Ch22.Branch and Bound

SNU Internet DataBase Lab.

BIRD'S-EYE VIEW

- A surefire way to solve a problem is to make a list of all candidate answers and check them
  - If the problem size is big, we can not get the answer in reasonable time using this approach
  - List all possible cases? ➔ exponential cases
- By a systematic examination of the candidate list, we can find the answer without examining every candidate answer
  - *Backtracking* and *Branch and Bound* are most popular systematic algorithms
- Branch and Bound
  - Searches a solution space that is often organized as a tree (like backtracking)
  - Usually searches a tree in a breadth-first / least-cost manner (unlike backtracking)

SNU Internet DataBase Lab.

Table of Contents

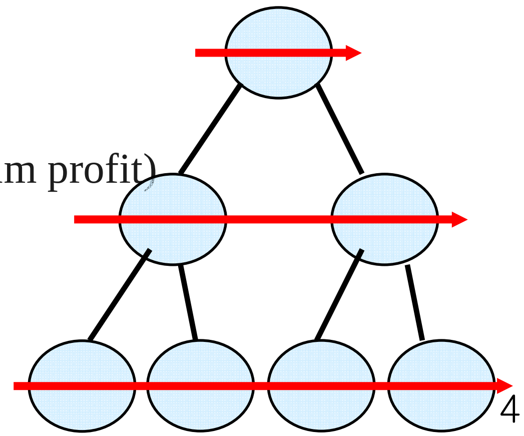
- The Branch and Bound Method
- Application
  - Rat in a Maze
  - Container Loading

SNU Internet DataBase Lab.

Branch and Bound

- Another way to systematically search a solution space
- Usually searches trees in either a breadth-first or least-cost manner
  - But not exactly breadth-first search
- Each live node becomes an E-node exactly once

- Selection options of the next E-node
  - First In, First Out (FIFO)
    - The live node list - queue
    - Extracts nodes in the same order as they are put into it
  - Least Cost (or Max Profit)
    - The live node list - min heap (or max heap)
    - The next E-node – the live node with least cost (or maximum profit)



Backtracking vs. Branch and Bound

Backtracking		**Branch and Bound**
Depth-first	Search order	Breadth-first or Least cost
More	Execution time	Less*
Less: stack O(length of longest path)	Space requirement	More: queue O(size of solution space)

•It might be expected to examine fewer nodes on many inputs in a max-profit or least-cost branch and bound

• Backtracking may never find a solution if tree depth is infinite
• FIFO branch and bound finds solution closest to root
• Least-cost branch and bound directs the search to parts of the space most likely to contain the answer → So it could perform generally better than backtracking

Table of Contents

- The Branch and Bound Method
- Application
  - Rat in a Maze
  - Container Loading

Rat in a Maze

- 3 x 3 rat-in-a-maze instance

entrance	0	0	0	
	0	1	1	
	0	0	0	exit

0 : road

1 : obstacle

- A maze is a tour puzzle in the form of a complex branching passage through which the solver must find a route
  - Path of a maze is a graph
  - So, we can traverse a maze using DFS / BFS
- Branch and Bound = Finding solution using BFS
- Worst-case time complexity of finding path to the exit of n*n maze is $O(n^2)$

SNU Internet DataBase Lab.

Branch and Bound in "Rat in a Maze"

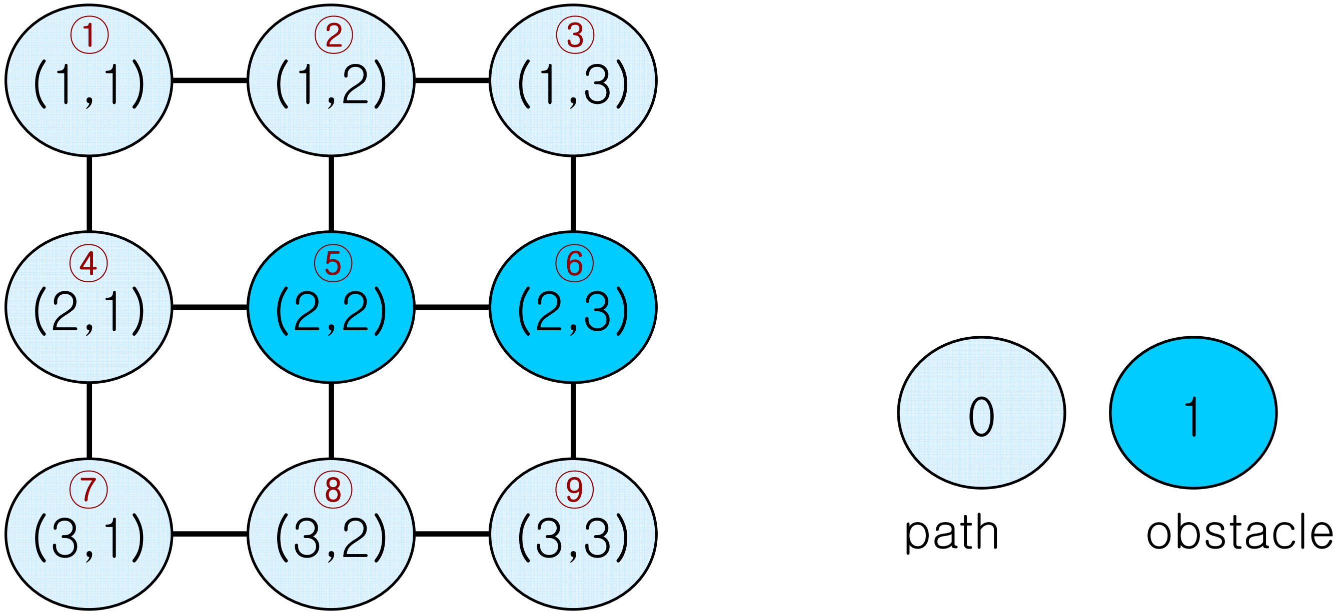
1. Prepare an empty queue and an empty 2D array
2. Initialize array elements with 1 where obstacles are, 0 elsewhere
3. Start at the upper left corner and push the position to the queue
4. Pop a position from the queue and set current position to it
5. Set the array value of current position to 1
6. Check adjacent (up, right, down and left) cells whose value is zero and push them into the queue
7. If we found such cells, push their positions into the queue
8. If we haven't reach to the goal, repeat from 4

Code for Rat in a Maze

```
Prepare an empty queue and an empty 2D array
Initialize array elements with 1 where obstacles are, 0 elsewhere
i ← 1
j ← 1
Repeat until reach to the goal {
    a[i][j] ← 1;
    if (a[i][j+1]==0) {   put (i,j) into the queue
                          j++;  }
    if (a[i+1][j]==0) {   put (i,j) into the queue
                          i++; }
    if (a[i][j-1]==0) {   put (i,j) into the queue
                          j--; }
    if (a[i-1][j]==0) {   put (i,j) into the queue
                          i--; }
    pop (i,j) from the queue
}
```

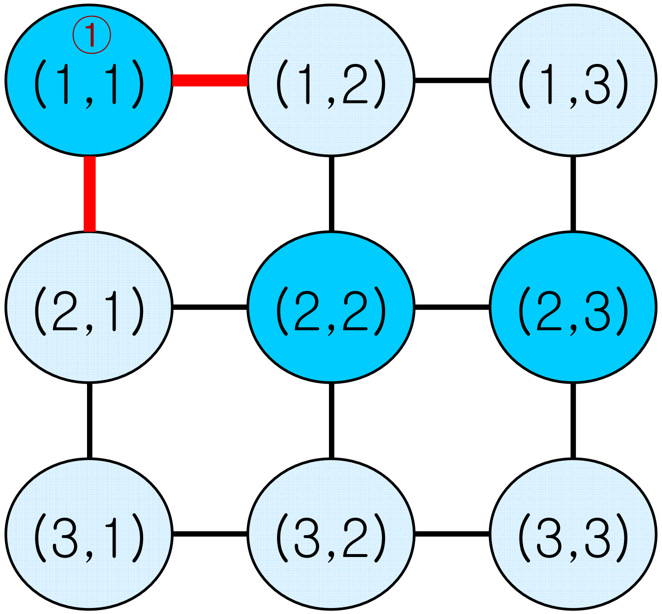
Rat in a Maze Example (1)

- Organize the solution space



Rat in a Maze Example (2)

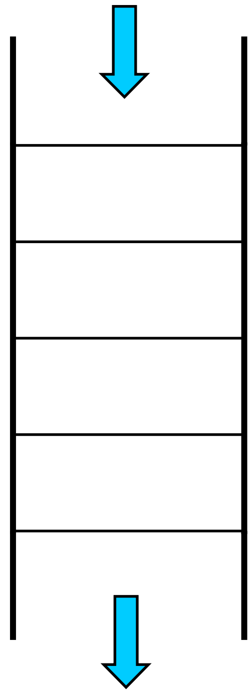
- FIFO Branch and Bound



E-node

(1,1)

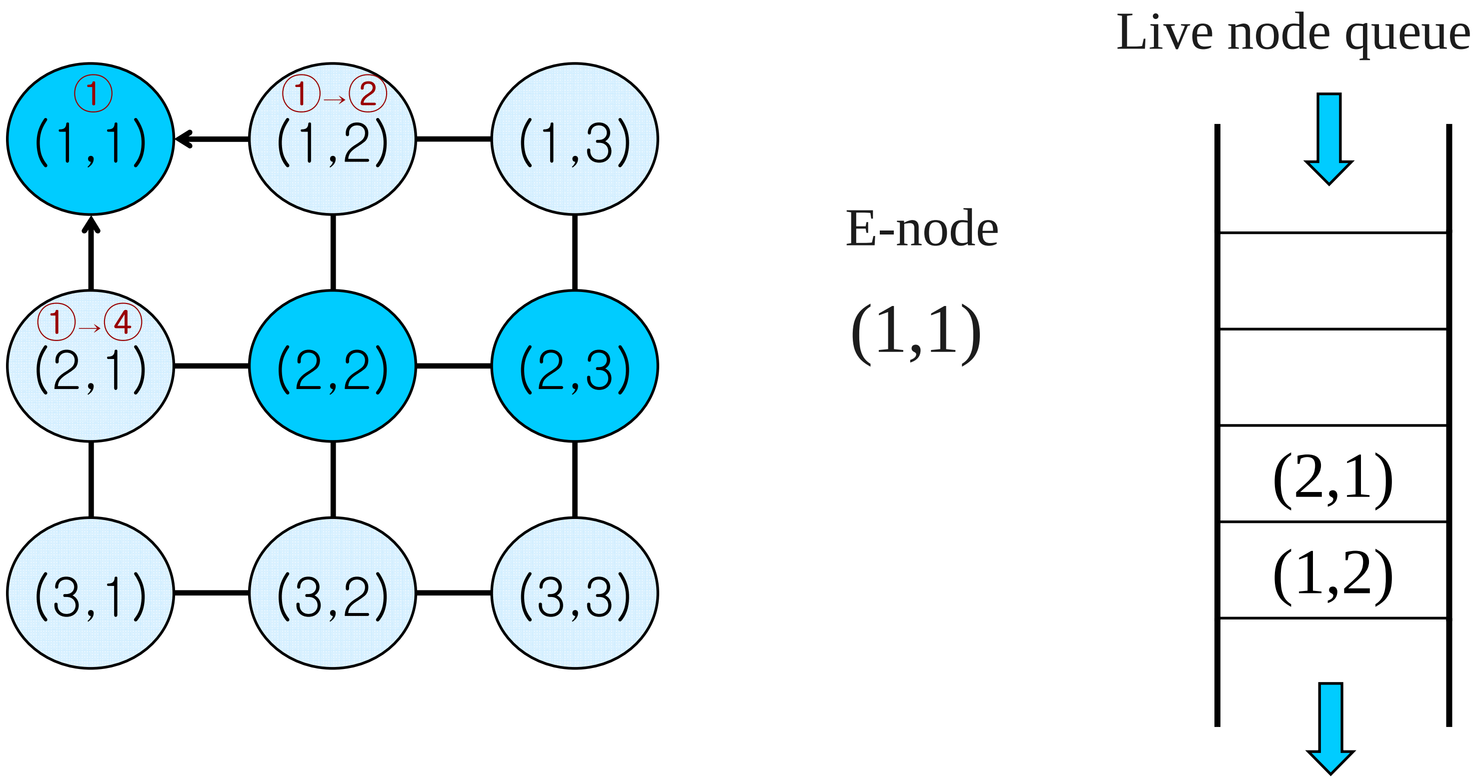
Live node queue



Push (1,2) and (2,1) // Branch

Rat in a Maze Example (3)

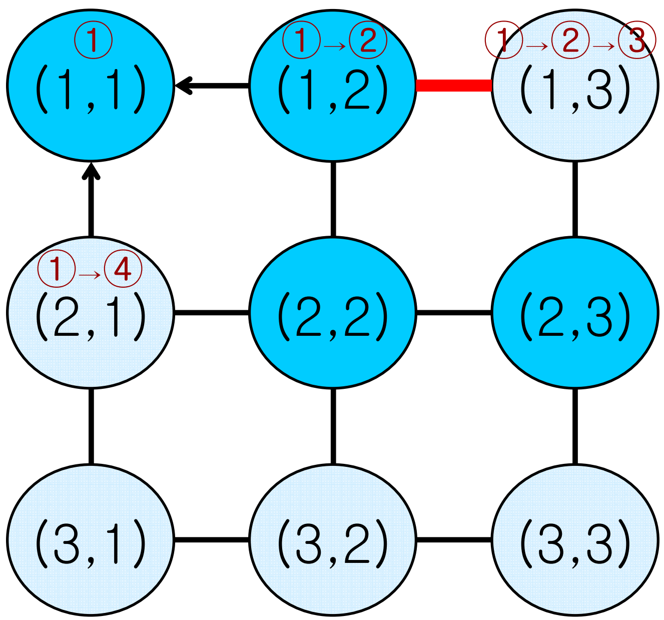
- FIFO Branch and Bound



Pop (1,2) and Move (Bound) to (1,2)

Rat in a Maze Example (4)

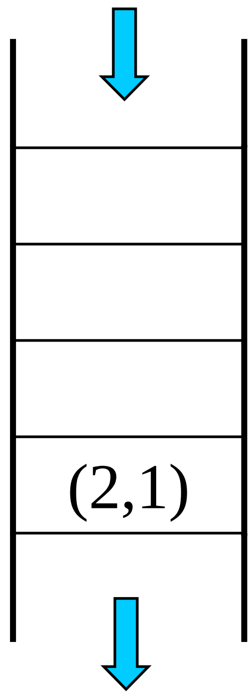
- FIFO Branch and Bound



E-node

(1,2)

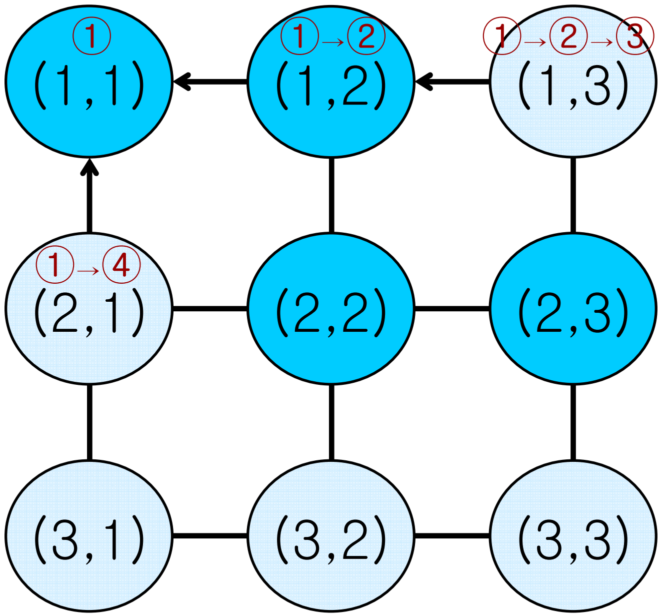
Live node queue



Push (1,3) // Branch

Rat in a Maze Example (5)

