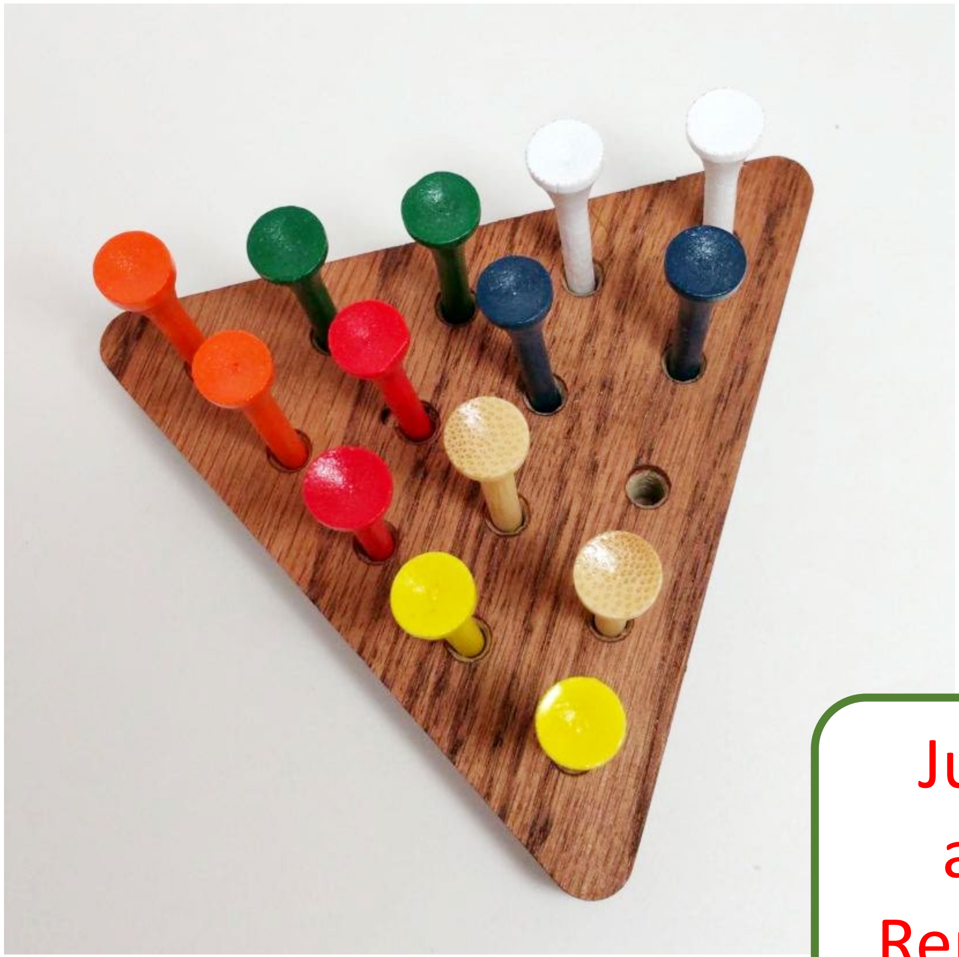
Lecture 11: Backtracking, Branch & Bound

