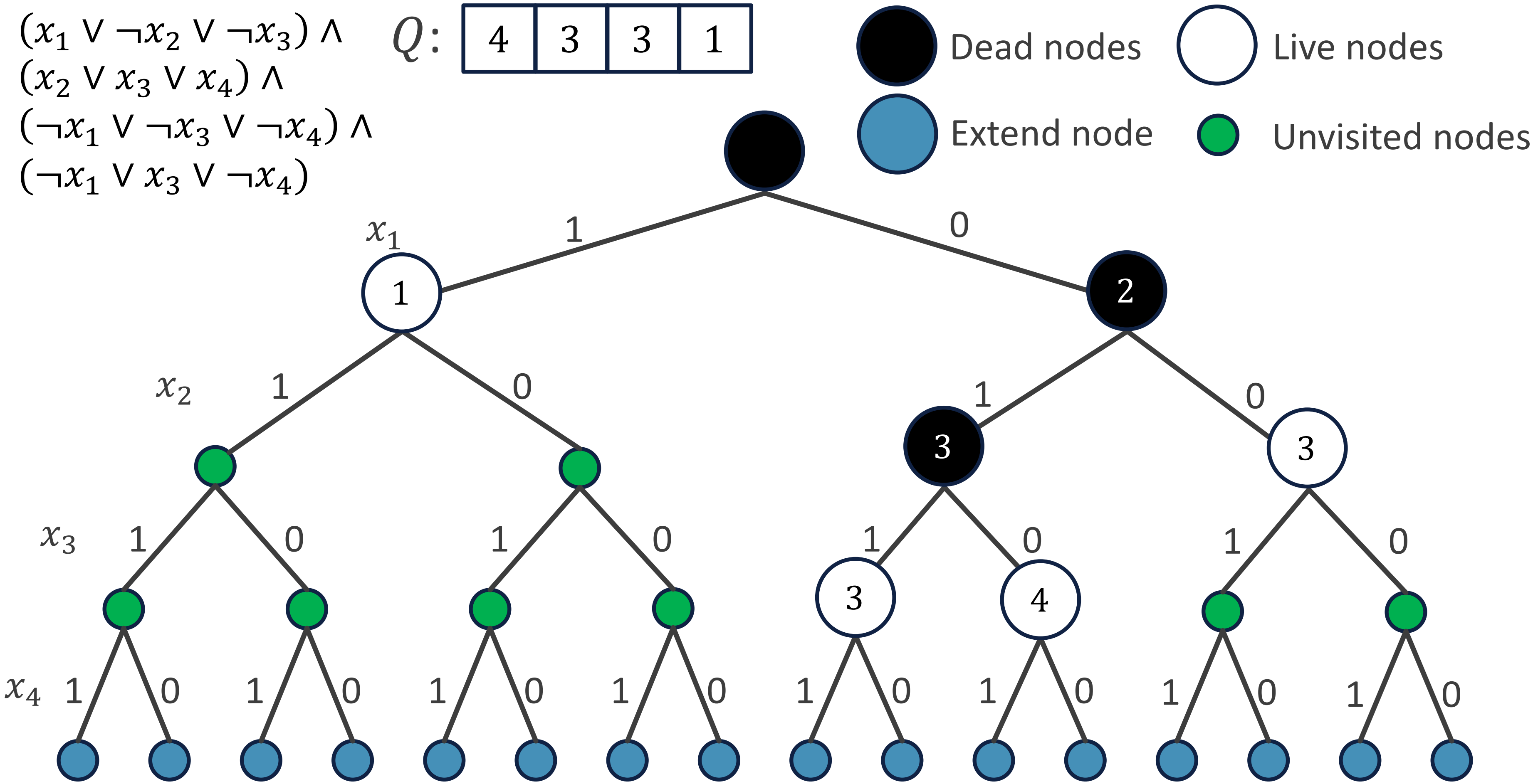


Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge$
$(x_2 \vee x_3 \vee x_4) \wedge$
$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4)$

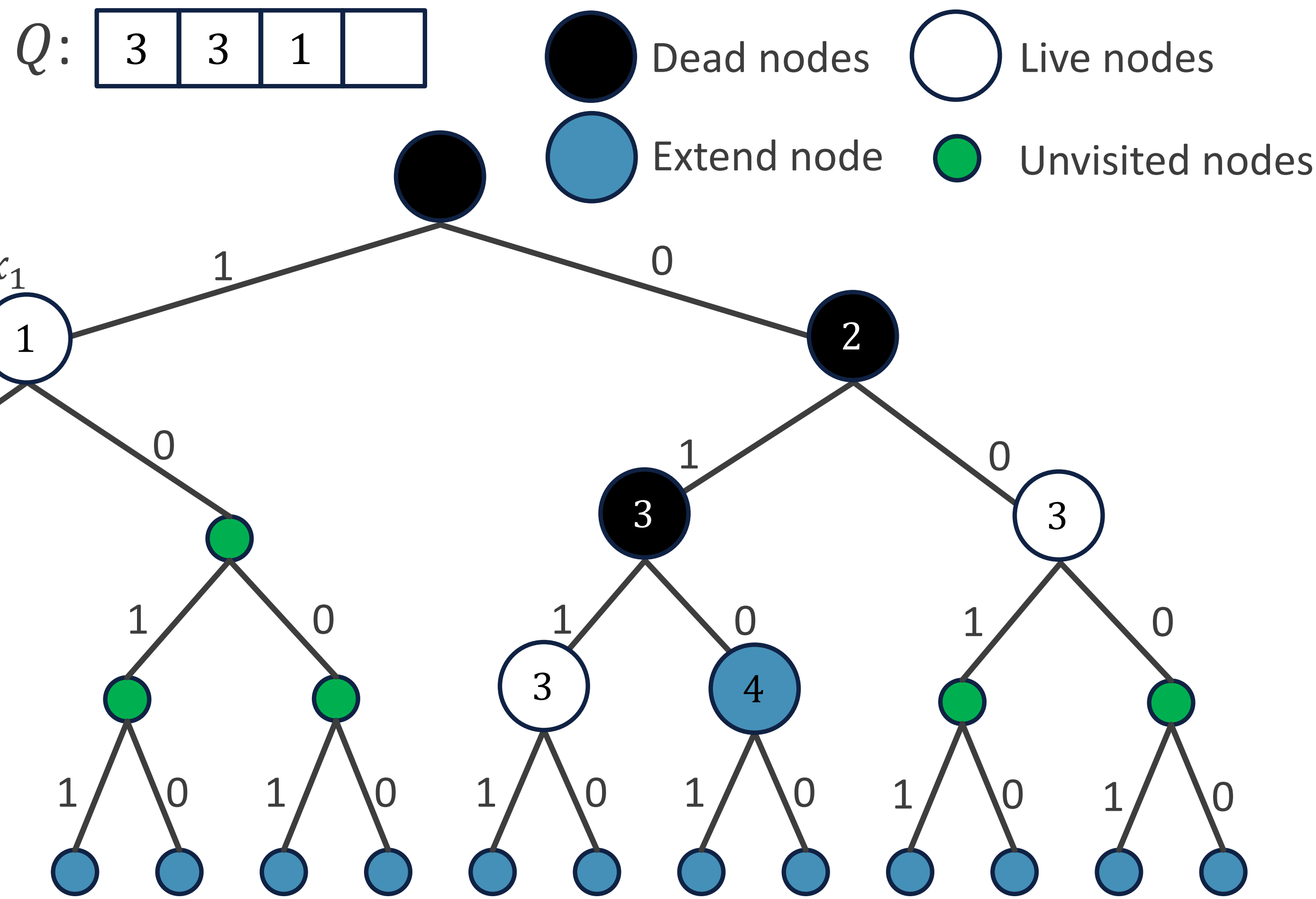
Q:

4	3	3	1



Example

$(x_1 \lor \neg x_2 \lor \neg x_3) \land$
$(x_2 \lor x_3 \lor x_4) \land$
$(\neg x_1 \lor \neg x_3 \lor \neg x_4) \land$
$(\neg x_1 \lor x_3 \lor \neg x_4)$



Example

$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge$
$(x_2 \vee x_3 \vee x_4) \wedge$
$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4)$

Q:

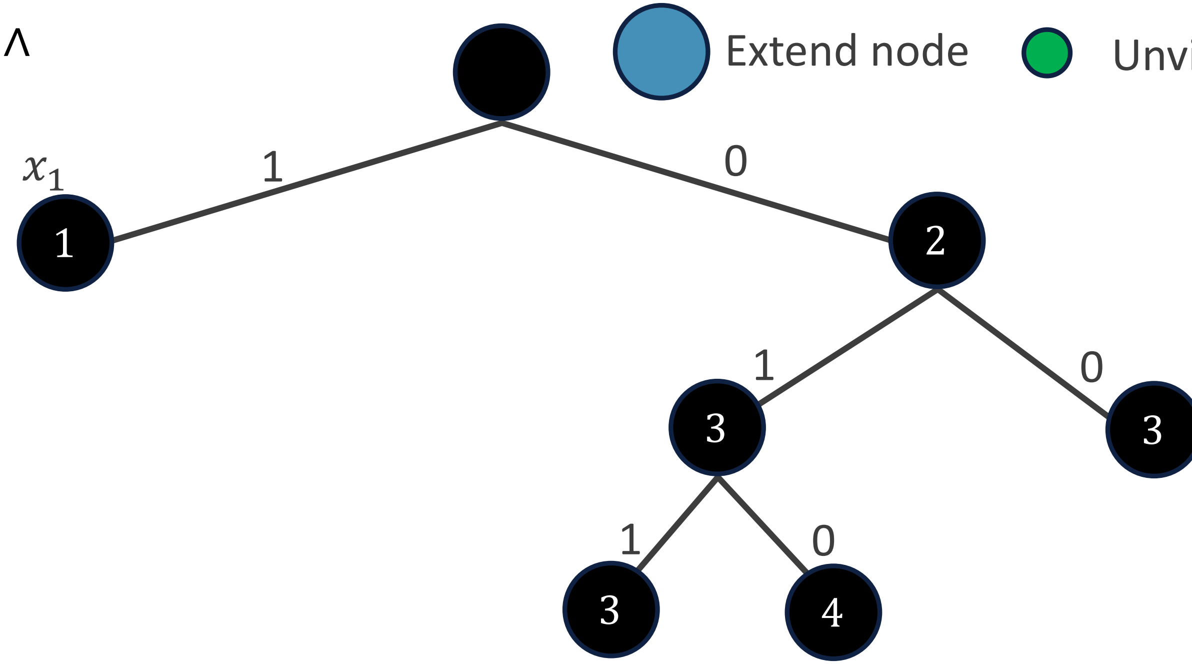
3	3	1	

Dead nodes

Live nodes

Extend node

Unvisited nodes



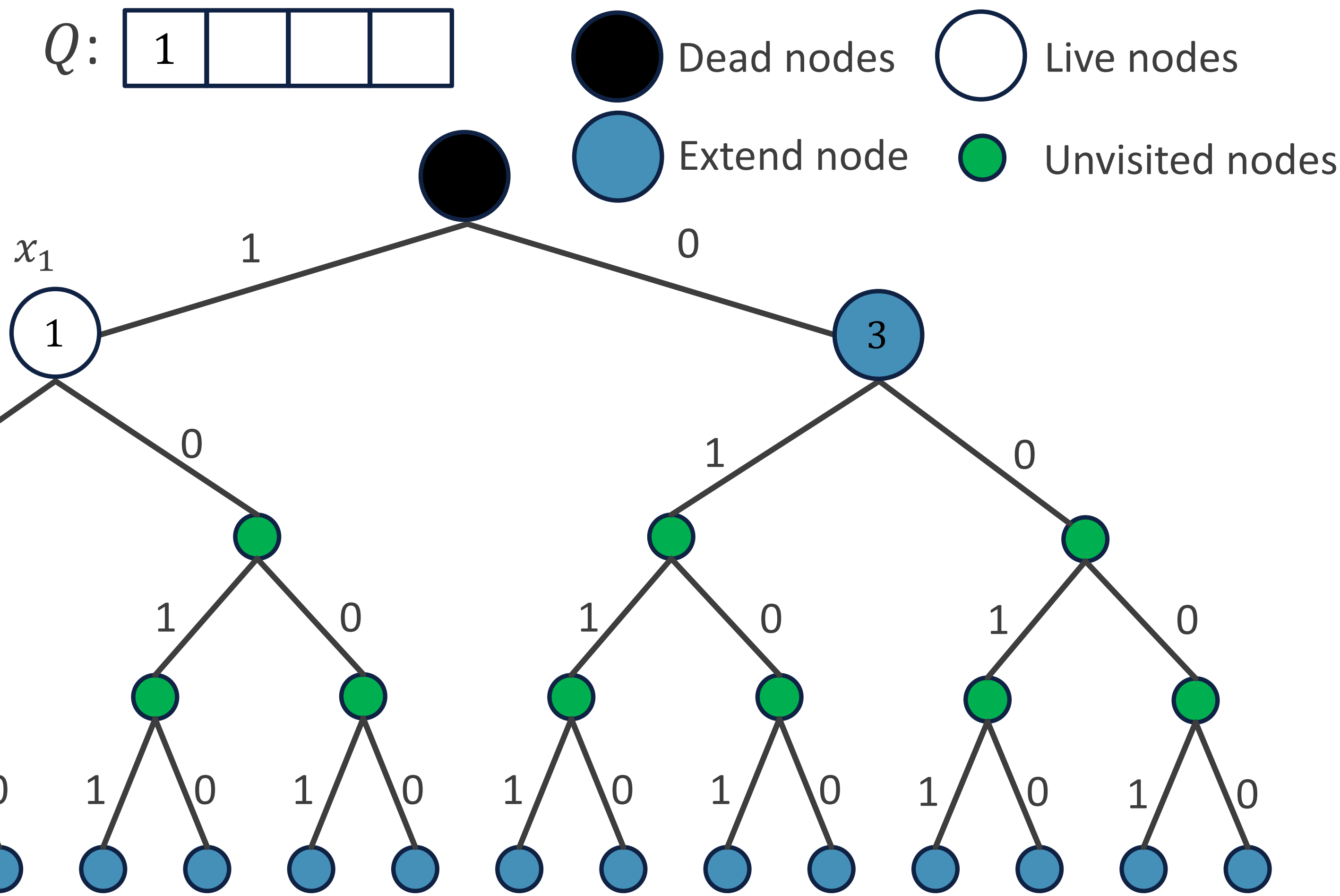
Classroom Exercise

- Draw the pruned solution space tree for the following k-CNF-SAT problem instance by max-profit branch-and-bound.

$$\phi = (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$$

Classroom Exercise

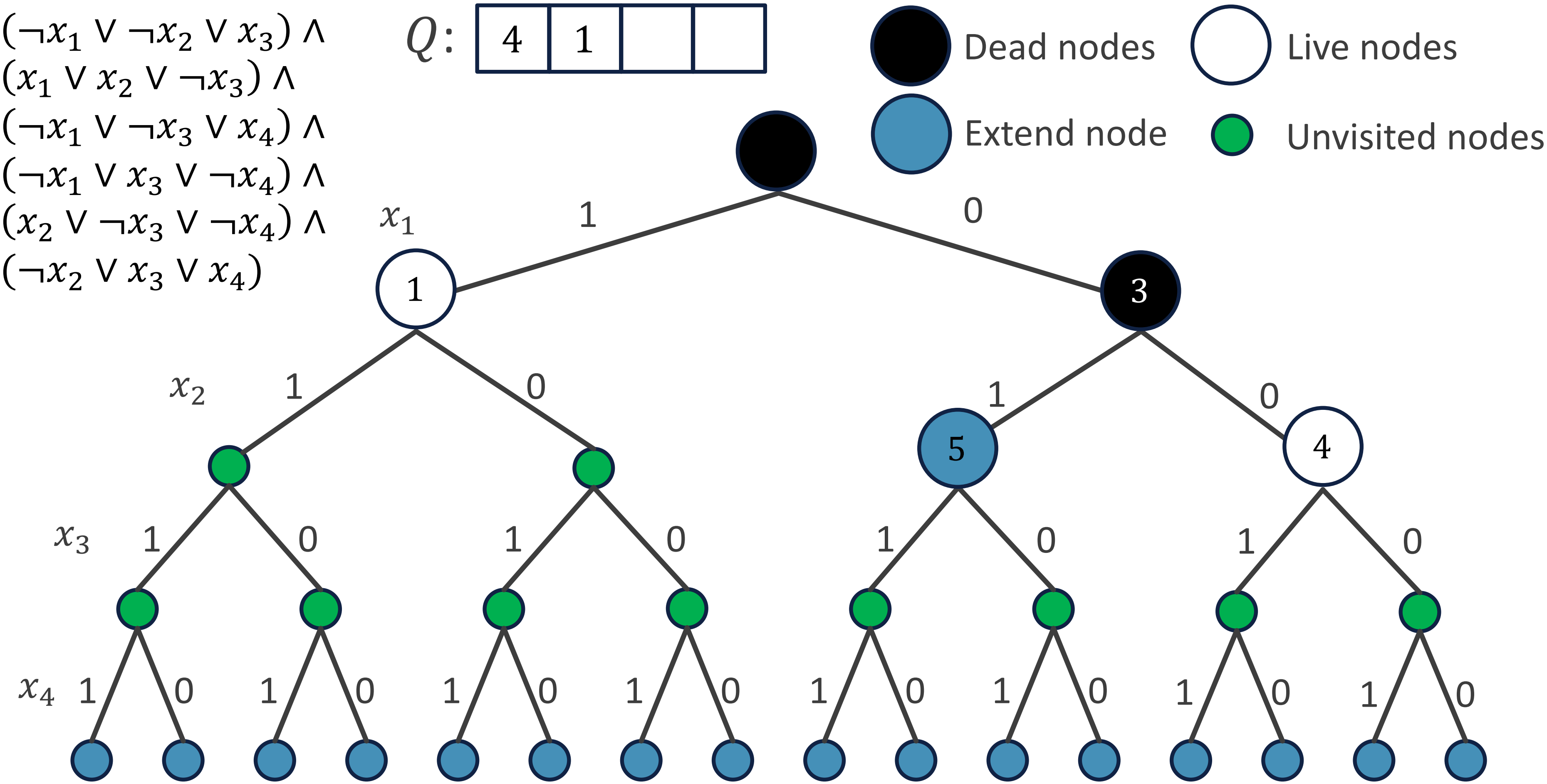
$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge$
$(x_1 \vee x_2 \vee \neg x_3) \wedge$
$(\neg x_1 \vee \neg x_3 \vee x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4) \wedge$
$(x_2 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_2 \vee x_3 \vee x_4)$



Classroom Exercise

$(\neg x_1 \lor \neg x_2 \lor x_3) \land$
$(x_1 \lor x_2 \lor \neg x_3) \land$
$(\neg x_1 \lor \neg x_3 \lor x_4) \land$
$(\neg x_1 \lor x_3 \lor \neg x_4) \land$
$(x_2 \lor \neg x_3 \lor \neg x_4) \land$
$(\neg x_2 \lor x_3 \lor x_4)$

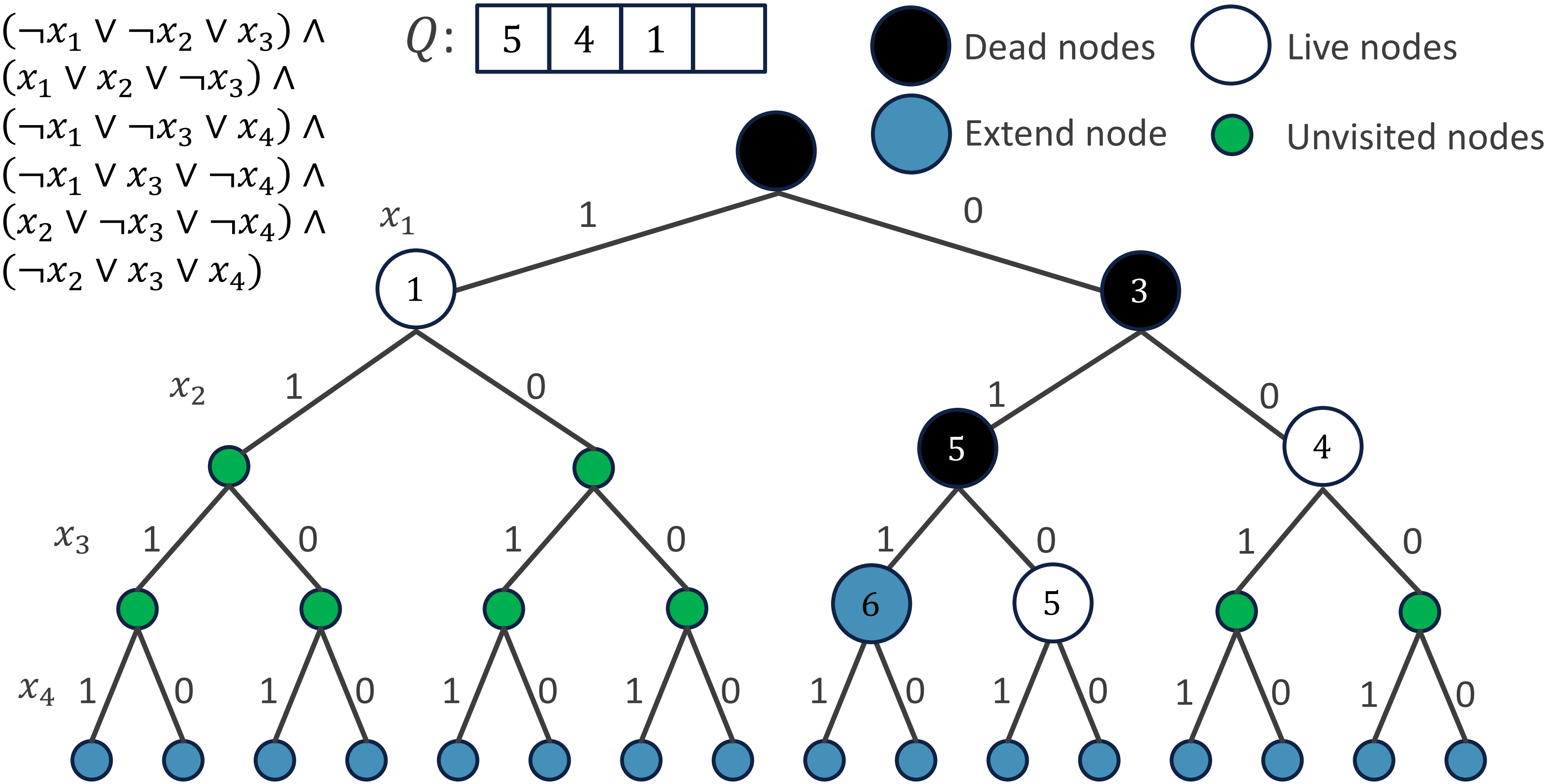
Q:	4	1		



Classroom Exercise

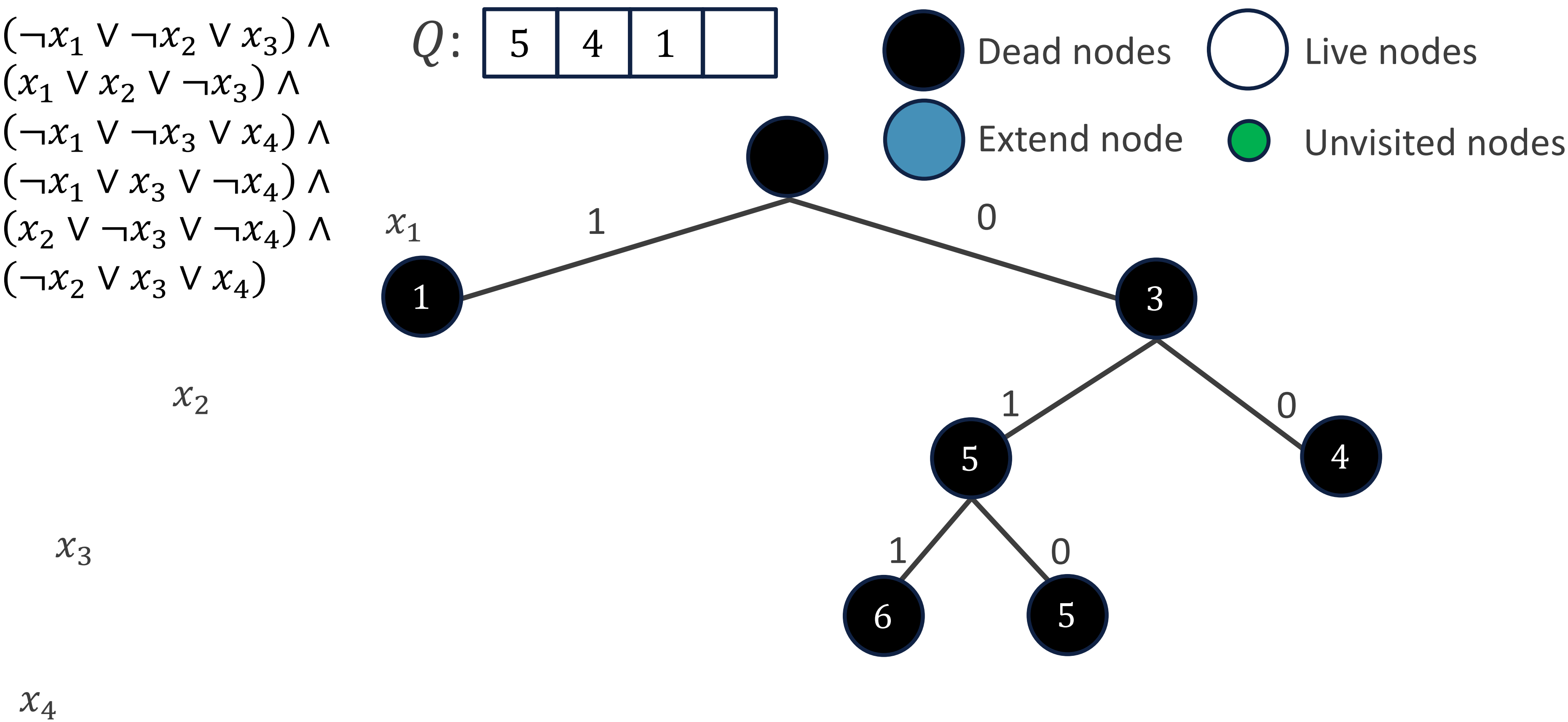
$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge$
$(x_1 \vee x_2 \vee \neg x_3) \wedge$
$(\neg x_1 \vee \neg x_3 \vee x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4) \wedge$
$(x_2 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_2 \vee x_3 \vee x_4)$

Q: | 5 | 4 | 1 | |



Classroom Exercise

$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge$
$(x_1 \vee x_2 \vee \neg x_3) \wedge$
$(\neg x_1 \vee \neg x_3 \vee x_4) \wedge$
$(\neg x_1 \vee x_3 \vee \neg x_4) \wedge$
$(x_2 \vee \neg x_3 \vee \neg x_4) \wedge$
$(\neg x_2 \vee x_3 \vee x_4)$



TRAVELING SALESPERSON PROBLEM

Traveling Salesperson Problem

- Constraint function $C(i)$ is to simply check if the next vertex is connected to the current vertex:

$$C(i) = w\big[x[i], x[j]\big]$$

Check if $C(i) \neq \infty$.