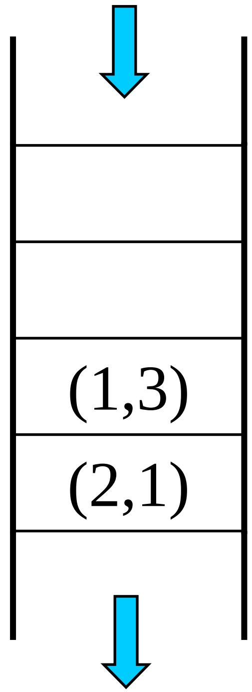
- FIFO Branch and Bound



E-node

(1,2)

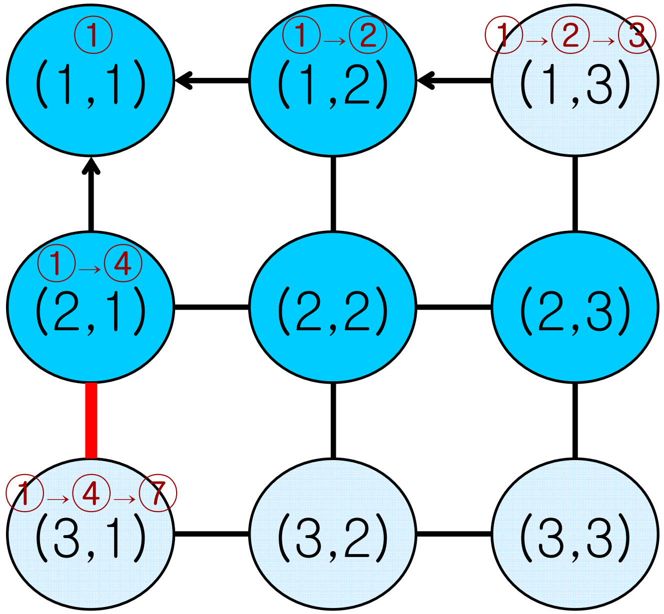
Live node queue



Pop (2,1) and Move (Bound) to (2,1)

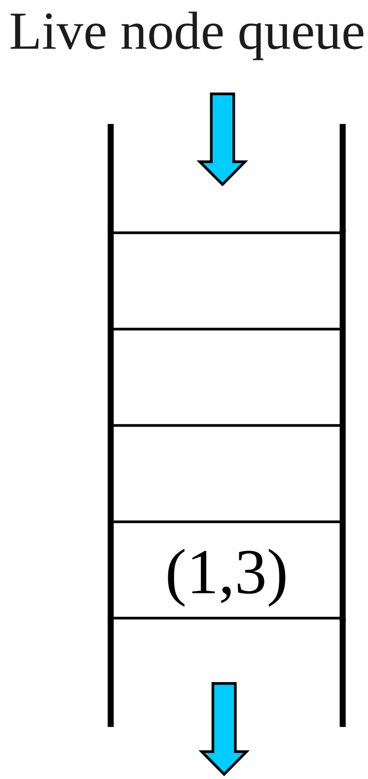
Rat in a Maze Example (6)

- FIFO Branch and Bound



E-node

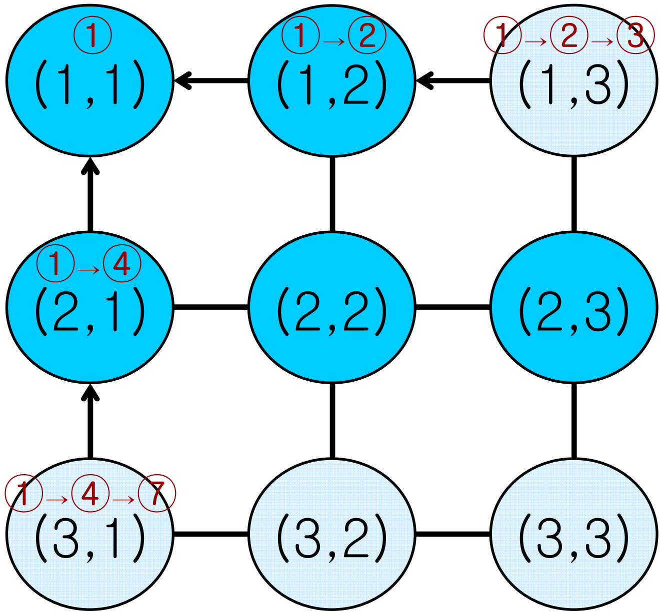
(2,1)

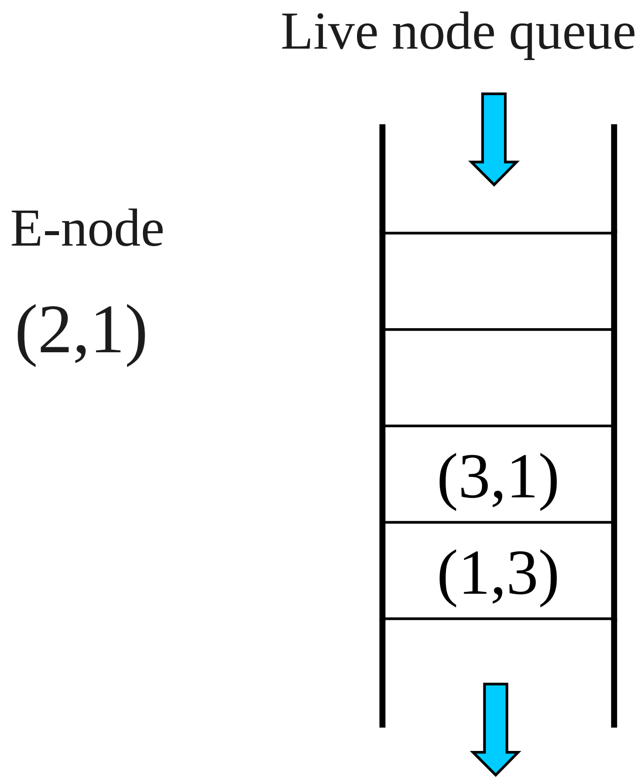


Push (3,1) // Branch

Rat in a Maze Example (7)

- FIFO Branch and Bound

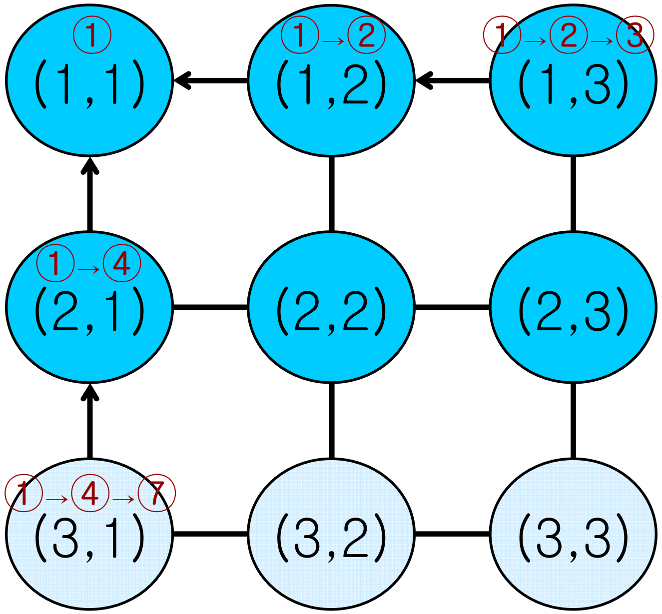




Pop (1,3) & Move (Bound) to (1,3); no more progress

Rat in a Maze Example (8)

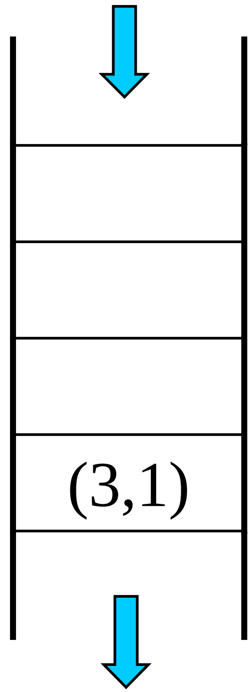
- FIFO Branch and Bound



E-node

(1,3)

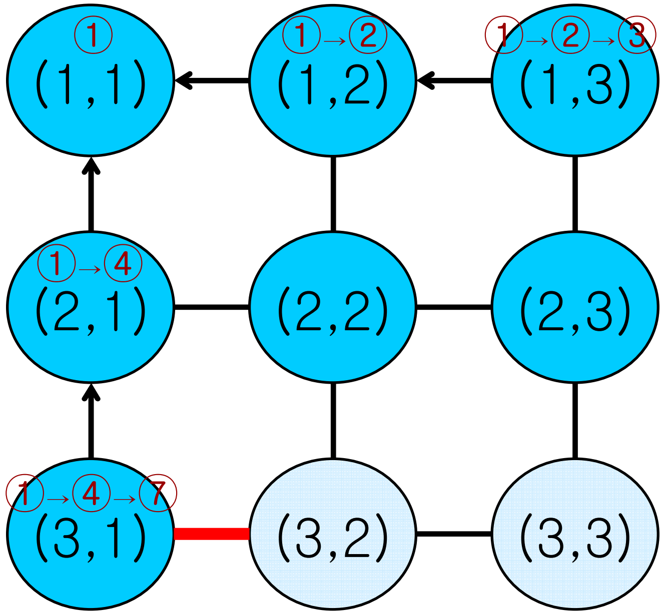
Live node queue



Pop (3,1) and Move (Bound) to (3,1)

Rat in a Maze Example (9)

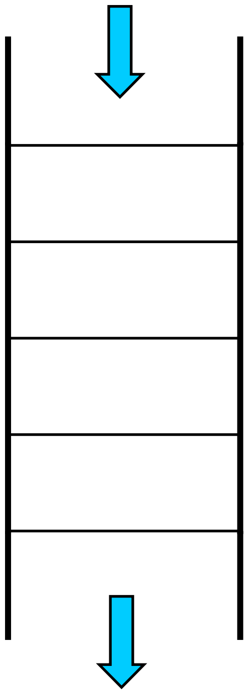
- FIFO Branch and Bound



E-node

(3,1)

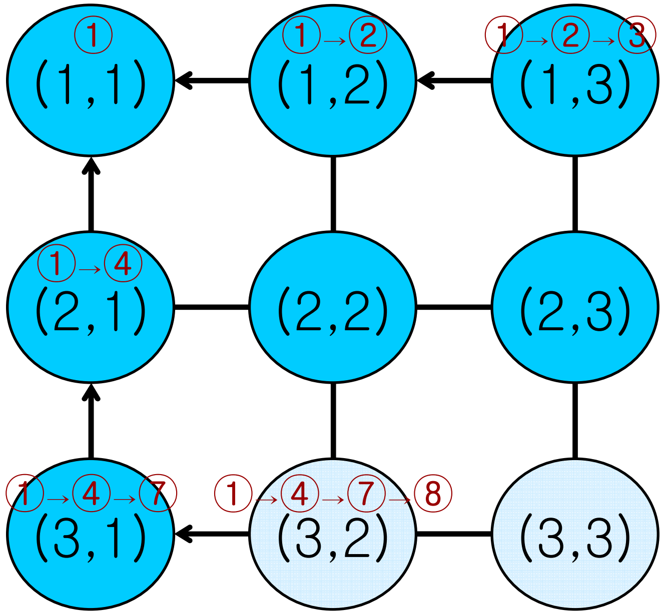
Live node queue



Push (3,2) // Branch

Rat in a Maze Example (10)

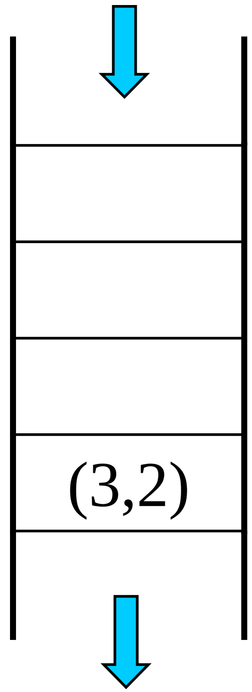
- FIFO Branch and Bound



E-node

(3,1)

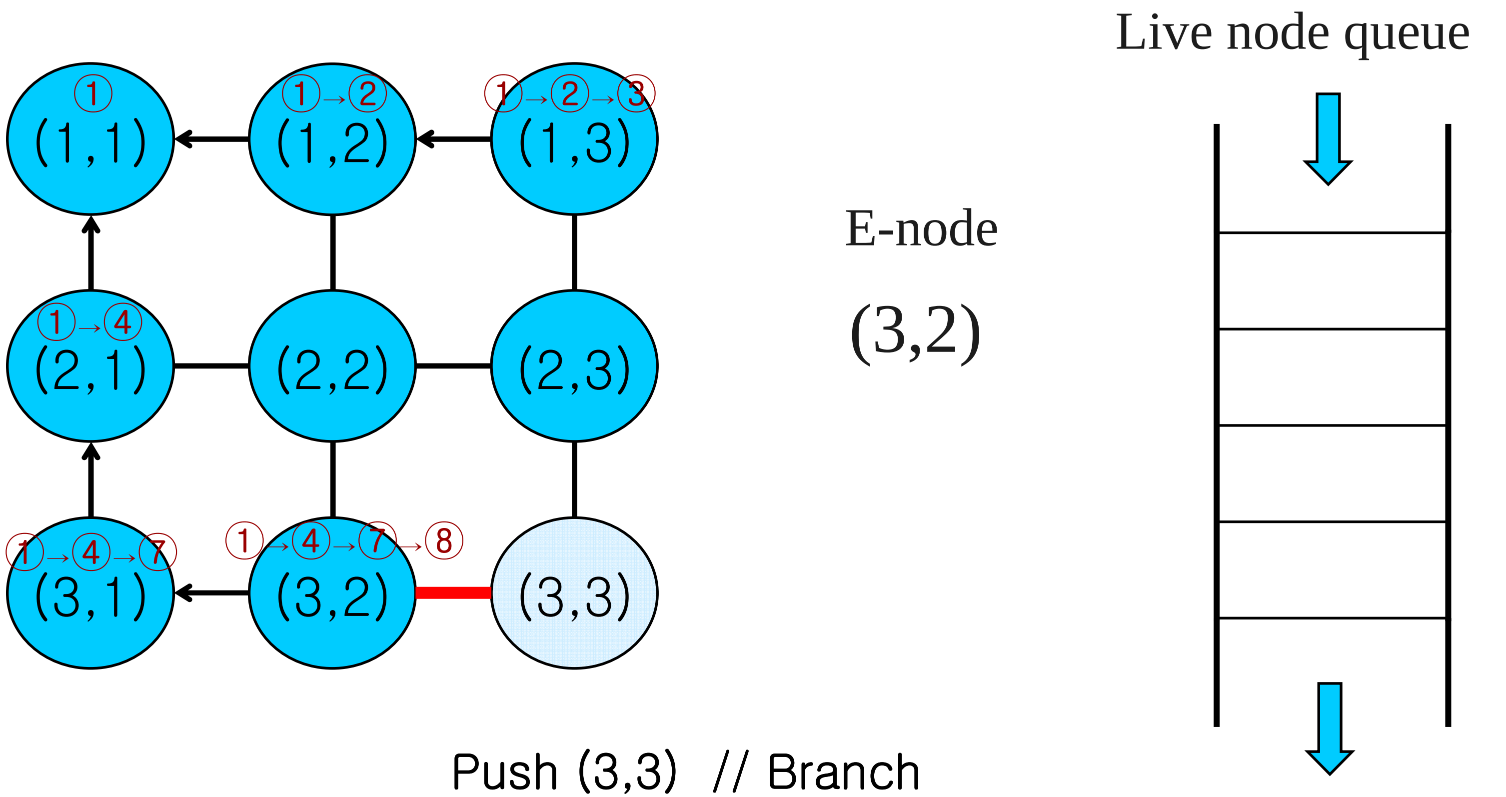
Live node queue



Pop (3,2) and Move (Bound) to (3,2)

Rat in a Maze Example (11)

