Kourosh Davoudi
kourosh@ontariotechu.ca

Lecture 8: Branch and Bound Algorithms

CSCI 3070U: Design and Analysis of Algorithms

Learning Outcomes

- Branch and Bound Foundation
- Case Studies:
  - Project Management
  - 0/1 Knapsack

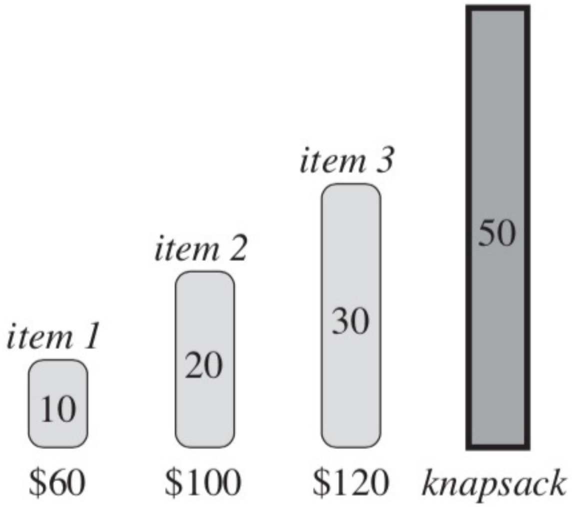
Branch and Bound Foundation

- When branch and bound is useful?
  - It is generally used for solving discrete optimization problem, where exhaustive search is not possible !

- Example:

  0/1 – Knapsack problem:

$$
\begin{aligned}
\text{maximize} \quad & 60x_1 + 100x_2 + 120x_3 \\
& 10x_1 + 20x_2 + 30x_3 \leq 50 \\
& x_i \in \{0, 1\}
\end{aligned}
$$



(a)

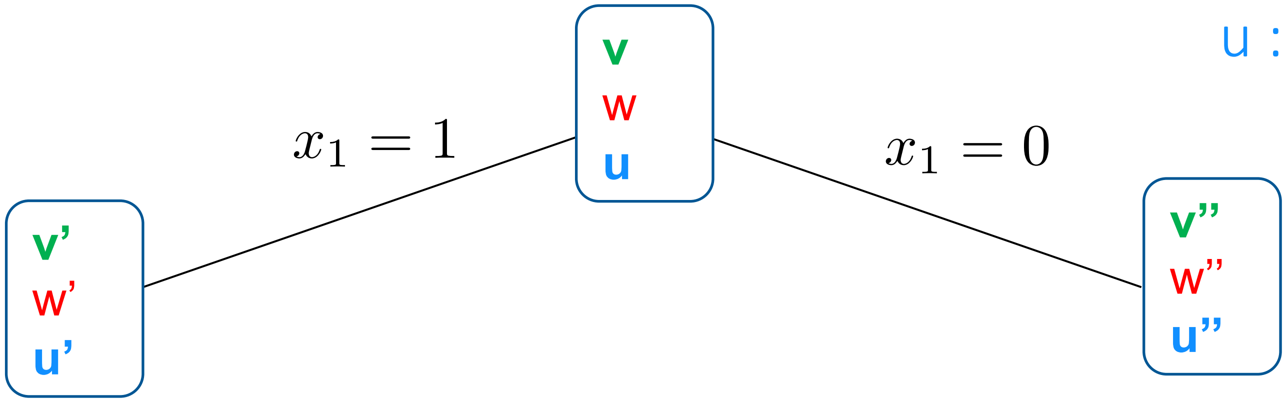
Branch and Bound Foundation

- General Idea:
  - We compute bound (best solution) for every node and compare the bound with current best solution before exploring the node.

Branch and Bound Foundation

- Branch
  - Continuously break the problem into sub-problems

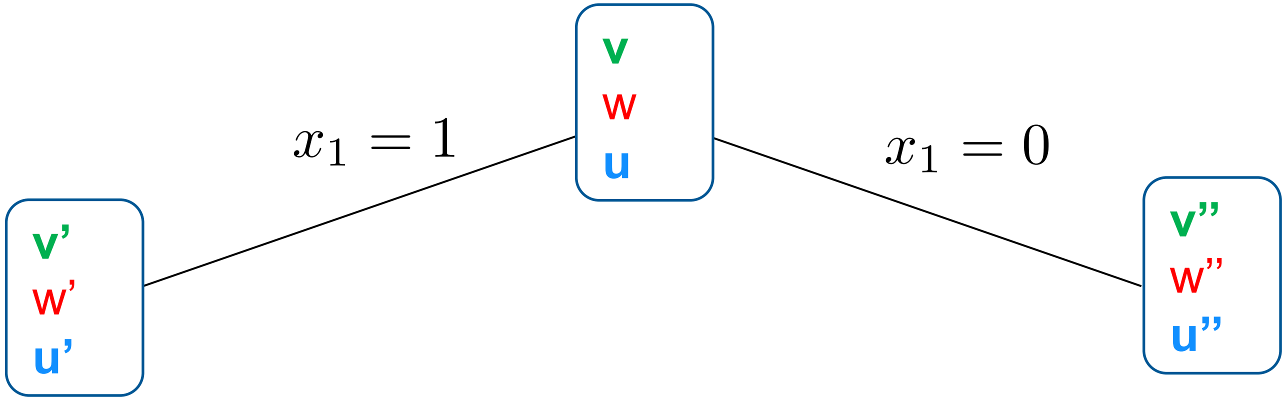
v : value in knapsack
w : available room
u : upper bound heuristic



  - Upper bound heuristic is "optimistic value" that we can obtain

Branch and Bound Foundation

- Bound
  - Compare the optimistic value with the best value obtained so far. If it is less, stop the search in that branch



If u'' < best value so far then stop

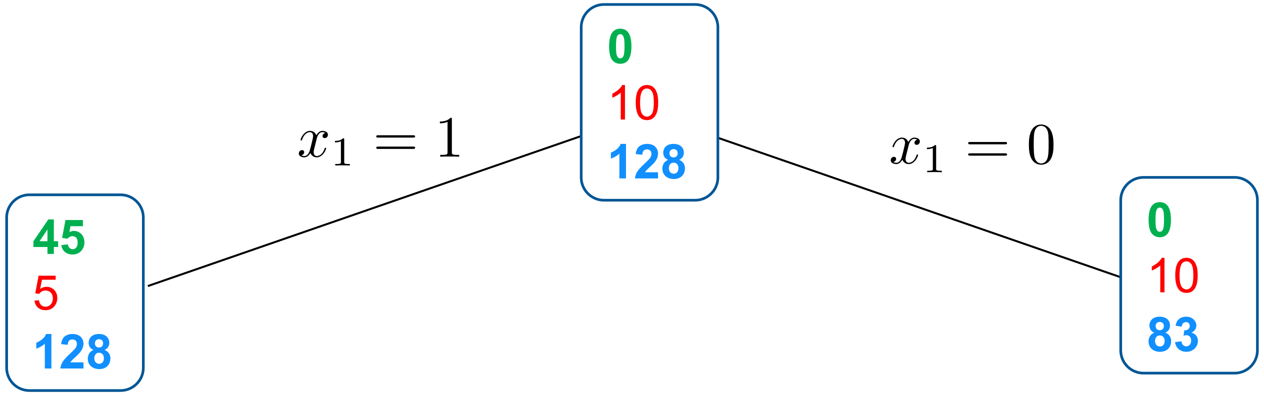
Heuristic Bound

- Heuristic bound
  - In maximization problem is optimistic upper bound (highest value)
  - In minimization problem is optimistic lower bound (lowest cost)

- How to design the heuristic ?
  - Depends on the problem
  - The tighter one is better
  - It should not remove the optimum solution form search space !

  - Example : sum of value by selecting remining items !