Textbook: Chapter 12

Golf-tee puzzle

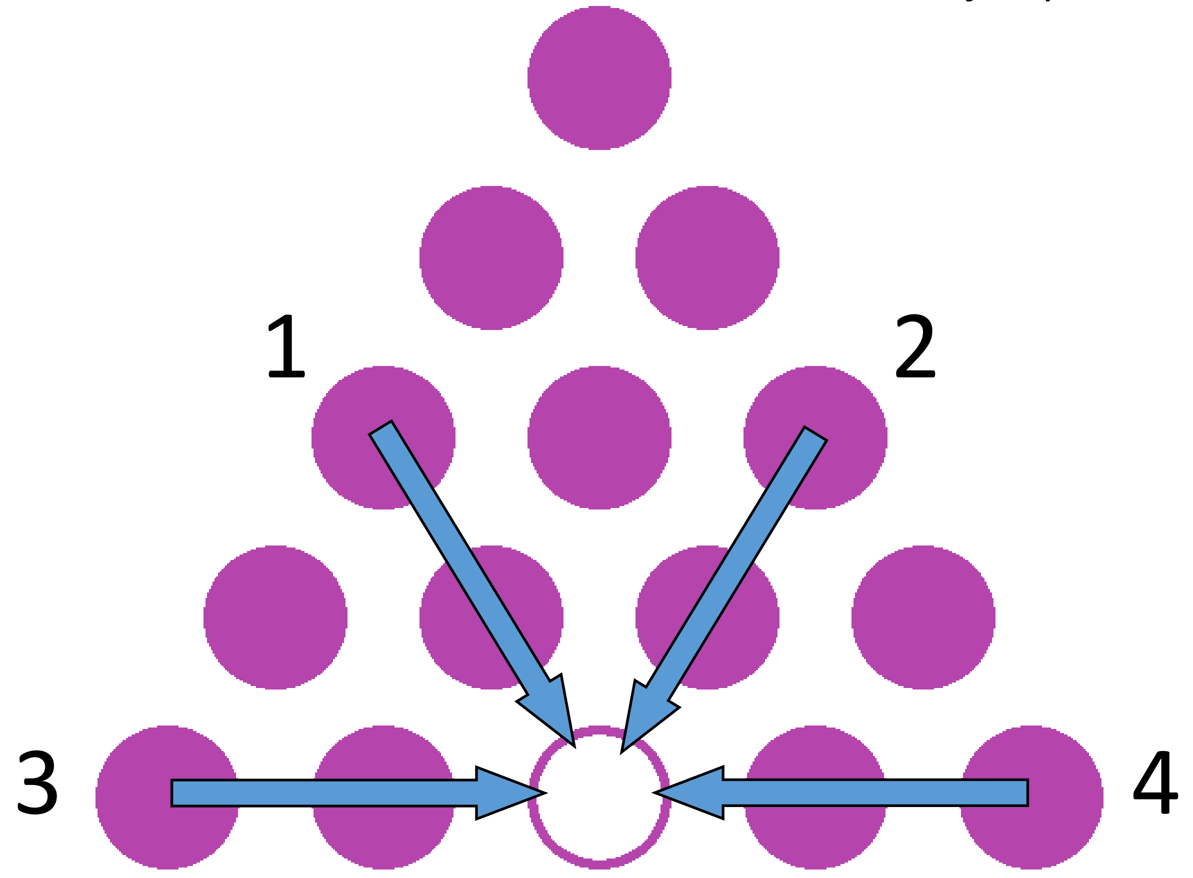


Jump and Remove

Valid moves

To win:
13 consecutive valid jumps; one peg left

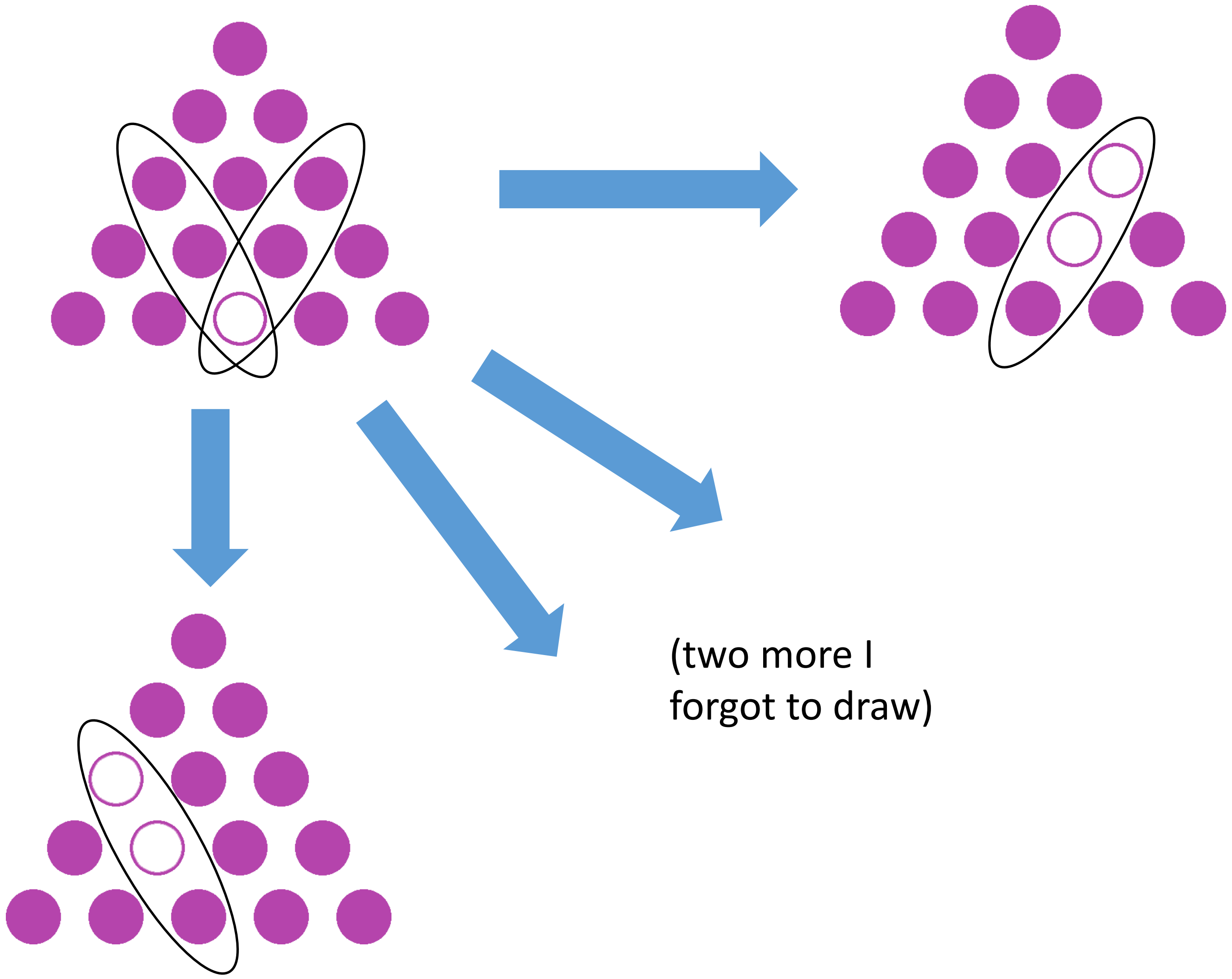
This position has four valid moves:



Backtracking

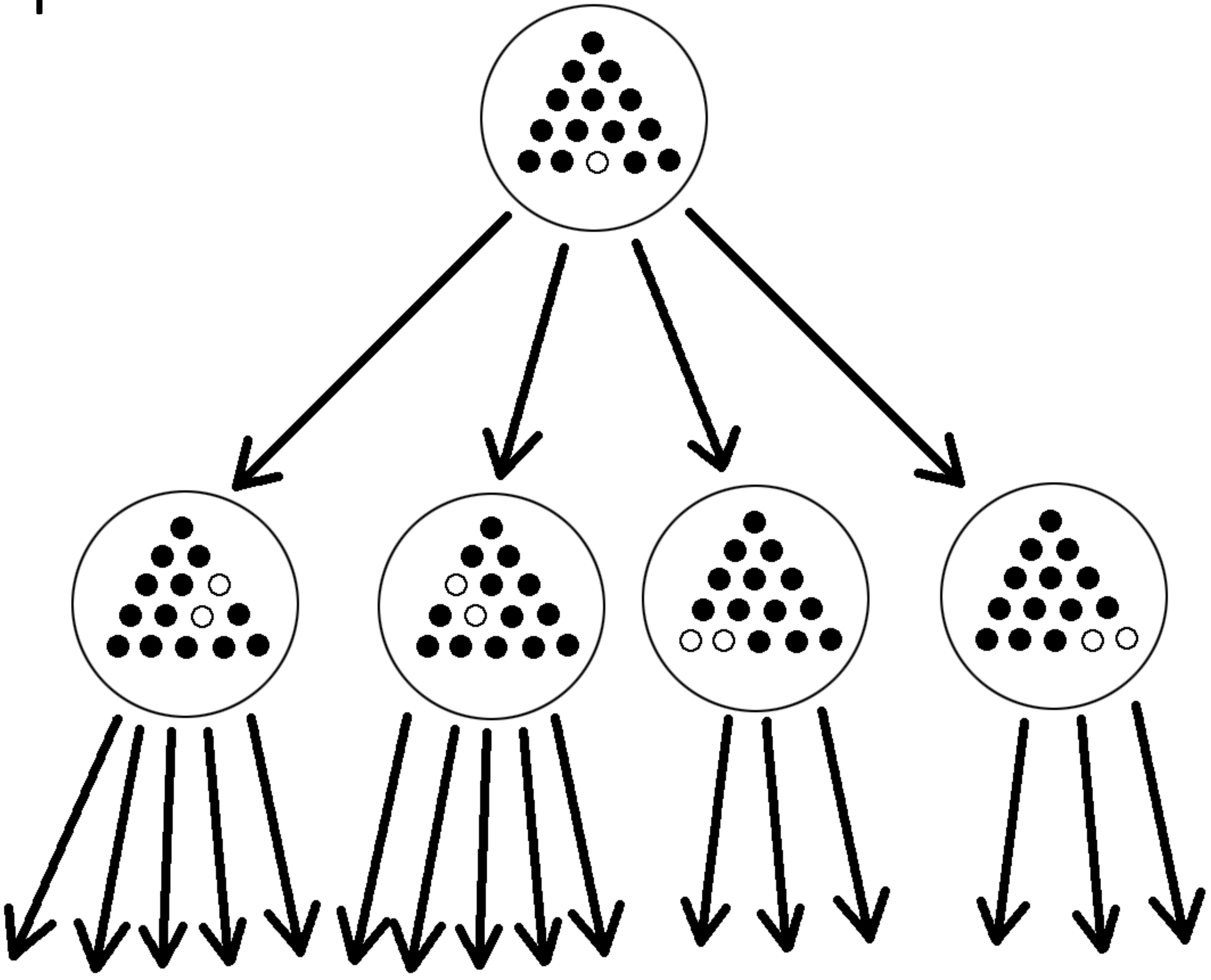
- Suppose you have to make a series of *decisions,* among various *choices,* where
  - You don’t have enough information to know what to choose
  - Each decision leads to a new set of choices
  - Some sequence of choices (possibly more than one) may be a solution to your problem
- Backtracking is a systematic way of trying out various sequences of decisions, until you find one that “works”

Changing state



(two more I
forgot to draw)

State-space tree



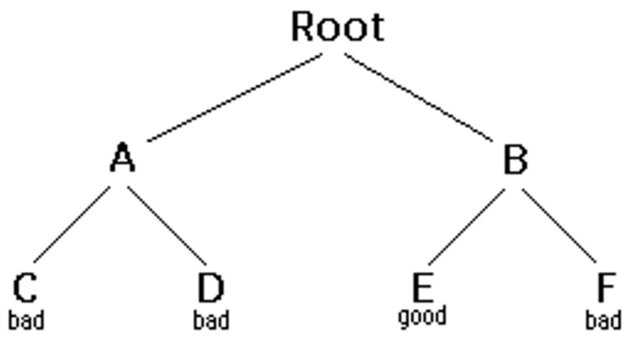
So many more places to go from here!