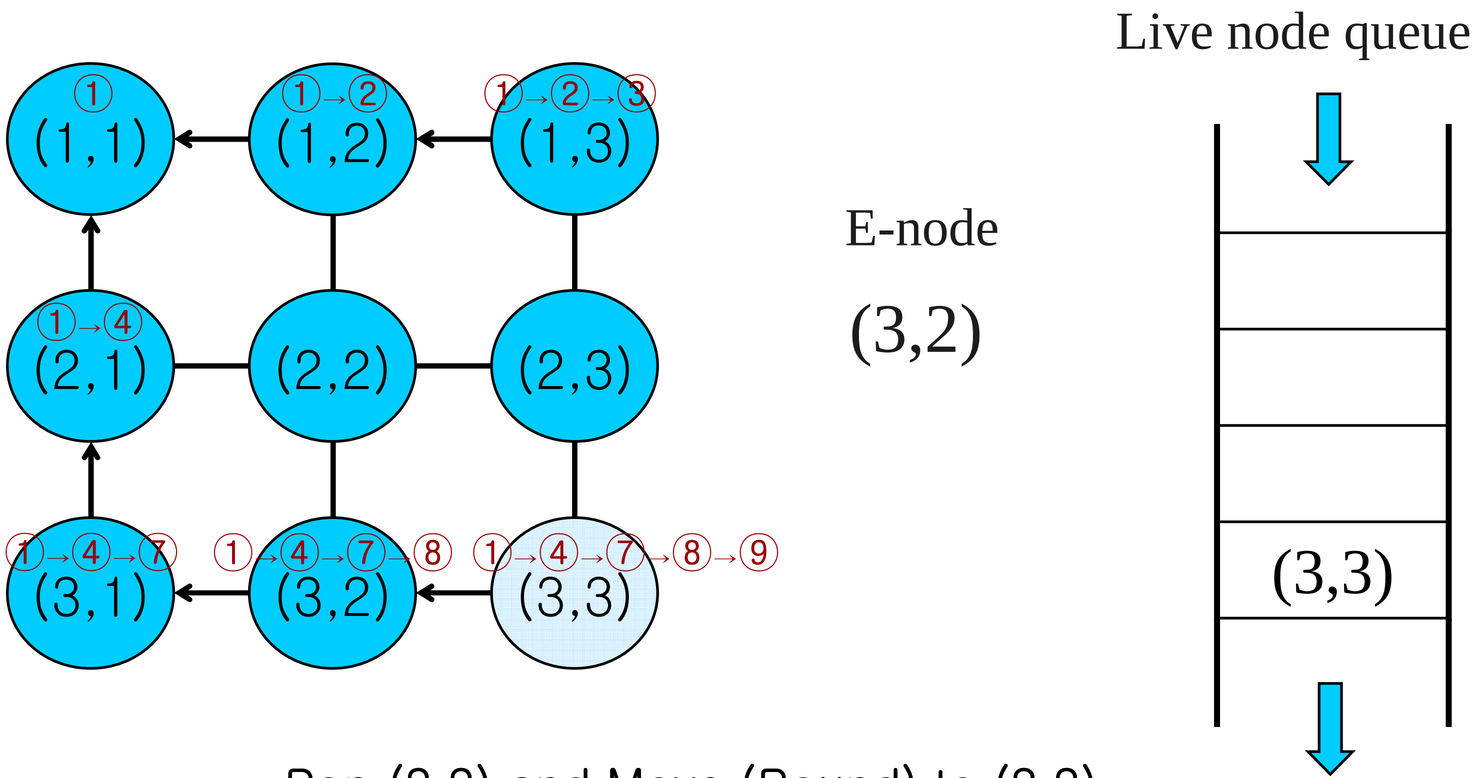
- FIFO Branch and Bound



Push (3,3) // Branch

Rat in a Maze Example (12)

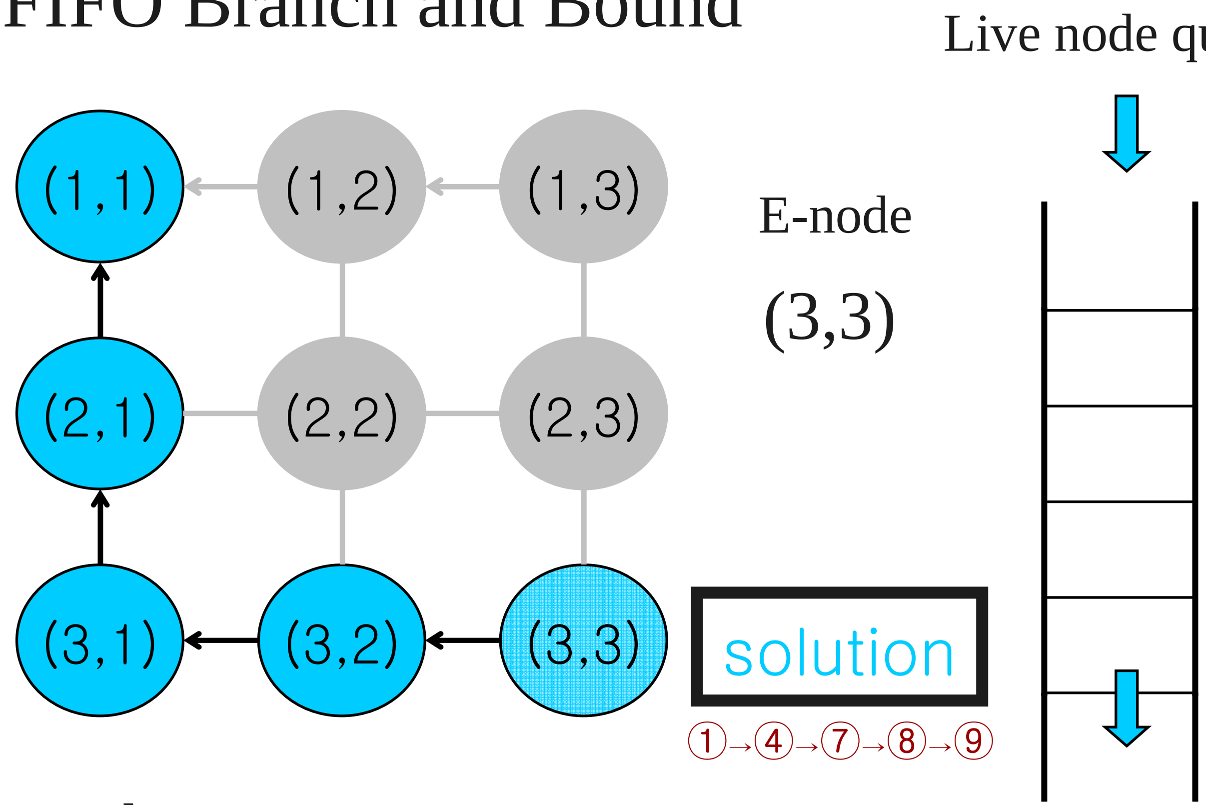
- FIFO Branch and Bound



Pop (3,3) and Move (Bound) to (3,3)

Rat in a Maze Example (13)

- FIFO Branch and Bound



- Observation
  - FIFO search solution is a shortest path from the entrance to the exit
  - Remember that backtracking solution may not be a shortest path

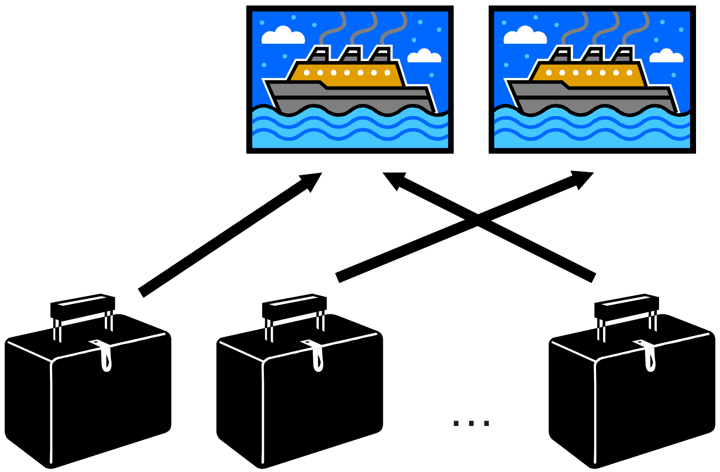
Table of Contents

- The Branch and Bound Method
- Application
  - Rat in a Maze
  - Container Loading

Container Loading

- Container Loading Problem
  - 2 ships and *n* containers
  - The ship capacity: c_1, c_2
  - The weight of container *i*: w_i

$$\sum_{i=1}^{n} w_i \le c_1 + c_2$$



  - Is there a way to load all *n* containers?

- Container Loading Instance
  - n = 4
  - $c_1 = 12$, $c_2 = 9$
  - w = [8, 6, 2, 3]
- Find a subset of the weights with sum as close to c_1 as possible

Solving without Branch and Bound

- We can find a solution with brute-force search

1. Generate n random numbers $x_1, x_2, \ldots, x_n$
   where x_i = 0 or 1 ($i = 1,\ldots,n$)
2. If $x_i = 1$, we put i-th container into ship 1
   If $x_i = 0$, we put i-th container into ship 2
3. Check if sum of weights in both ships are less than their maximum capacity

3-1. If so, we found a solution!

3-2. Otherwise, repeat from 1

- Above method are too naïve and not duplicate-free

➔ Branch and bound provides a systematic way to search feasible solutions (still NP-complete, though)

SNU Internet DataBase Lab.

Container Loading and Branch & Bound

- Container loading is one of NP-complete problems
  - There are 2^n possible partitionings
- If we represent the decision of location of each container with a *branch*, we can represent container loading problem with a *tree*
  - So, we can traverse the tree using DFS / BFS
- Branch and bound = Finding solution using BFS
- Worst-case time complexity is $O(2^n)$ if there are n containers

- FIFO branch and bound finds solution closest to root
  - Rat in Maze
- Least-cost branch and bound directs the search to parts of the space most likely to contain the answer
  - Container Loading

SNU Internet DataBase Lab.

Considering only One Ship

- Original problem: Is there any way to load n containers with

$$\sum_{i \text{ belongs to } ship_1} w_i \le c_1, \qquad \sum_{i \text{ belongs to } ship_2} w_i \le c_2$$

- Because $\sum_{i \text{ belongs to } ship_1} w_i + \sum_{i \text{ belongs to } ship_2} w_i = \sum_{i=1}^{n} w_i$ is constant,

$$\max\left(\sum_{i \text{ belongs to } ship_1} w_i\right) = \min\left(\sum_{i \text{ belongs to } ship_2} w_i\right)$$

- So, all we need to do is trying to load containers at ship 1 as much as possible and check if the sum of weights of remaining containers is less than or equal to c_2

Branch and Bound in Container Loading

1. Prepare an empty queue Q & a complete binary tree T with depth *n*
2. Initialize the *max* to zero
3. Start from root of T and put the root node into the queue
4. Pop a node from the queue and set *t* to it
5. If we haven't visit left child and have space to load $w_{depth(t)}$,
   then load it, push *t* into Q and move to left child
6. If we haven't visit right child, push t into Q and move to right child
7. If current sum of weights is greater than *max*, update *max*
8. Repeat from 4 until we have checked all nodes

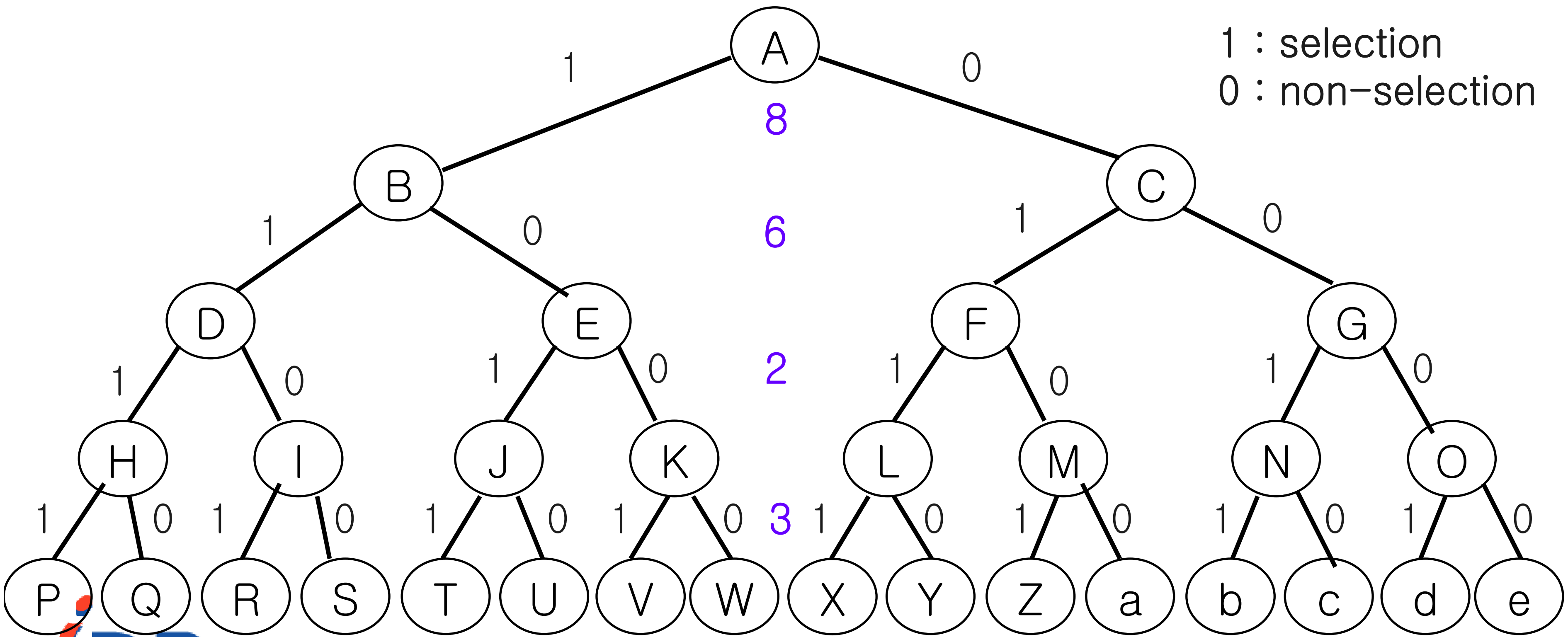
SNU Internet DataBase Lab.