- Bounding function $B(i)$ is the total weight if we connect $x[i]$:

$$B(i) = cw(i-1) + w[x[i-1], x[i]]$$

$$cw(i) = \sum_{j=2}^{i} w[x[j-1], x[j]]$$

Check if $B(i) < bestw$.

Traveling Salesperson Problem

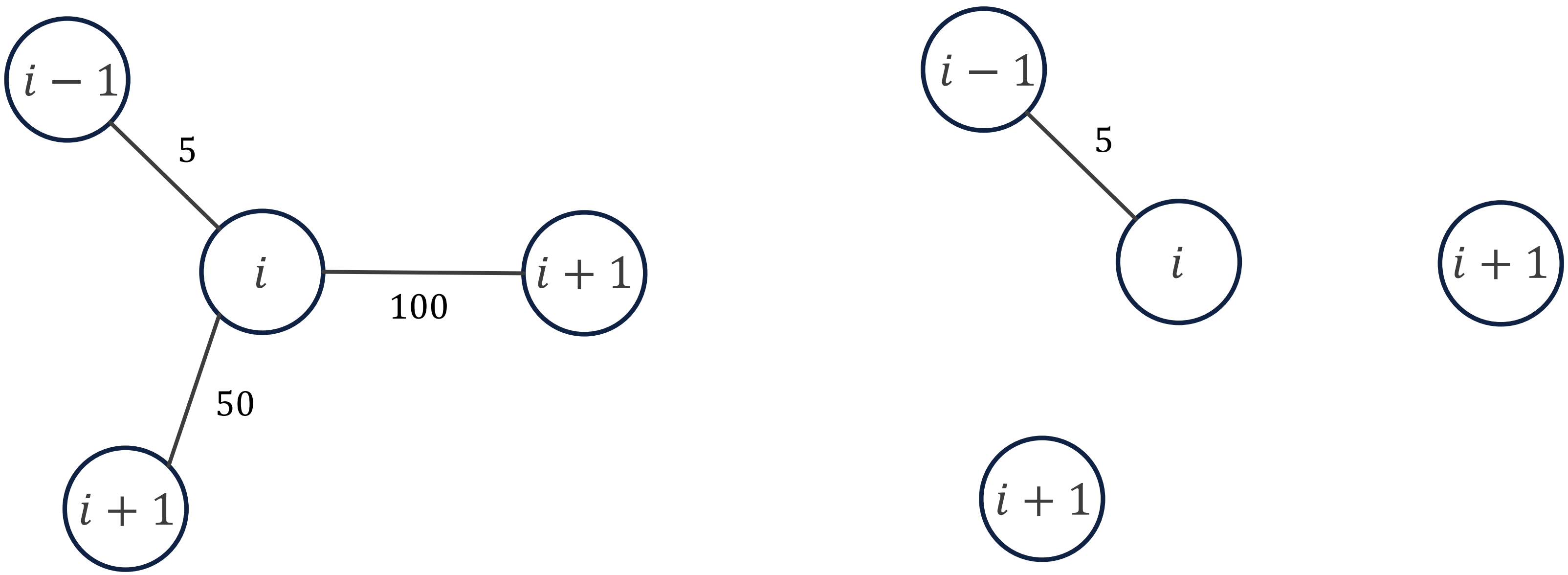
- $B(i) \geq bestw$ is the condition to prune. If we want to prune more branches, we need to increase $B(i)$ as much as possible.
- Now, the bounding function

$$B(i) = cw(i-1) + w[x[i-1], x[i]]$$

only calculates the weight between $x[i-1]$ and $x[i]$, but ignores all the remaining paths.

Traveling Salesperson Problem

- Now, consider these cases:



After going to node i, the rest paths are hopeless.

Traveling Salesperson Problem

- How to obtain the lower bound if we consider all the nodes that we haven't visited?
- Just pick the outgoing edge of each unvisited node with minimum weight, and sum them up.
  - Although it may not form a solution (a path), but it is a lower bound.
  - Just like the bound of 0/1 knapsack problem.

The Improved Bounding Function

- Let the cost of the extend node i be

$$B(i) = cw(i) + rw(i)$$

where, $cw(i)$ is as before, $rw(i)$ is the sum of costs of least-cost outgoing edges from each remaining vertices, namely

$$rw(i) = \sum_{j=i}^{n} \min_{i<k<n, k \neq j} \{w[x[j], x[k]]\}$$

- If $B(i) \ \geq bestw$, then stop search the extend node i and the following level, otherwise, continue to search.
- At the same time, we adopt min-priority queue to extract the live node with min cost.

```
MinWeightTSP()
MinSum ← 0
for i ← 1 to n do
    Min ← ∞
    for j ← 1 to n do
        if  w[i,j] ≠ ∞ and w[i,j] < Min  then Min ← w[i,j]
    if Min = ∞ then return ∞
    MinOut[i] ← Min
    MinSum ← MinSum + Min
for i ← 1 to n do E.x[i] ← i
E.s ← 1;  E.cw ← 0;  E.rw ← MinSum;   bestw ← ∞
while E.s < n do
    if E.s = n − 1 then
        if w[E.x[n − 1], E.x[n]] ≠ ∞ and w[E.x[n], E.x[1]] ≠ ∞ and
              E.cw +  w[E.x[n − 1], E.x[n]] + w[E.x[n], E.x[1]] <  bestw then
            bestw ← E.cw +  w[E.x[n − 1], E.x[n]] + w[E.x[n], E.x[1]]
            E.cw ← bestw;  E.lw ← bestw
            E.s ← E.s + 1
            Insert(Q , E)
```

Initialize $MinOut$

Isolated vertex

Initialize data structure

Constraint function for node $n-1 \rightarrow n \rightarrow 1$

Increase level

Not finish here. We still put enqueue it. The algorithm terminates when a solution is dequeued ($E.s = n$).

$E.s$: Current node

i: Next node

Once moved, subtract $MinOut$ of the current node.

else for $i \leftarrow E.s + 1$ **to** n **do**
  if $w[E.x[E.s], E.x[i]] \neq \infty$ **then**
    $cw \leftarrow E.cw + w[E.x[E.s], E.x[i]]$
    $rw \leftarrow E.rw - MinOut[E.x[E.s]]$
    $B(i) \leftarrow cw + rw$
    if $B(i) < bestw$ **then**
      for $j \leftarrow 1$ **to** n **do** $N.x[j] \leftarrow E.x[j]$
      $N.x[E.s + 1] \leftarrow E.x[i]$
      $N.x[i] \leftarrow E.x[E.s + 1]$
      $N.cw \leftarrow cw$; $N.s \leftarrow E.s + 1$
      $N.lw \leftarrow B(i)$; $N.rw \leftarrow rw$
      Insert(Q, N)
  $E \leftarrow$ExtractMin(Q)
if $bestw = \infty$ **return** ∞
for $i \leftarrow 1$ **to** n **do** $bestx[i] \leftarrow E.x[i]$
return $bestw$

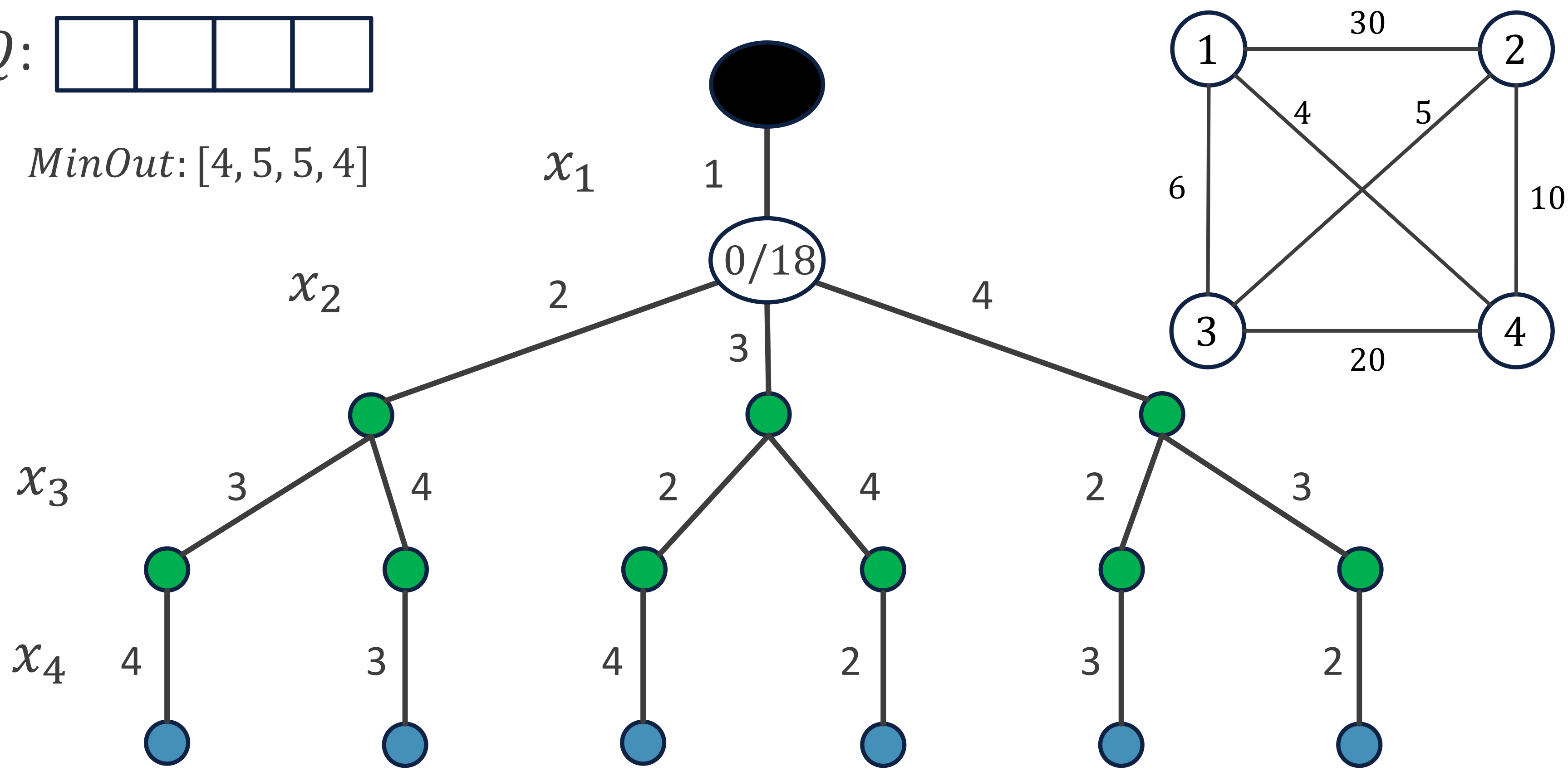
Calculate $B(i)$

Switch selected vertex $(E.s + 1)$ to i.

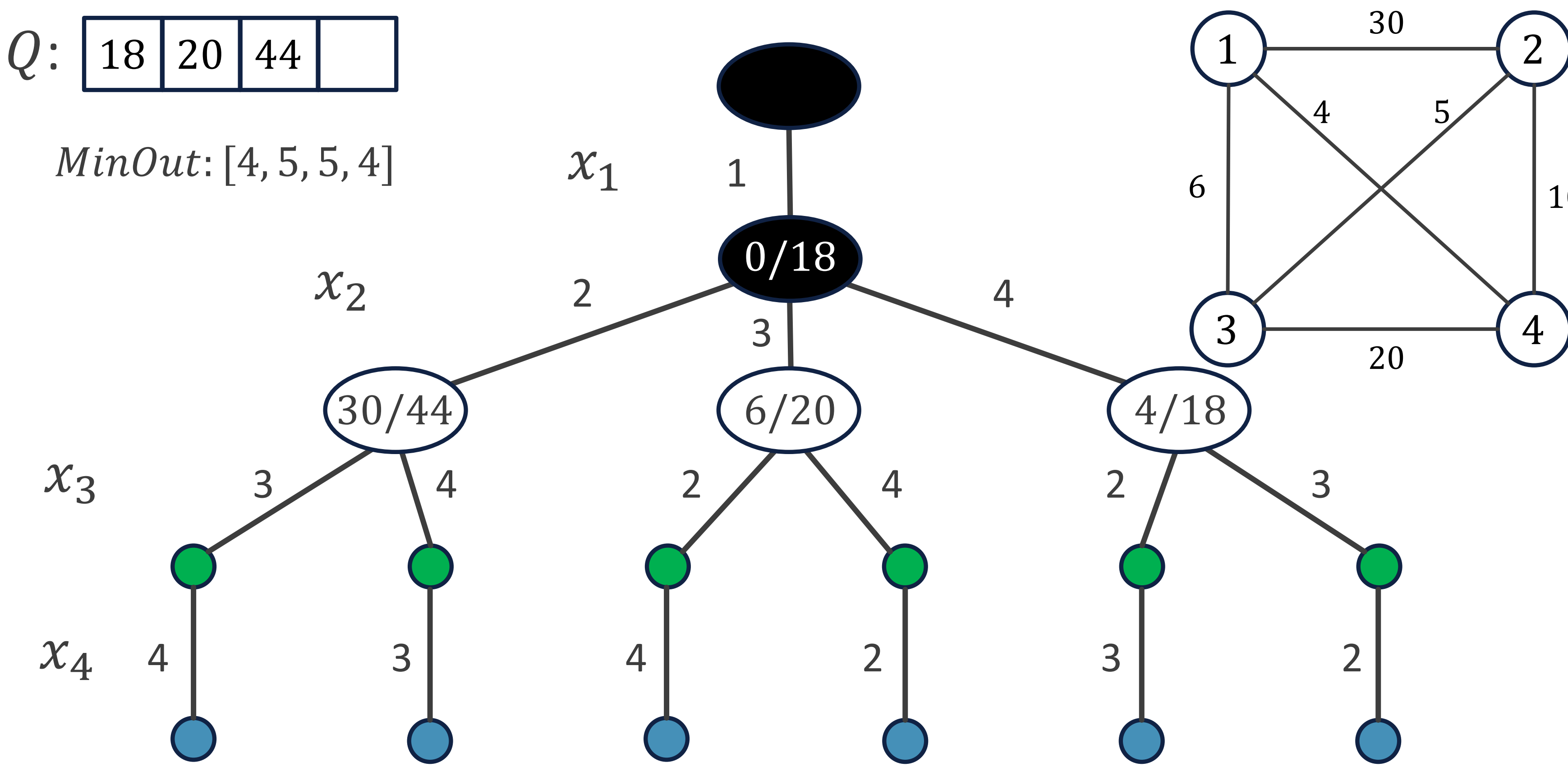
Example

Q:

$MinOut$: [4, 5, 5, 4]



Example

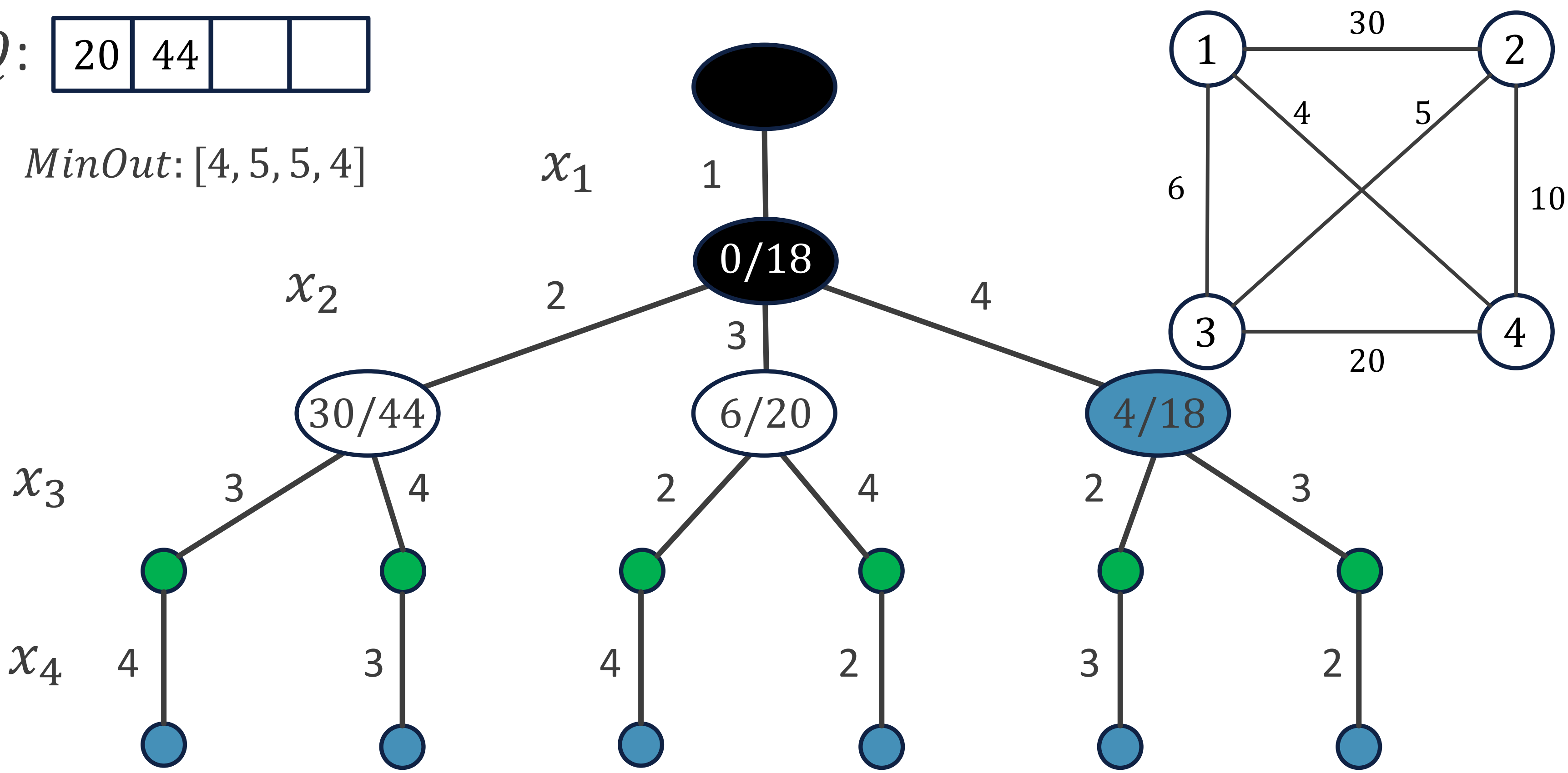


Q: 18 20 44
MinOut: [4, 5, 5, 4]
x1 x2 x3 x4
0/18
30/44
6/20
4/18
1 2 3 4
30 4 5 6 10 20