Example 1: Loose upper bound



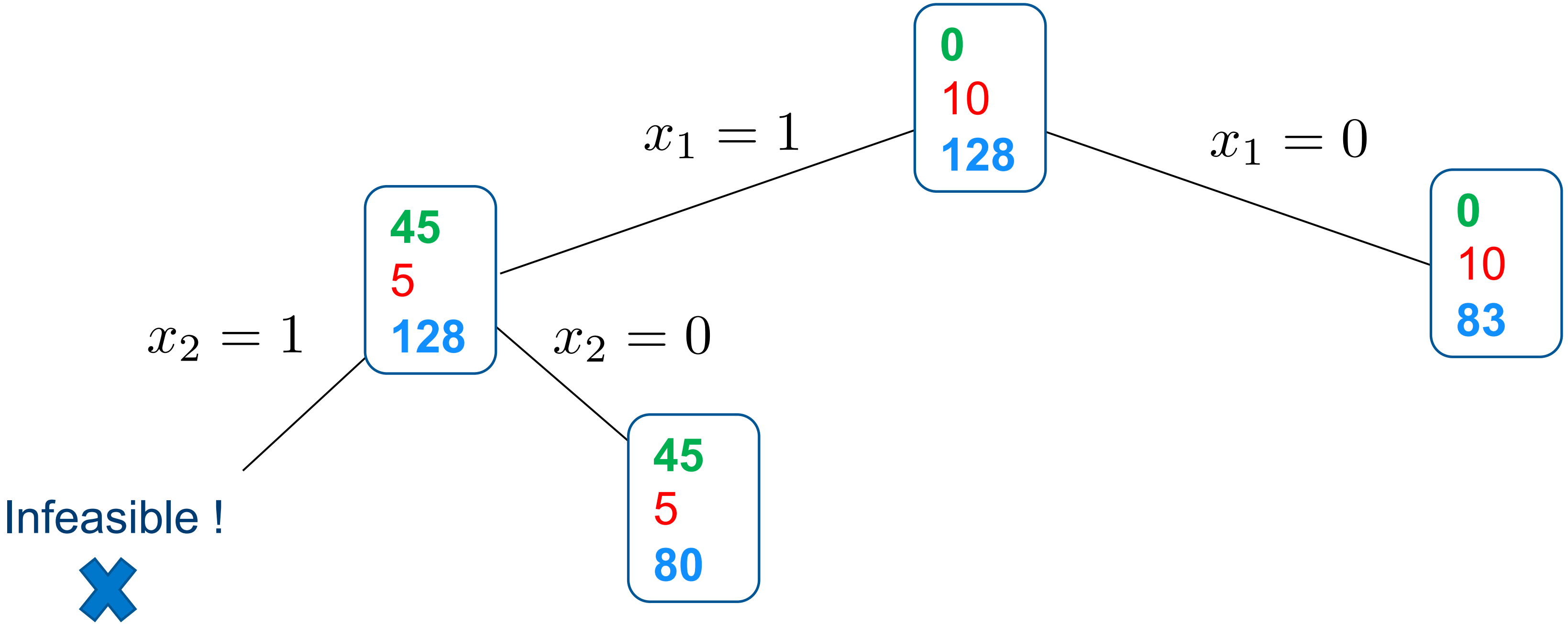
item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

v : value in knapsack
w : available room
u : upper bound heuristic

Heuristic upper bound : sum of value by selecting remining items !

Example 1: Loose upper bound



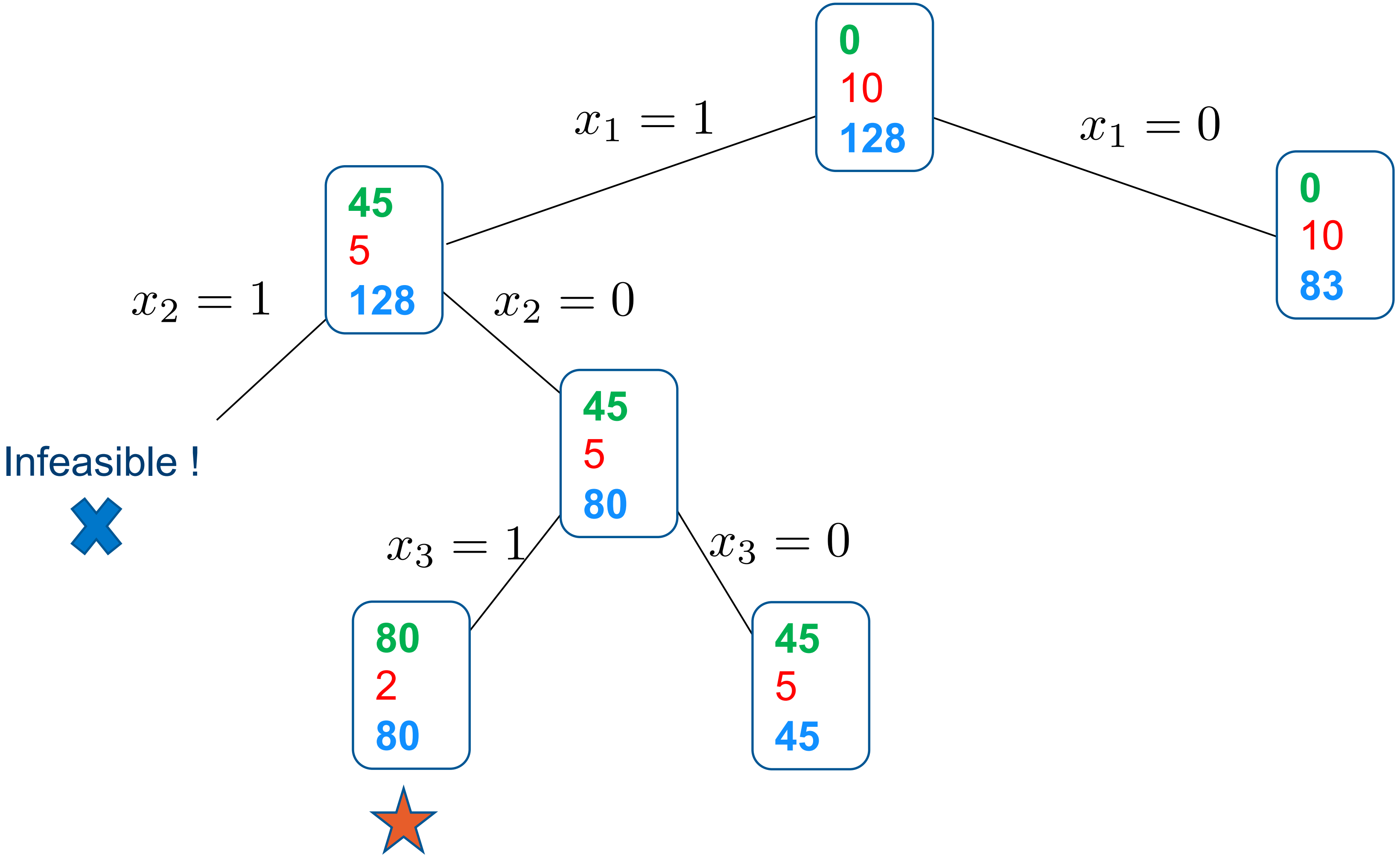
item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

v : value in knapsack
w : available room
u : upper bound heuristic

Heuristic upper bound : sum of value by selecting remining items !