Container Loading Code

```
Consider n, c1, c2, w are given
Construct a complete binary tree with depth n
Prepare an empty max-priority queue
max ← 0
sum ← 0
x ← root node of the tree
while(true){
    if (x.depth < n && !x.visitedLeft && c1 – sum ≥ w[x.depth]) {
        sum ← sum + w[x.depth]
        if (sum > max) max = sum;
        Put (x,sum) into the queue
        x.visitedLeft ← true;
        x ← x.leftChild;
    }
    if (x.depth < n && !x.visitedRight) {
        Put (x,sum) into the queue
        x.visitedRight ← true;
        x ← x.rightChild;
    }
    if (the queue is empty) {
        If sum(w) – max <= c2, max is the optimal weight
        Otherwise, it is impossible to load all containers
        Quit the program }
    Pop (x,sum) from the queue
}
```

Container Loading Example (1)

- Organize the solution space: $n = 4; c_1 = 12, c_2 = 9, w = [8, 6, 2, 3]$
- Max Profit Branch Bound ➔ Priority Queue

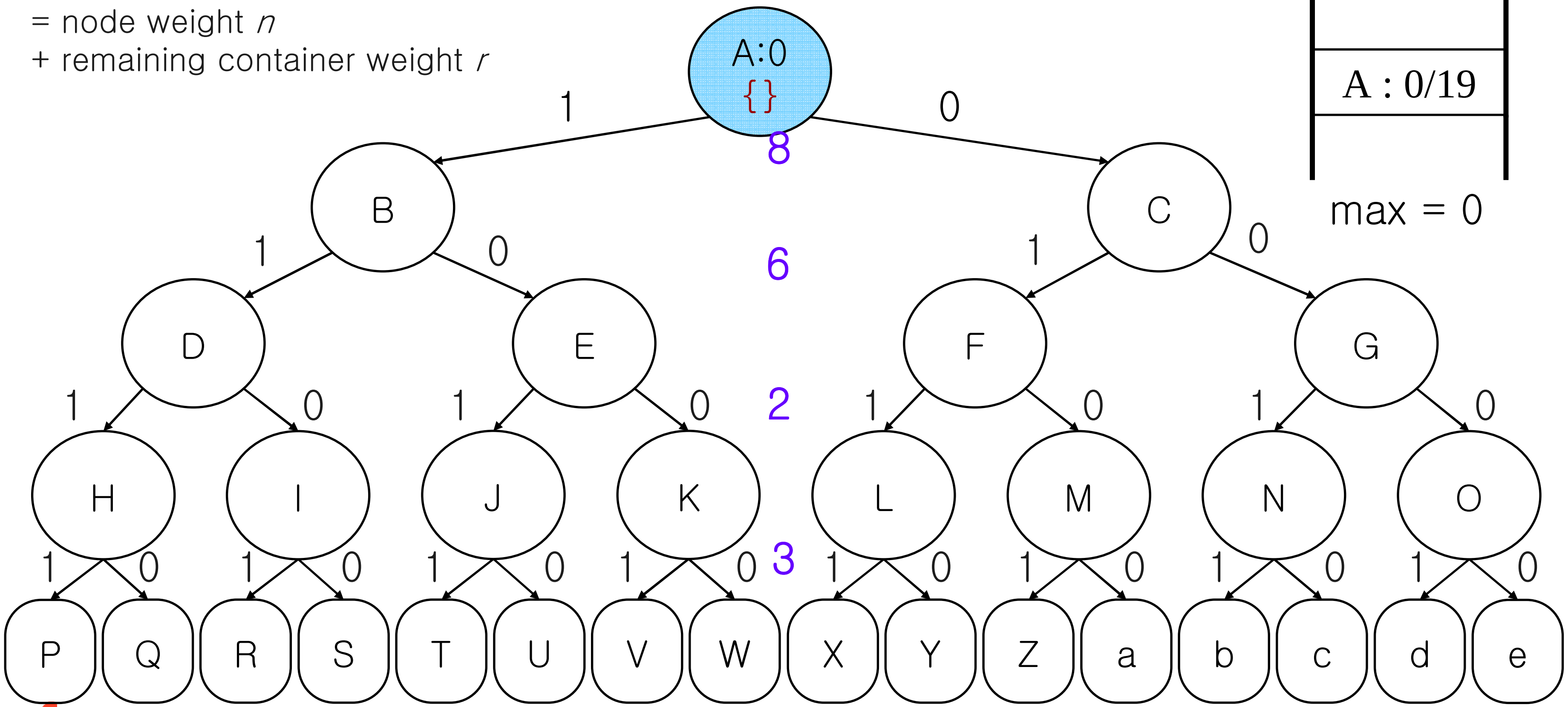


Push A to Live Node Queue

Container Loading Example (2)

- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight n
+ remaining container weight r



Live node queue
(max-priority)

n / r
A : 0/19

max = 0

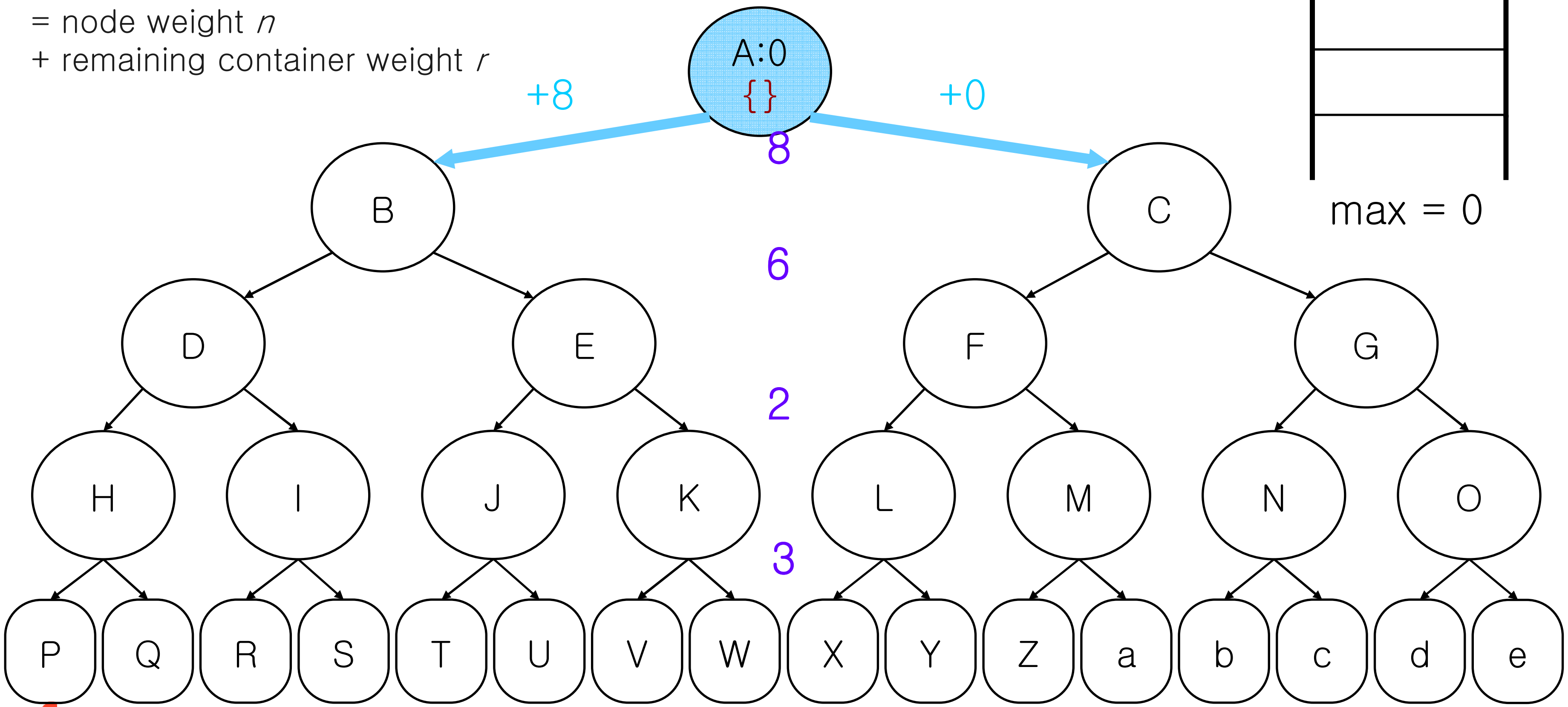
Pop A and Move to A

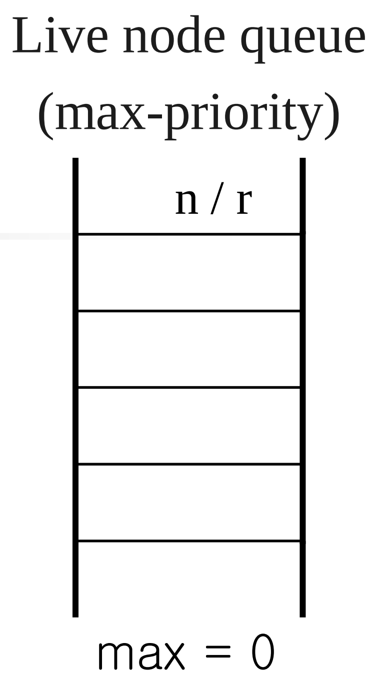
SNU Internet DataBase Lab.

Container Loading Example (3)

- Max Profit Branch Bound: $n = 4; c_1 = 12, c_2 = 9, w = [8, 6, 2, 3]$

priority = upper weight
= node weight n
+ remaining container weight r





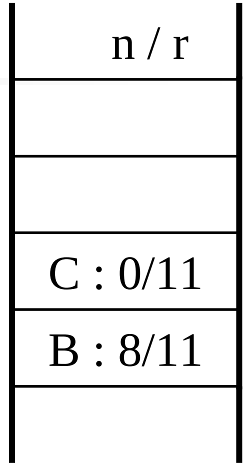
Push B and C // Branch

Container Loading Example (4)

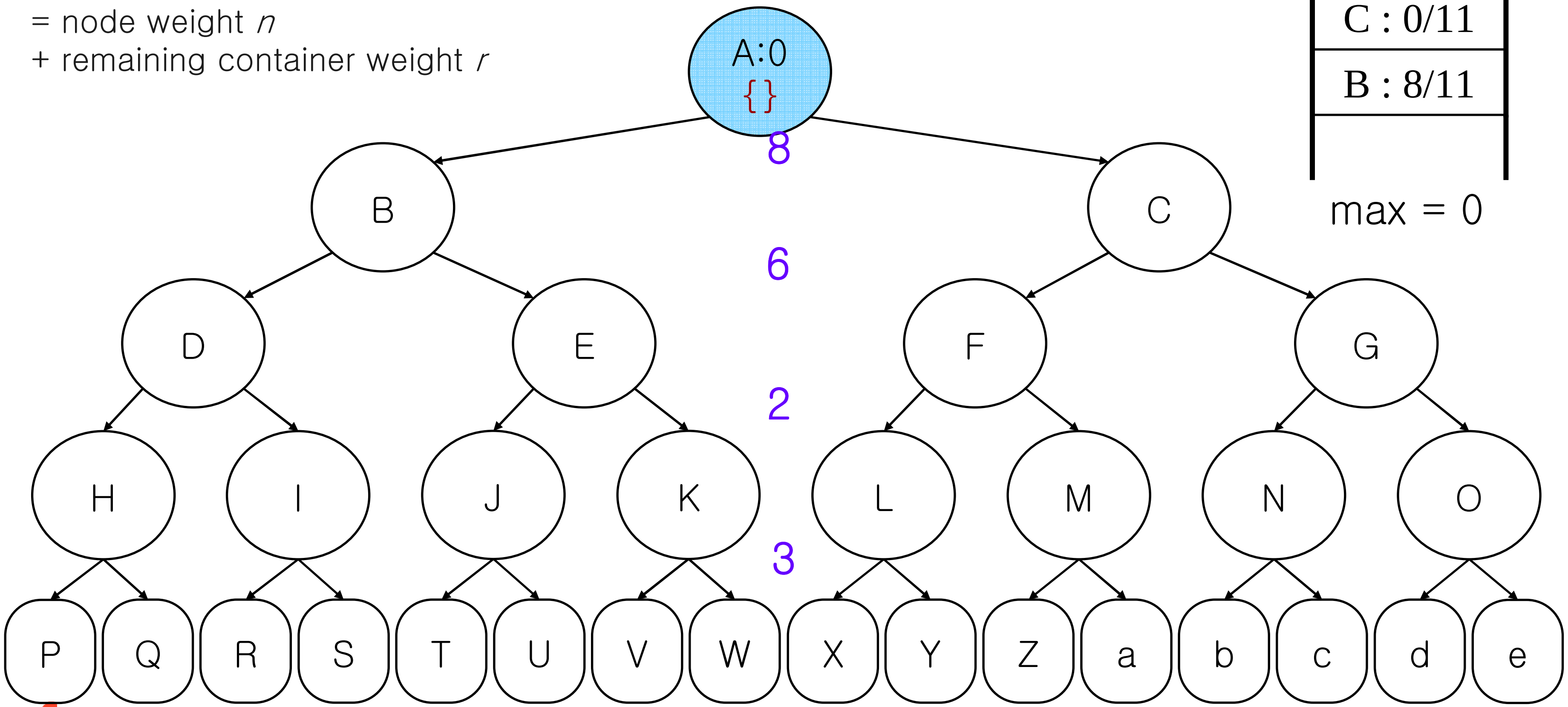
- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*

Live node queue
(max-priority)



max = 0

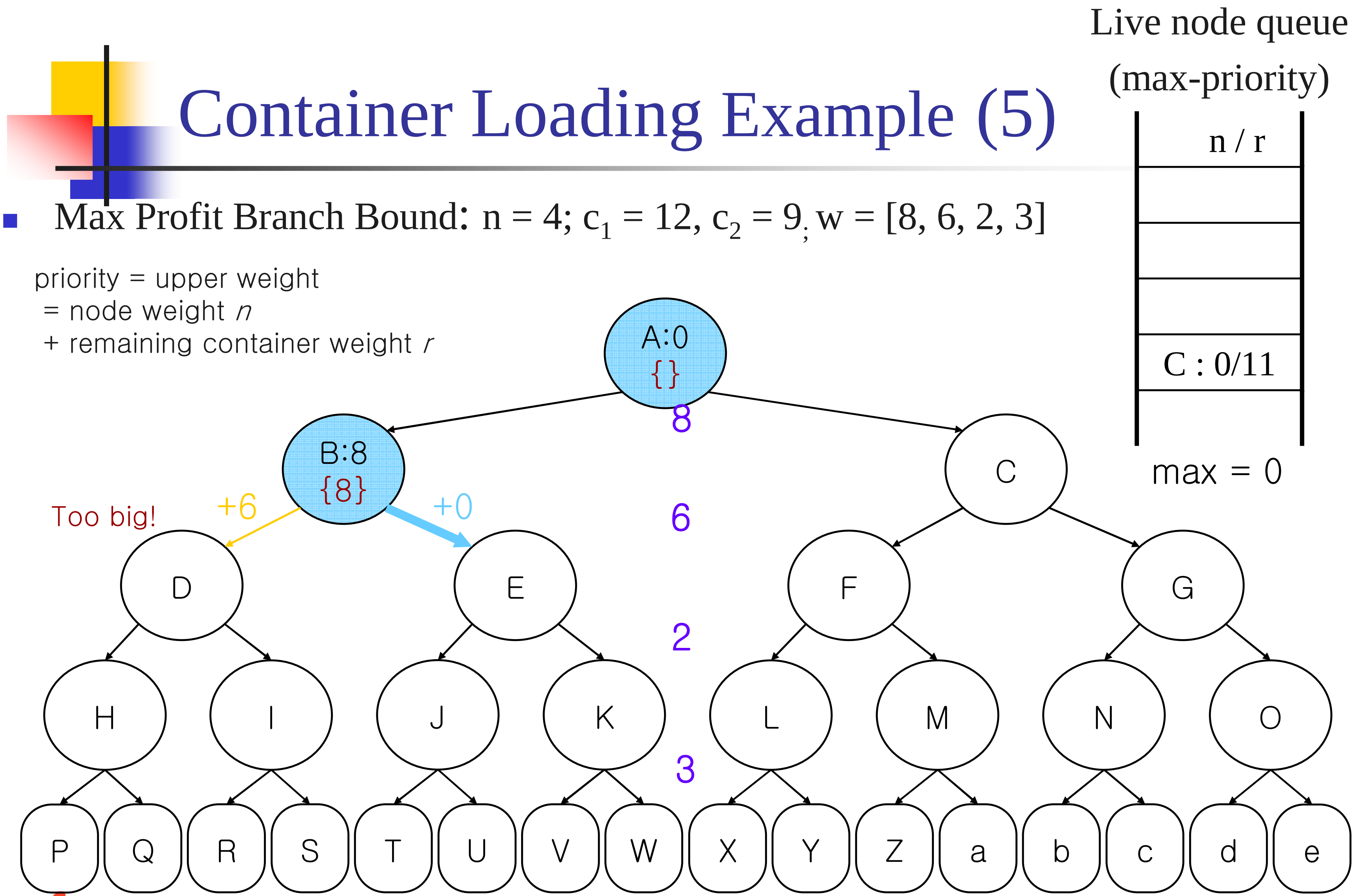


Pop B and Move (Bound) to B

Container Loading Example (5)

- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight n
+ remaining container weight r



Push E // Branch

Container Loading Example (6)