Example

Q: | 20 | 44 | | |

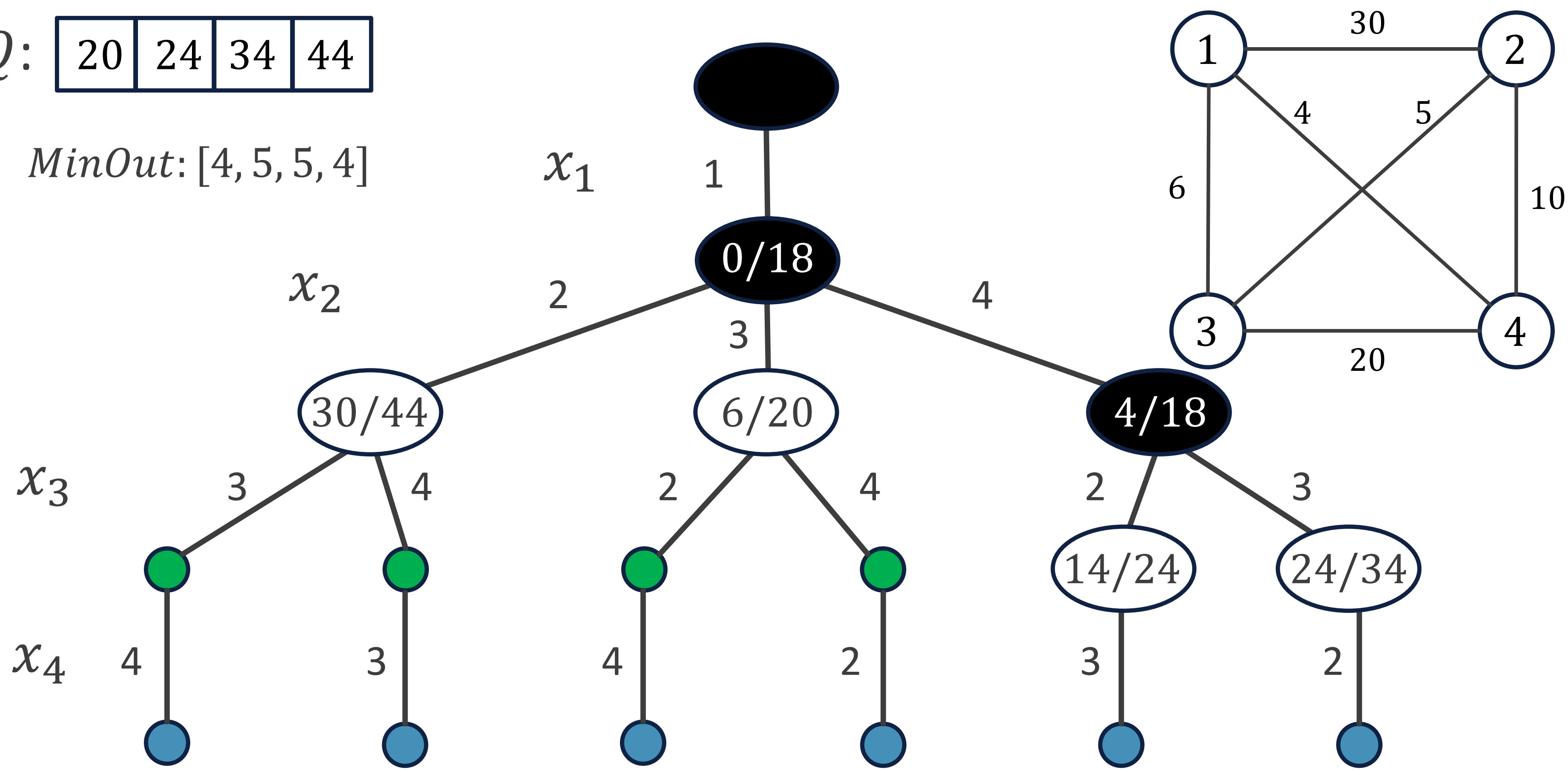
$MinOut$: $[4, 5, 5, 4]$



Example

Q: | 20 | 24 | 34 | 44 |

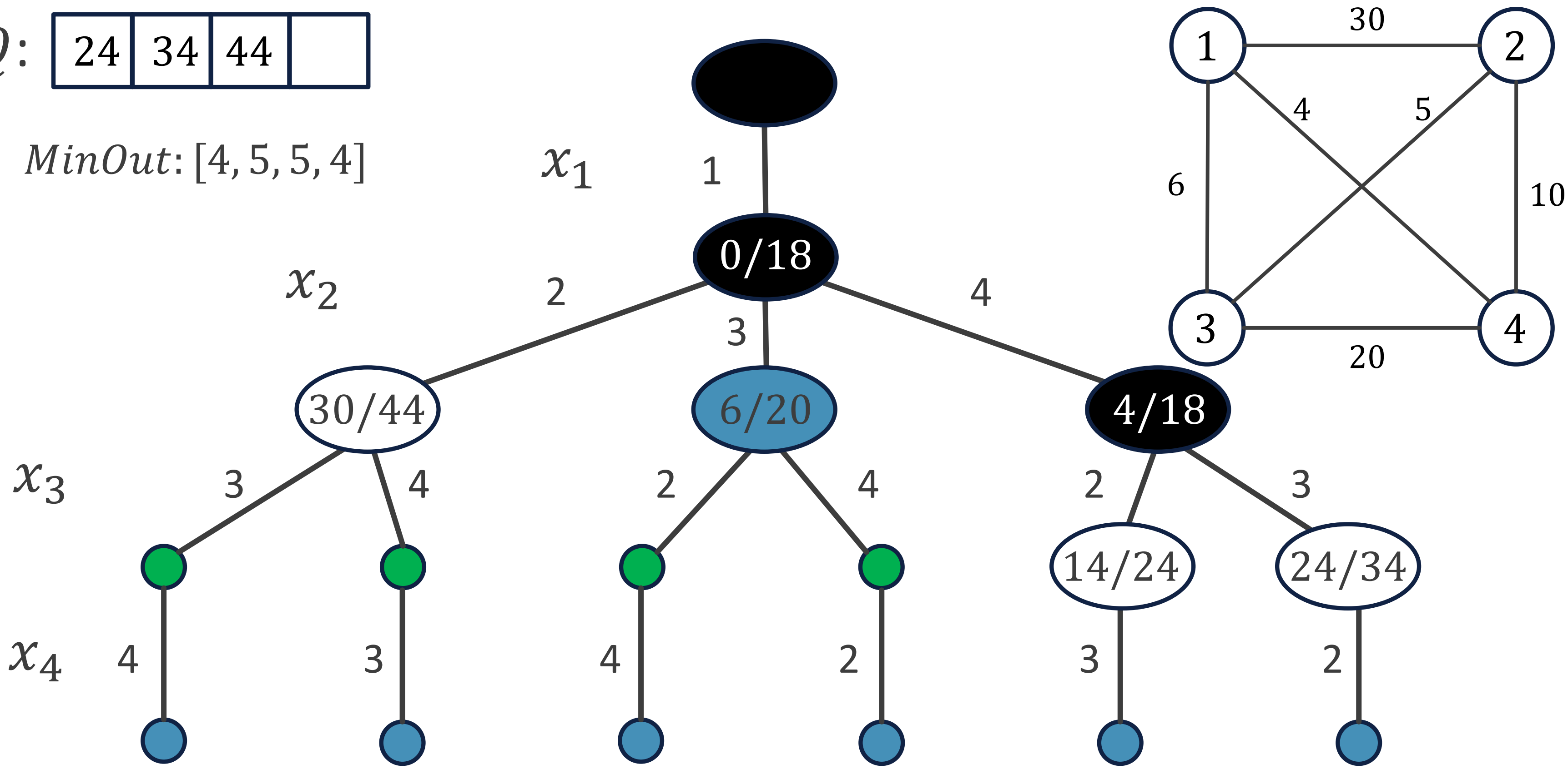
$MinOut: [4, 5, 5, 4]$



Example

Q: | 24 | 34 | 44 | |

$MinOut: [4, 5, 5, 4]$

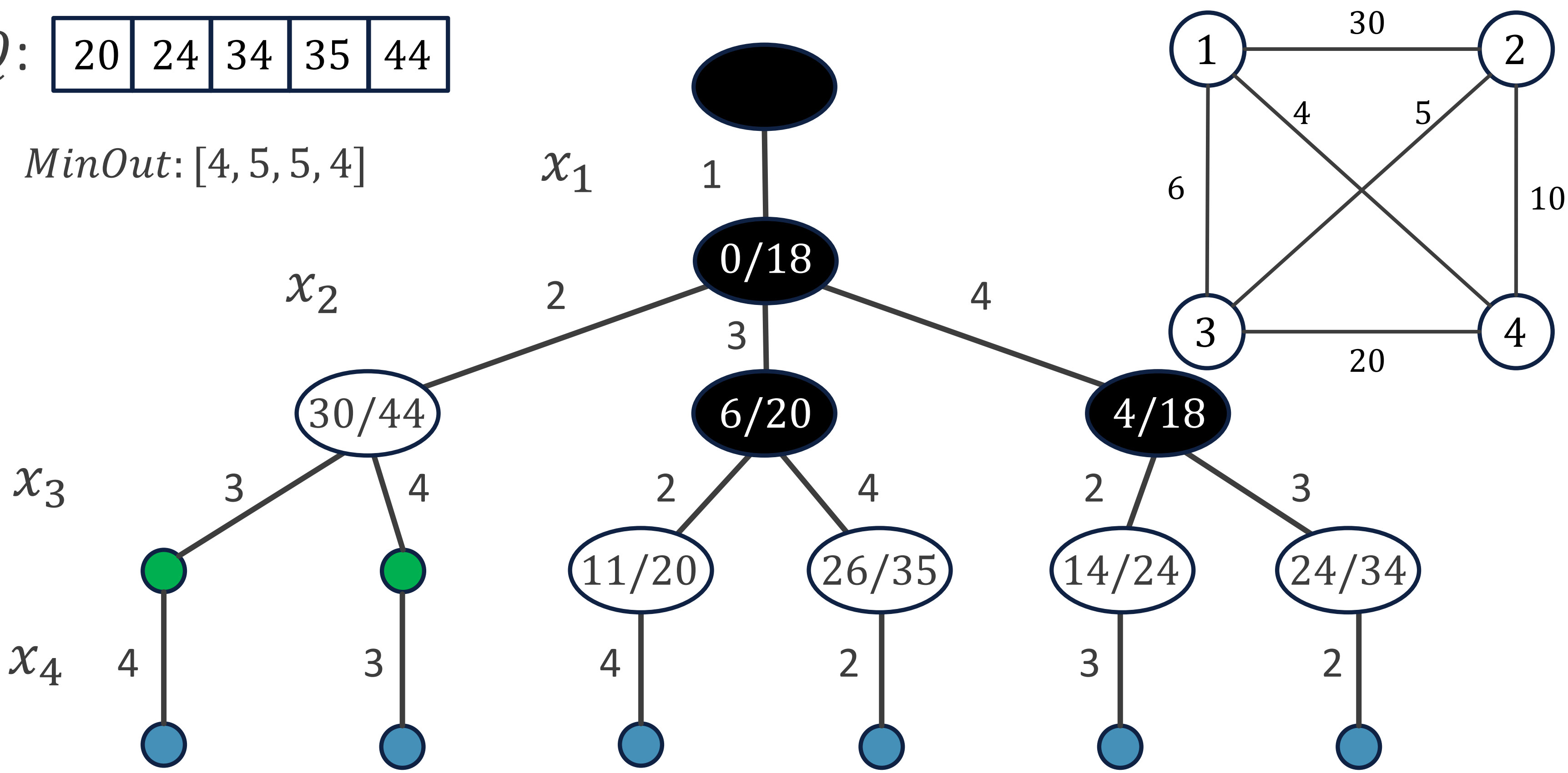


Example

Q:

20	24	34	35	44

$MinOut$: [4, 5, 5, 4]

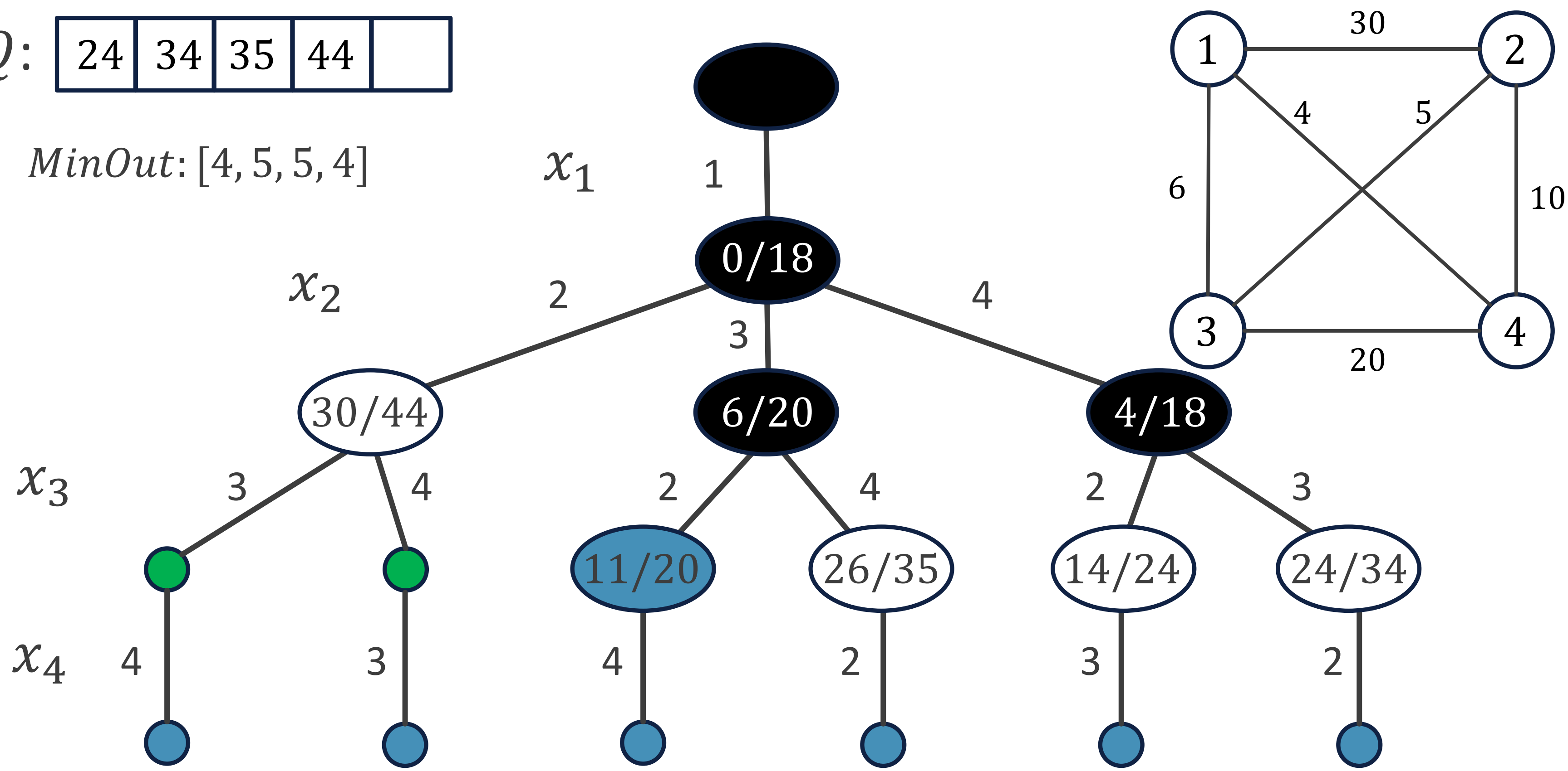


Example

Q:

24	34	35	44	

$MinOut$: $[4, 5, 5, 4]$

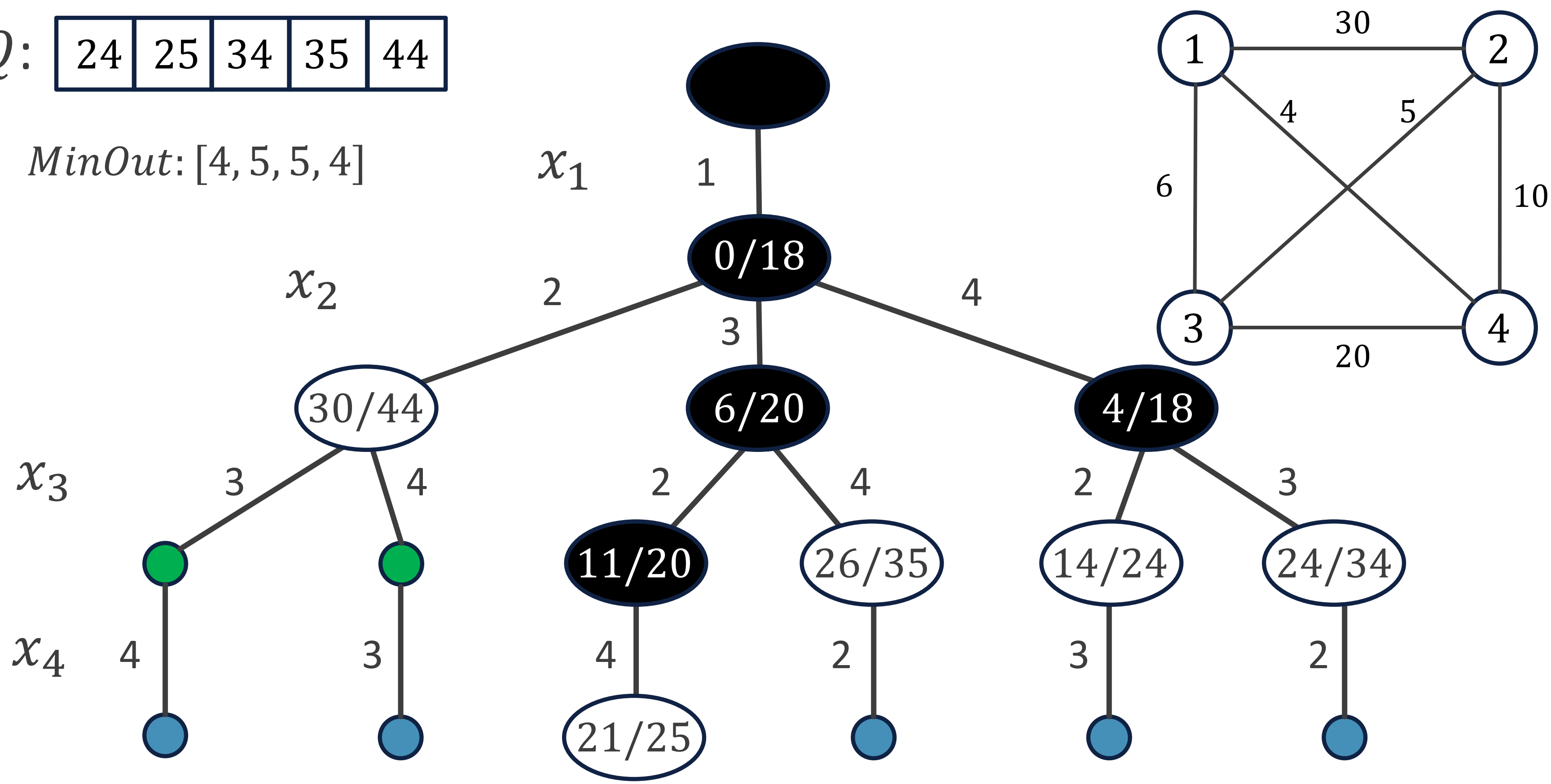


Example

Q:

24	25	34	35	44

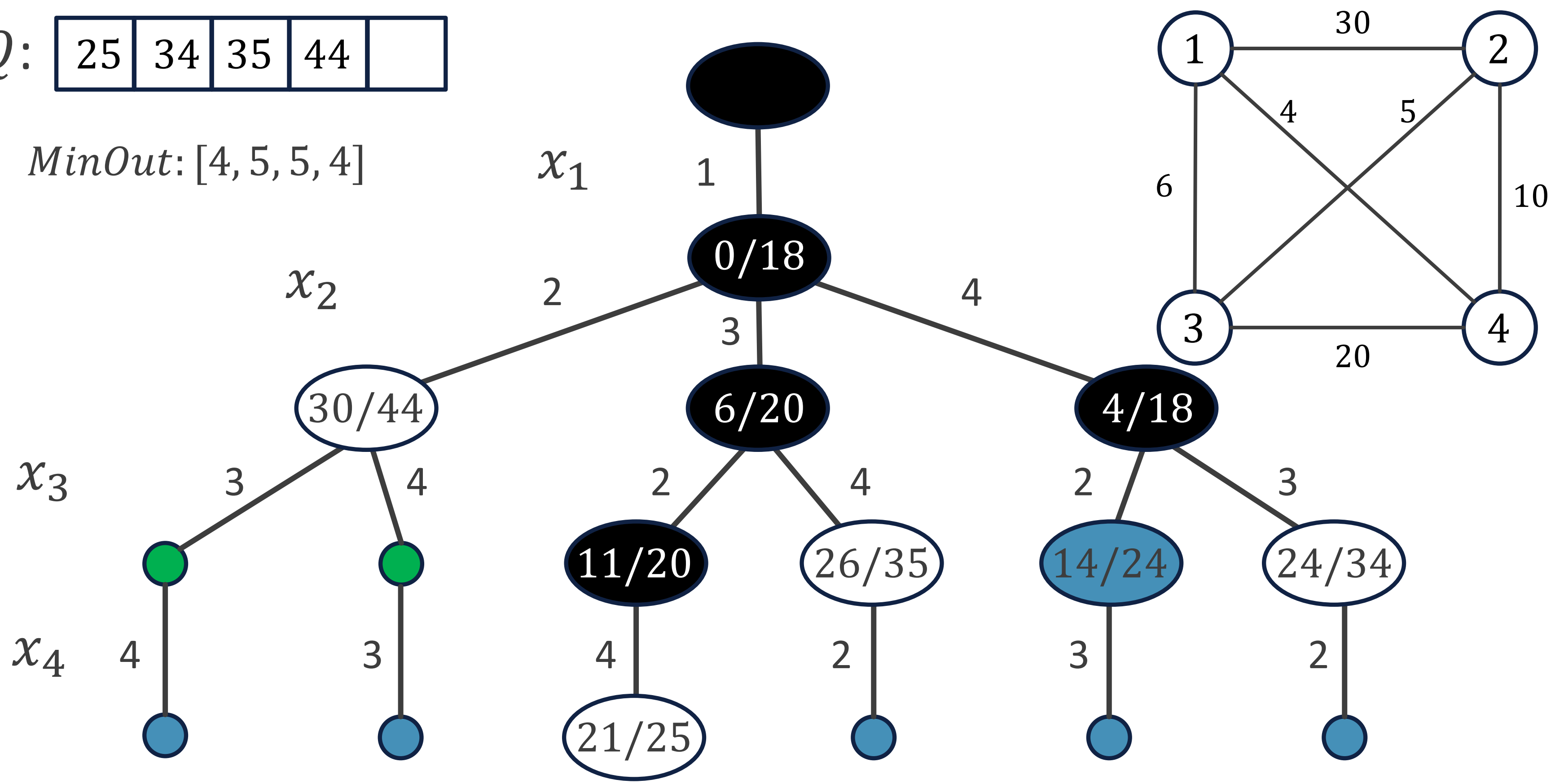
$MinOut$: [4, 5, 5, 4]



Example

Q: | 25 | 34 | 35 | 44 | |

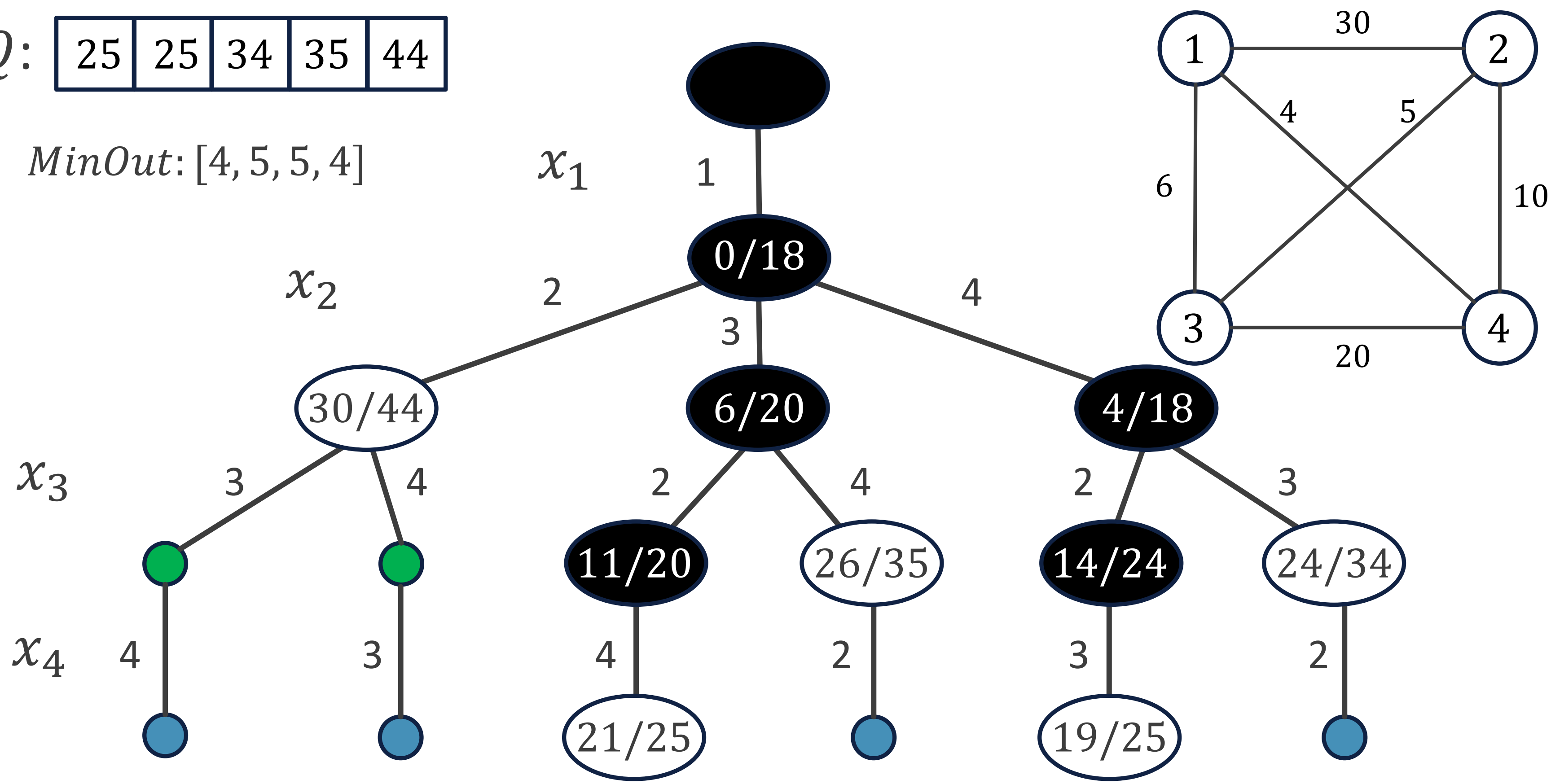
$MinOut$: [4, 5, 5, 4]



Example

Q: | 25 | 25 | 34 | 35 | 44 |

$MinOut$: [4, 5, 5, 4]