Example 1: Loose upper bound



item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

v : value in knapsack
w : available room
u : upper bound heuristic

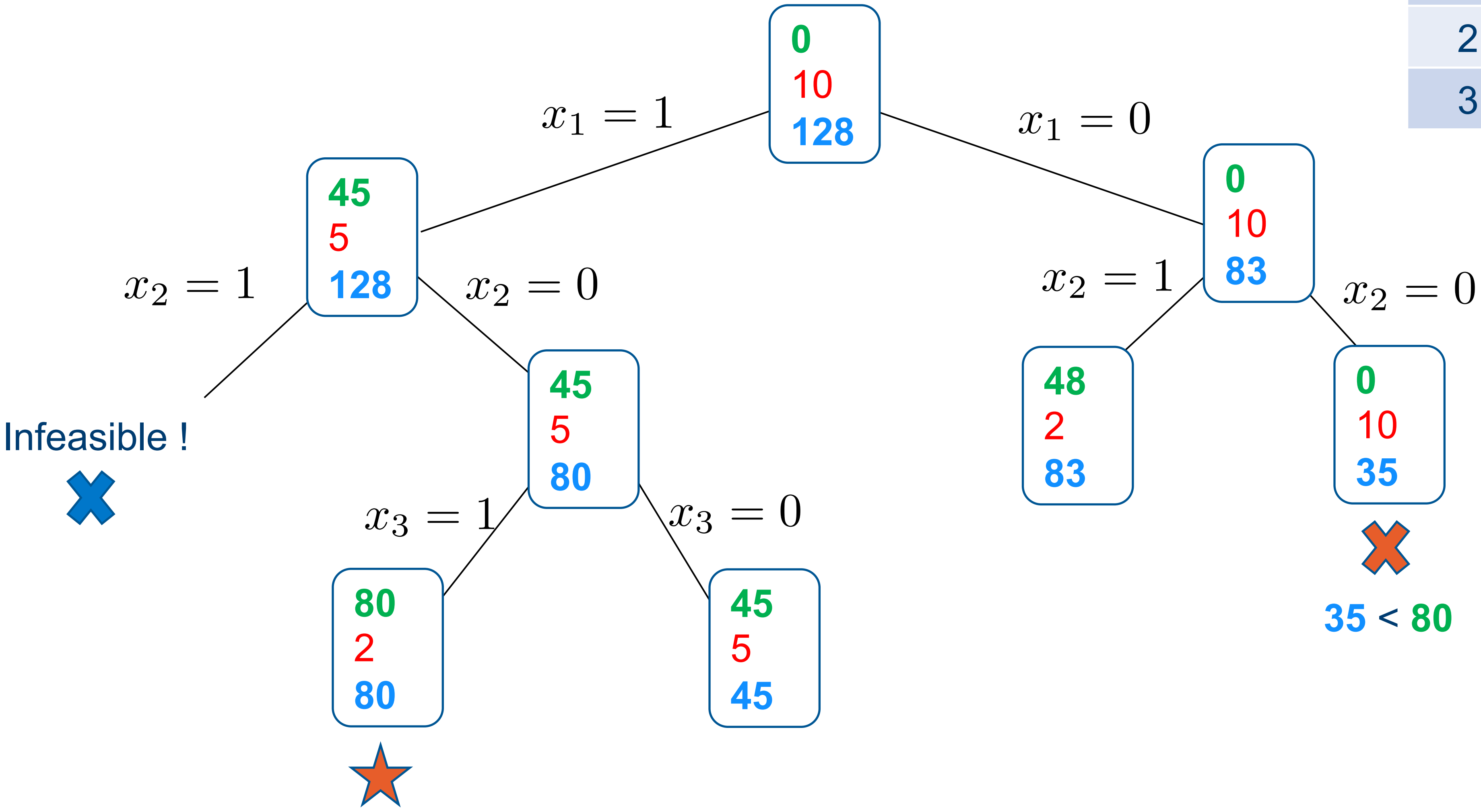
Heuristic upper bound : sum of value by selecting remining items !

Example 1: Loose upper bound

item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

v : value in knapsack
w : available room
u : upper bound heuristic



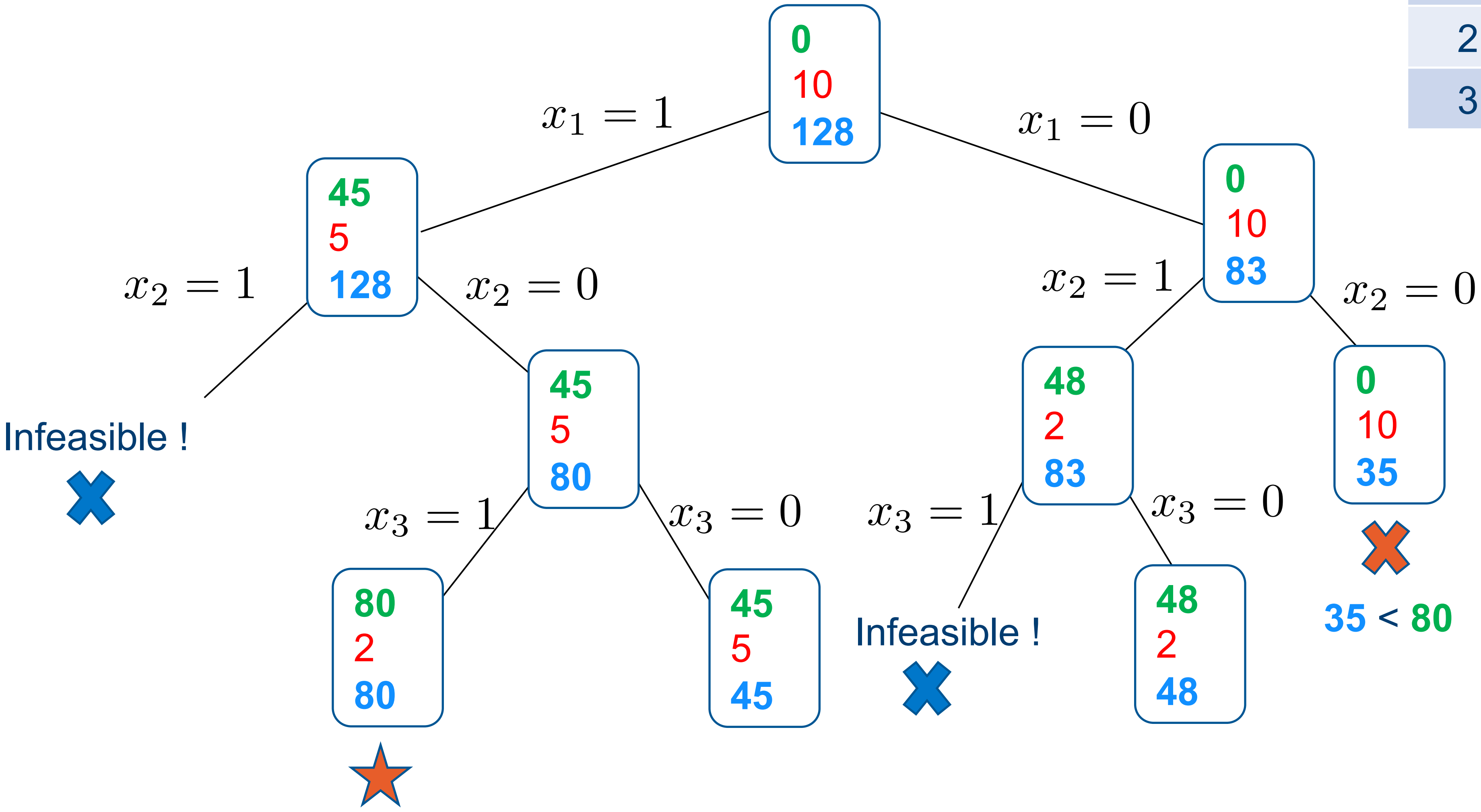
Heuristic upper bound : sum of value by selecting remining items !

Example 1: Loose upper bound

item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

v : value in knapsack
w : available room
u : upper bound heuristic



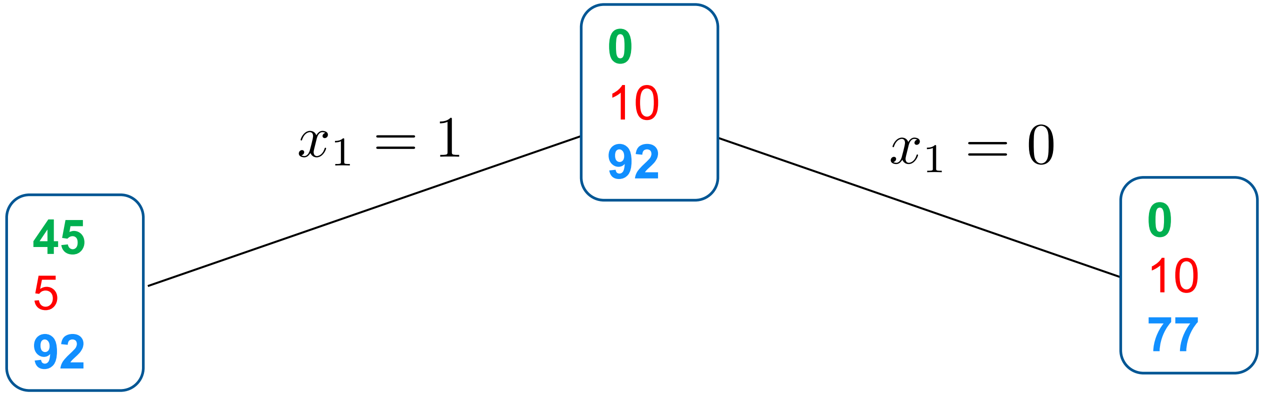
Heuristic upper bound : sum of value by seleting remining items !