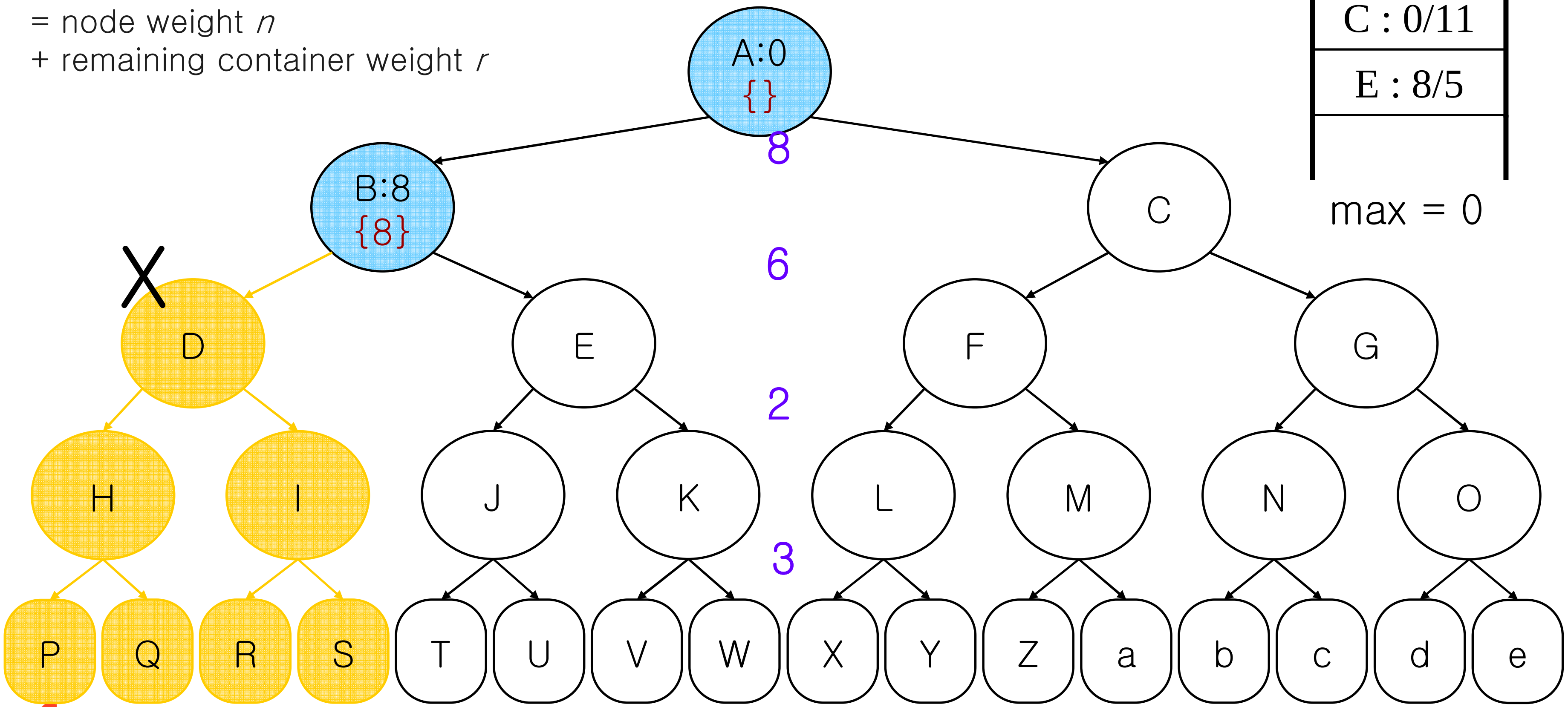
- Max Profit Branch Bound: n = 4; c_1 = 12, c_2 = 9; w = [8, 6, 2, 3]

priority = upper weight
= node weight *n*
+ remaining container weight *r*

Live node queue
(max-priority)

n / r
C : 0/11
E : 8/5

max = 0

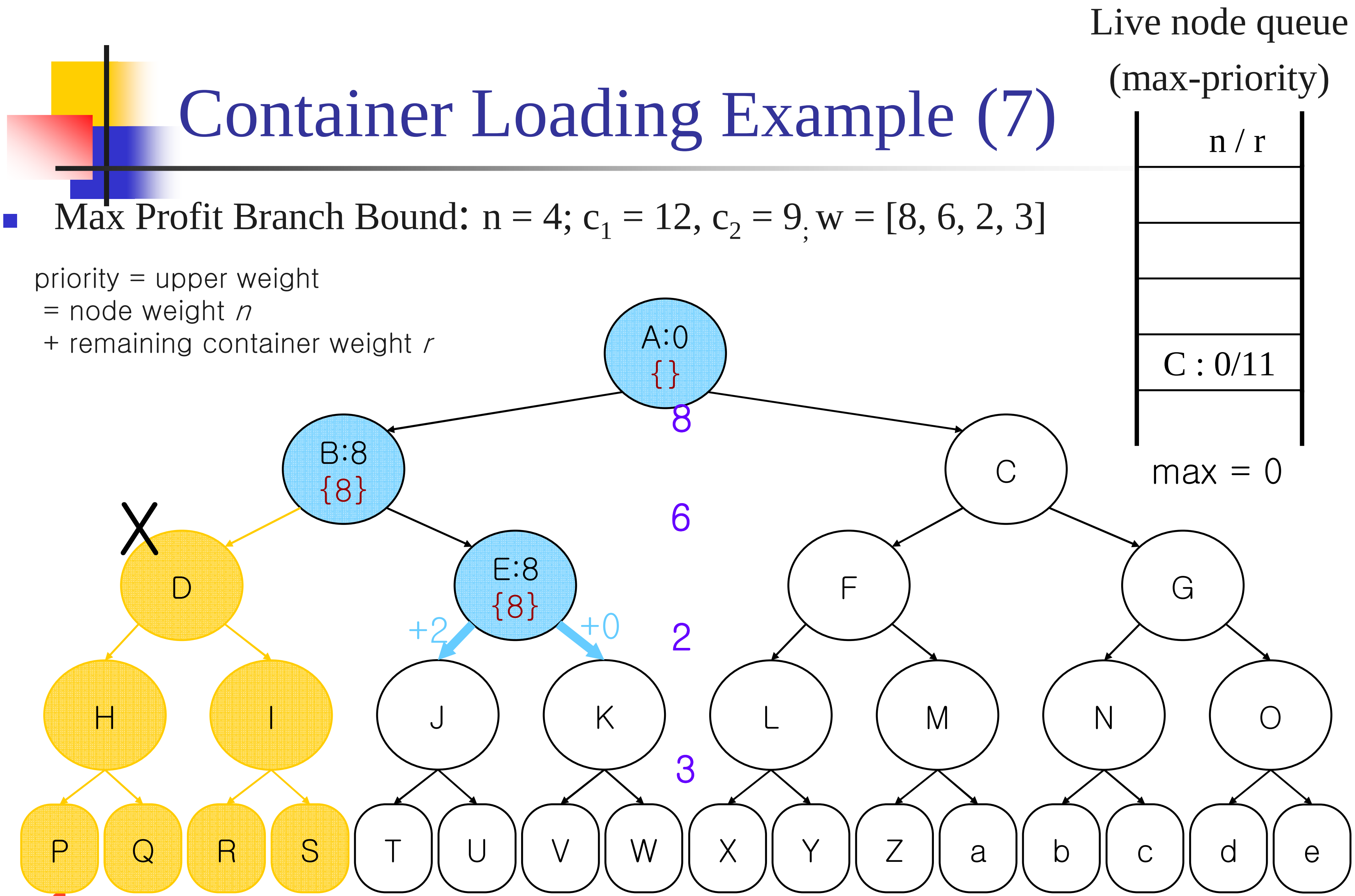


Pop E and Move (Bound) to E

Container Loading Example (7)

- Max Profit Branch Bound: n = 4; $c_1 = 12$, $c_2 = 9$, w = [8, 6, 2, 3]

priority = upper weight
= node weight *n*
+ remaining container weight *r*



Push J and K // Branch

Container Loading Example (8)

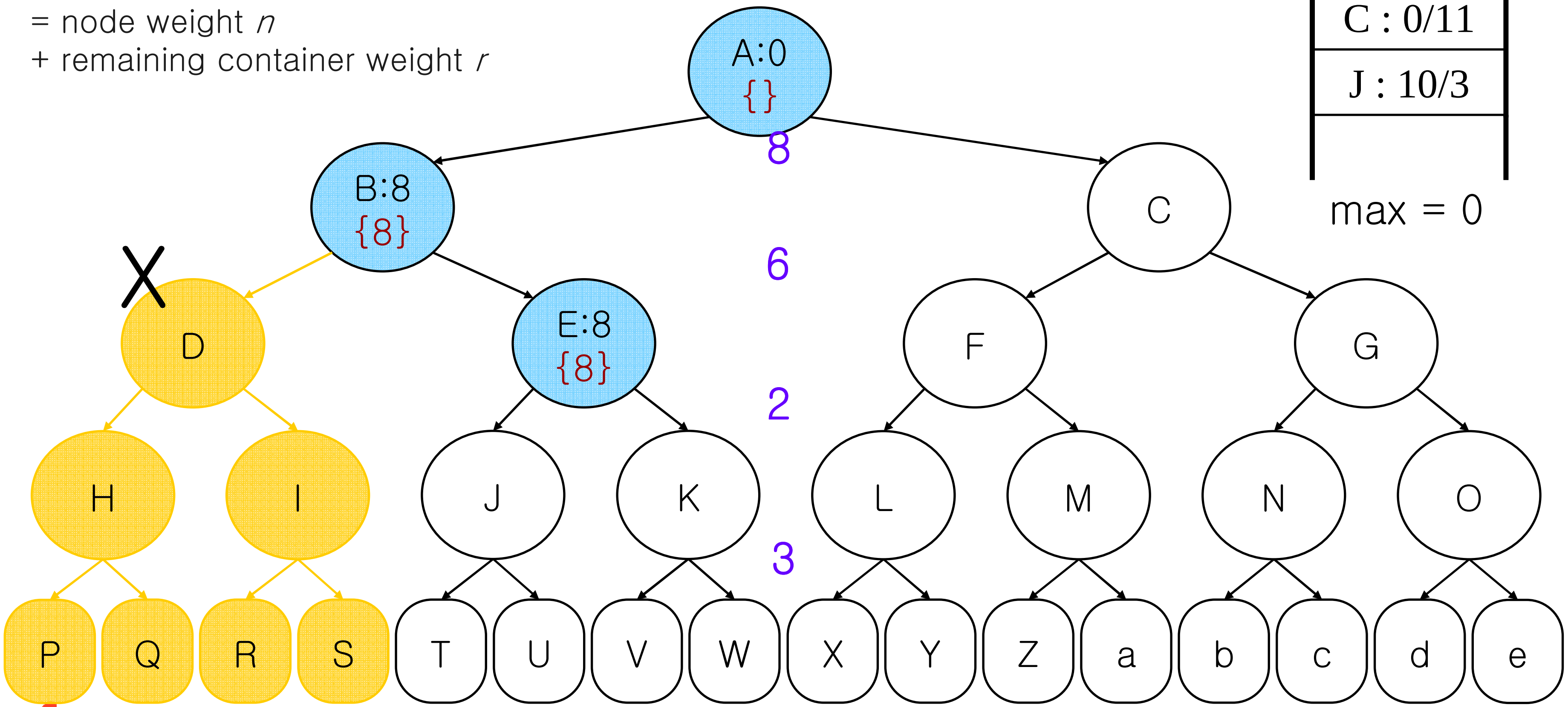
- Max Profit Branch Bound: n = 4; c_1 = 12, c_2 = 9; w = [8, 6, 2, 3]

priority = upper weight
= node weight *n*
+ remaining container weight *r*

Live node queue (max-priority)

n / r
K : 8/3
C : 0/11
J : 10/3

max = 0

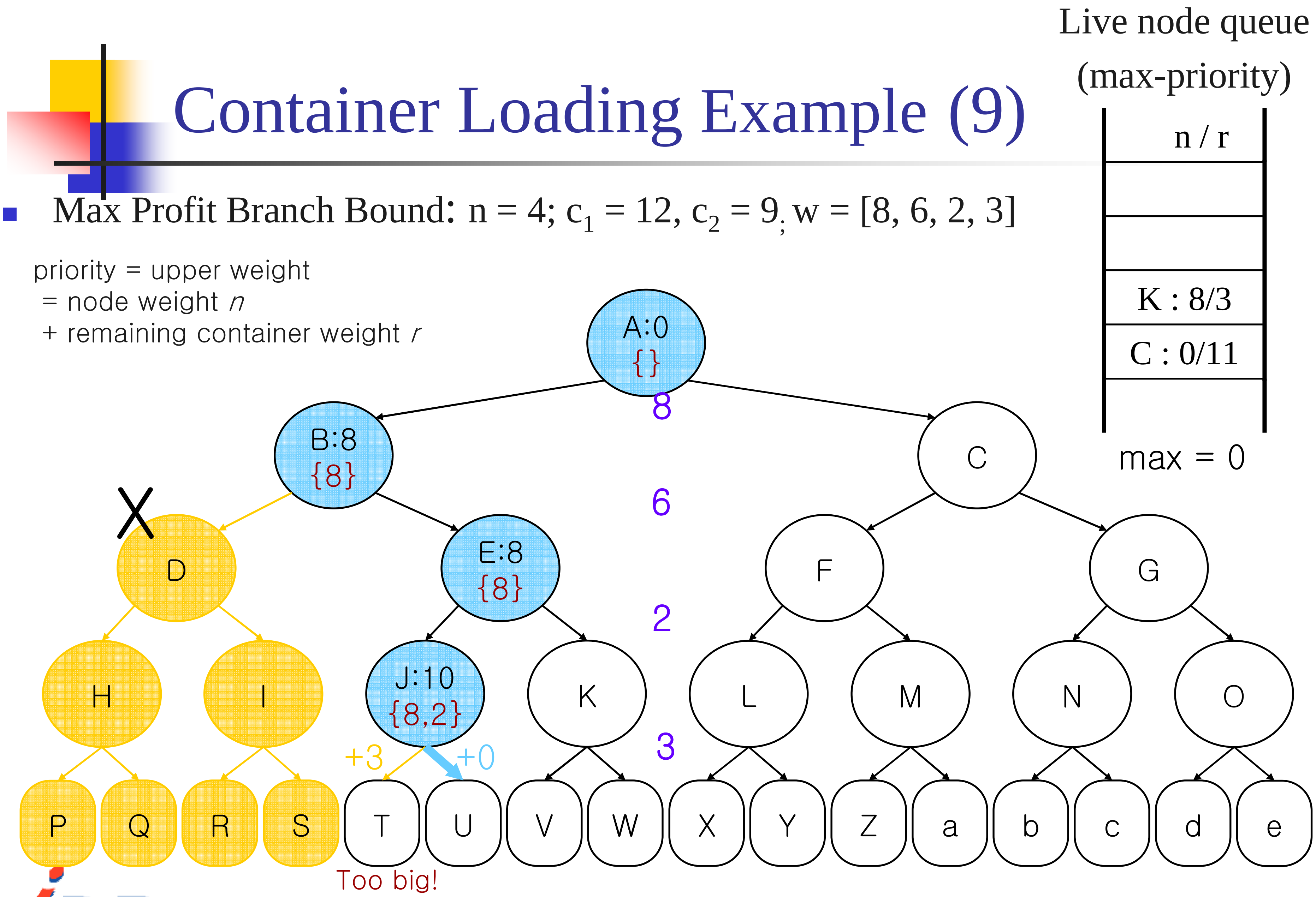


Pop J and Move (Bound) to J

Container Loading Example (9)

- Max Profit Branch Bound: n = 4; c_1 = 12, c_2 = 9, w = [8, 6, 2, 3]

priority = upper weight
= node weight *n*
+ remaining container weight *r*



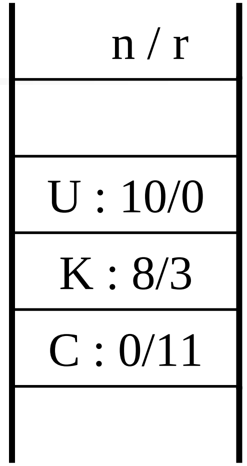
Push U // Branch

Container Loading Example (10)