Backtracking in words

- IDEA:
    - Construct solutions one component at a time
    - If a partial solution can be developed further without violating constraints:
        - Choose first legitimate option for the next component
    - If there is *no option* for the next component
        - Backtrack to replace the last component of partial solution

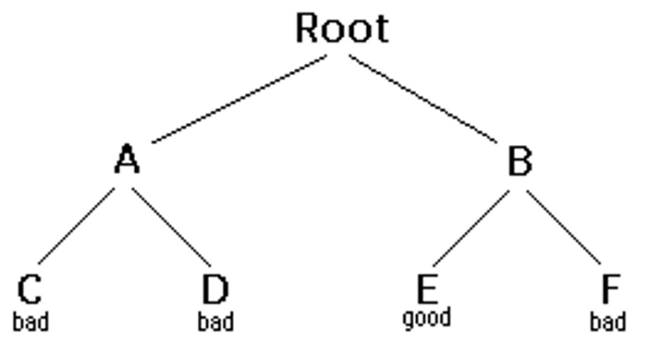
Backtracking

- Think of the solutions as being organized in a tree
  - Each node represents the “state” at one stage of the solution
  - The root represents initial state before the search begins
  - Nodes at first level represent first choice
  - Second level ... second choice ... etc.



Backtracking – Abstract Example

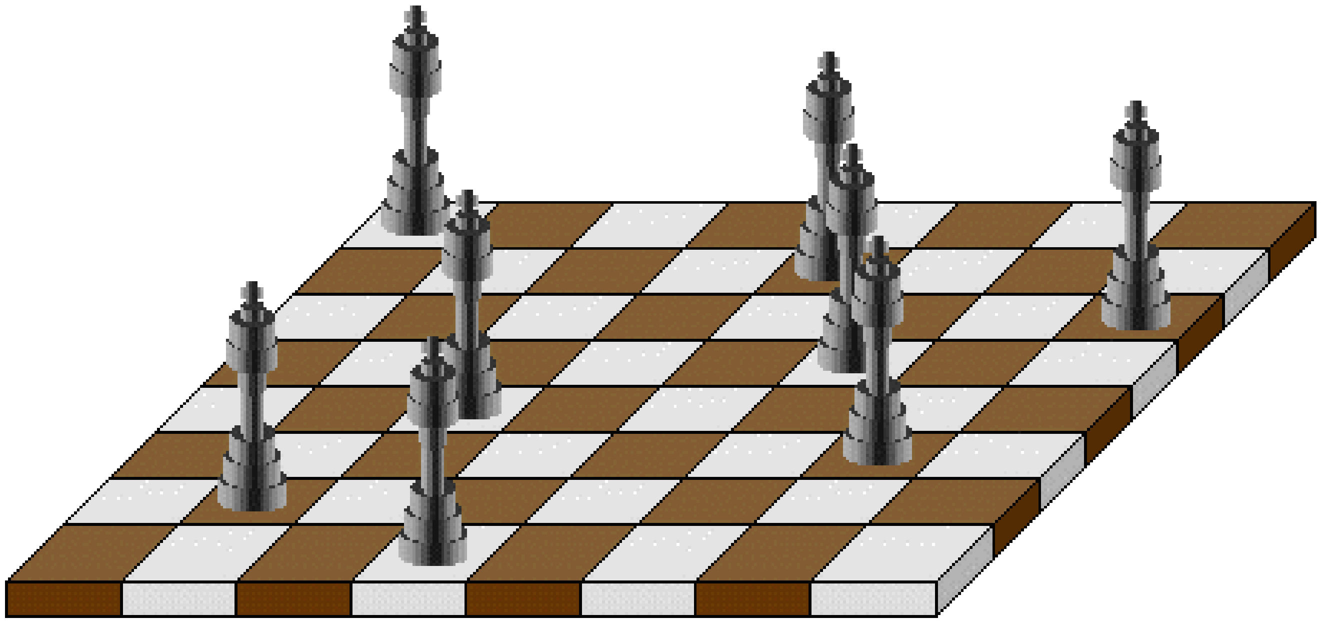
- Starting at Root, your options are A and B. You choose A.
- At A, your options are C and D. You choose C.
- C is bad. Go back to A.
- At A, you have already tried C, and it failed. Try D.
- D is bad. Go back to A.
- At A, you have no options left to try. Go back to Root.
- At Root, you have already tried A. Try B.
- At B, your options are E and F. Try E.
- E is good. Congratulations!



The tree used to build solutions is called the *state-space tree*
The nodes are *partial solutions*
The edges are *choices*

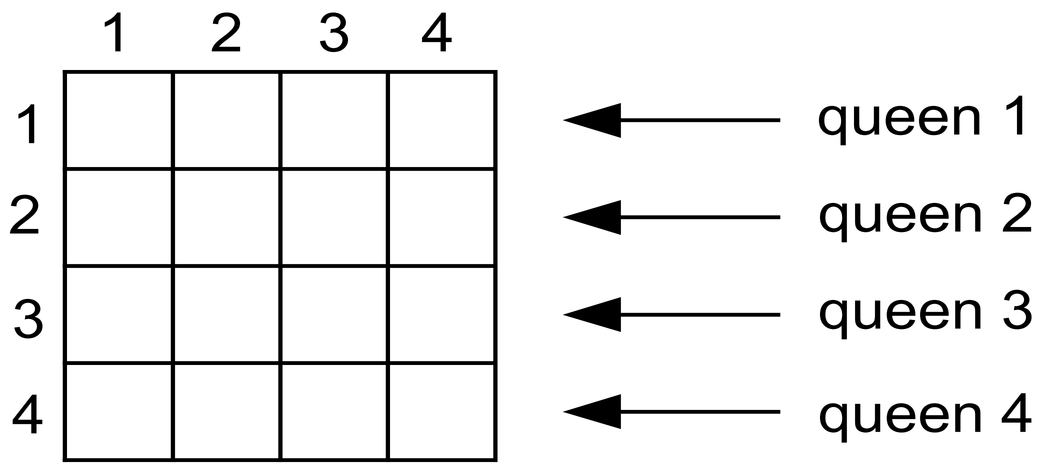
Example: n-Queens Problem

- Place *n* queens on an *n*-by-*n* chess board so that no two are in the same row, column or diagonal
  - i.e. no queens are attacking each other