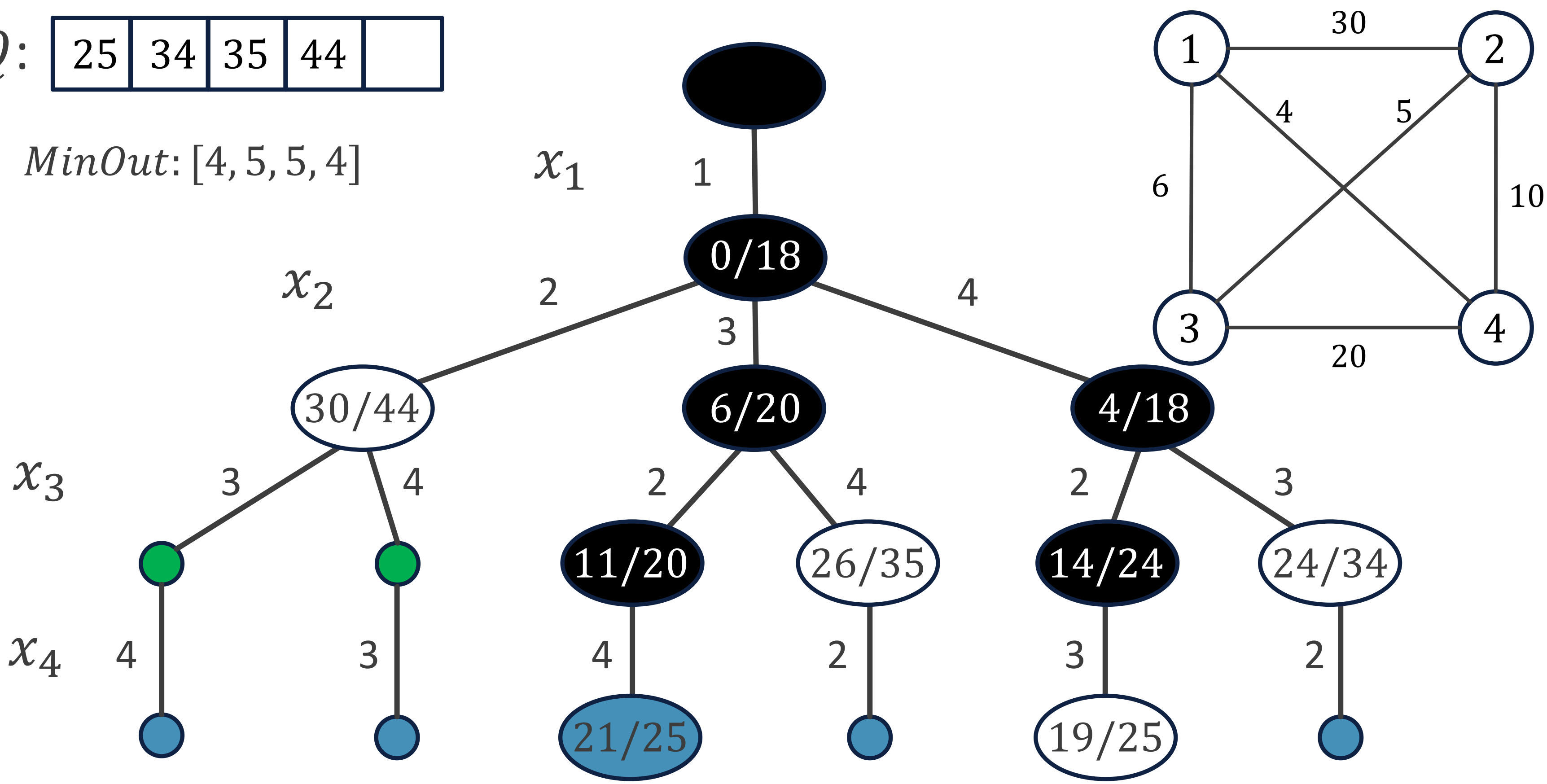


Example

Q:

25	34	35	44	

$MinOut: [4, 5, 5, 4]$

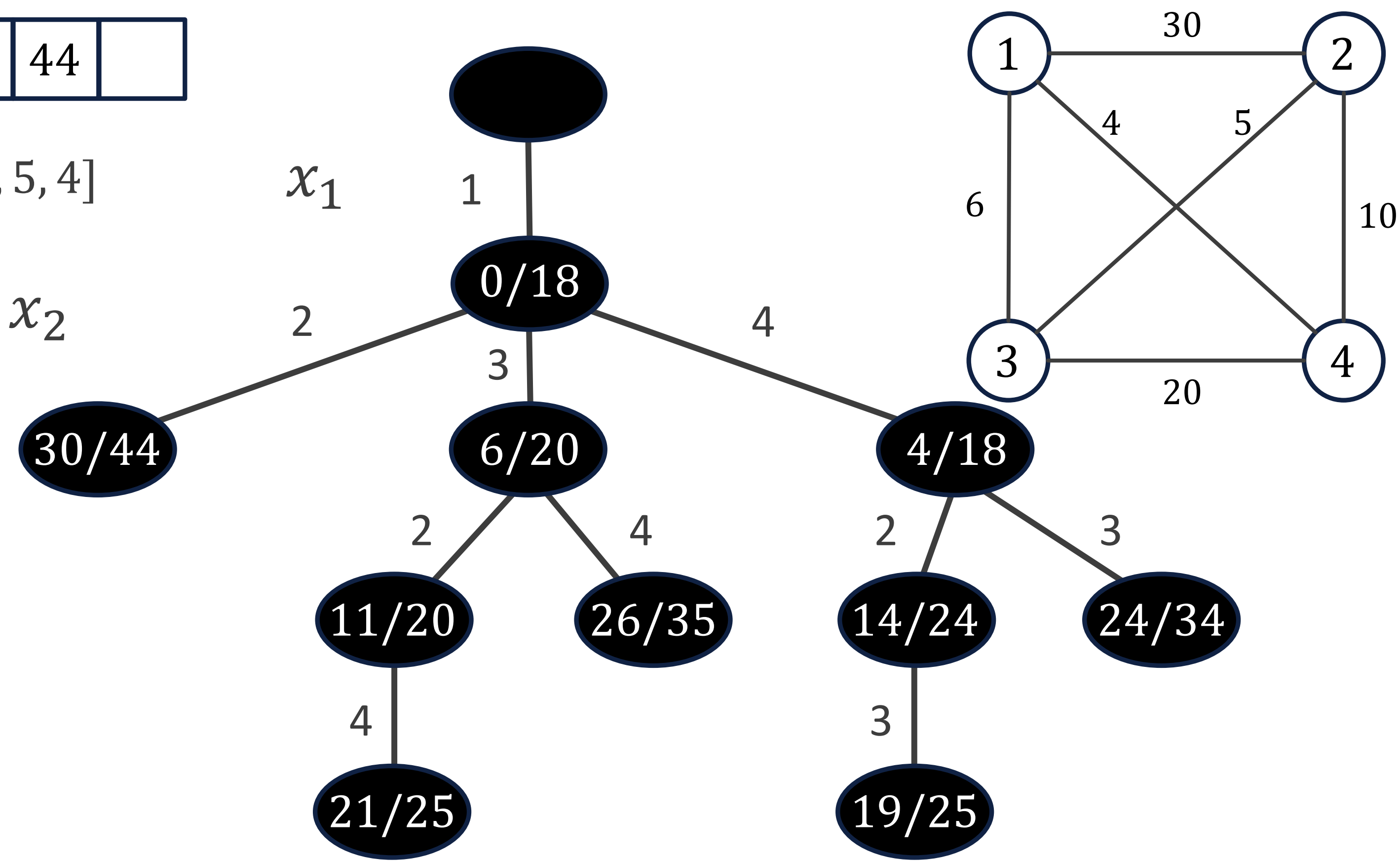


Example

Q:

25	34	35	44	

$MinOut$: [4, 5, 5, 4]

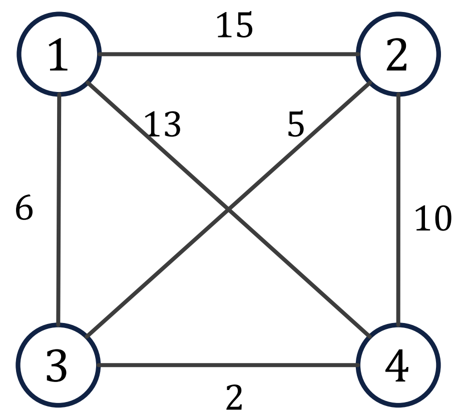


(1,3,2,4,1), 25

厦门大学信息学院
SCHOOL OF INFORMATICS XIAMEN UNIVERSITY
厦门大学计算机科学系
Computer Science Department of Xiamen University

Classroom Exercise

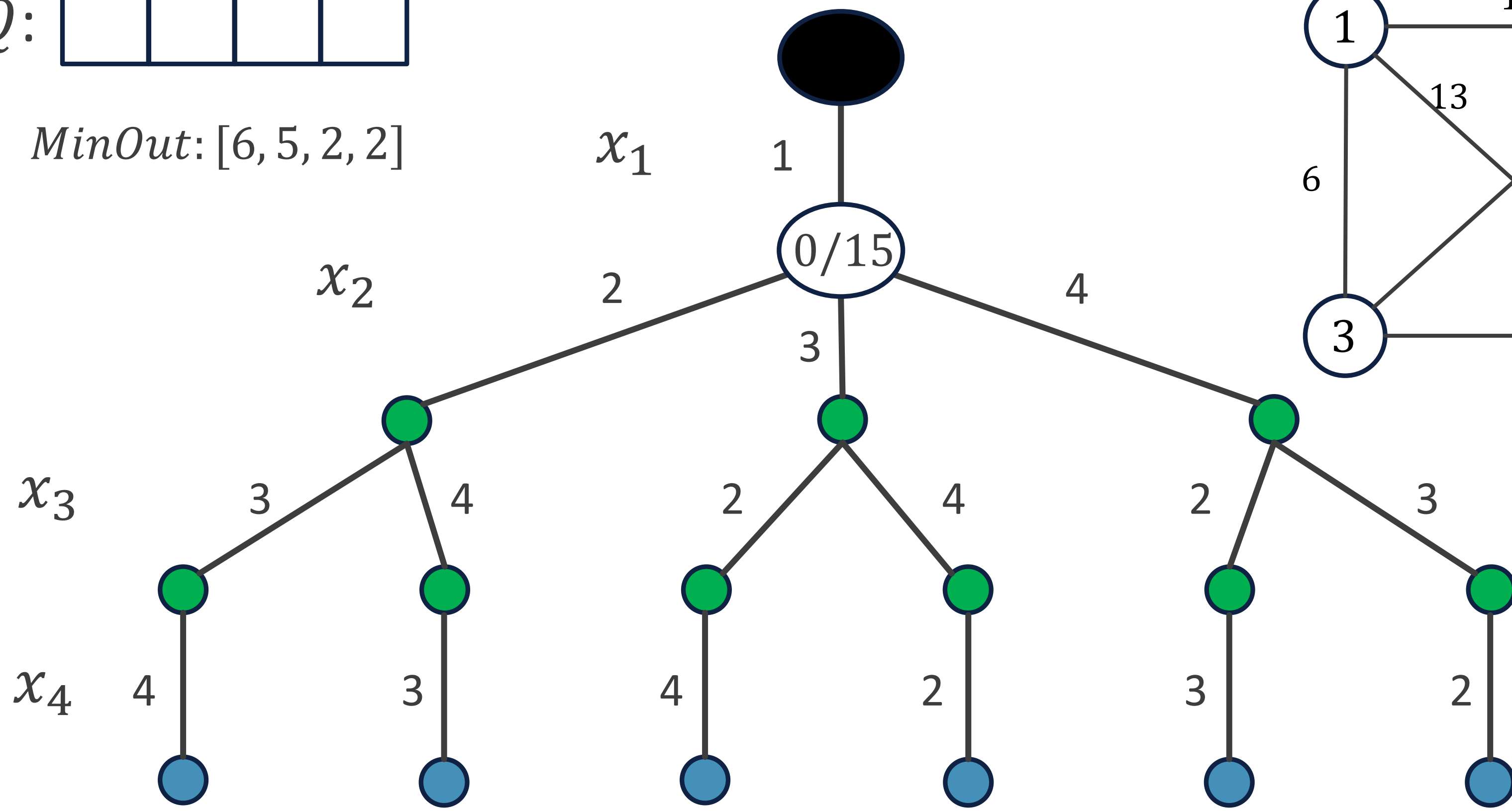
- Draw the pruned solution space tree for the following TSP instance by max-profit branch-and-bound.

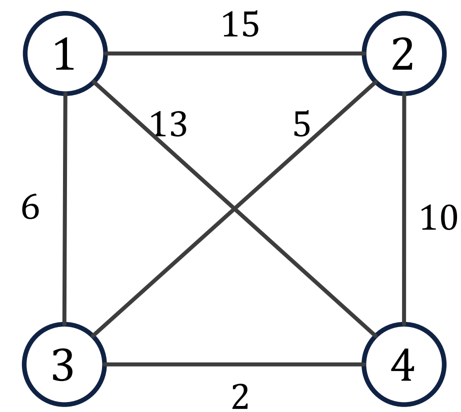


Classroom Exercise

Q: | | | | |
|---|---|---|---|
| | | | |

$MinOut: [6, 5, 2, 2]$



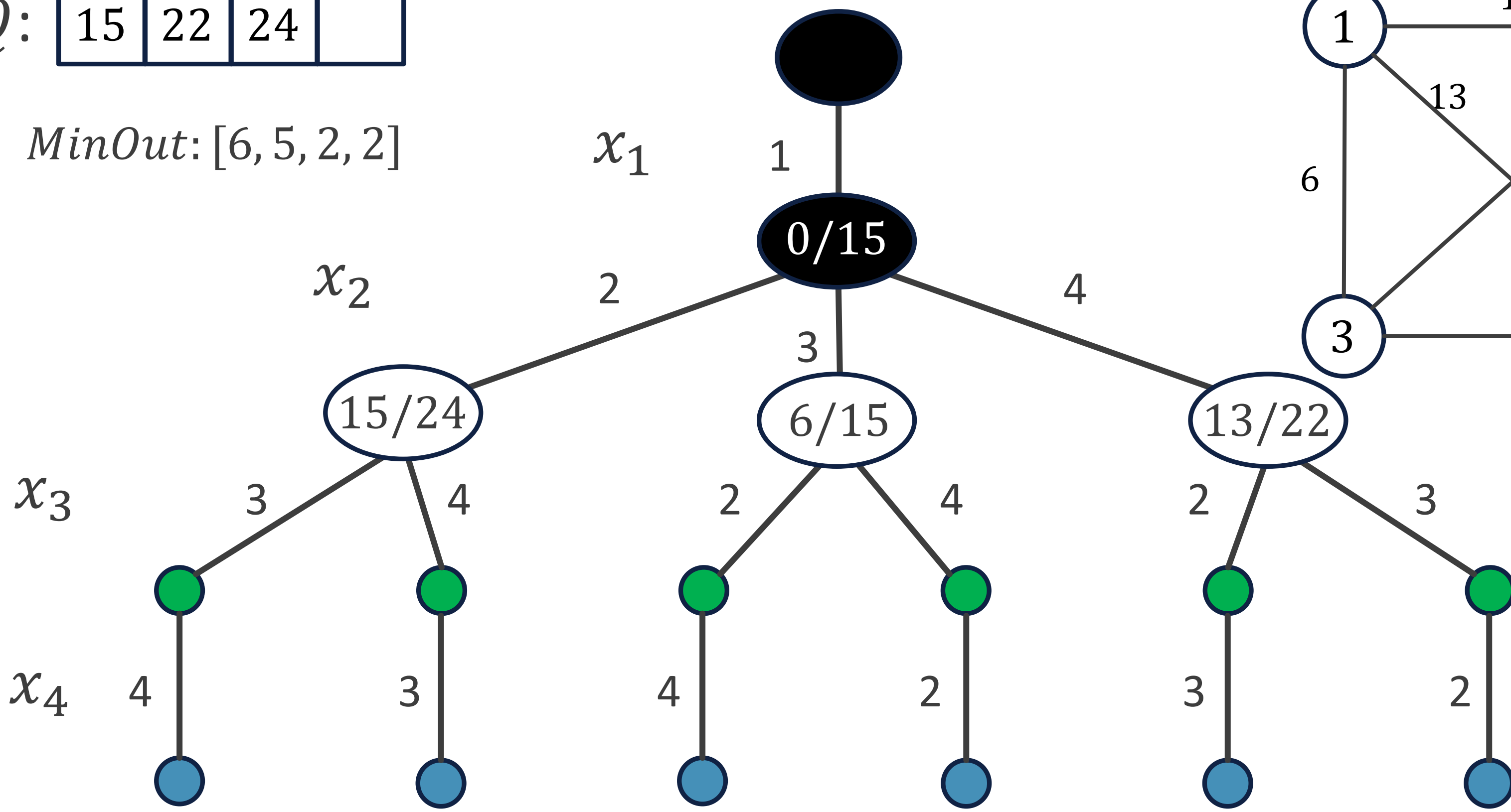


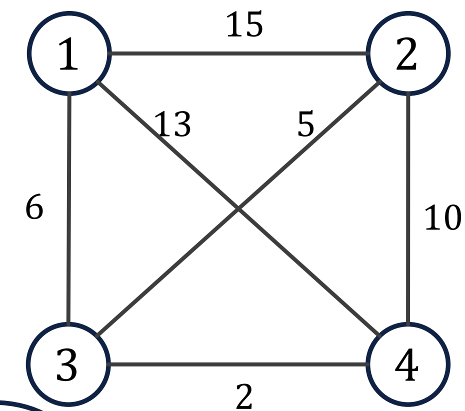
Classroom Exercise

Q:

15	22	24	

$MinOut$: $[6, 5, 2, 2]$



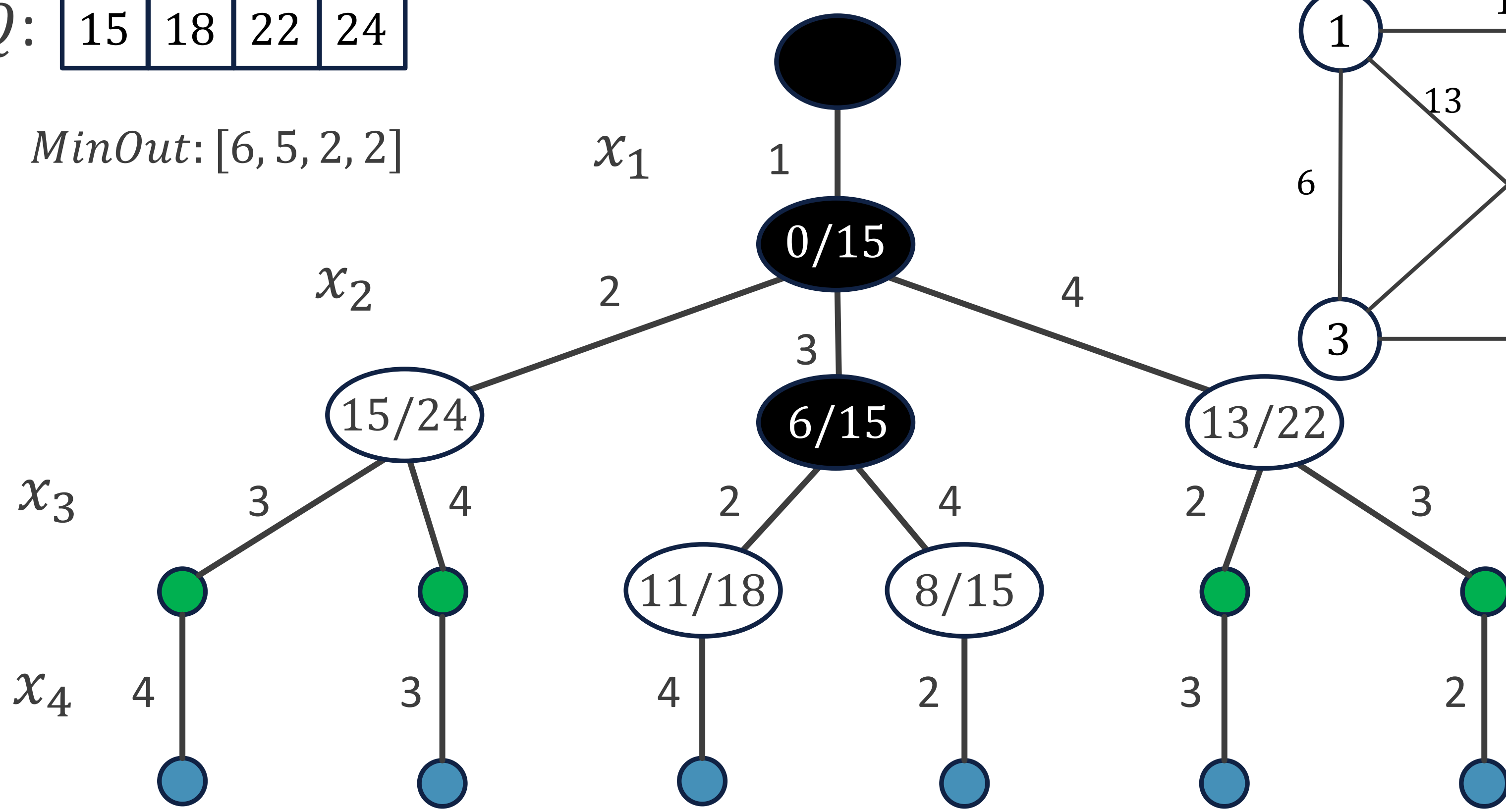


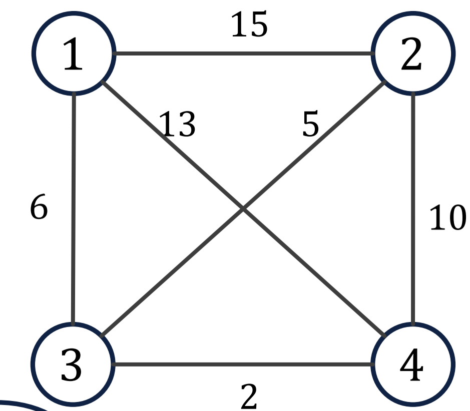
Classroom Exercise

Q:

15	18	22	24

$MinOut$: [6, 5, 2, 2]

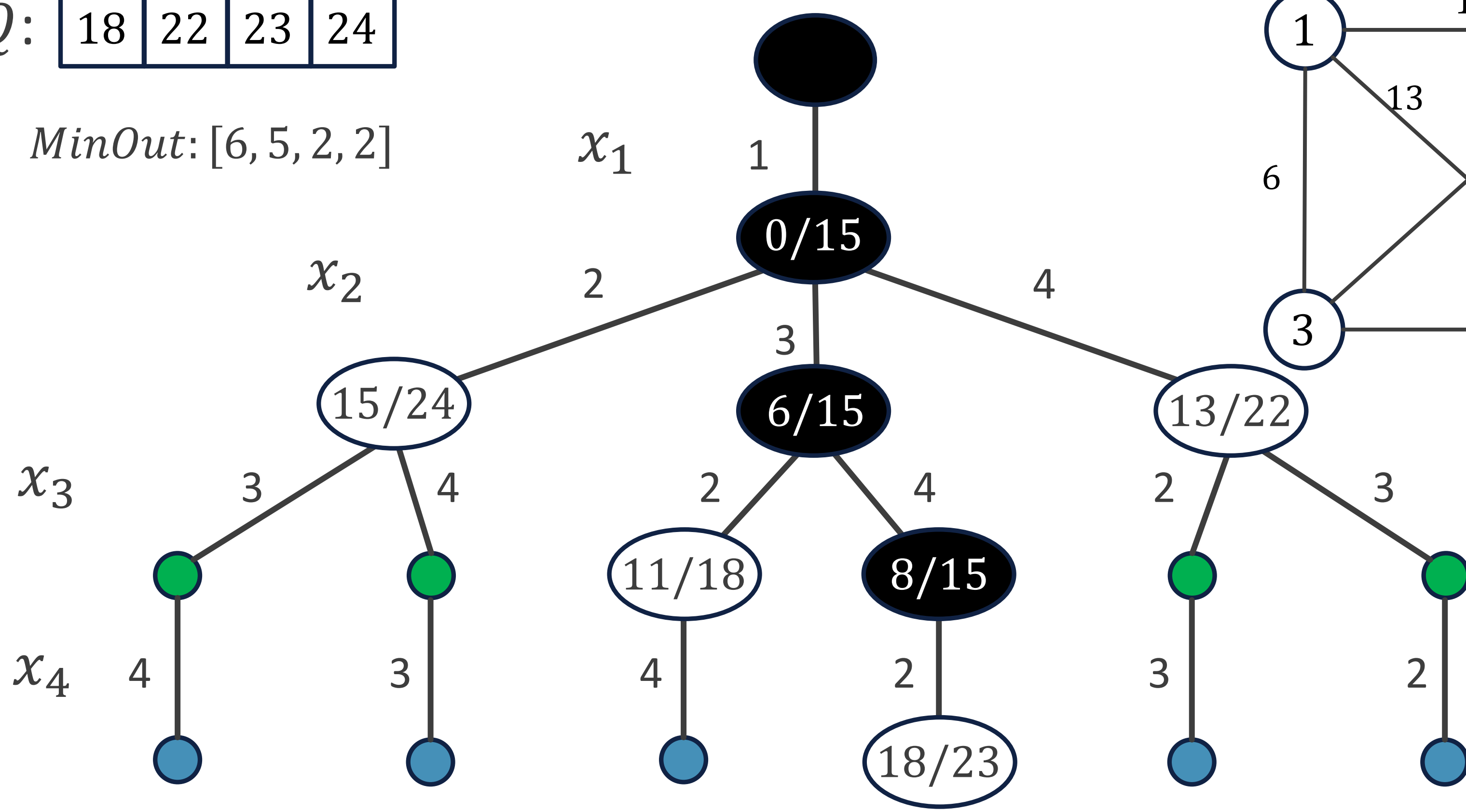


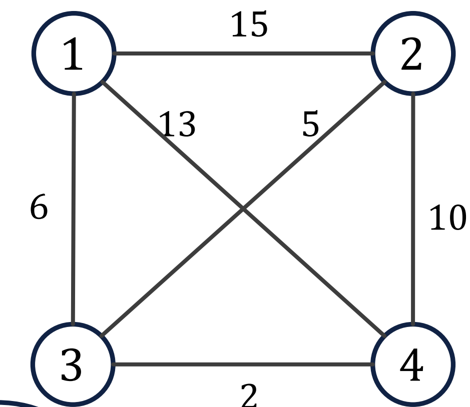


Classroom Exercise

Q: | 18 | 22 | 23 | 24 |

$MinOut$: [6, 5, 2, 2]