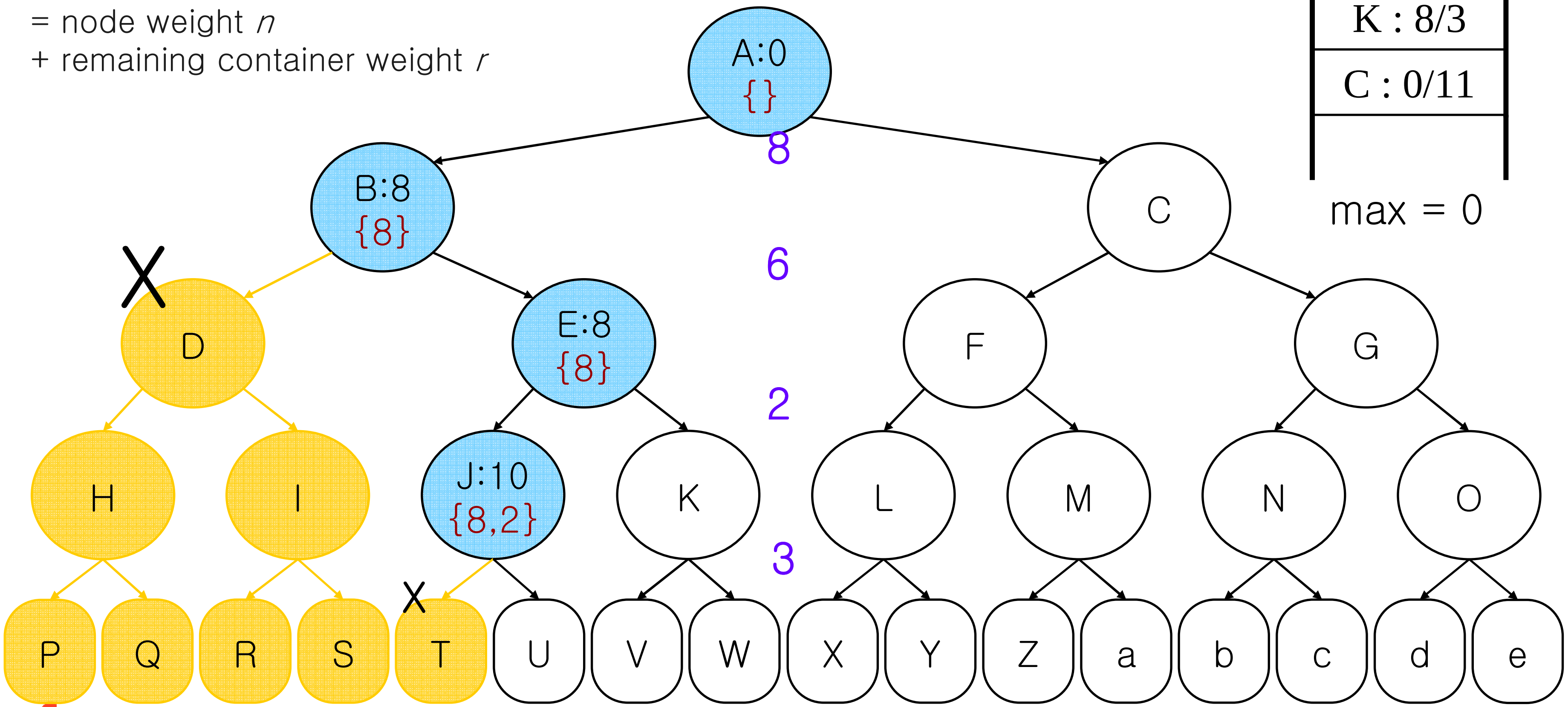
- Max Profit Branch Bound: n = 4; c_1 = 12, c_2 = 9, w = [8, 6, 2, 3]

priority = upper weight
 = node weight *n*
 + remaining container weight *r*

Live node queue
(max-priority)



max = 0



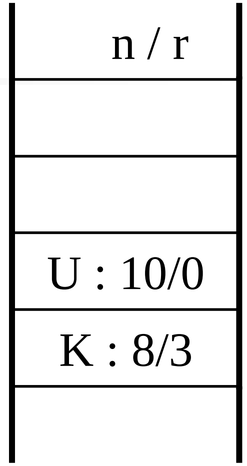
Pop C and Move (Bound) to C

Container Loading Example (11)

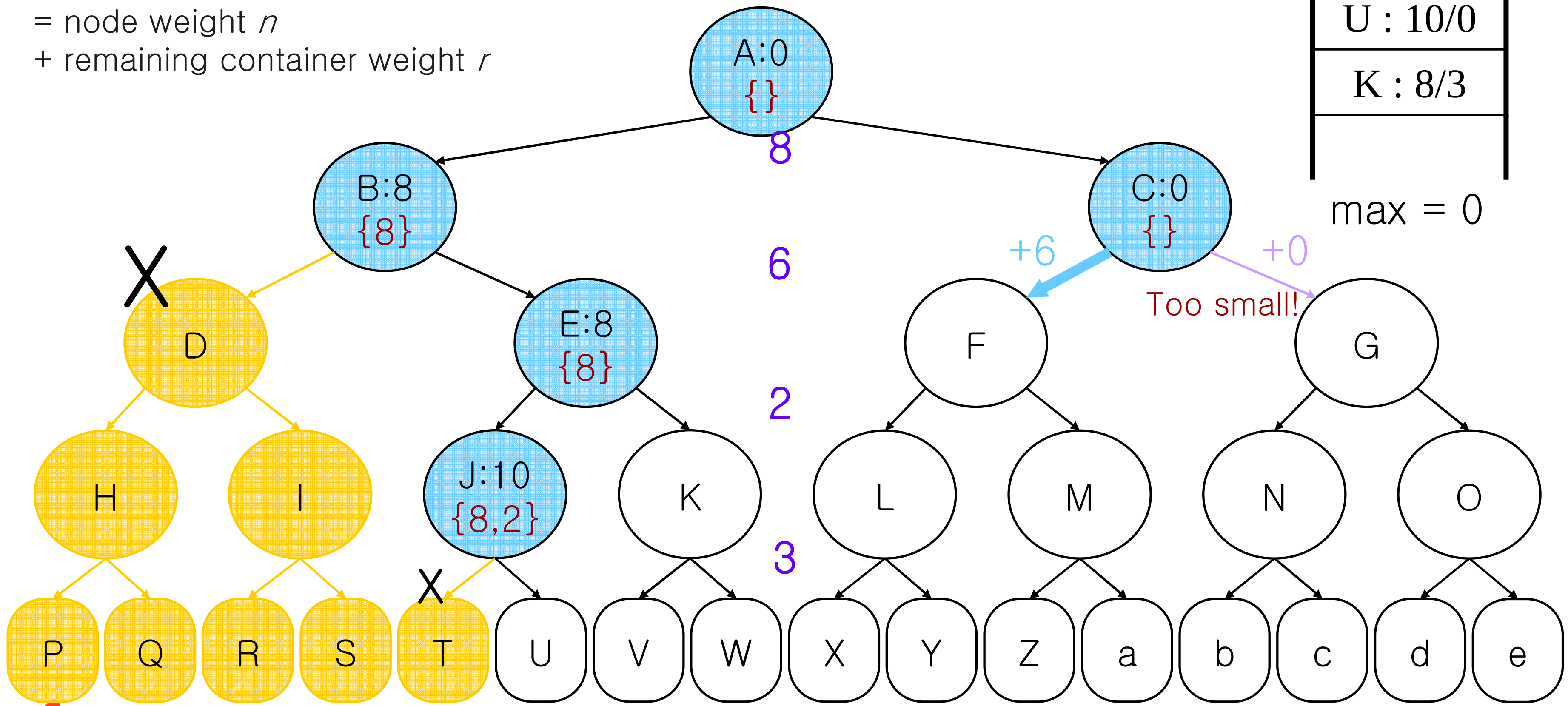
- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*

Live node queue (max-priority)



max = 0

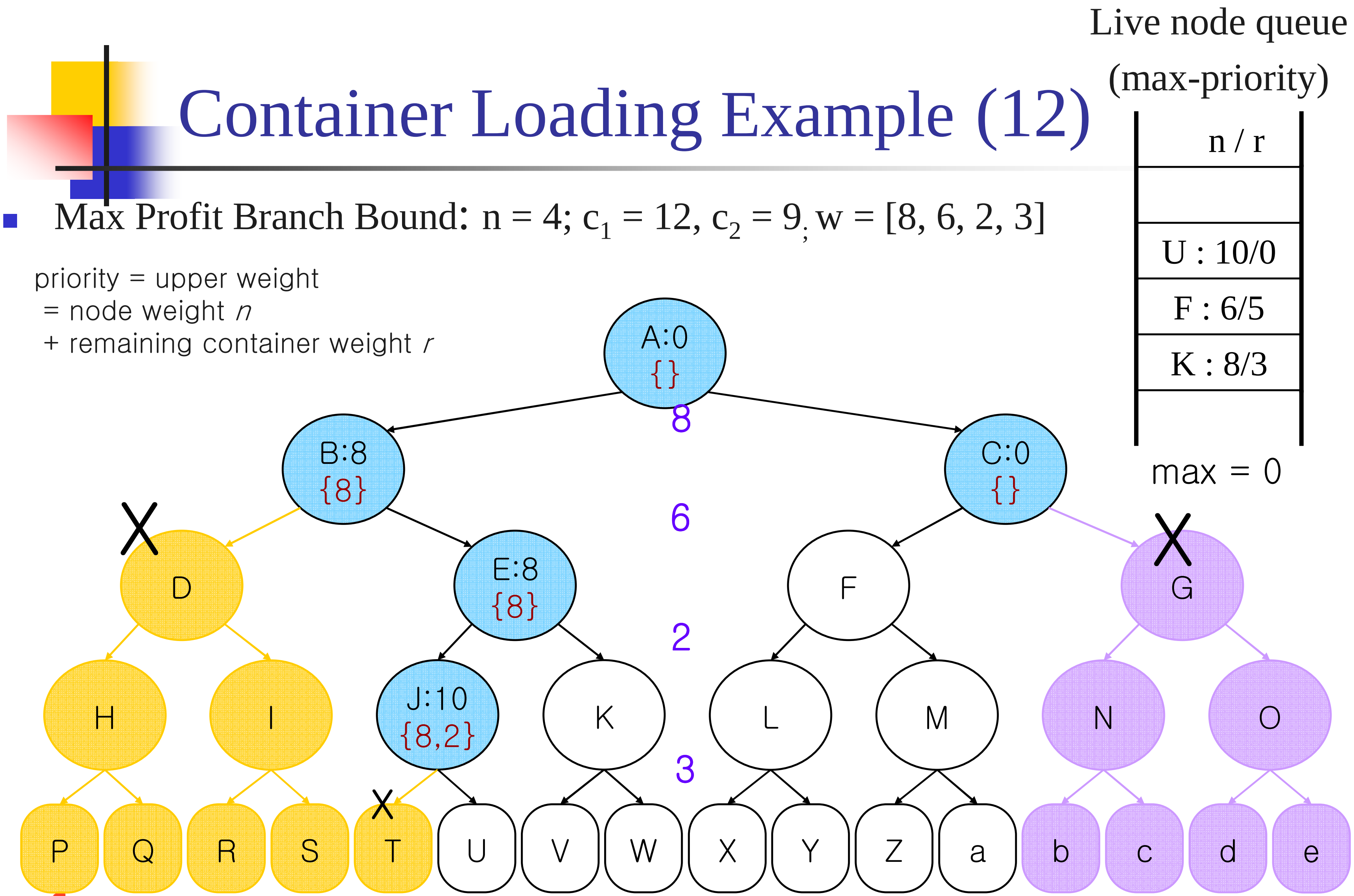


Push F // Branch

Container Loading Example (12)

- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*



Pop K and Move (Bound) to K

Container Loading Example (13)

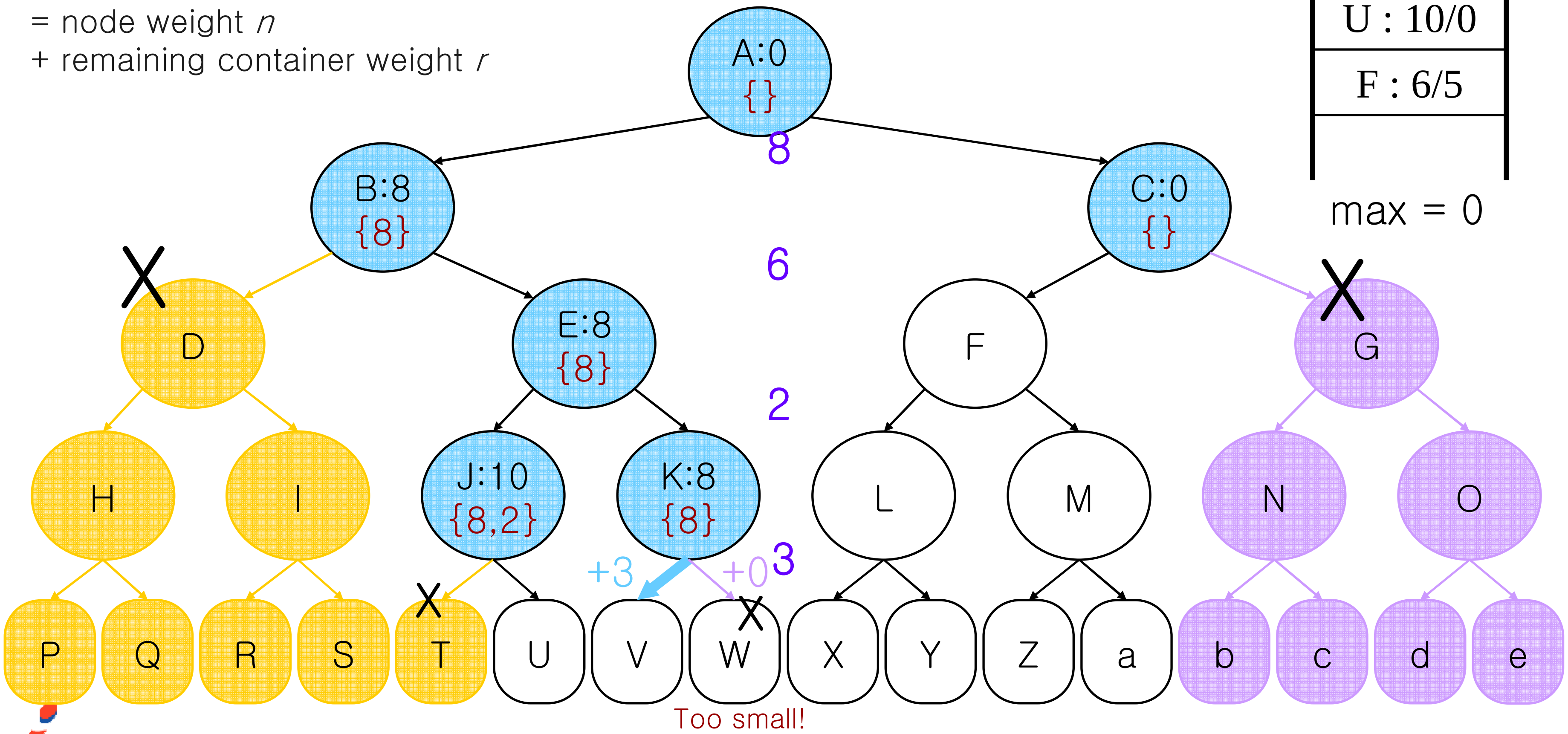
- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$, $w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*