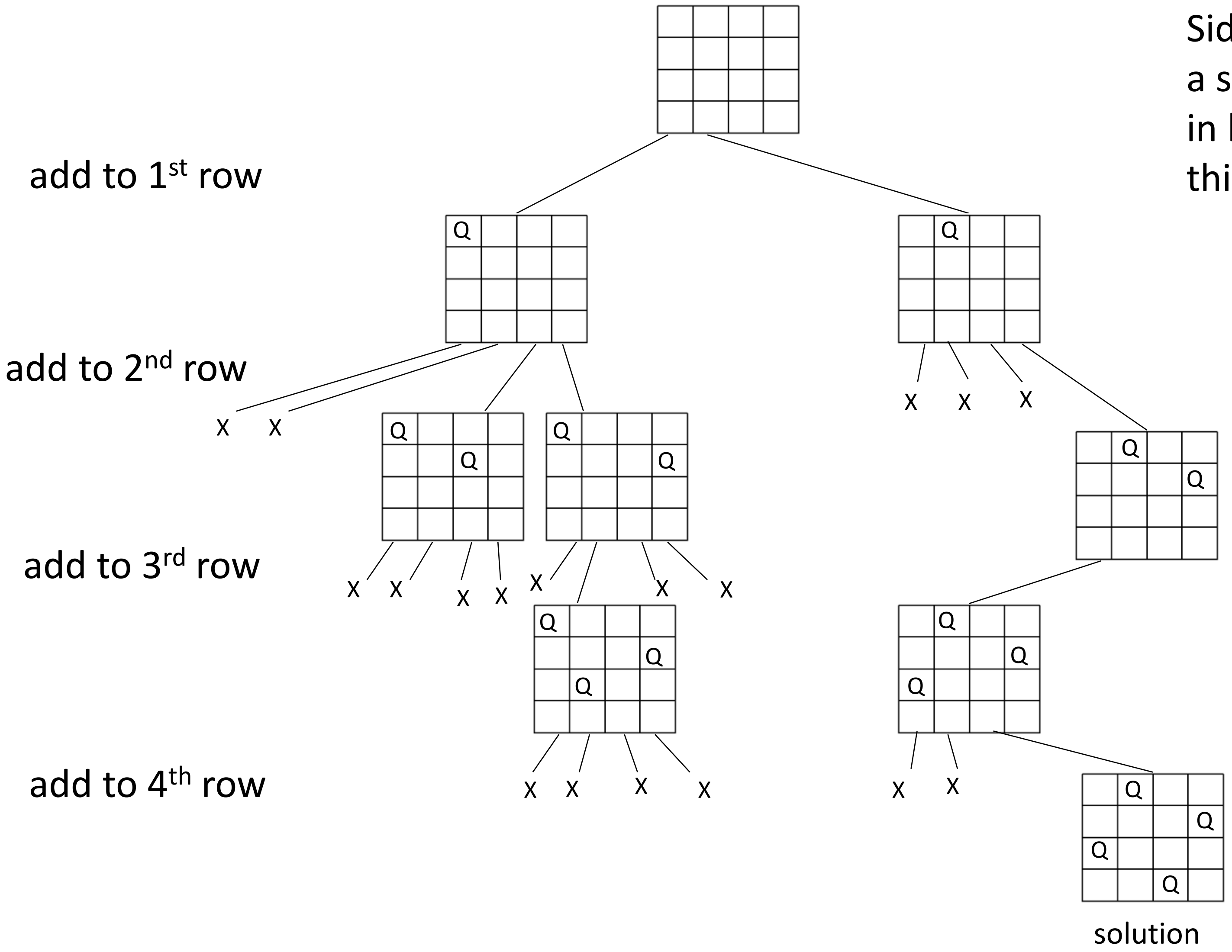
Example: 4-Queens

- n=4



- We can solve it by backtracking
  - Root is empty board
  - At step i (level i)… put a queen in row i

State-Space Tree of 4-Queens

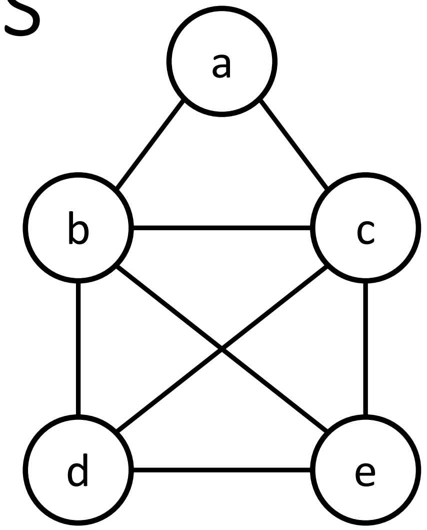


Side note: for any *n>3*, a solution can be found in linear time (not with this algorithm)

Takeaways from the N- queens demonstration

- Moving around in a DFS-like way through the State Space Tree
- This is the essence of a backtracking algorithm
- Proceed to the next possible choice; examine the choice; if "promising", we continue; if "non-promising", we backtrack (go back up the tree)
- At each LEVEL of the tree we have partial solutions of increasing sizes -- growing towards a complete solution
- LEAVES of the tree can be dead ends, or (if they get far enough down the tree) SOLUTIONS

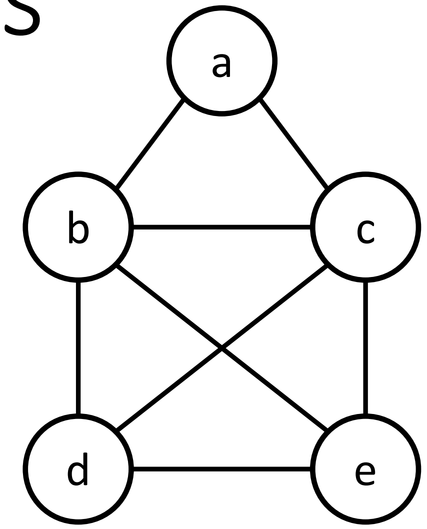
Example: Hamiltonian cycles



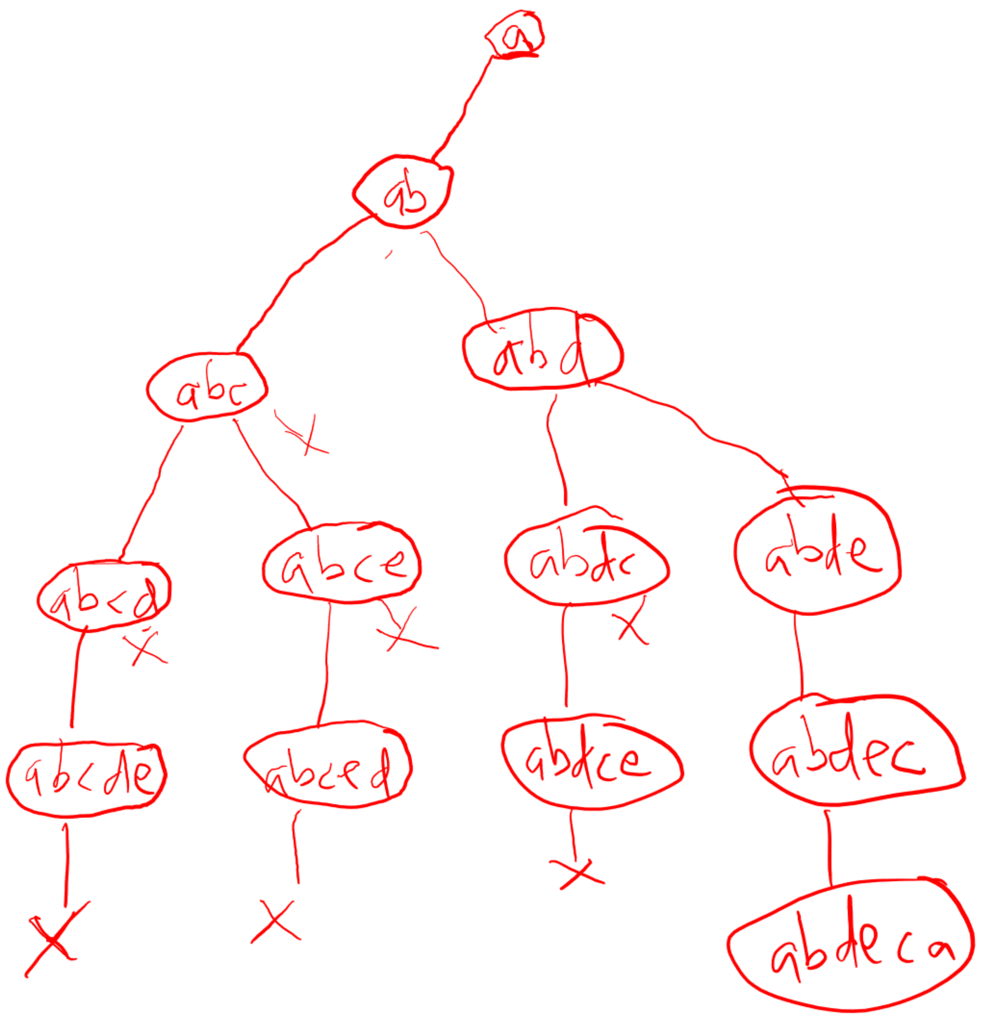
- Start at any vertex
- Successively build a path
- At each "level", try adding each remaining neighbor
- Backtrack at dead ends