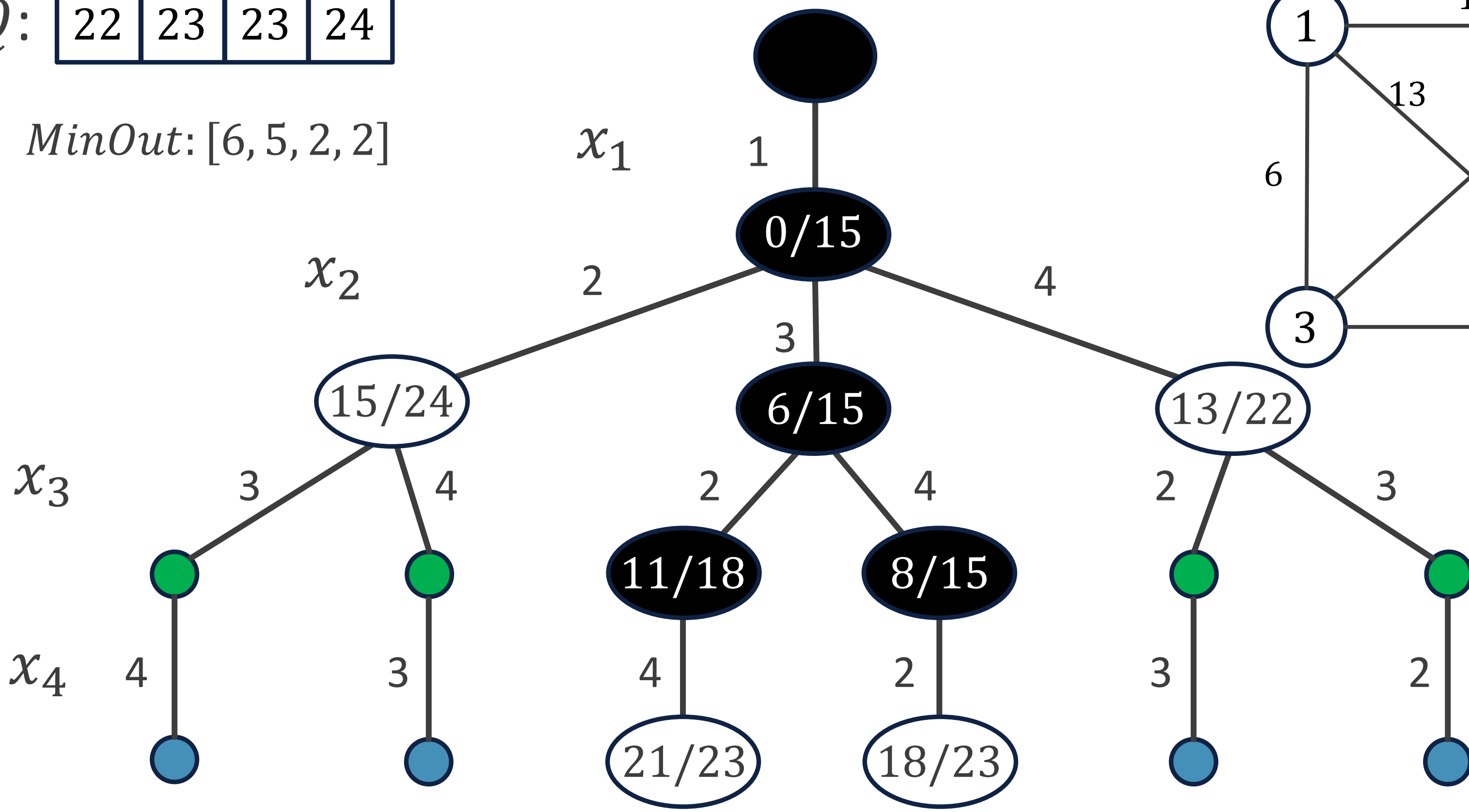


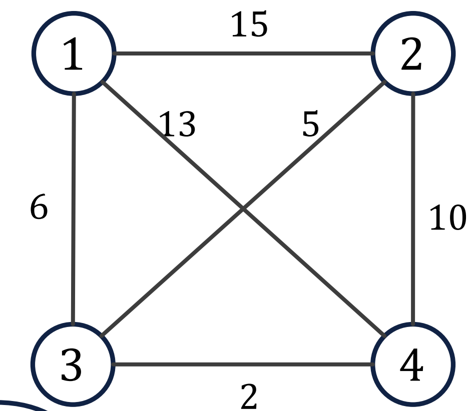


Classroom Exercise

Q: | 22 | 23 | 23 | 24 |

$MinOut$: [6, 5, 2, 2]



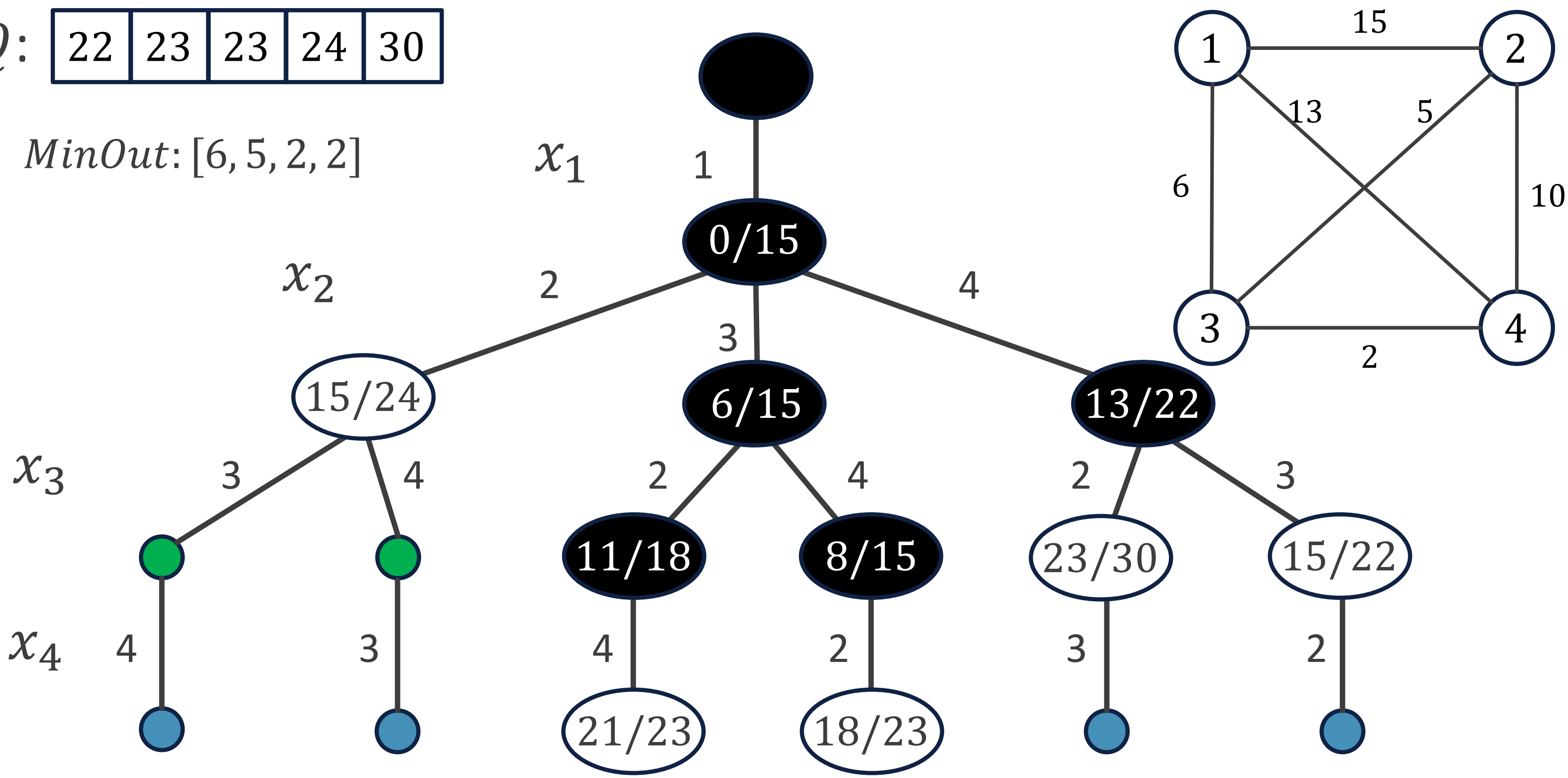


Classroom Exercise

Q:

22	23	23	24	30

$MinOut$: [6, 5, 2, 2]

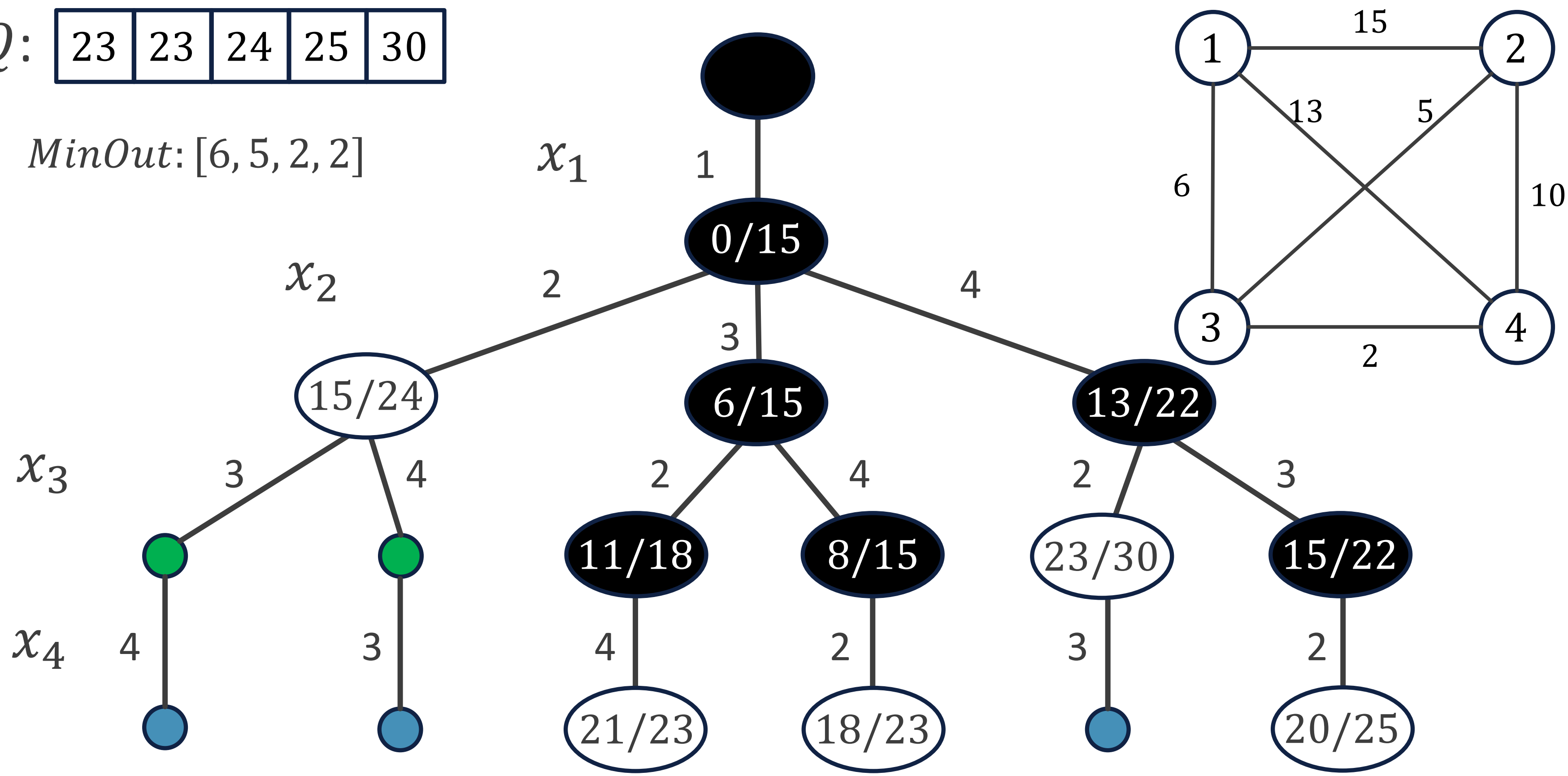


Classroom Exercise

Q:

23	23	24	25	30

$MinOut$: [6, 5, 2, 2]

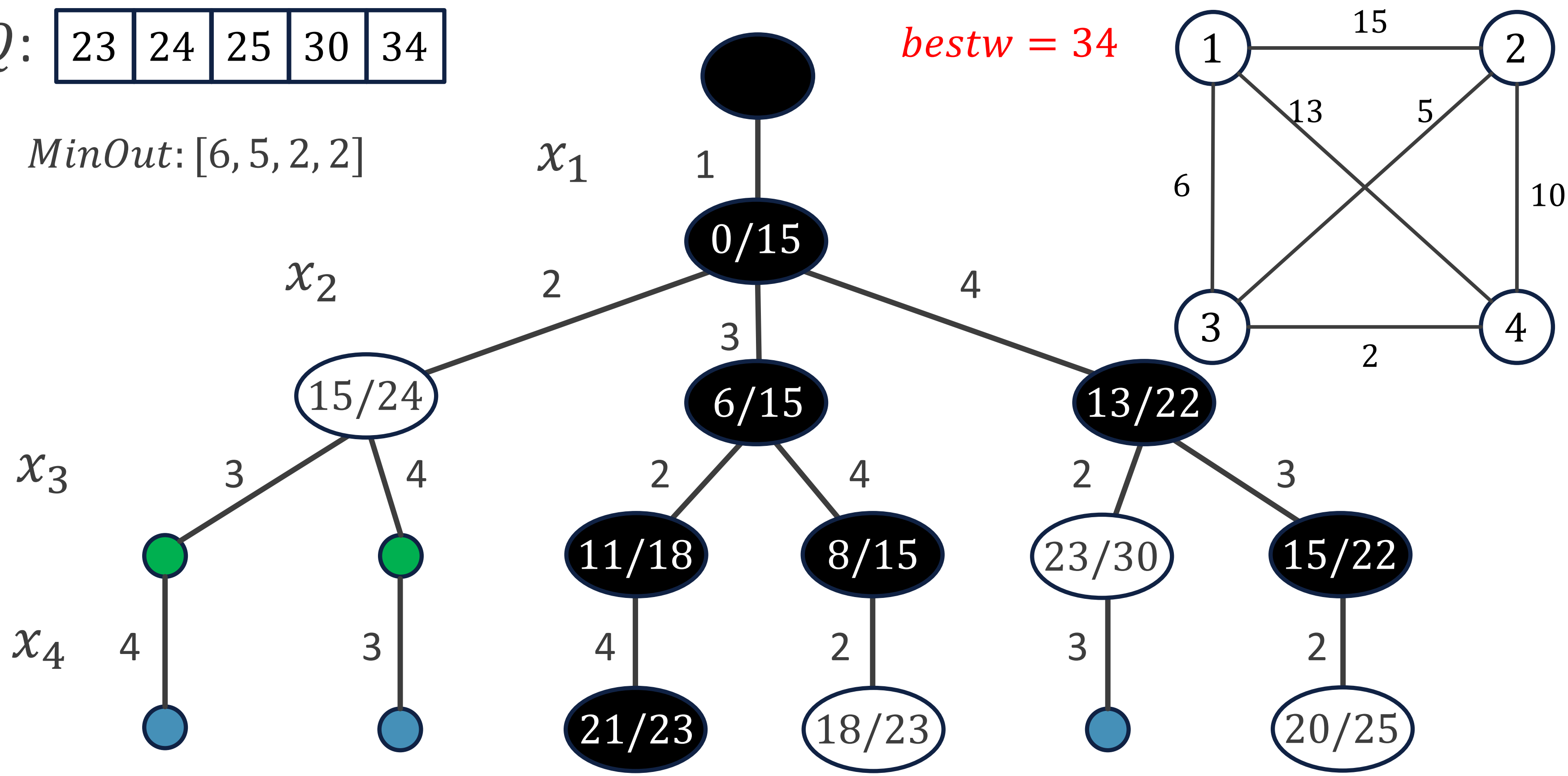


Classroom Exercise

Q: | 23 | 24 | 25 | 30 | 34 |

$MinOut$: [6, 5, 2, 2]

$bestw = 34$



(1,3,2,4,1), 34

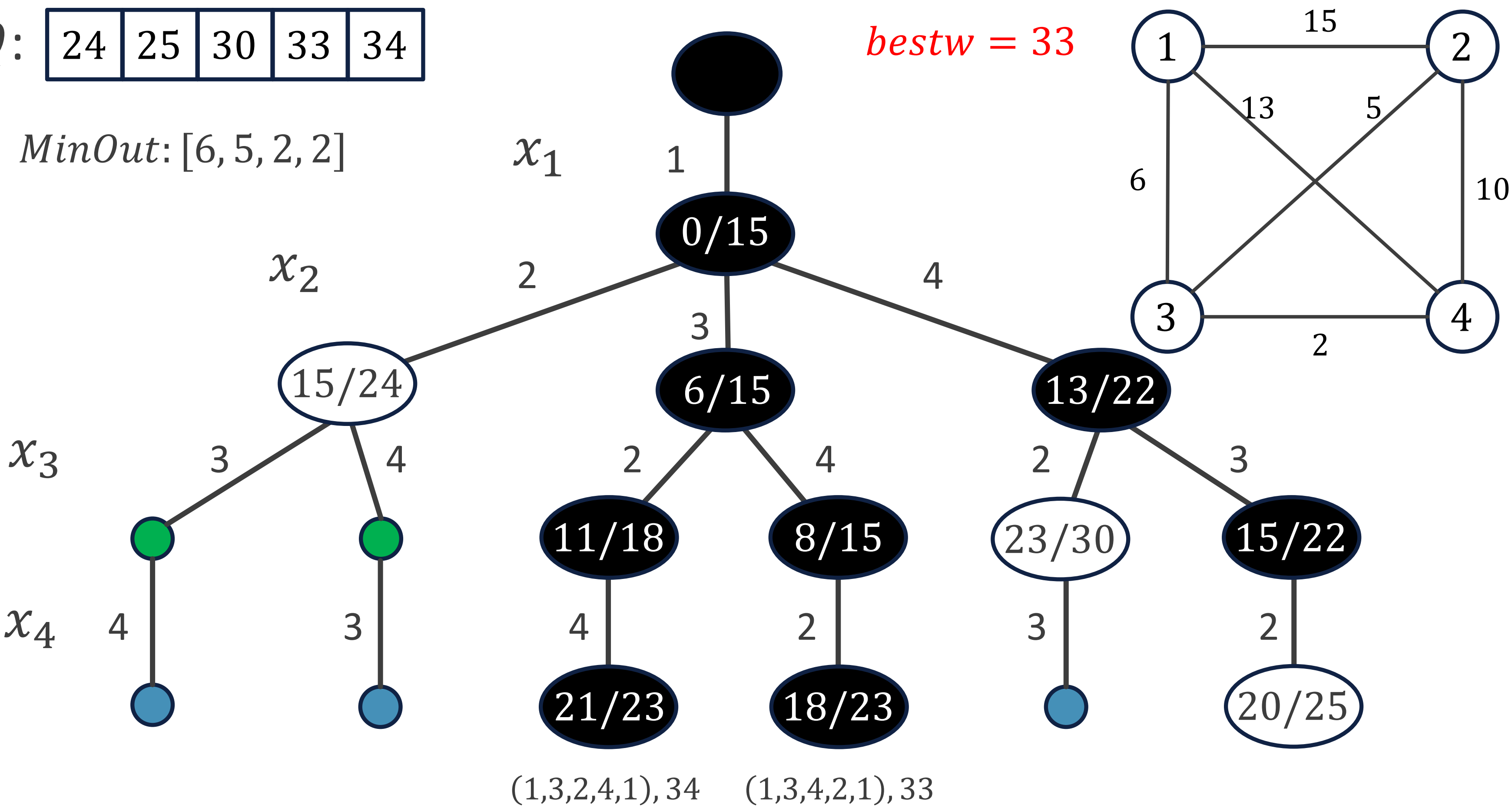
Not finish yet!