Example 2: Tighter upper bound

Example 2: Tighter upper bound

item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

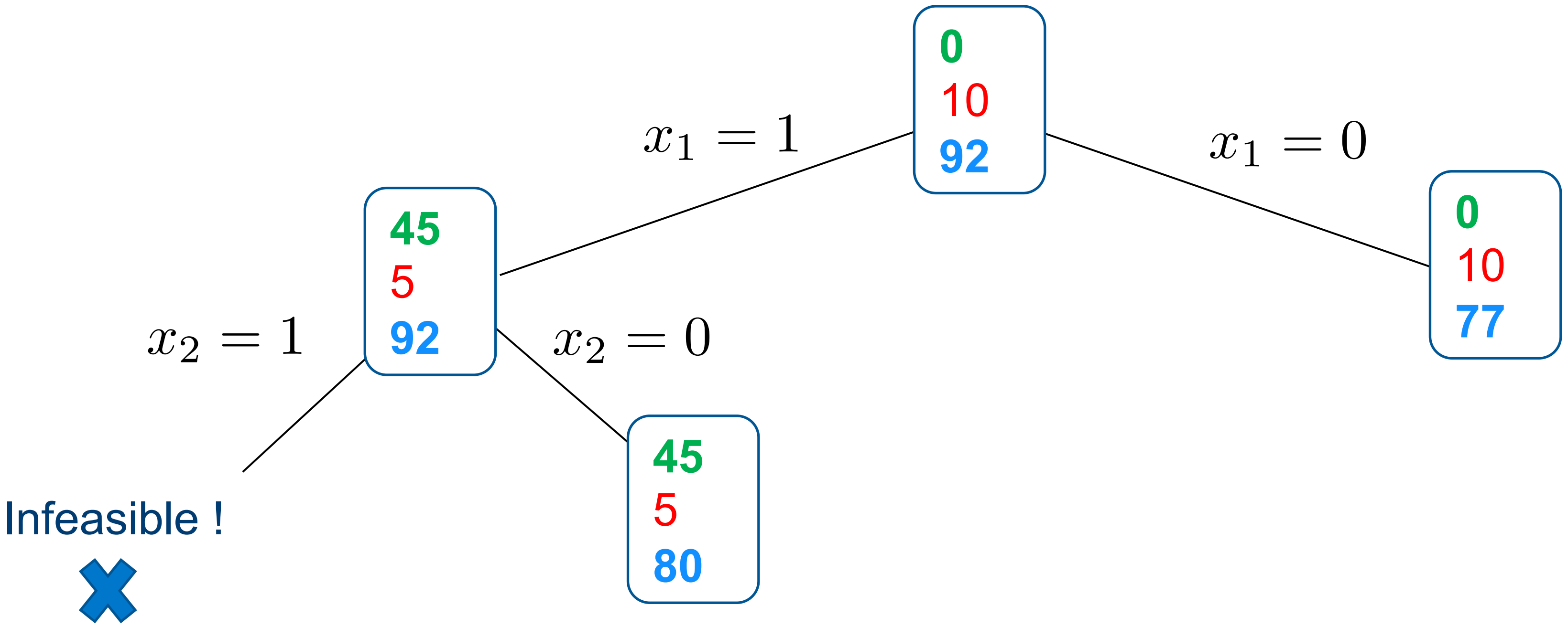


Heuristic upper bound : sum of value of remining items based on fractional knapsack

Example 2: Tighter upper bound

item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

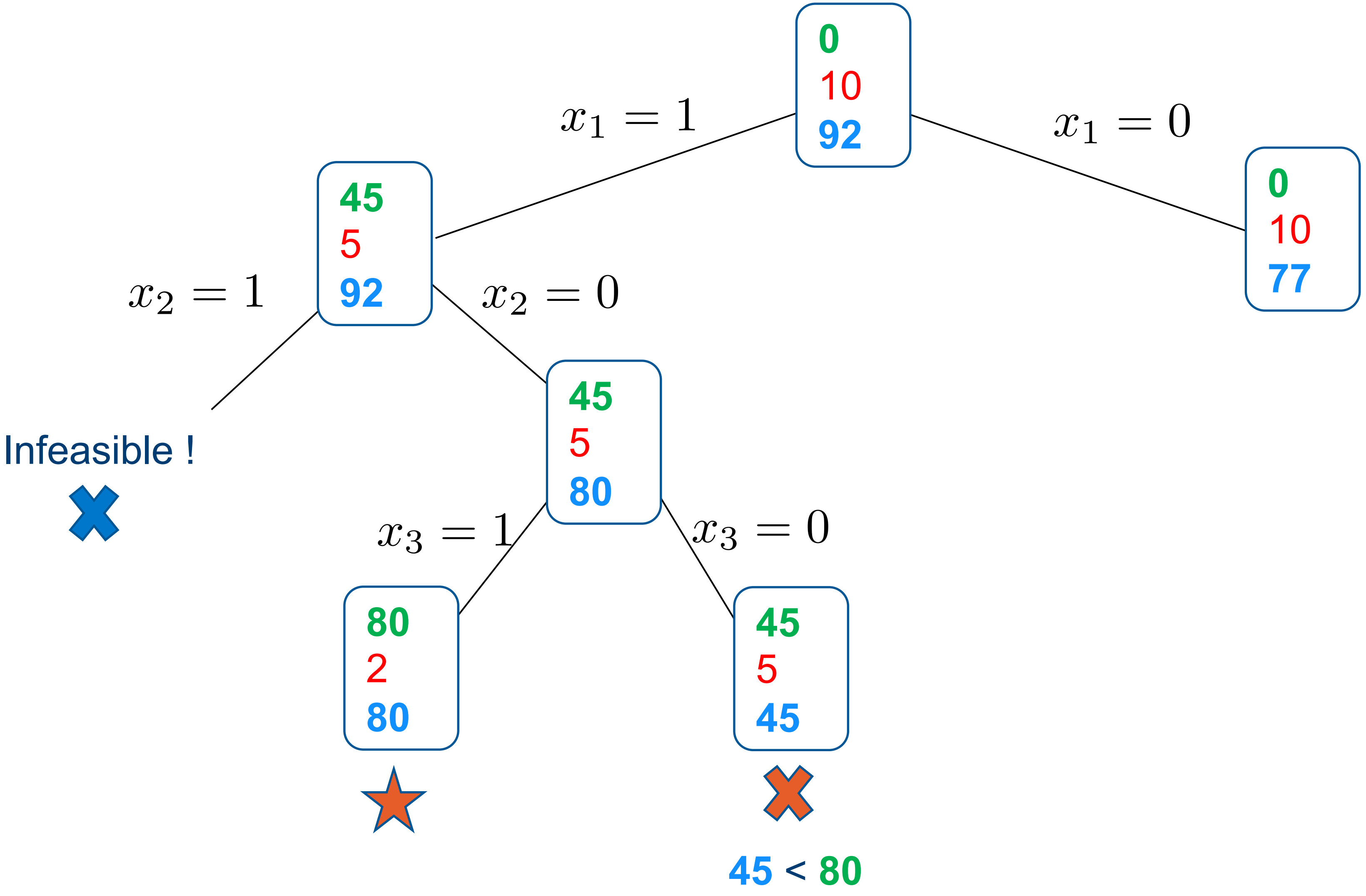


Heuristic upper bound : sum of value of remining items based on fractional knapsack