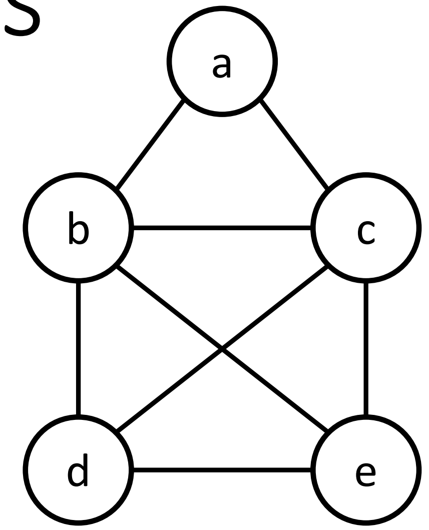
- What is the state space?

Example: Hamiltonian cycles



Example: Hamiltonian cycles





Branch and Bound

Branch and Bound

- The idea:

Set up a **bounding function**, which is used to compute a **bound** (for the value of the objective function) **at a node** on a state-space tree and determine **if it is promising**

  - **Promising** (if the bound is better than the value of the best solution so far): expand beyond the node.
  - **Non-promising** (if the bound is no better than the value of the best solution so far): do not expand beyond the node (pruning the state-space tree).

Assignment problem

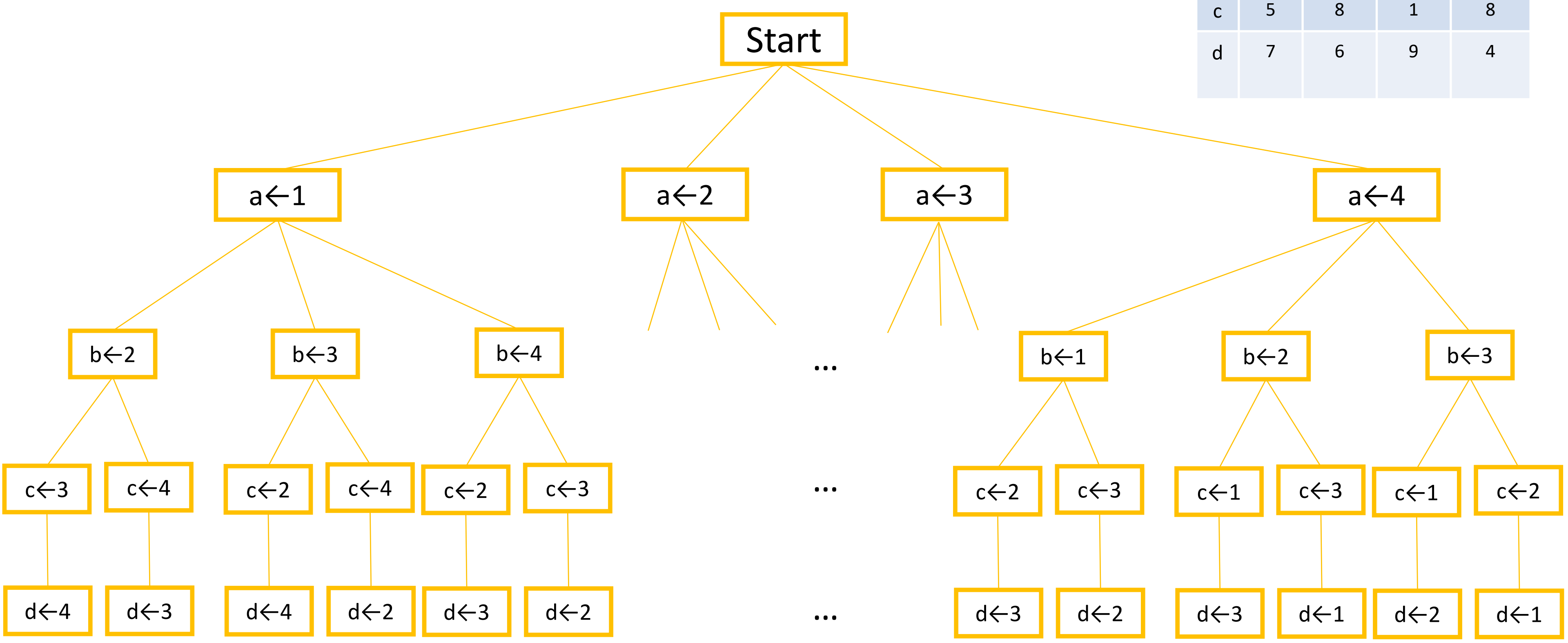
Select one element in each row of the cost matrix C so that:

- no two selected elements are in the same column
- the sum is minimized

	Job 1	Job 2	Job 3	Job 4
Person a	9	2	7	8
Person b	6	4	3	7
Person c	5	8	1	8
Person d	7	6	9	4

Assignment Problem (Brute Force)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

Lower bound: Any solution to this problem will have total cost at least 10

lb = 2+3+1+4 =10

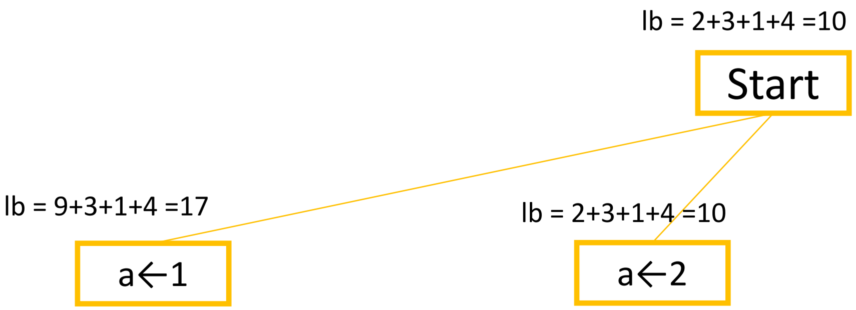
Start

Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

$$\begin{tikzcd}
 & \overset{lb = 2+3+1+4 = 10}{\boxed{\text{Start}}} \arrow[dl, dash] \\
\underset{}{\overset{lb = 9+3+1+4 = 17}{\boxed{a \leftarrow 1}}} &
\end{tikzcd}$$