Classroom Exercise

Q: | 24 | 25 | 30 | 33 | 34 |

$MinOut$: $[6, 5, 2, 2]$

$bestw = 33$



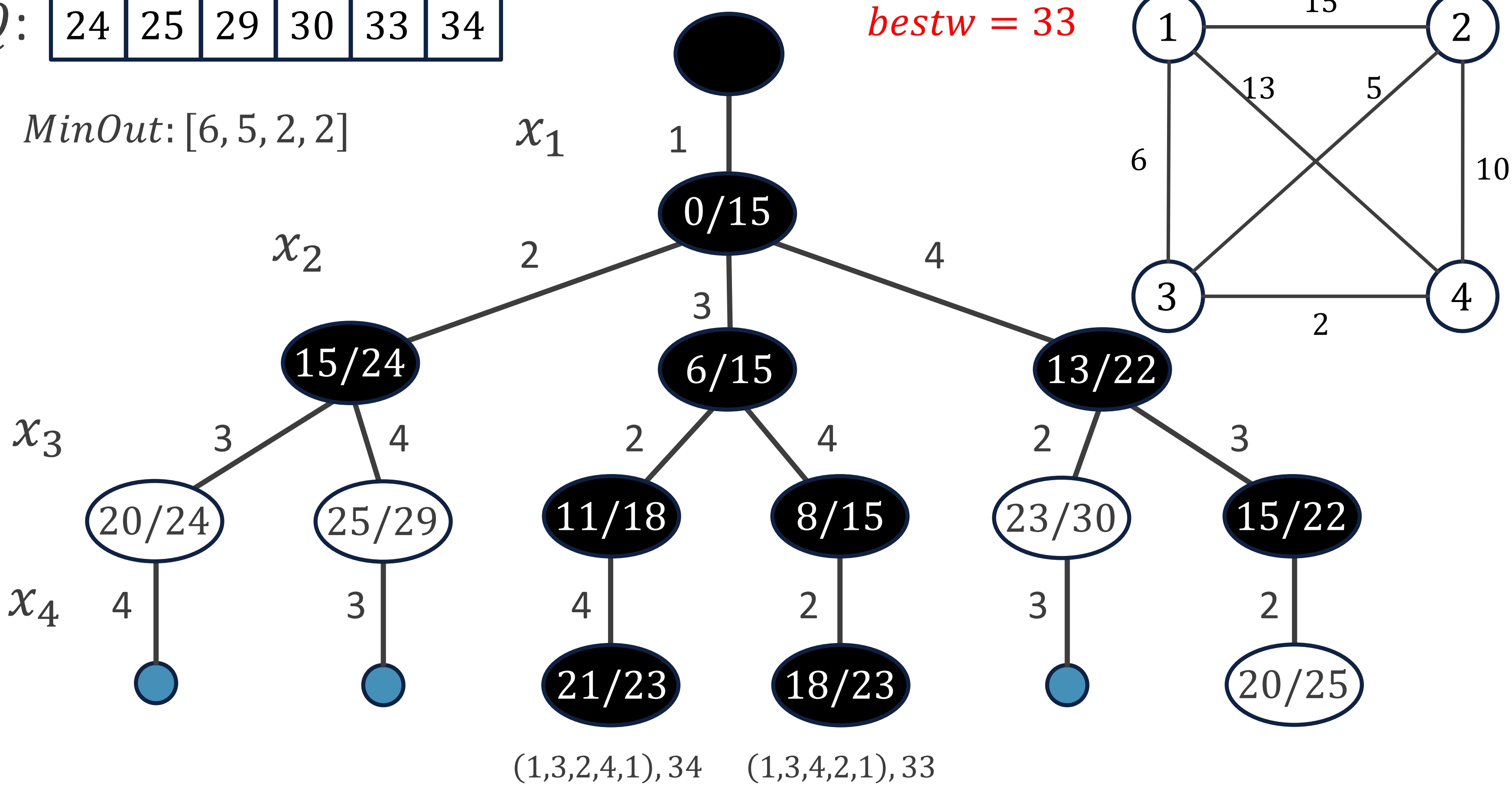
Classroom Exercise

Q:

24	25	29	30	33	34

$MinOut$: [6, 5, 2, 2]

$bestw = 33$

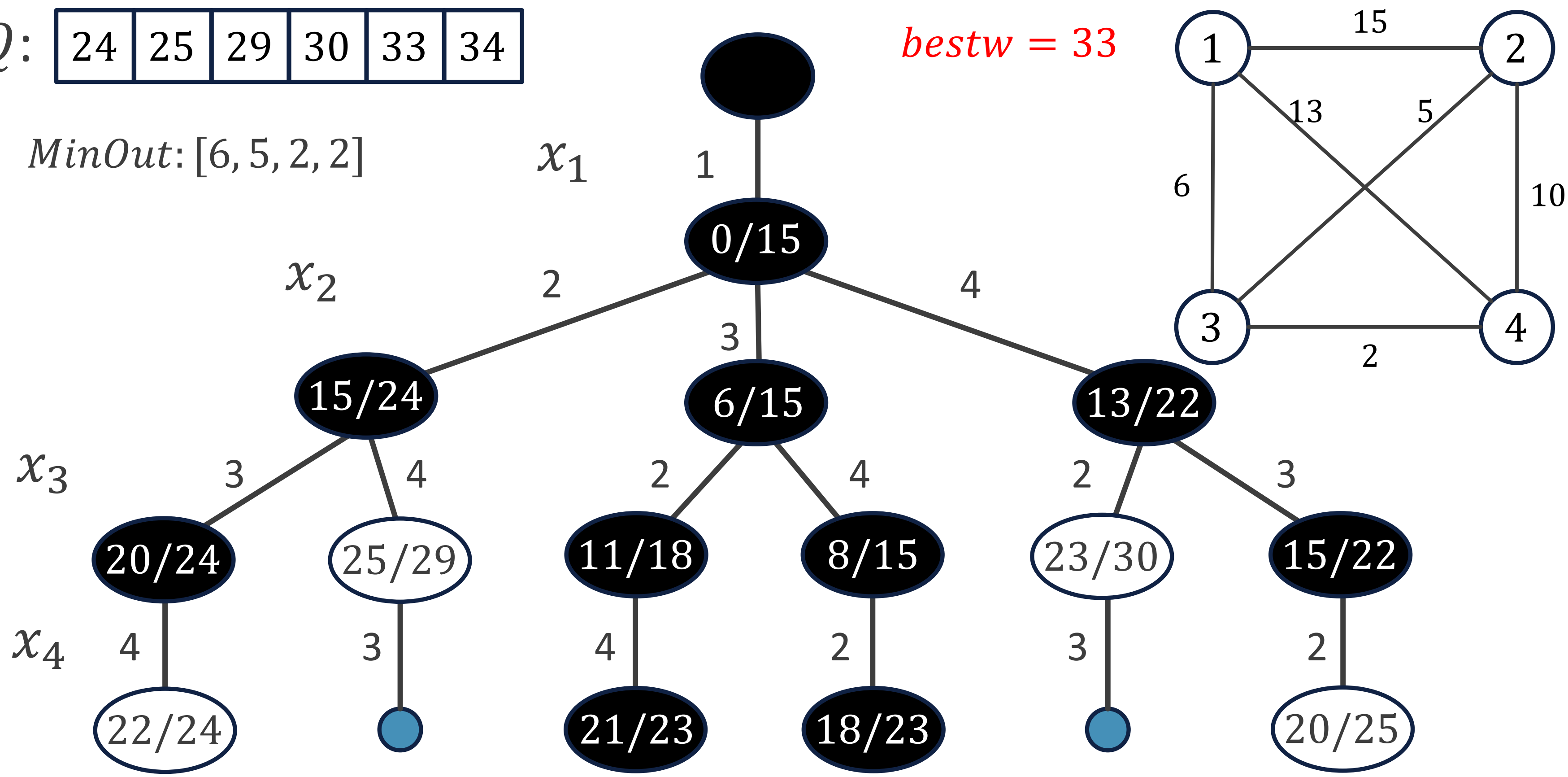


Classroom Exercise

Q: | 24 | 25 | 29 | 30 | 33 | 34 |

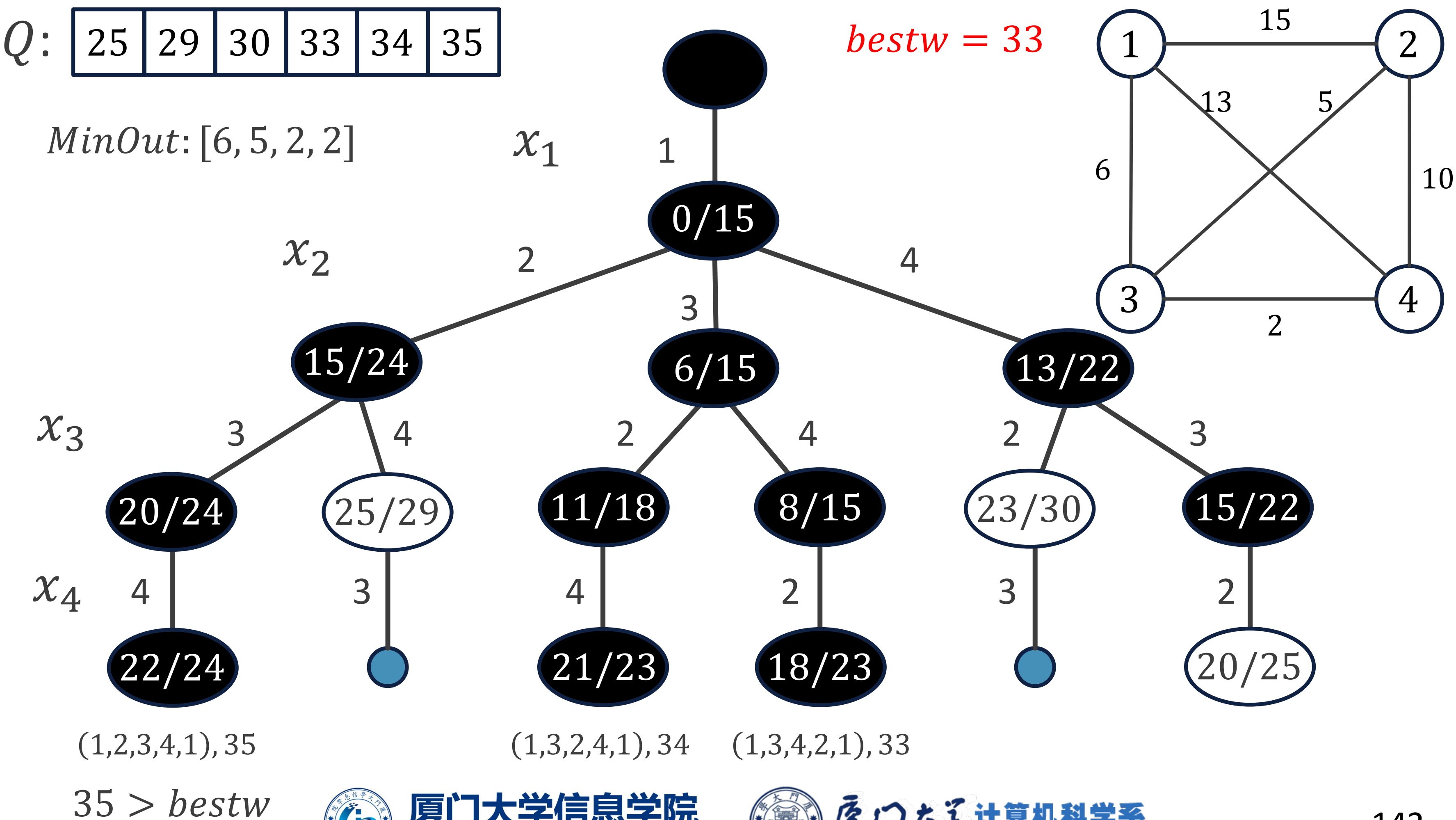
$bestw = 33$

$MinOut$: $[6, 5, 2, 2]$



(1,3,2,4,1), 34   (1,3,4,2,1), 33

Classroom Exercise



厦门大学信息学院 SCHOOL OF INFORMATICS XIAMEN UNIVERSITY 厦门大学计算机科学系 Computer Science Department of Xiamen University

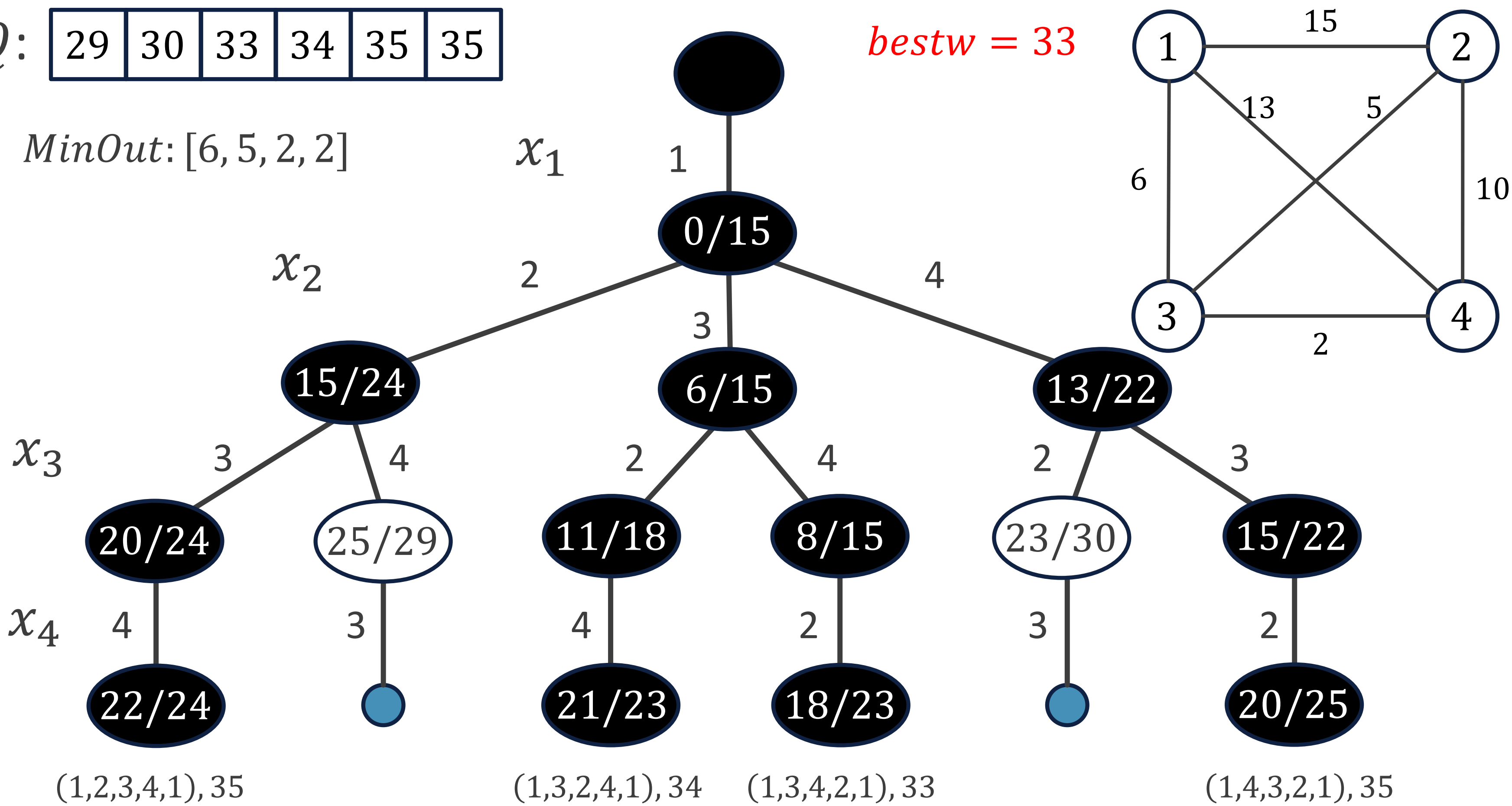
Classroom Exercise

Q:

29	30	33	34	35	35

$MinOut$: [6, 5, 2, 2]

$bestw = 33$



$35 > bestw$

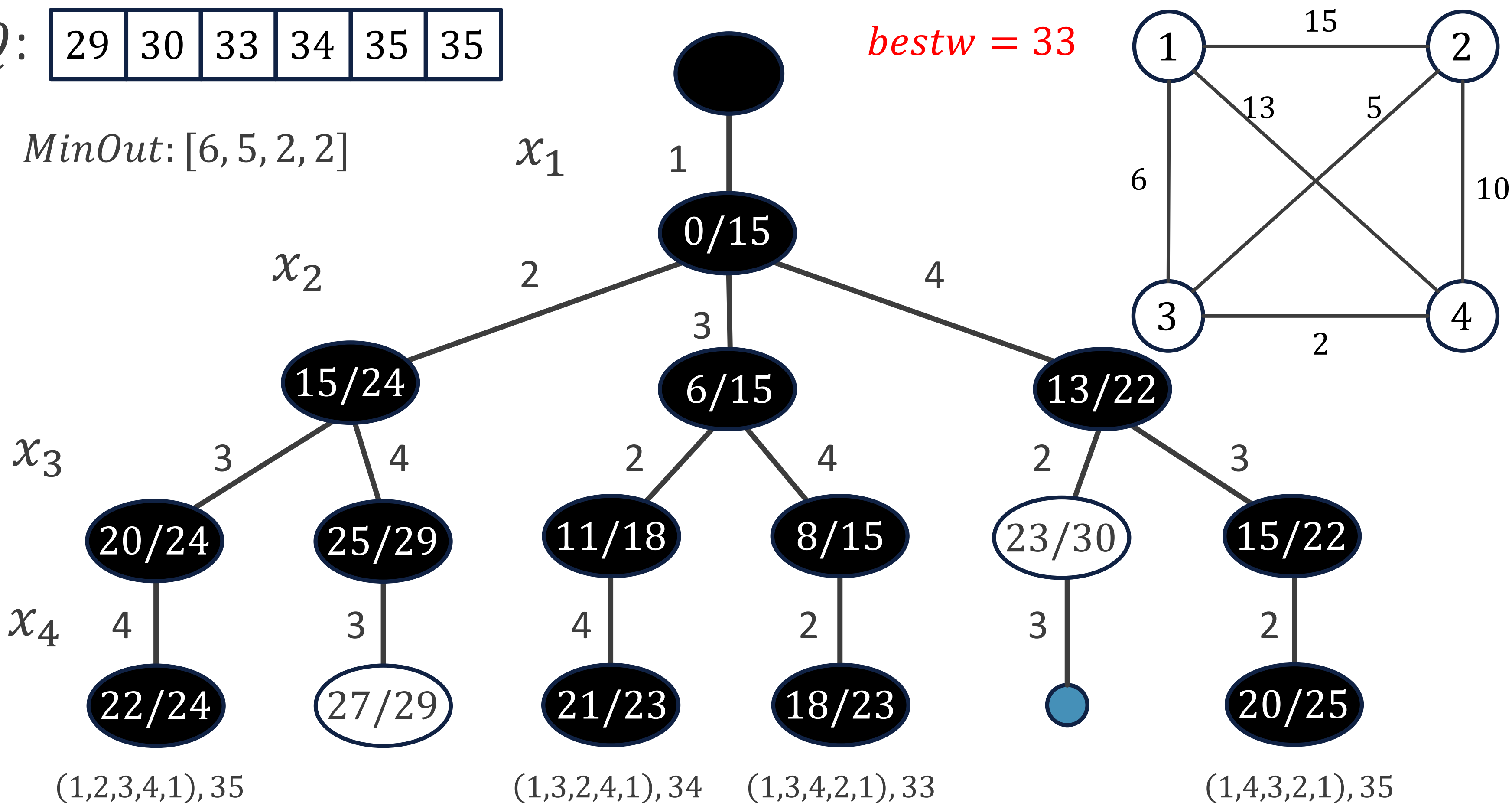
Classroom Exercise

Q:

29	30	33	34	35	35

$MinOut$: [6, 5, 2, 2]

$bestw = 33$



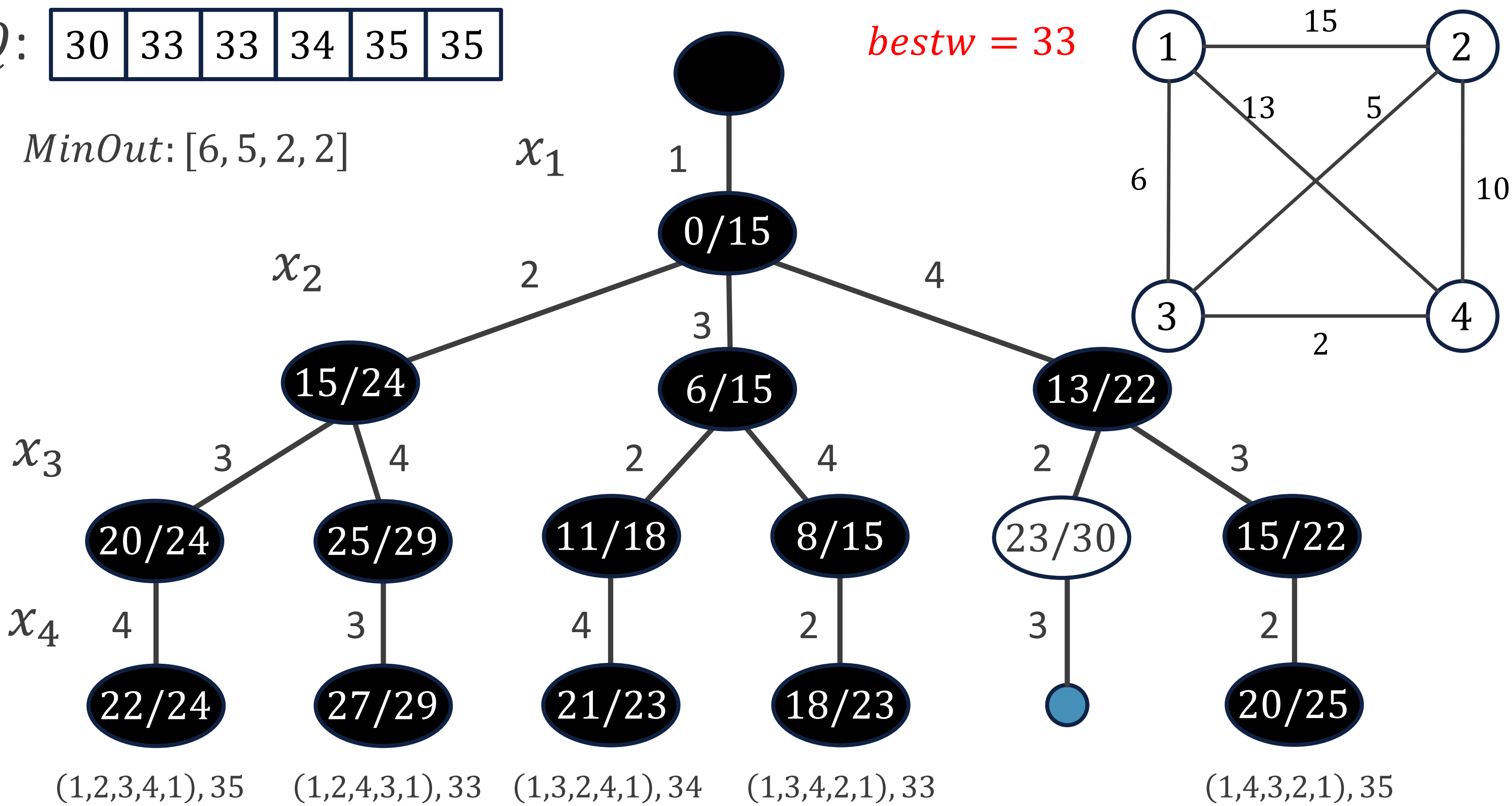
Classroom Exercise

Q:

30	33	33	34	35	35

$MinOut$: $[6, 5, 2, 2]$

$bestw = 33$

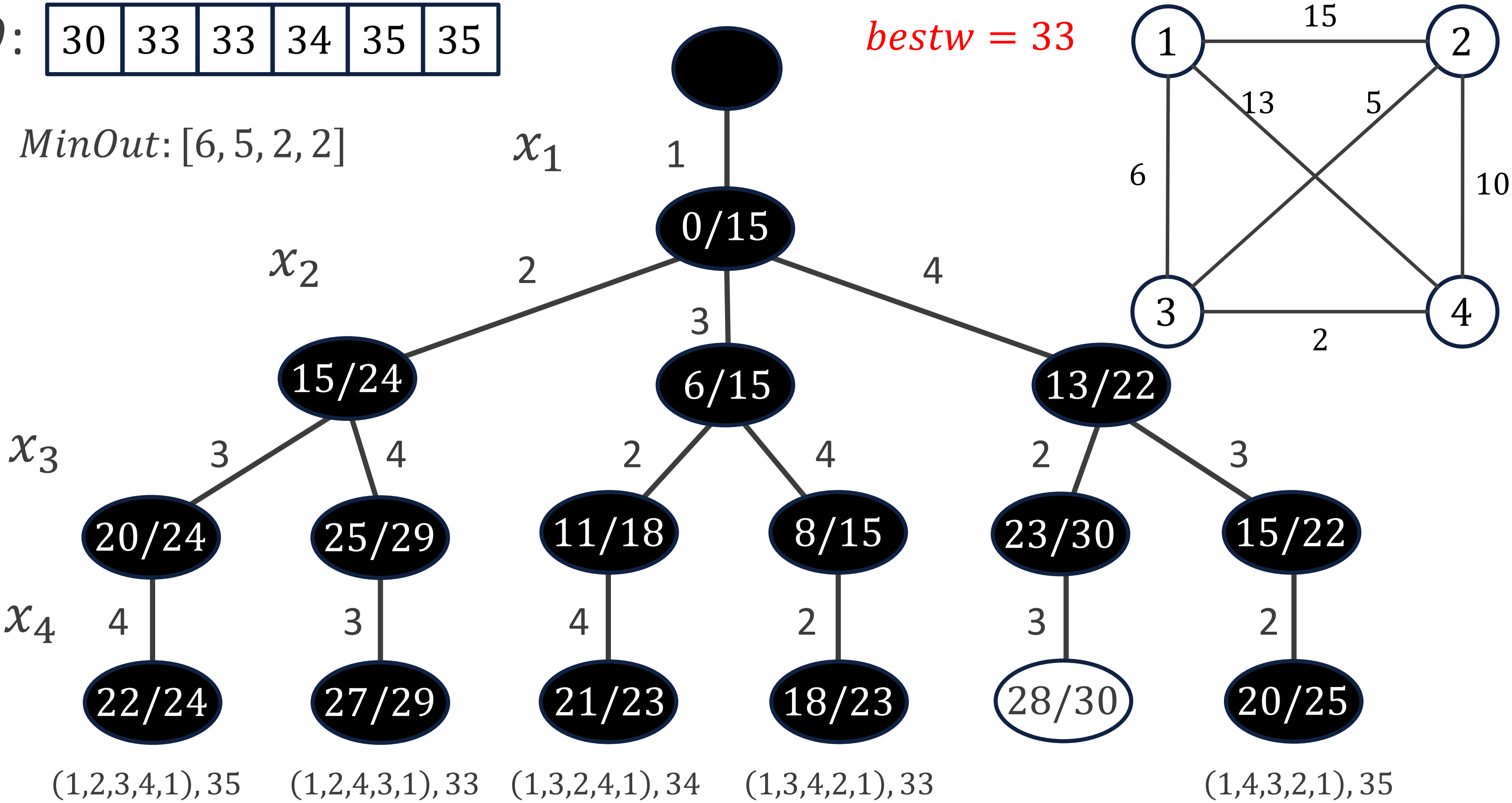


$33 = bestw$

Classroom Exercise

Q: | 30 | 33 | 33 | 34 | 35 | 35 |

$MinOut$: [6, 5, 2, 2]