Live node queue
(max-priority)

n / r
U : 10/0
F : 6/5

max = 0

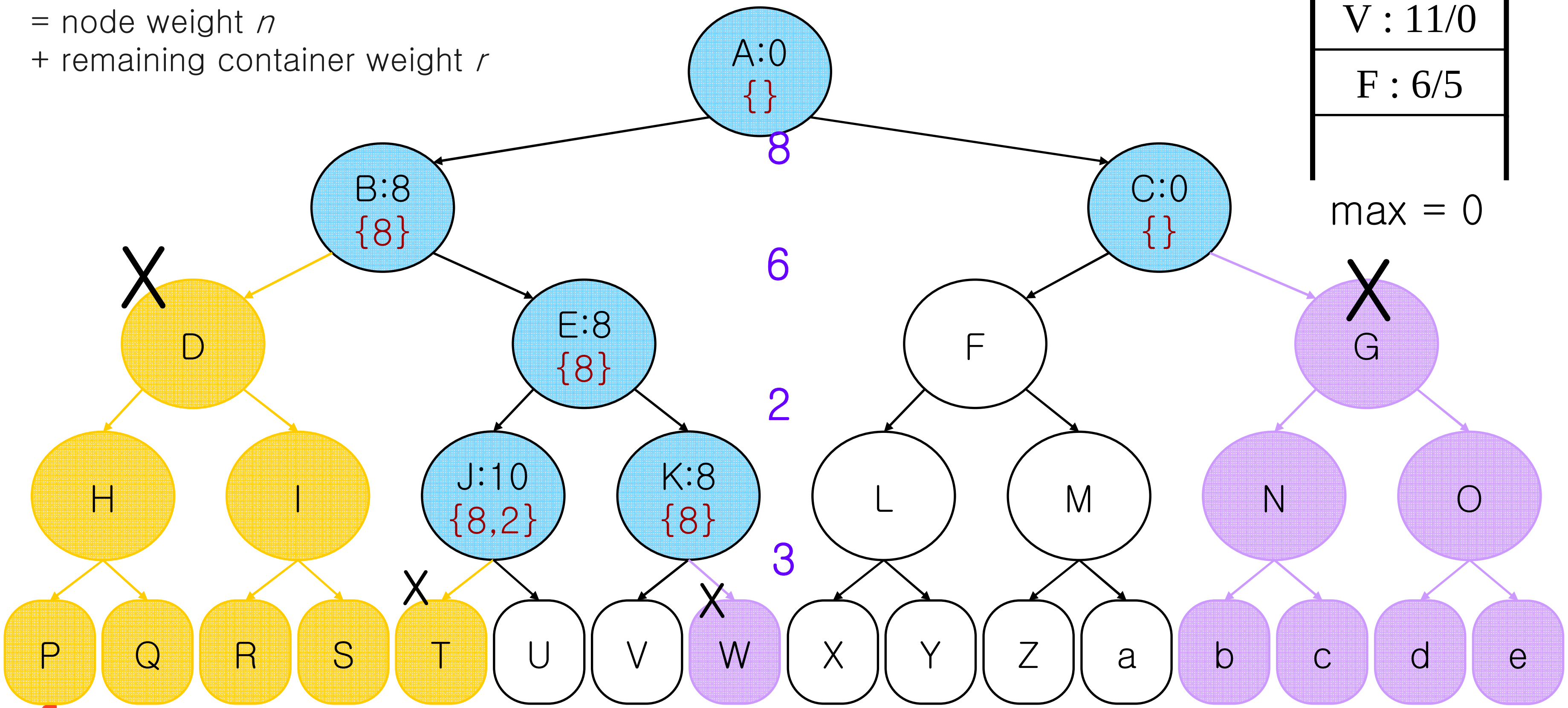


Push V // Branch

Container Loading Example (14)

- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*



Live node queue (max-priority)

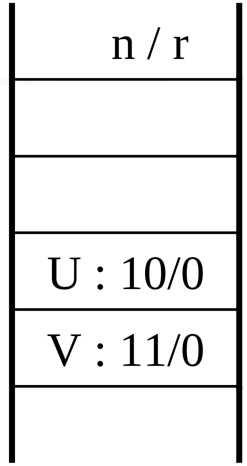
n / r
U : 10/0
V : 11/0
F : 6/5

max = 0

Pop F and Move (Bound) to F

Container Loading Example (15)

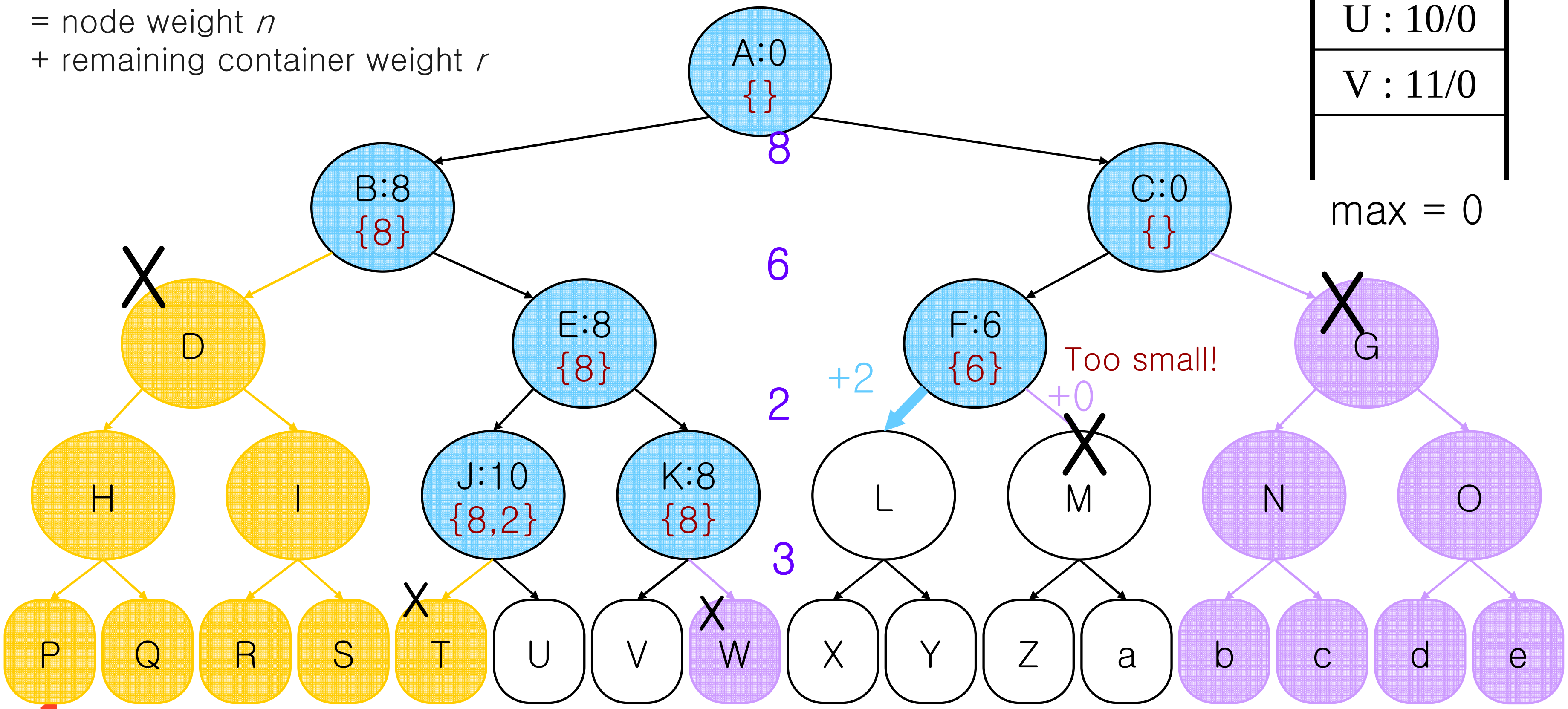
Live node queue
(max-priority)



max = 0

- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight n
+ remaining container weight r

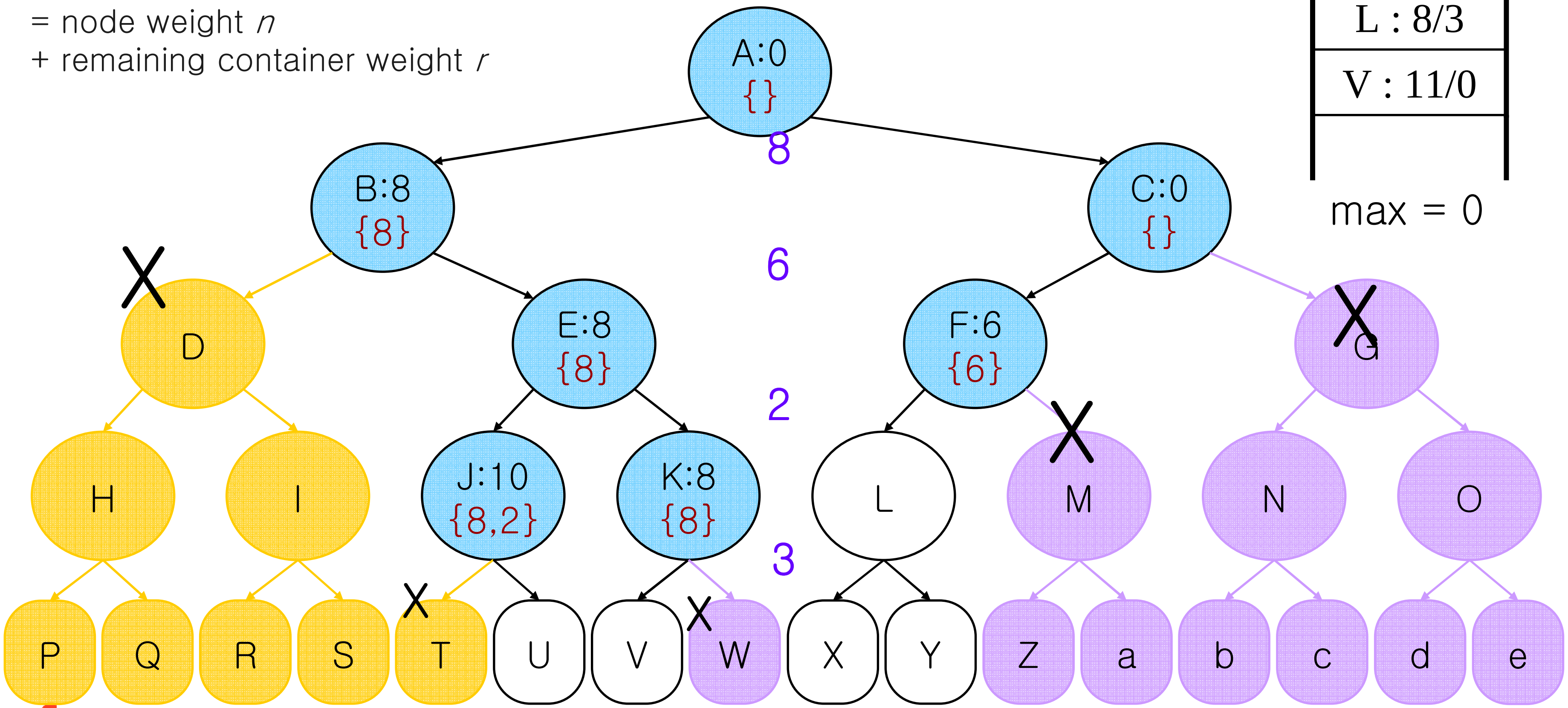


Push L // Branch

Container Loading Example (16)

- Max Profit Branch Bound: n = 4; c_1 = 12, c_2 = 9; w = [8, 6, 2, 3]

priority = upper weight
= node weight *n*
+ remaining container weight *r*



Live node queue
(max-priority)

n / r
U : 10/0
L : 8/3
V : 11/0

max = 0

Pop V and Move (Bound) to V

Container Loading Example (17)

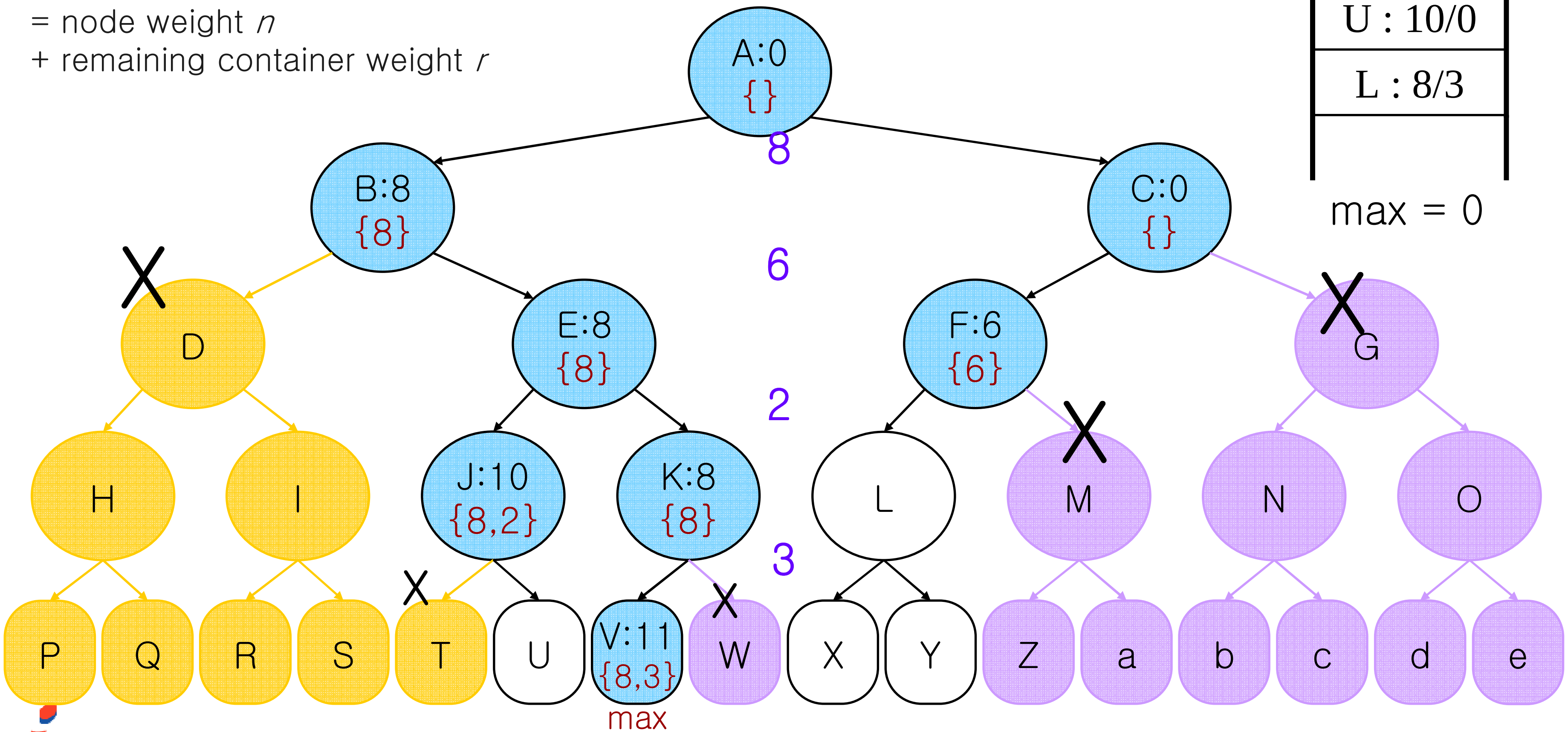
- Max Profit Branch Bound: $n = 4; c_1 = 12, c_2 = 9, w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*

Live node queue
(max-priority)

n / r
U : 10/0
L : 8/3

max = 0



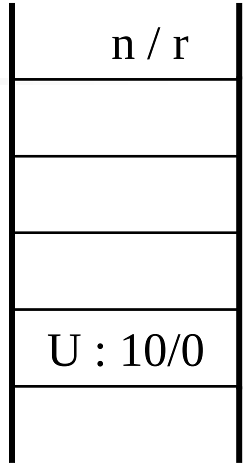
Set max←11, Pop L and Move (Bound) to L

Container Loading Example (18)