Example 2: Tighter upper bound

item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10

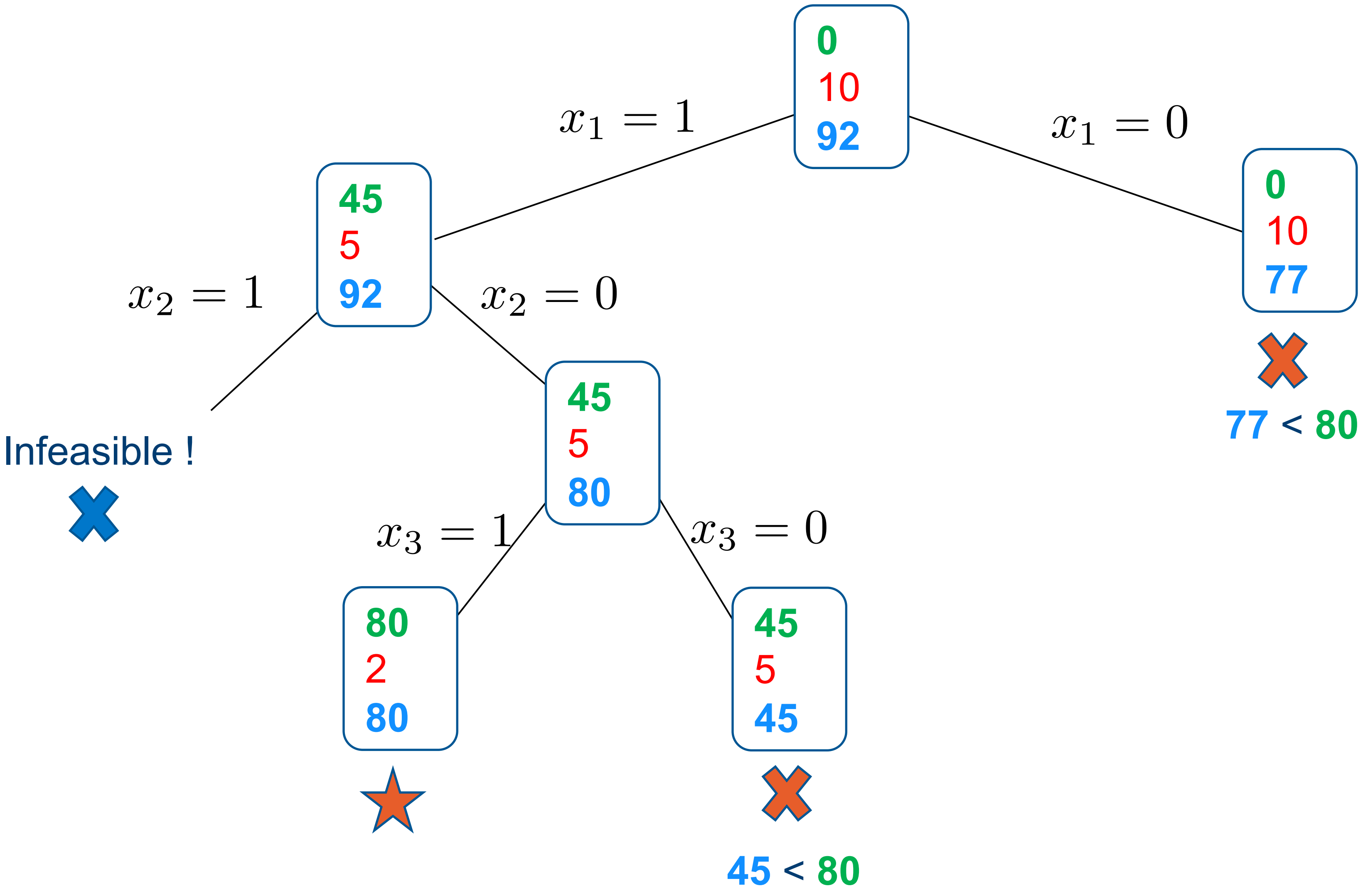


Heuristic upper bound : sum of value of remining items based on fractional knapsack

Example 2: Tighter upper bound

item	value	Weight
1	45	5
2	48	8
3	35	3

W = 10



Heuristic upper bound : sum of value of remining items based on fractional knapsack

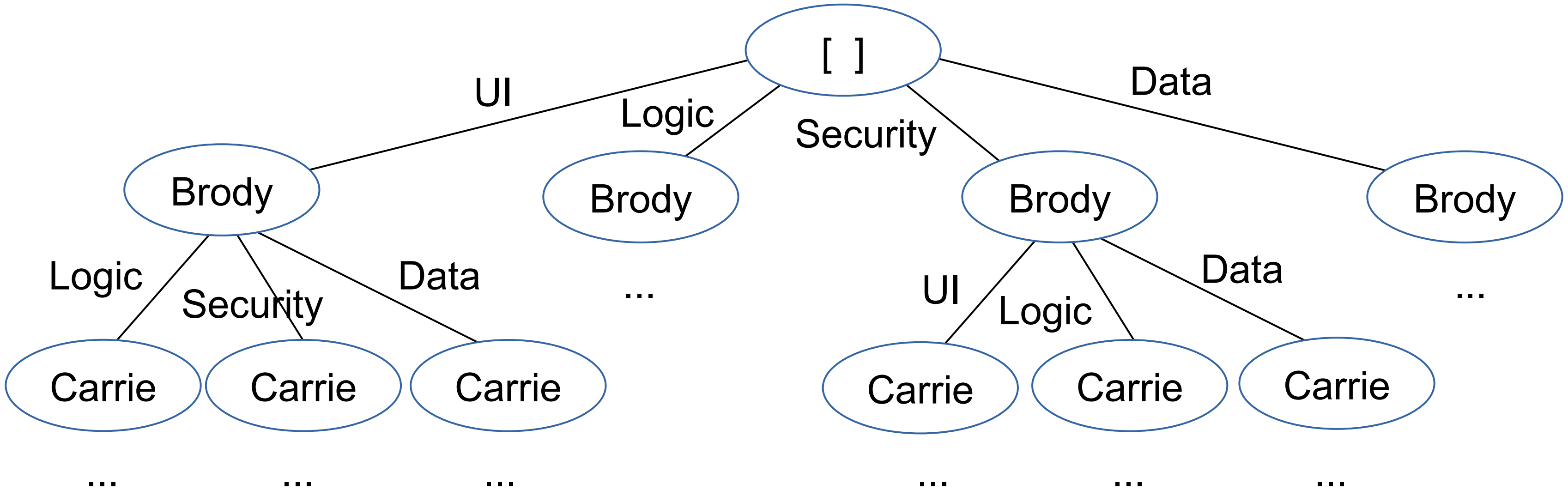
Case Study: Task Assignment

- We have a team of software developers and each (developer, task) combination has a cost.
  - The goal is to find an optimal assignment of tasks to developers to minimize work cost
- Example:

Developer	UI	Logic	Security	Data
Alexis	1	9	2	9
Brody	4	5	4	6
Carrie	6	5	5	9
Devon	7	9	2	3