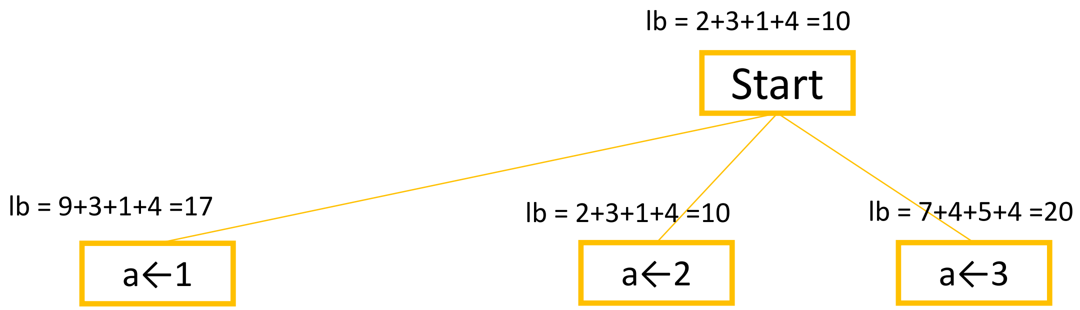
Assignment Problem (Branch & Bound)



	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

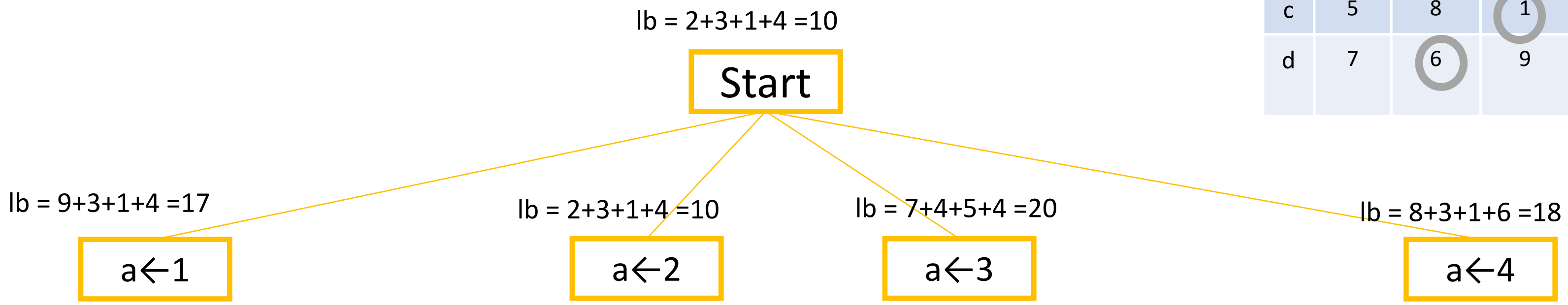
Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



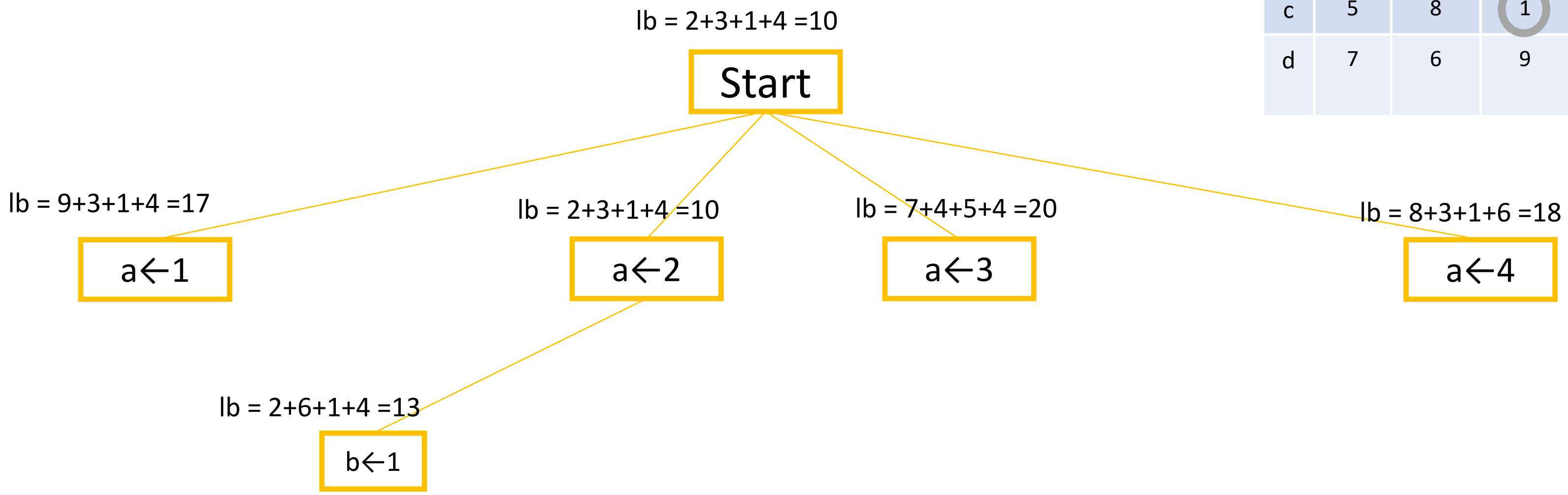
Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



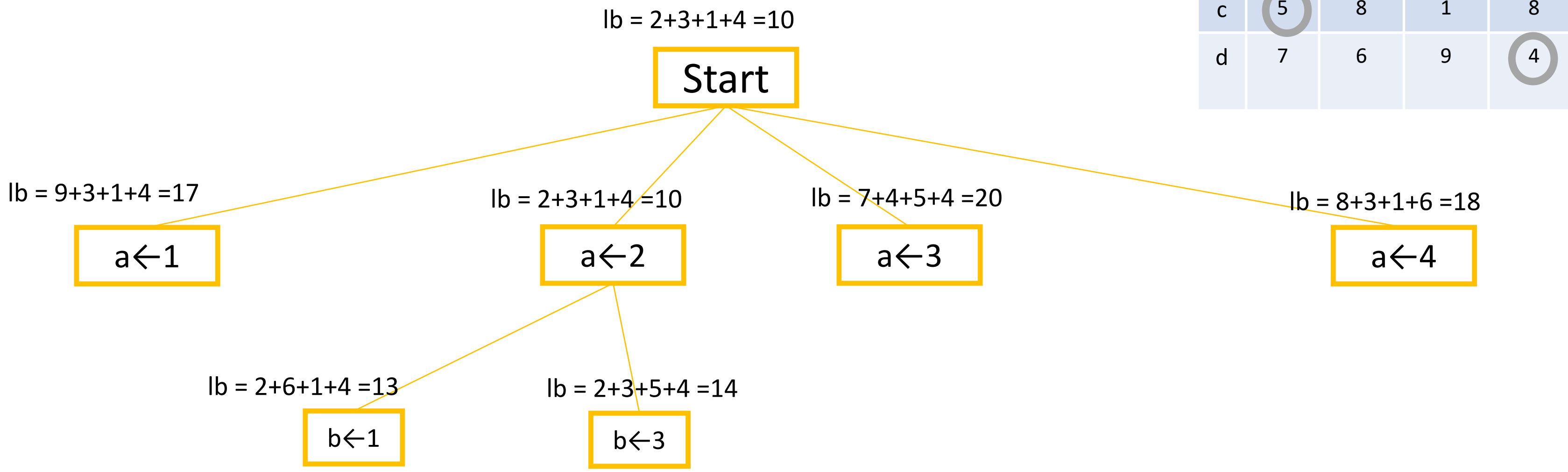
Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