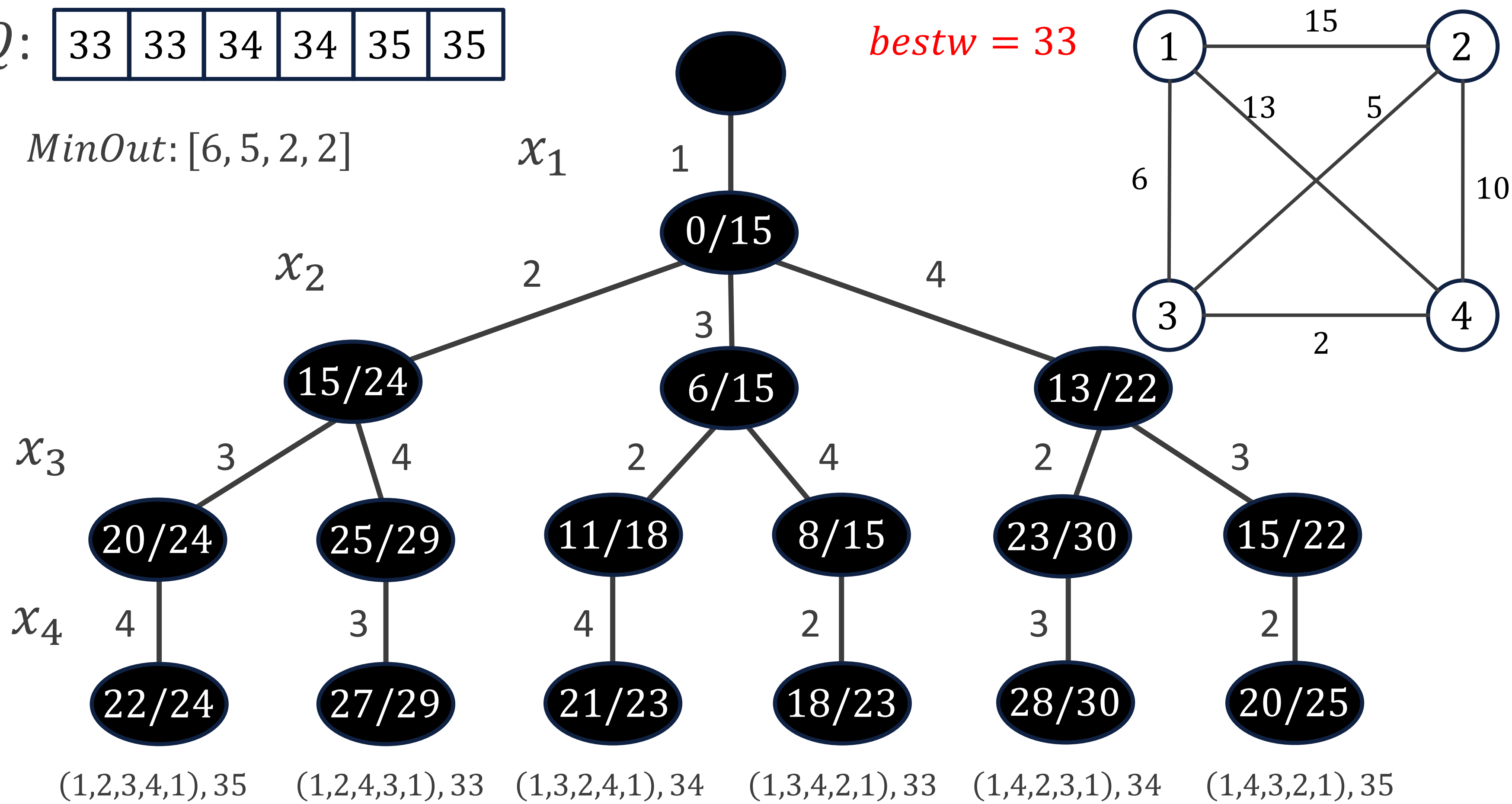


$33 = bestw$

Classroom Exercise

Q: | 33 | 33 | 34 | 34 | 35 | 35 |

$MinOut$: [6, 5, 2, 2]

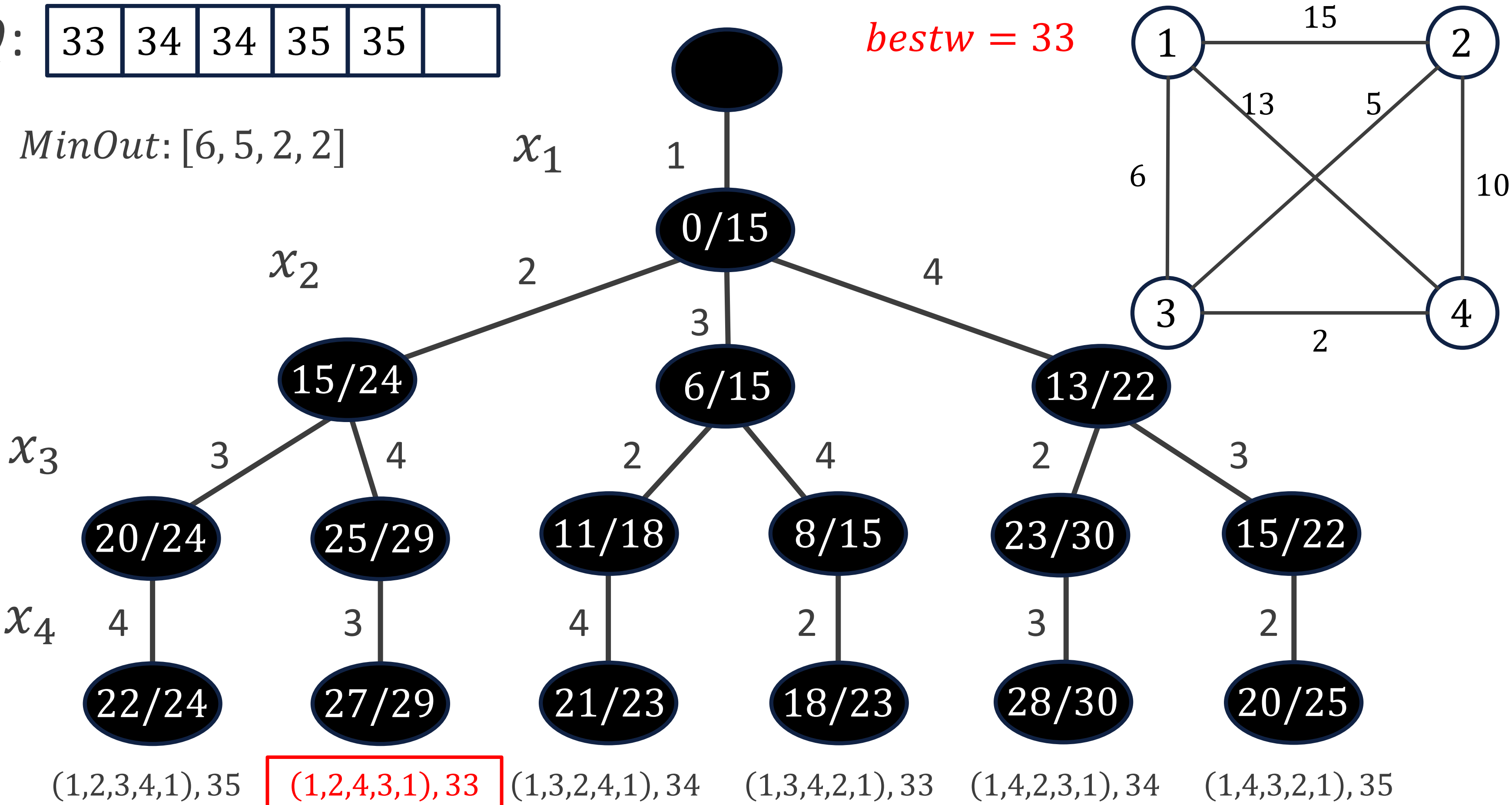


Classroom Exercise

Q: | 33 | 34 | 34 | 35 | 35 | |

$MinOut$: [6, 5, 2, 2]

$bestw = 33$



Worst case happens!

FLOW SHOP SCHEDULING PROBLEM

Flow Shop Scheduling Problem

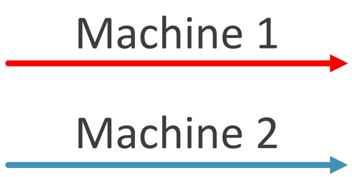
- Given n jobs $J = (j_1, j_2, \dots, j_n)$, each job has two operations processed by two machines.
- One machine can only process a single job at a time, and processing must be completed once initiated.
- Furthermore, machine 2 cannot begin processing a job until machine 1 has completed processing of the same job, namely, each job must be processed by machine 1 and machine 2 in turn.

Flow Shop Scheduling Problem

- Each job i requires a processing time of $t[i,j]$ on machine j.
- Given a scheduling solution, $F[i,j]$ denotes the finish time for job i on machine j.
- The task is to find an optimal scheduling that minimizes the total finish time:

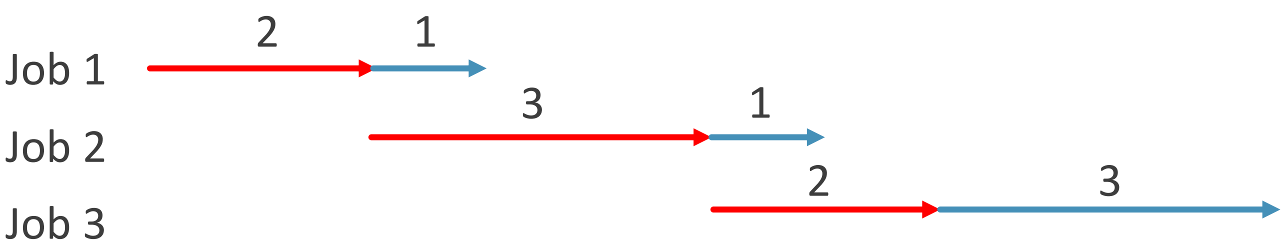
$$f = \sum_{i=1}^{n} F[i,2]$$

Example



$t[i,j]$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

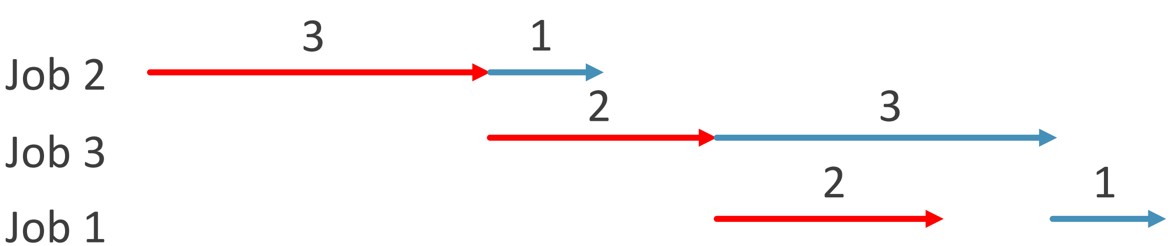
The goal is not to achieve the earliest finish time, but the earliest total finish time.



$F[1,2] = 3$

$F[2,2] = 6$ $\quad f = 19$

$F[2,2] = 10$



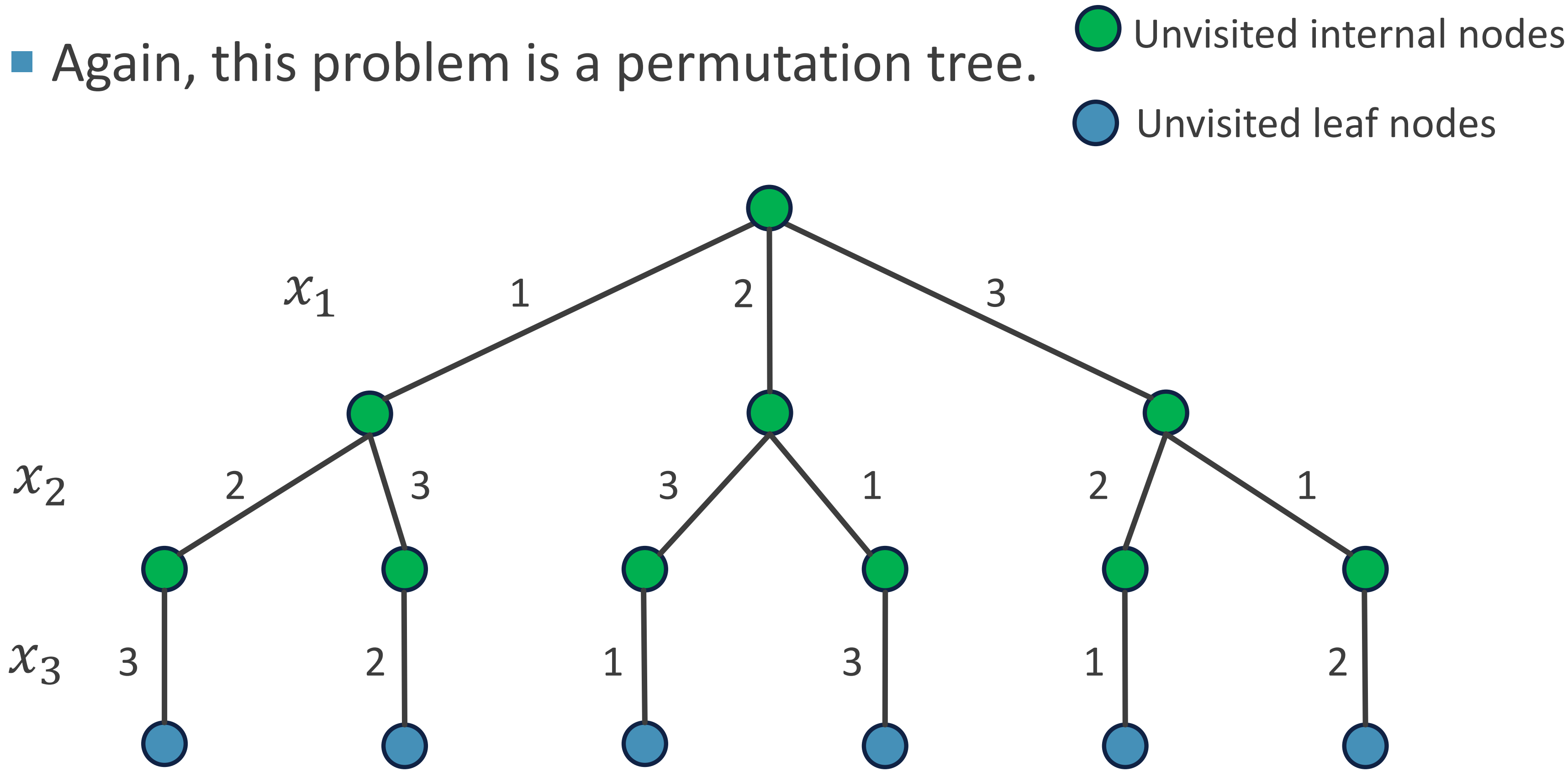
$F[2,2] = 4$

$F[2,2] = 8$ $\quad f = 21$

$F[1,2] = 9$

Flow Shop Scheduling Problem

- Again, this problem is a permutation tree.



Flow Shop Scheduling Problem

- What is the constraint function for this problem?

  There's no constraint function. Any permutation is a feasible solution.

Flow Shop Scheduling Problem

- Let $x = \{x[1], x[2], \ldots, x[i]\}$ be the set of jobs that has been processed up to the extend node i, then

$$f = \sum_{j=1}^{i} F[x[j], 2] + rf(i)$$

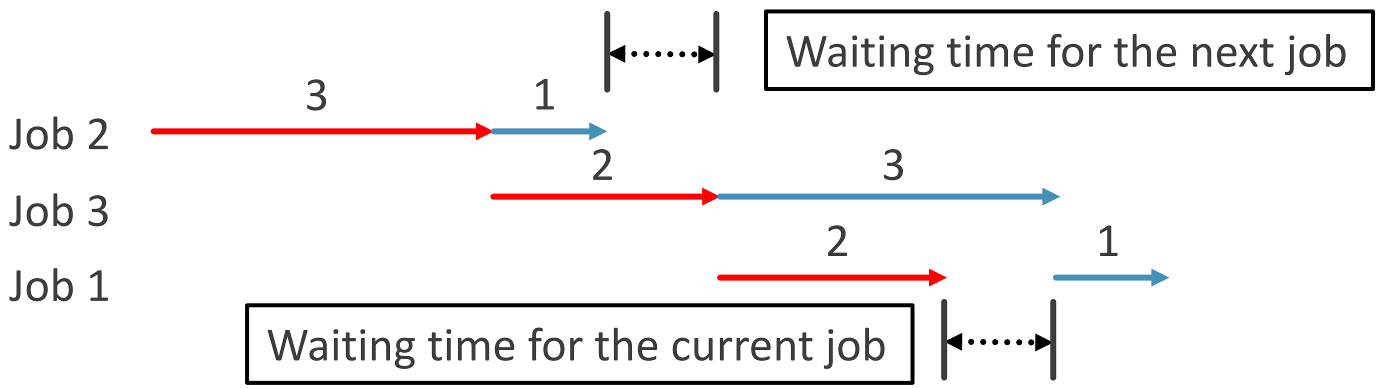
where

$$rf(i) = \sum_{j=i+1}^{n} F[x[j], 2]$$

- Computing $rf(i)$ is very difficult, we can estimate its lower bound?

Lower Bound

- Machine 1 is continuously working.
- The finish time is influenced by the waiting time of machine 2.



We can calculate the lower bound by assuming that there's no waiting time.

Lower Bound

Lower bound 1:

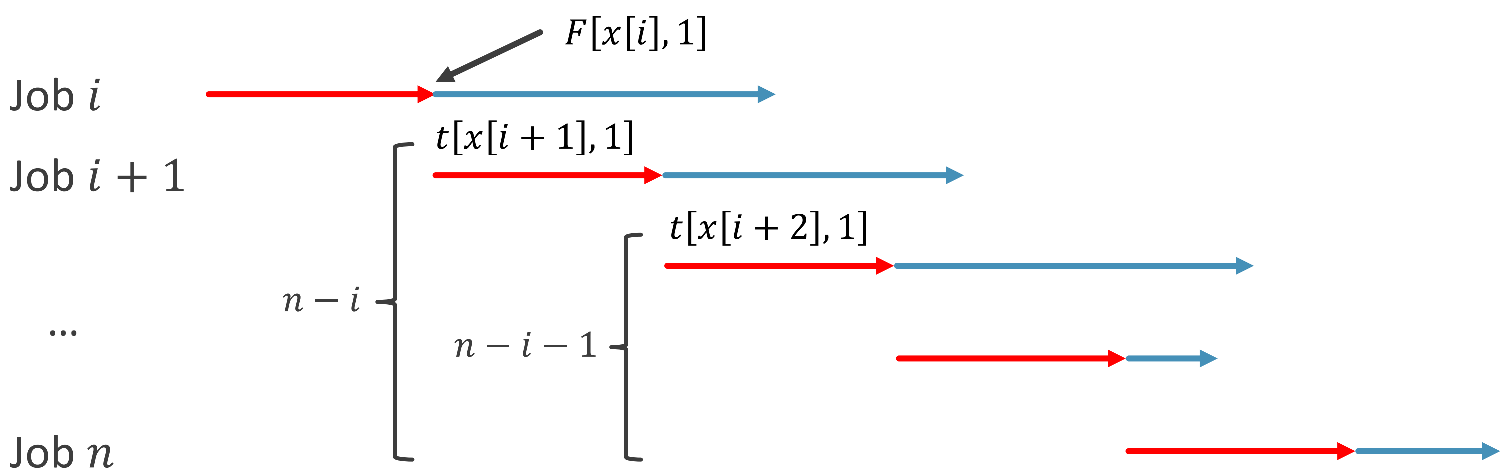
- Assume that machine 2 has no waiting time for the current job.
  - Each remaining job can be continuously processed in the machine 1 and 2 without waiting time.

$$rf1(i) = \sum_{j=i+1}^{n} (F[x[i],1] + (n-j+1)t[x[j],1] + t[x[j],2])$$

- Obviously, we have $rf(i) \geq rf1(i)$.

Lower Bound

Lower bound 1: $rf1(i) = \sum_{j=i+1}^{n} (F[x[i],1] + (n-j+1)t[x[j],1] + t[x[j],2])$



- The order from job $i+1$ to job n matters.
- Therefore, we can sort $t[x[j],1]$ in non-decreasing order to obtain smaller $rf1(i)' \leq rf1(i)$.

Lower Bound

Lower bound 2:

- Assume that machine 2 has no waiting time for the next job.
  - After the machine 2 finished one job, it can process the following job without waiting time.

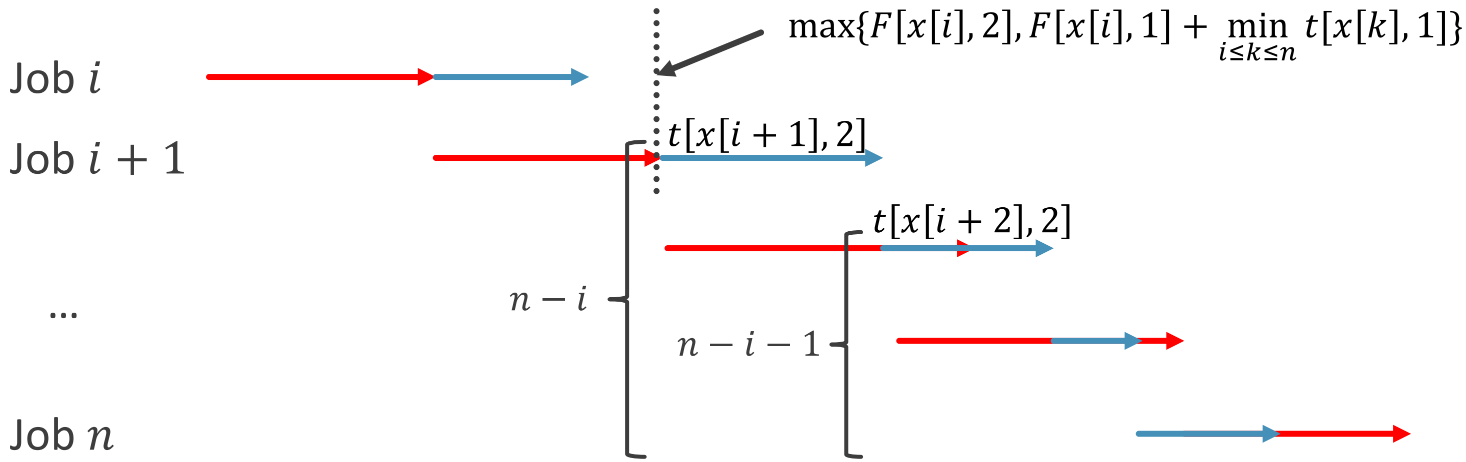
$$rf2(i) = \sum_{j=i+1}^{n} \Big(\max\{F[x[i],2], F[x[i],1] + \min_{i \le k \le n} t[x[k],1]\} + (n-j+1)t[x[j],2] \Big)$$

- Obviously, we have $rf(i) \geq rf2(i)$.

Lower Bound

Lower bound 2:

$$rf2(i) = \sum_{j=i+1}^{n} \Big(\max\{F[x[i],2], F[x[i],1] + \min_{i \le k \le n} t[x[k],1]\} + (n-j+1)t[x[j],2]\Big)$$



- The order from job $i+1$ to job n matters.
- Therefore, we can sort $t[x[j],2]$ in non-decreasing order to obtain smaller $rf2(i)' \le rf2(i)$.