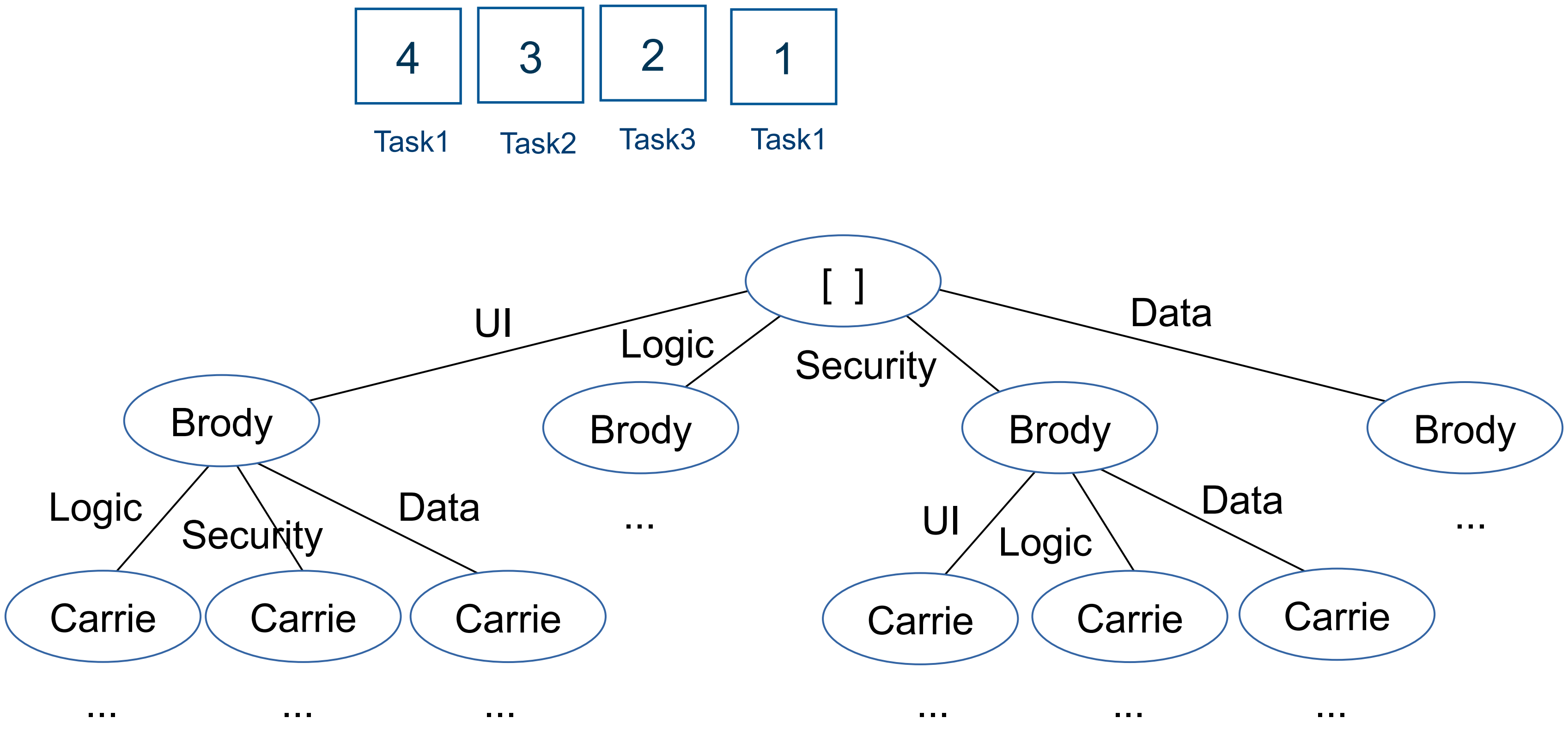
Brute Force Solution

- One way to solve this problem is to build a state space tree
  - Each node is a state, showing our assignment
  - Each relationship/edge is a decision
  - Each leaf node (final state) is one possible task assignment, so simply find the one with the lowest cost



Brute Force Solution

- The size of the state space tree is often problematic
  - In this case, there are 4! = 4 * 3 * 2 * 1 = 24 possible task assignments



Branch and Bound Solution

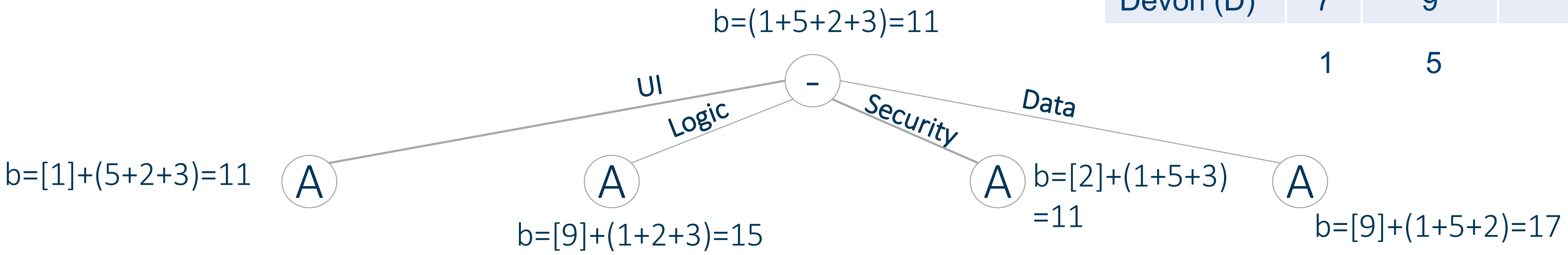
- We need to define a heuristic (lower bound) in order to prune the search space

- Lower bound heuristic:

  sum of minimum costs for each remaining task

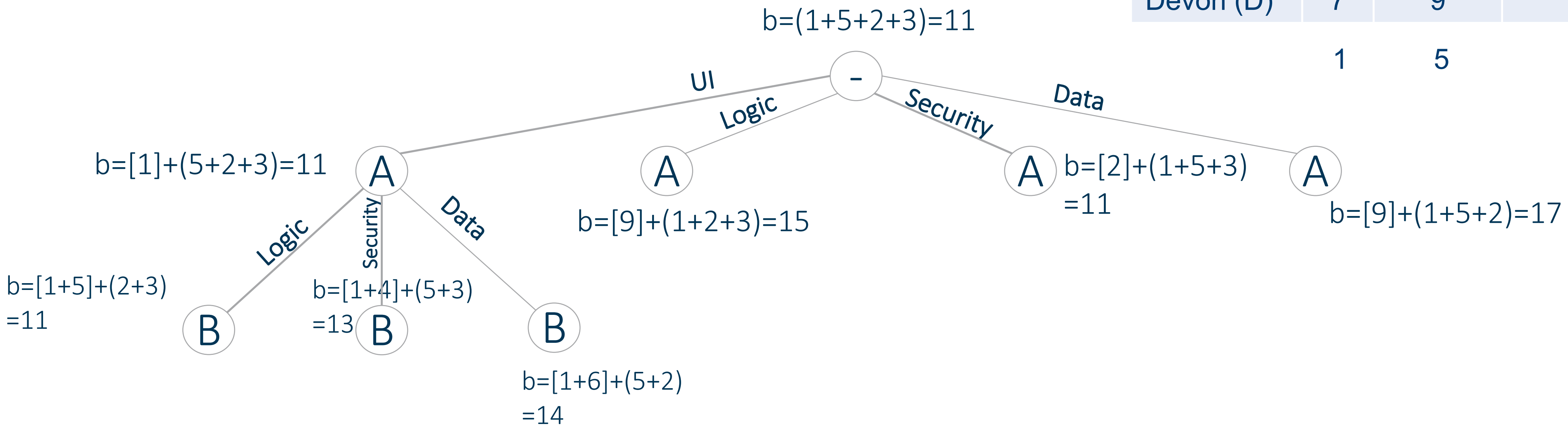
Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



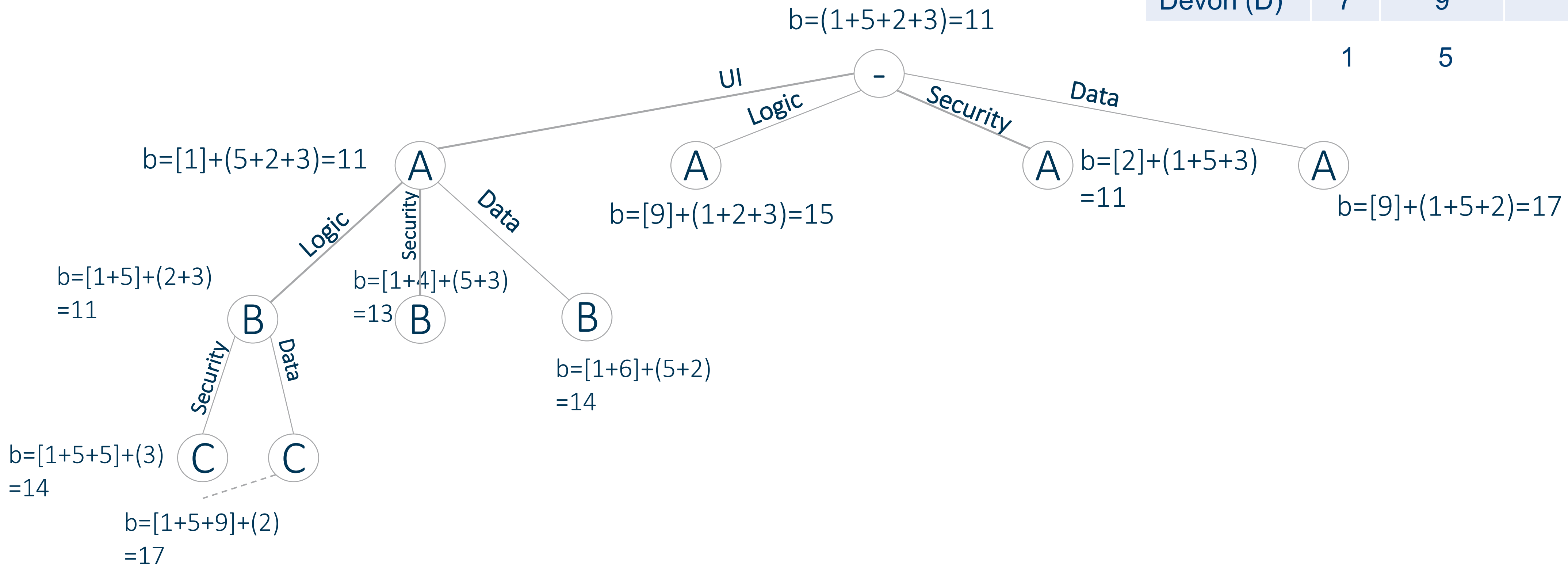
Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