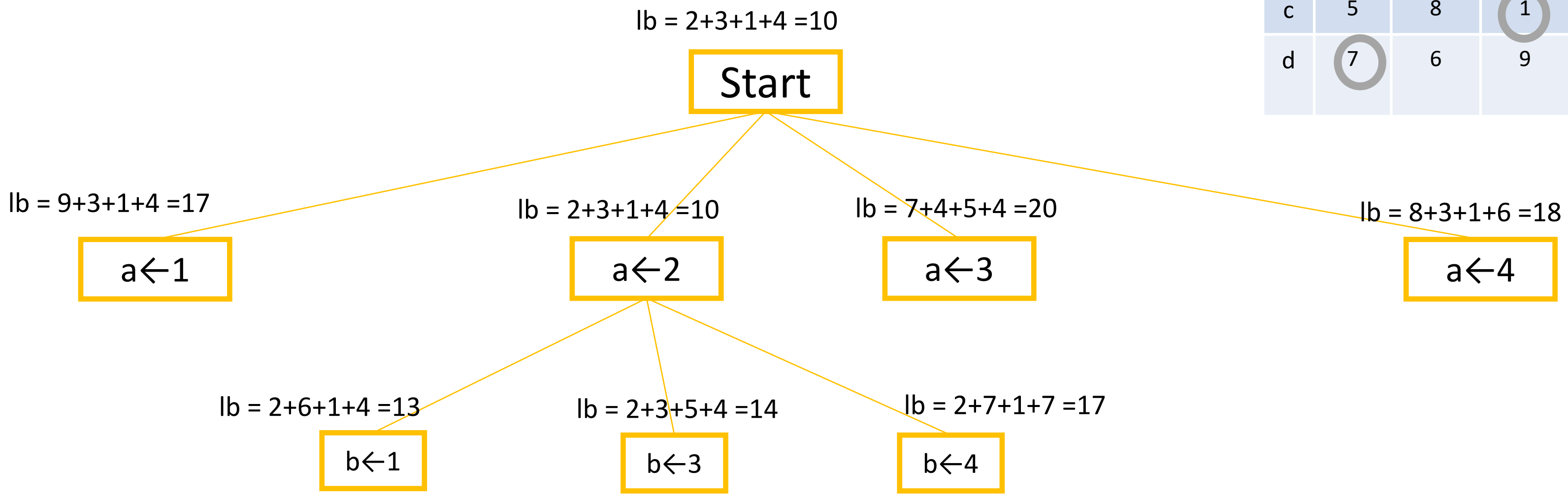
Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



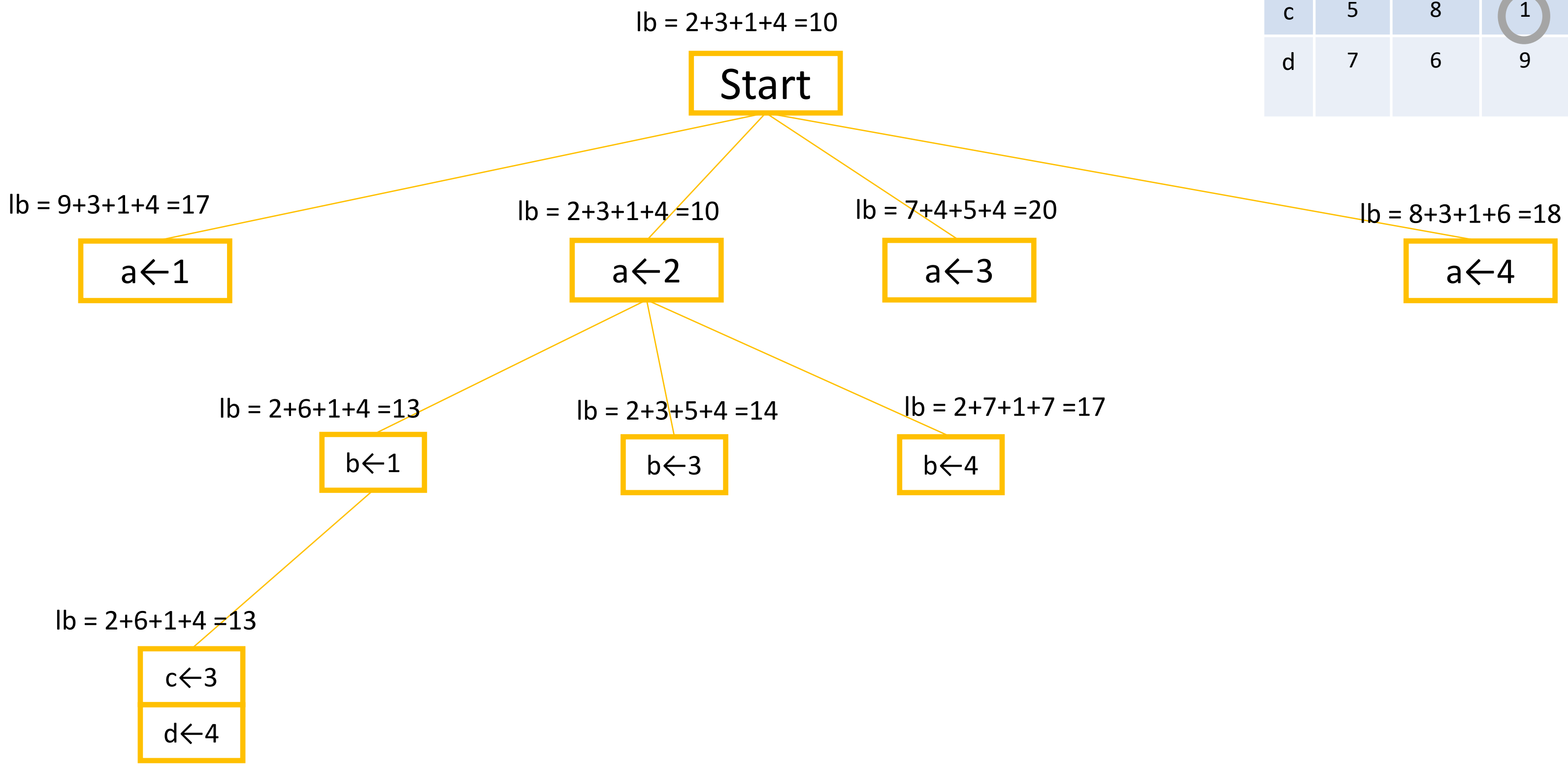
Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