Lower Bound

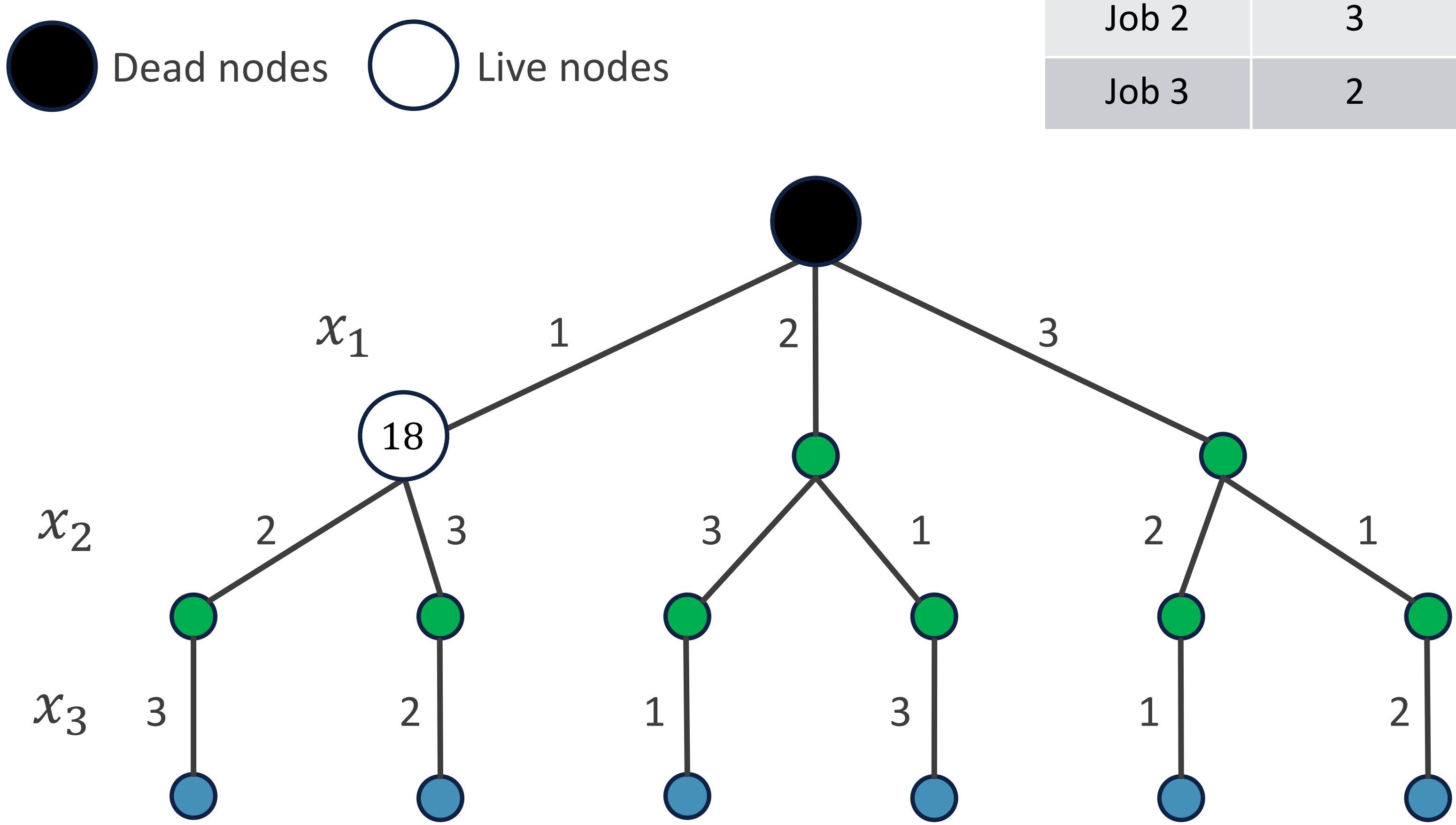
- So, we have

$$\begin{aligned} f &= \sum_{j=1}^{i} F[x[j],2] + rf(i) \\ &\geq \sum_{j=1}^{i} F[x[j],2] + \max\{rf1(i), rf2(i)\} \\ &\geq \sum_{j=1}^{i} F[x[j],2] + \max\{rf1(i)', rf2(i)'\} = B(i) \end{aligned}$$

- If $B(i) \geq bestf$, then stop search the node i and the following level, otherwise, continue to search. At the same time, we use min-priority queue to extend.

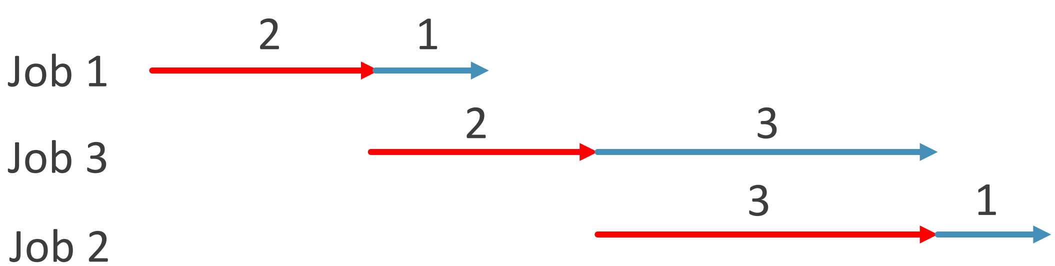
Example

$t[i,j]$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3



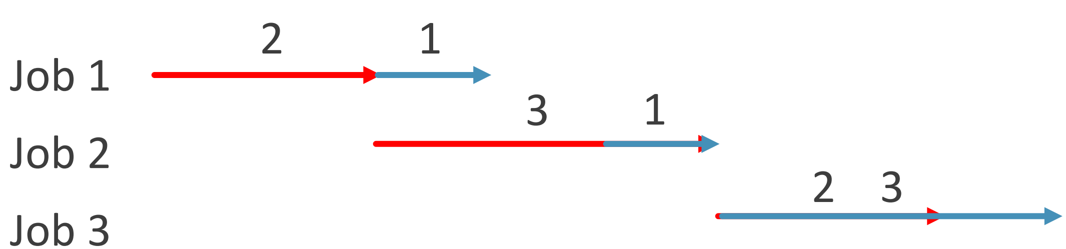
Example

$rf1(1)'$:



$$F[1,2] = 3$$
$$F[3,2] = 7 \qquad f = 18$$
$$F[2,2] = 8$$

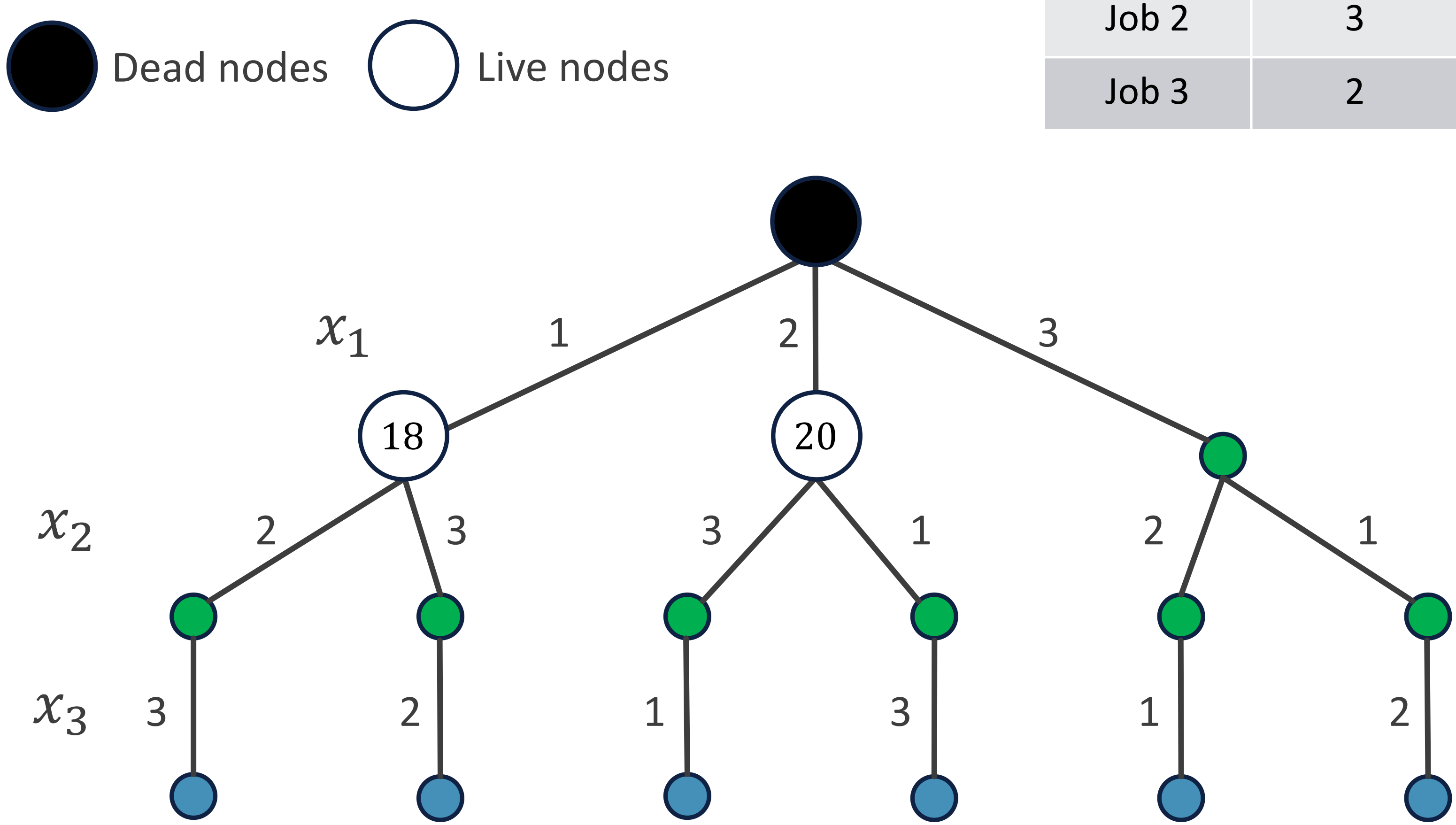
$rf2(1)'$:



$$F[1,2] = 3$$
$$F[2,2] = 5 \qquad f = 16$$
$$F[2,2] = 8$$

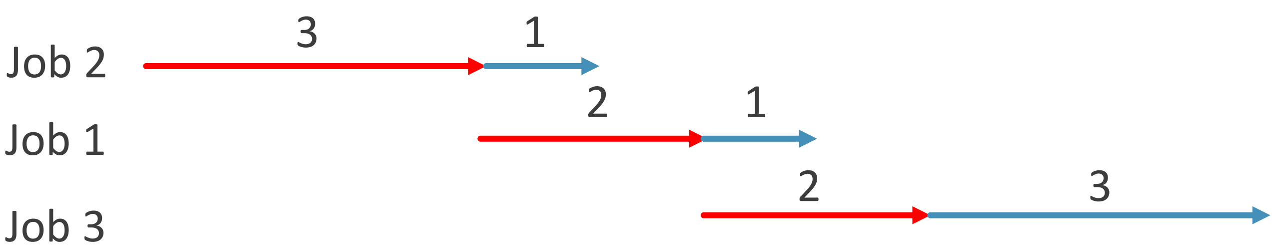
Example

$t[i,j]$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3



Example

$rf1(1)'$:

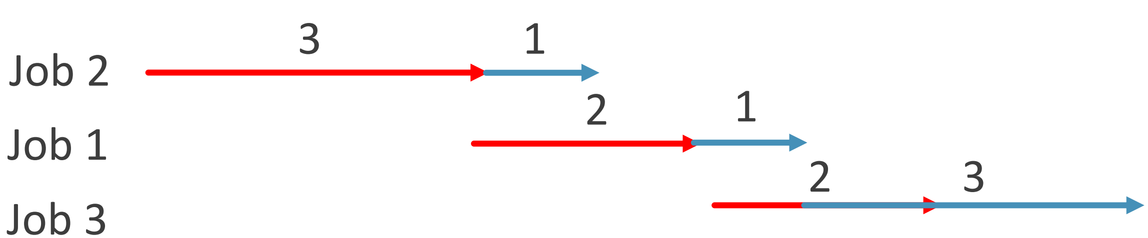


$F[2,2] = 4$

$F[1,2] = 6 \quad f = 20$

$F[3,2] = 10$

$rf2(1)'$:



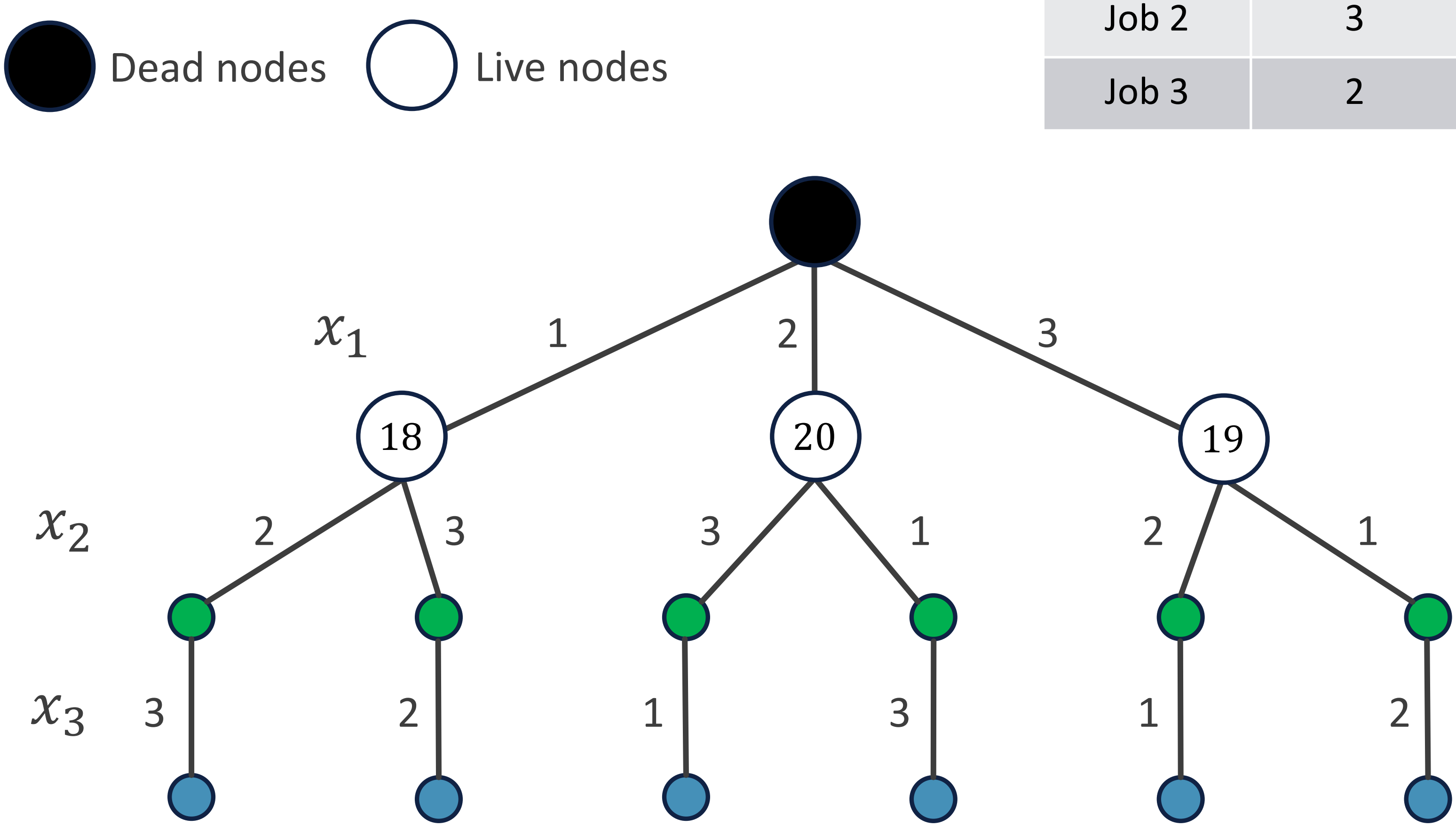
$F[2,2] = 4$

$F[1,2] = 6 \quad f = 19$

$F[2,2] = 9$

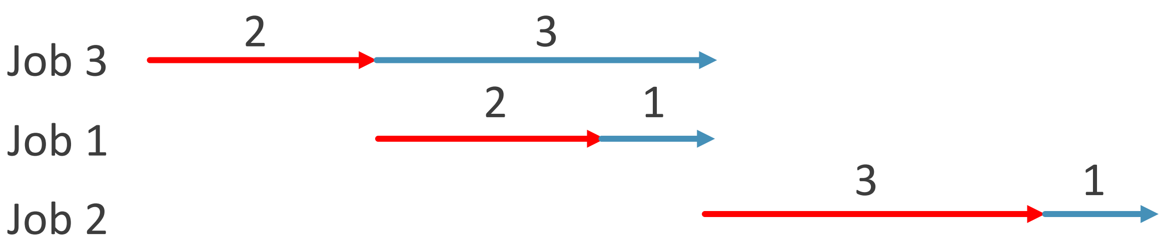
Example

$t[i,j]$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3



Example

$rf1(1)'$:

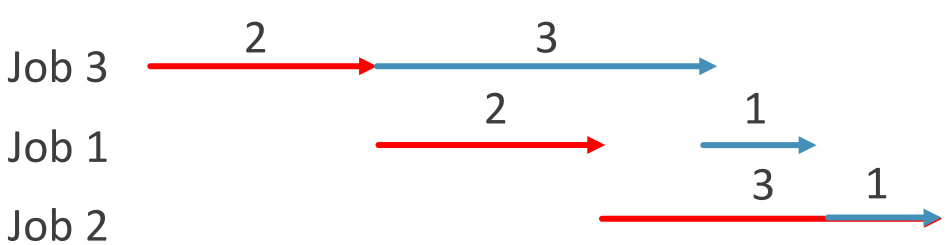


$F[3,2] = 5$

$F[1,2] = 5$ $\quad f = 19$

$F[2,2] = 9$

$rf2(1)'$:



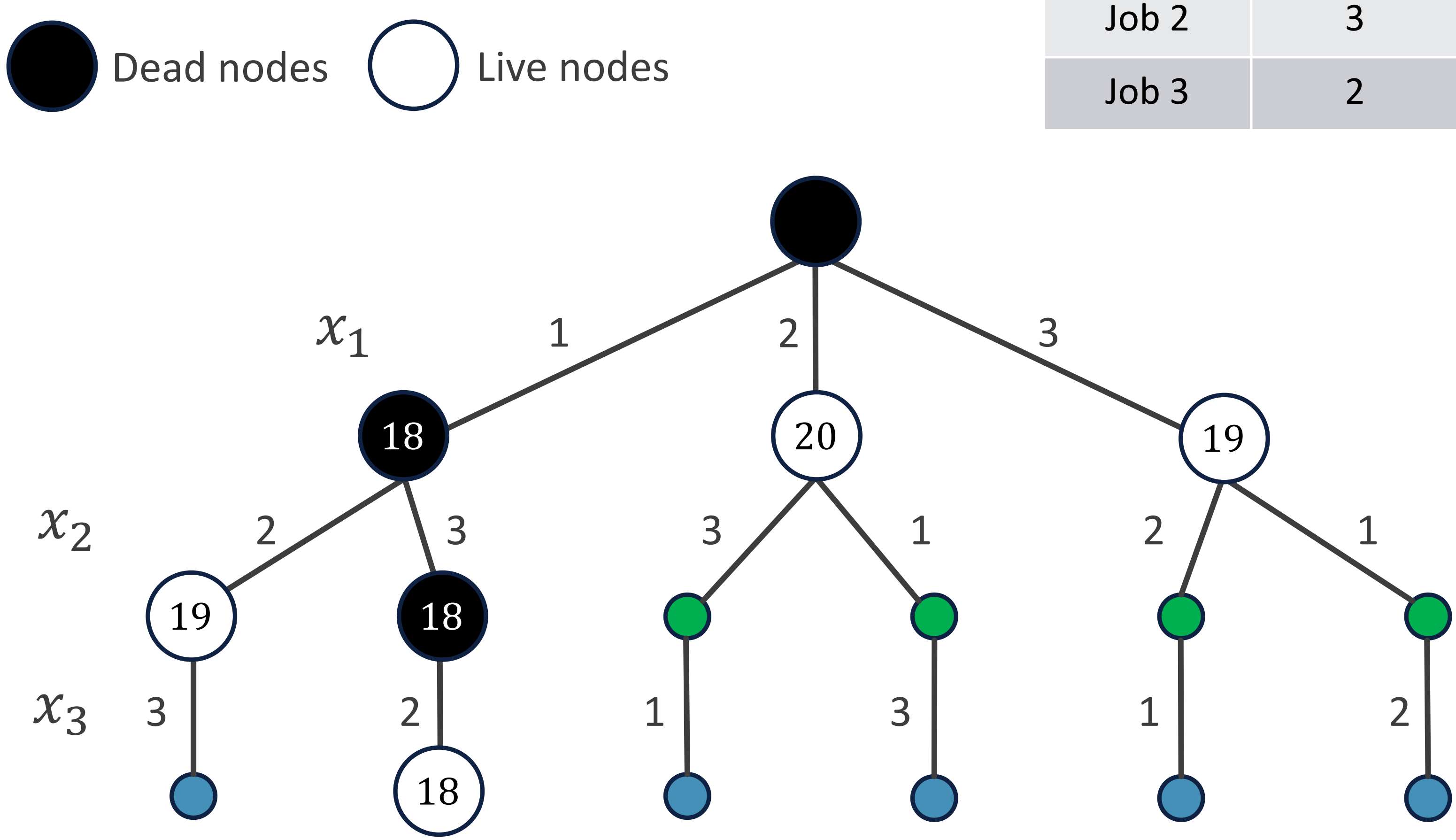
$F[3,2] = 5$

$F[1,2] = 6$ $\quad f = 18$

$F[2,2] = 7$

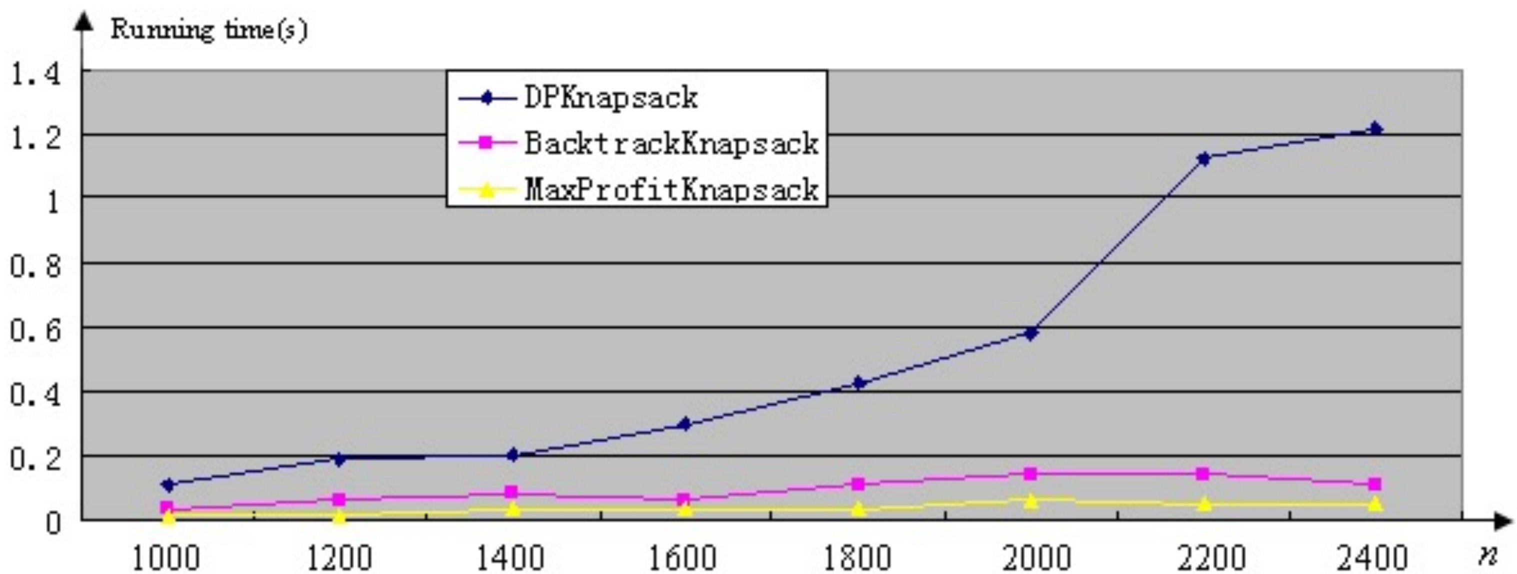
Example

$t[i,j]$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3



Experiments for 0/1 Knapsack

n	1000	1200	1400	1600	1800	2000	2200	2400
DPKnapsack	0.109	0.187	0.203	0.296	0.421	0.578	1.125	1.218
BacktrackKnapsack	0.031	0.063	0.078	0.063	0.11	0.14	0.14	0.109
MaxProfitKnapsack	0.015	0.015	0.031	0.031	0.031	0.062	0.046	0.046
Optimal value	282000	414610	455339	607732	748955	940129	1305502	1312372



Conclusion

After this lecture, you should know:

- What is the difference between backtracking and branch-and-bound.
- What kind of problem that we can use branch-and-bound.
- How can we improve the bounding function to eliminate more branches.

Homework

Page 262-263

13.1

13.2

13.4

Experiment

Choose one:

- P263, 13.11.
- 使用回溯解决石材切割问题.

谢谢

有问题欢迎随时跟我讨论