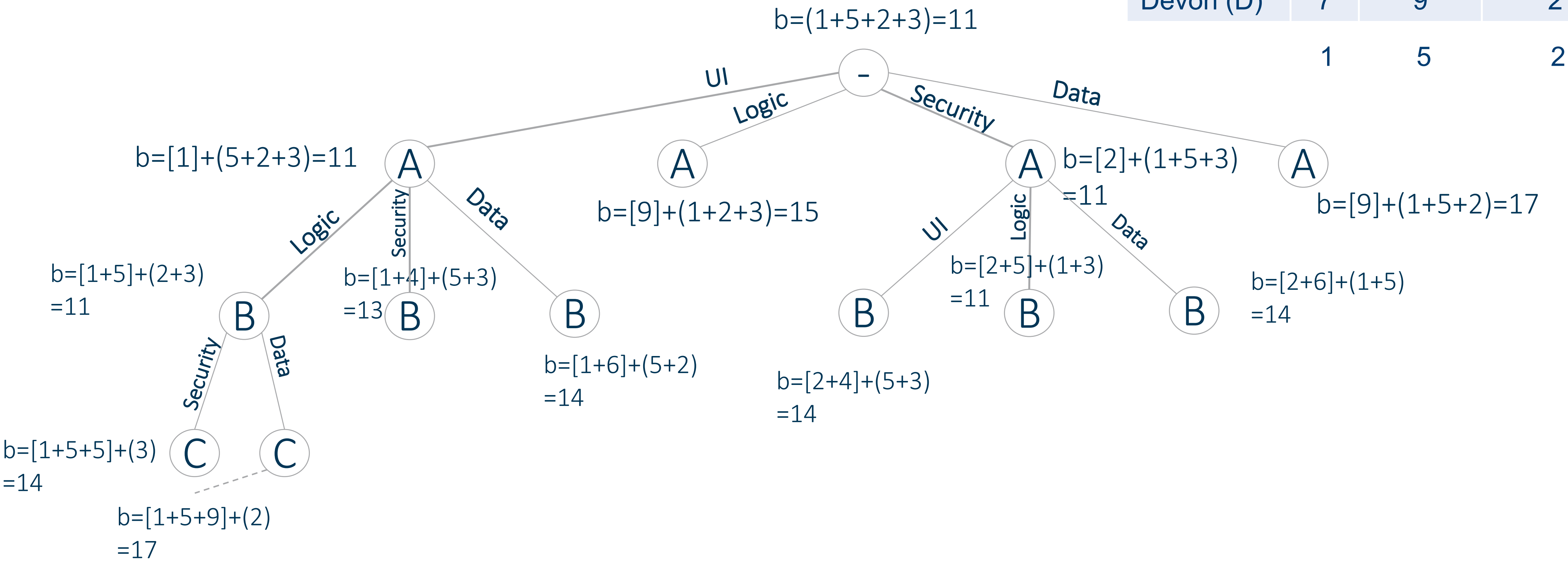
Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



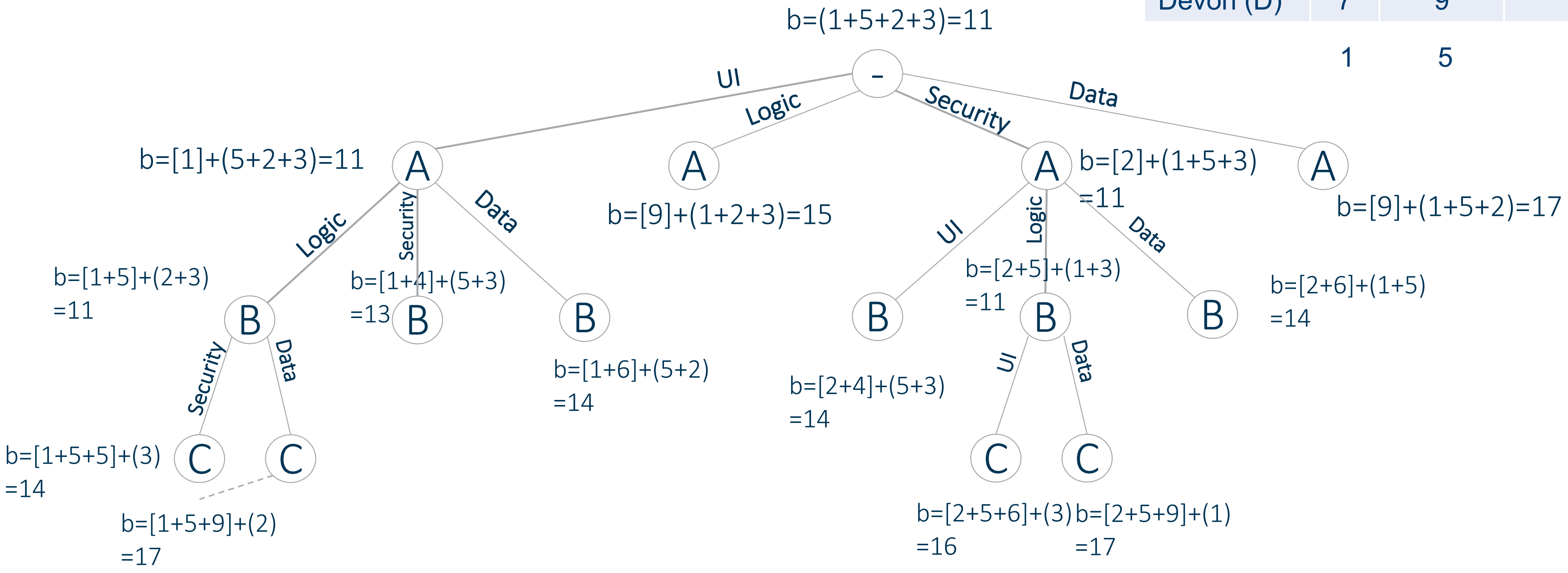
Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



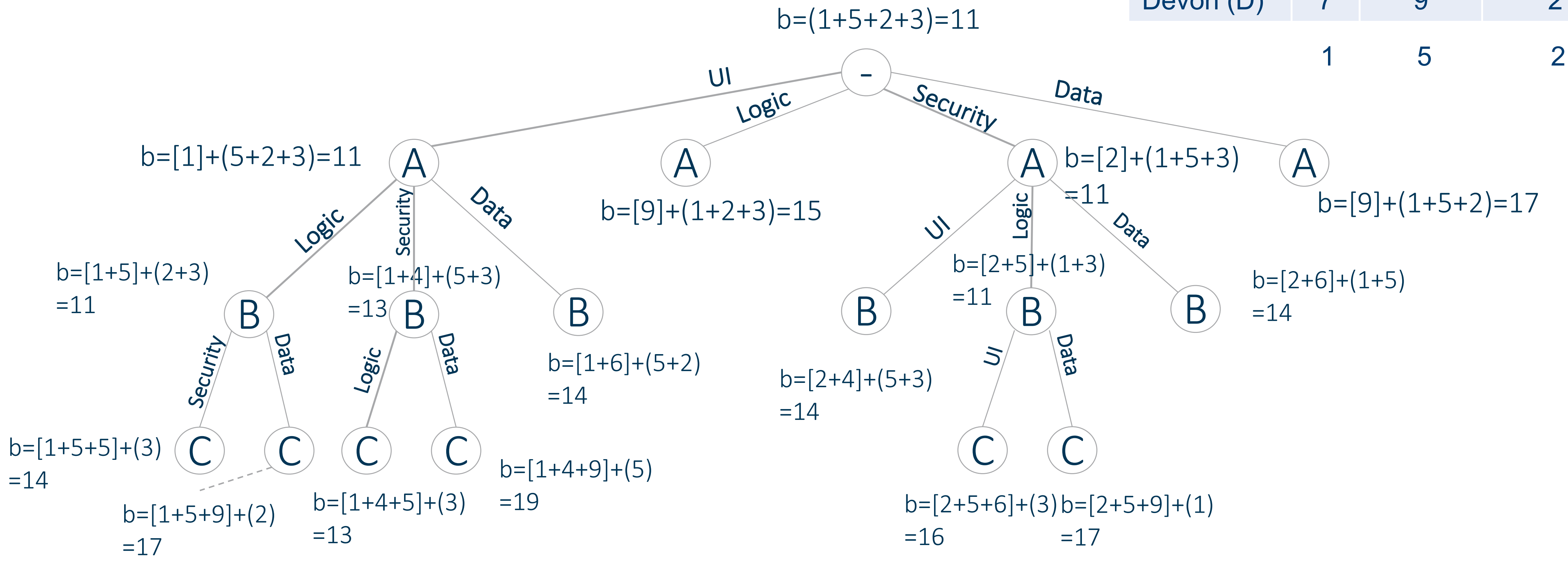
Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