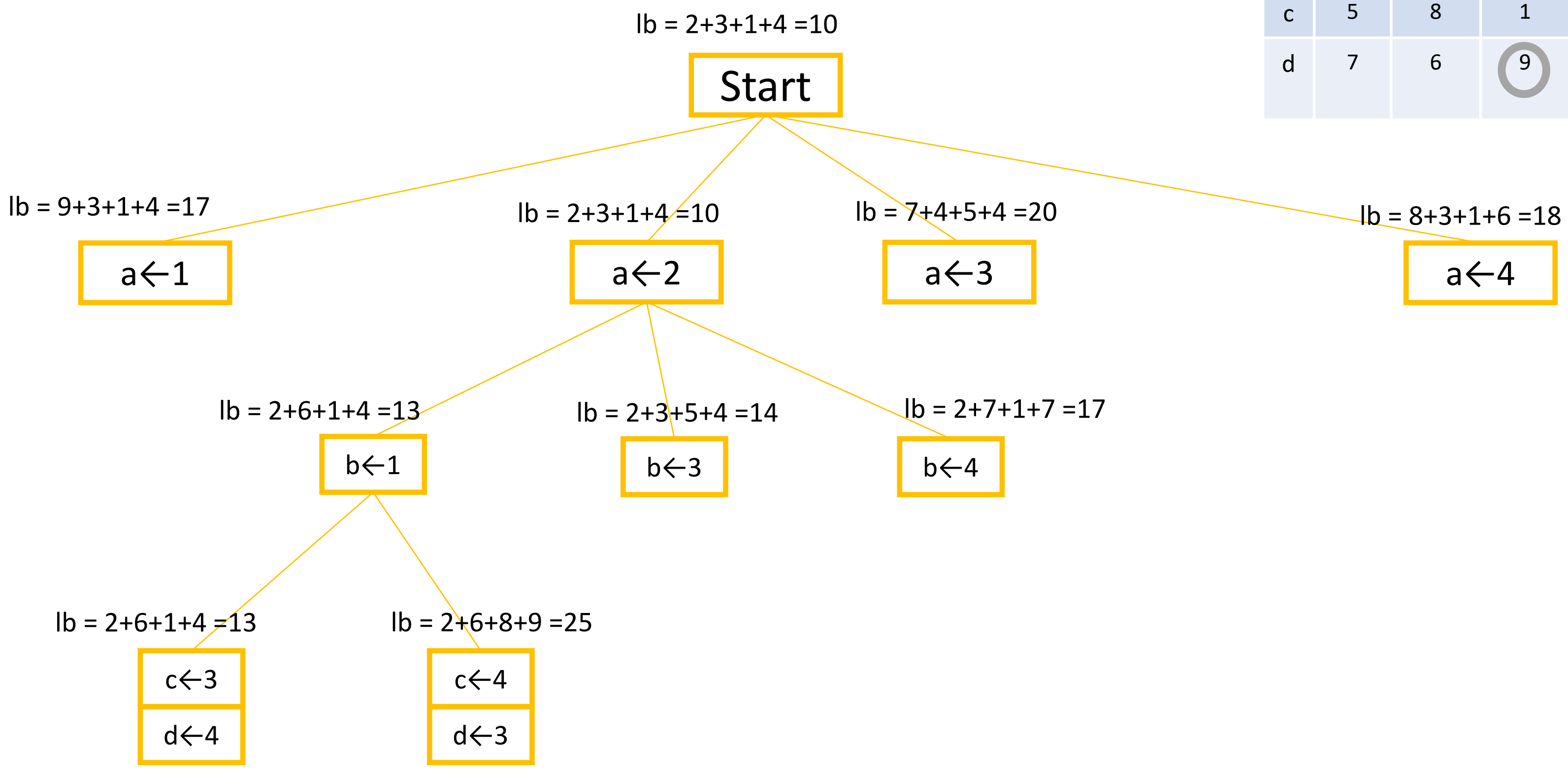
Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



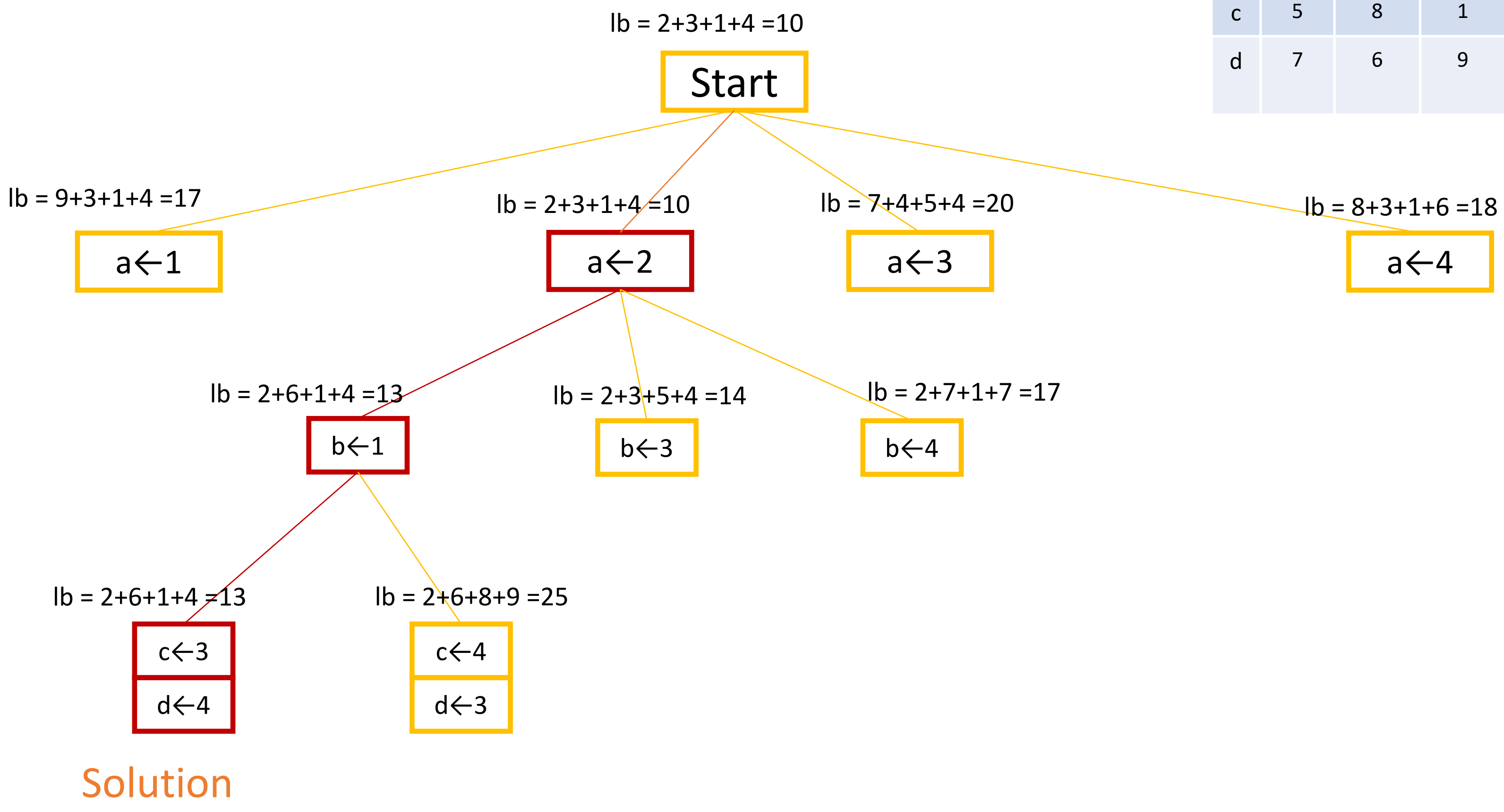
Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



Assignment Problem (Branch & Bound)

	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



Branch & Bound Example 2

	Job 1	Job 2	Job 3	Job 4
A	6	4	5	9
B	8	1	4	6
C	9	2	1	1
D	6	1	7	3

Branch & Bound Example 2

	Job 1	Job 2	Job 3	Job 4
A	6	4	5	9
B	8	1	4	6
C	9	2	1	1
D	6	1	7	3

Branch & Bound Example 2

	Job 1	Job 2	Job 3	Job 4
A	6	4	5	9
B	8	1	4	6
C	9	2	1	1
D	6	1	7	3