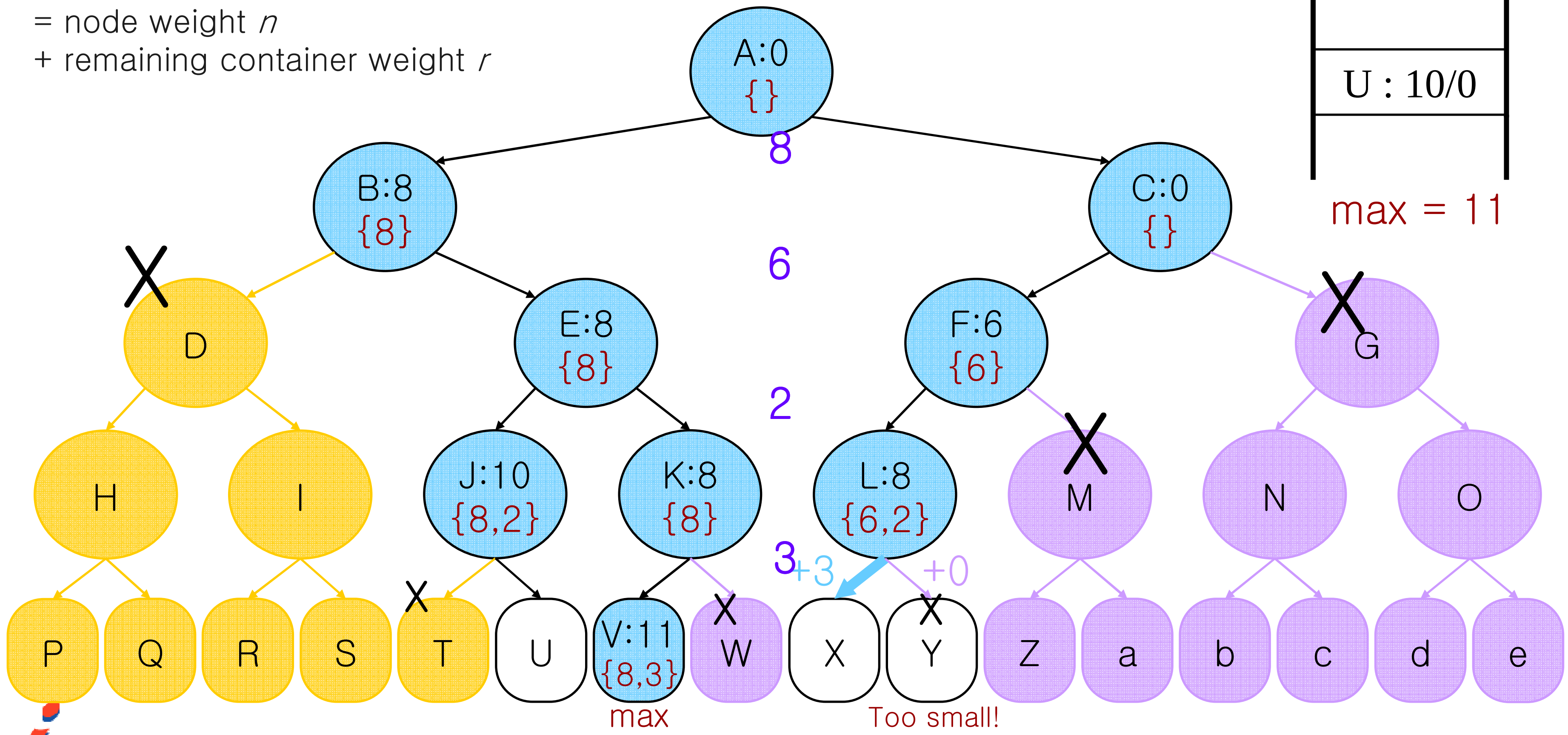
- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*

Live node queue
(max-priority)



max = 11

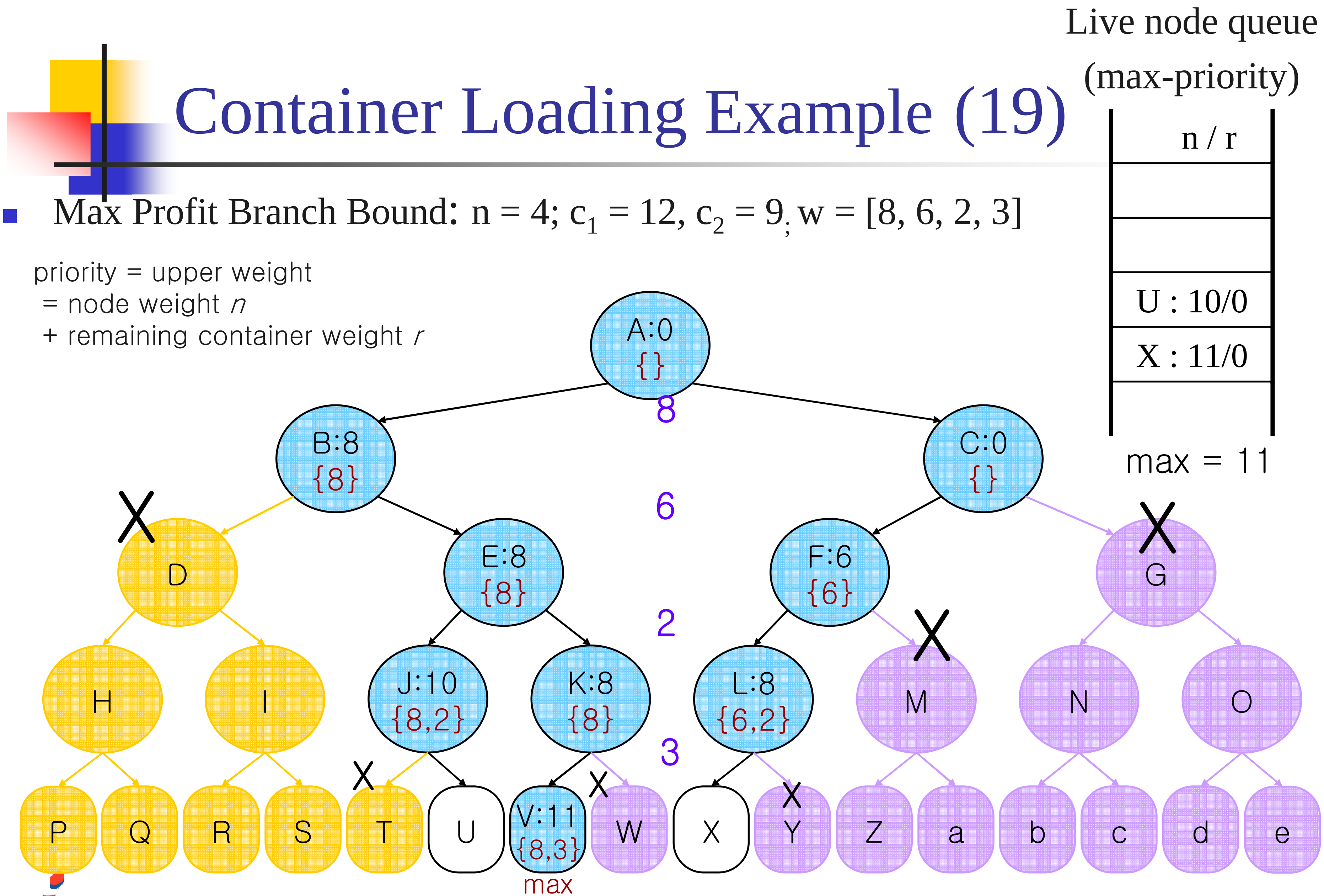


Push X // Branch

Container Loading Example (19)

- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight *n*
+ remaining container weight *r*

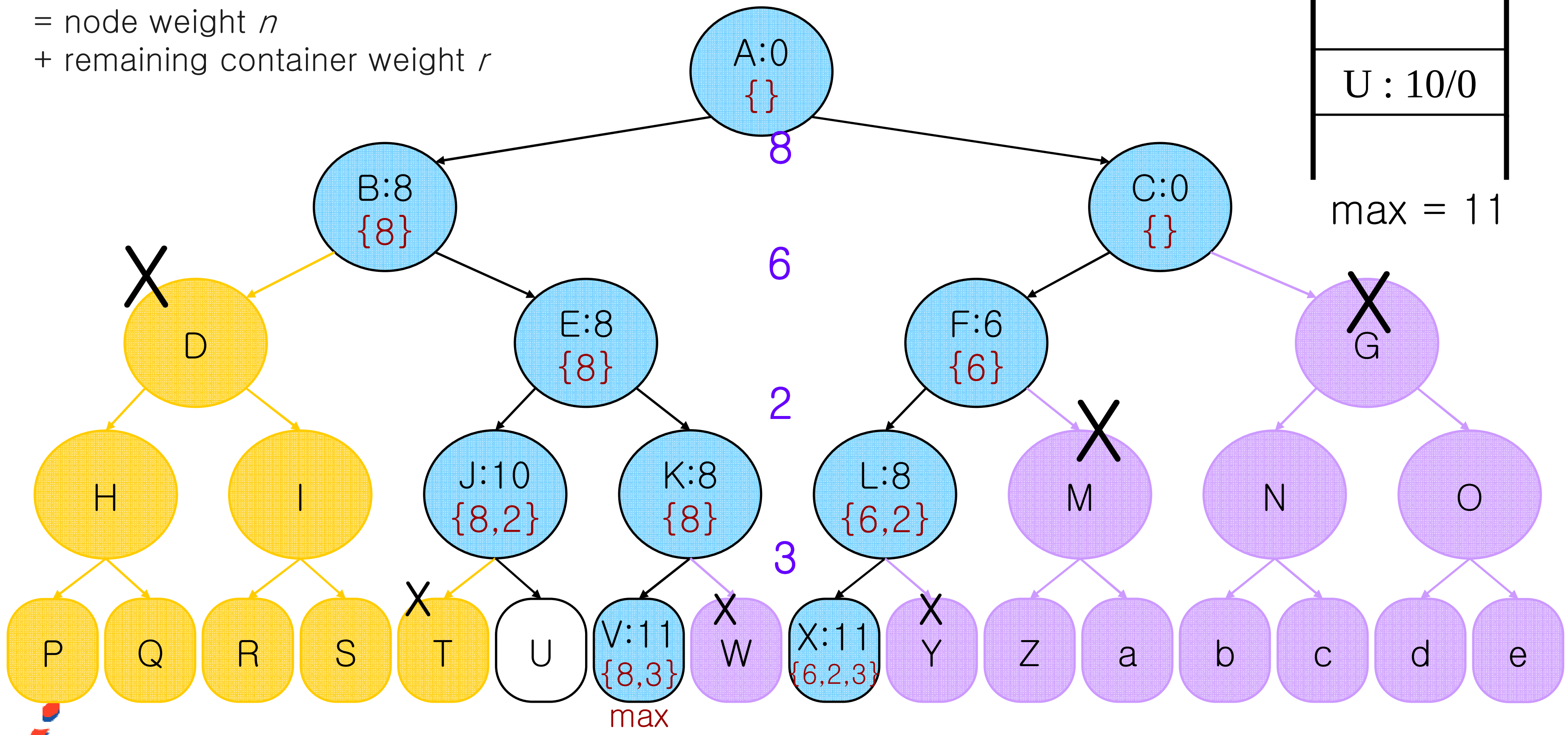


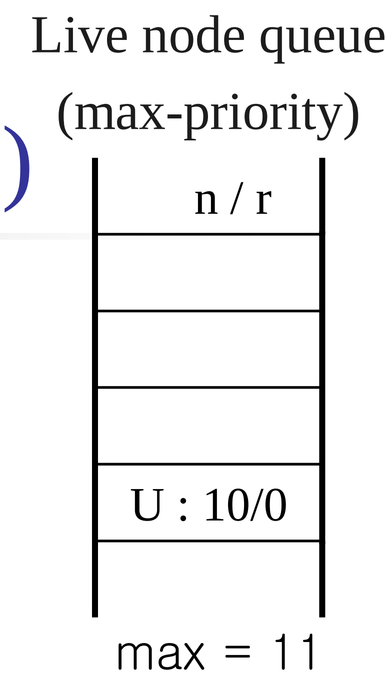
Pop X and Move (Bound) to X

Container Loading Example (20)

- Max Profit Branch Bound: n = 4; c_1 = 12, c_2 = 9; w = [8, 6, 2, 3]

priority = upper weight
= node weight *n*
+ remaining container weight *r*



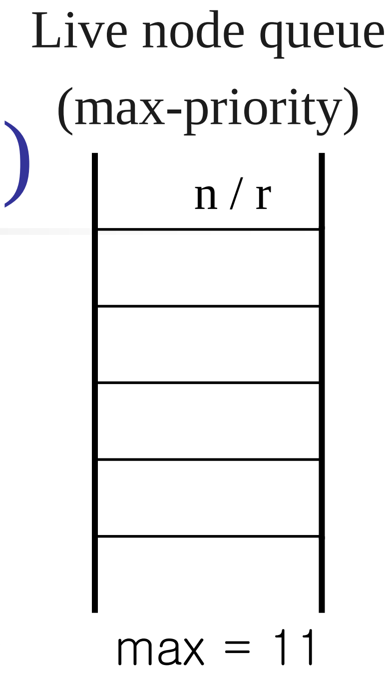


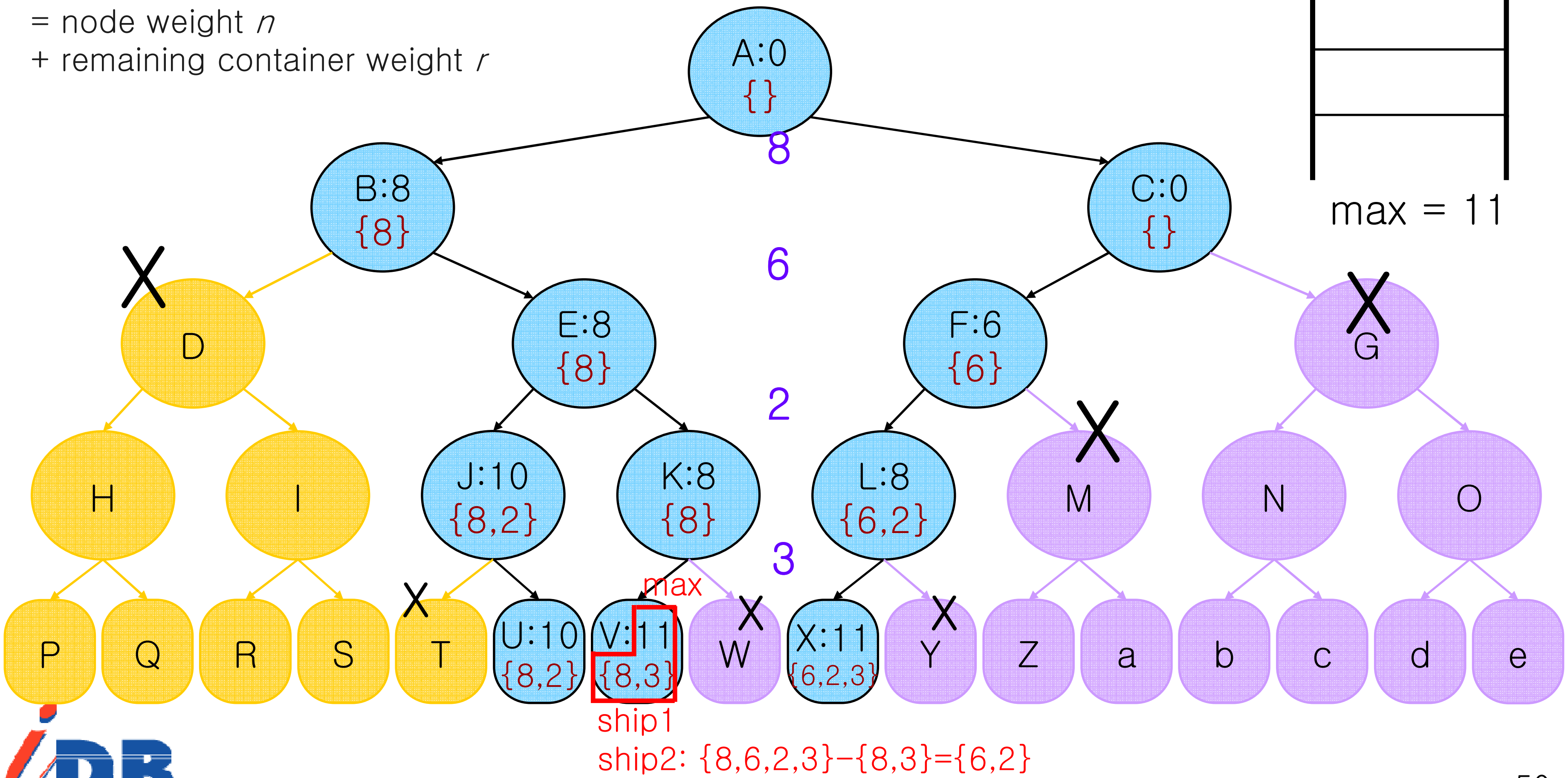
Pop U and Move (Bound) to U

Container Loading Example (21)

- Max Profit Branch Bound: $n = 4$; $c_1 = 12$, $c_2 = 9$; $w = [8, 6, 2, 3]$

priority = upper weight
= node weight n
+ remaining container weight r





BIRD'S-EYE VIEW

- A surefire way to solve a problem is to make a list of all candidate answers and check them
  - If the problem size is big, we can not get the answer in reasonable time using this approach
  - List all possible cases? → exponential cases
- By a systematic examination of the candidate list, we can find the answer without examining every candidate answer
  - *Backtracking* and *Branch and Bound* are most popular systematic algorithms
- Branch and Bound
  - Searches a solution space that is often organized as a tree (like backtracking)
  - Usually searches a tree in a breadth-first / least-cost manner (unlike backtracking)

SNU Internet DataBase Lab.

Table of Contents

- The Branch and Bound Method
- Application
  - Rat in a Maze
  - Container Loading