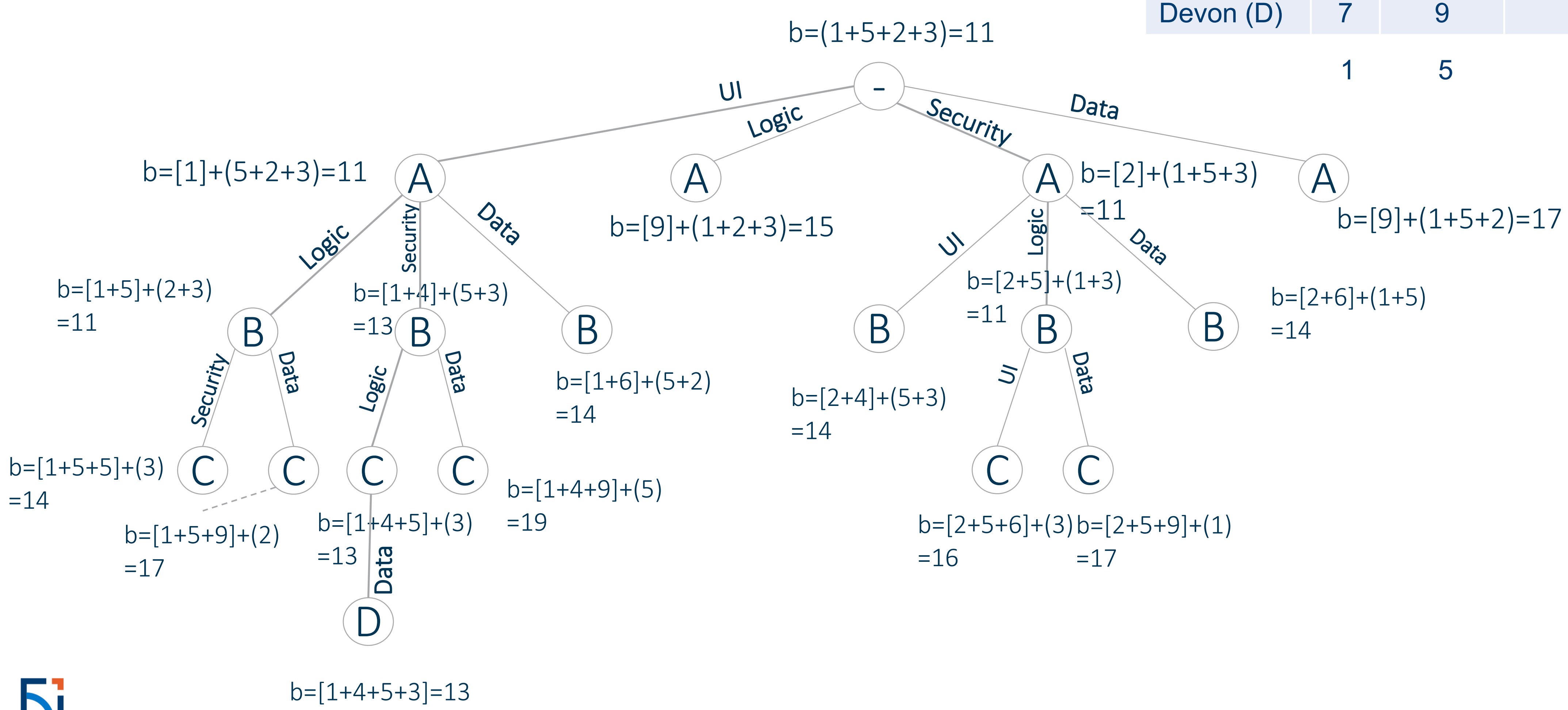
Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



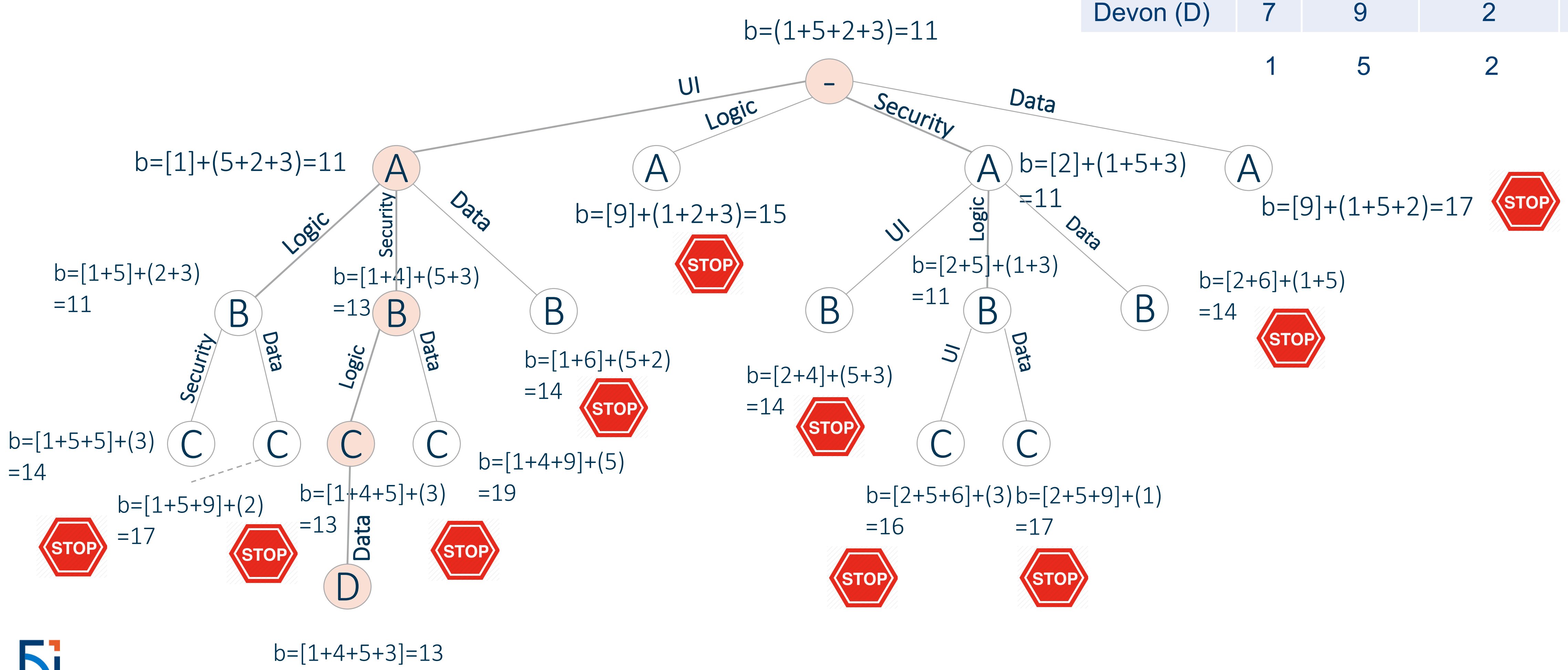
Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



Project Management - Solution

Developer	UI	Logic	Security	Data
Alexis (A)	1	9	2	9
Brody (B)	4	5	4	6
Carrie (C)	6	5	5	9
Devon (D)	7	9	2	3
	1	5	2	3



Practice: Tighter upper bound

item	value	Weight
1	8	2
2	24	4
3	10	5
4	18	6

$W = 12$

Heuristic upper bound : sum of value of remining items based on fractional knapsack

Practice: Project Management

Developer	UI	Logic	Security	Data
Alexis (A)	1	6	2	9
Brody (B)	4	5	3	6
Carrie (C)	6	7	5	9
Devon (D)	6	9	5	3

Wrap up

- Branch and Bound can help us prune the search space
- We learned branch and bound using
  - 0/1 knapsack
  - Project Management