Announcements

- No labs next week!
- Farnaz's lab students:
  - She will be available in her office (SW2-363) during lab times (and other times – see Learning Hub) and by email
- My lab students:
  - I'll be in DTC Thursday 5 Dec and BBY Friday 6 Dec; email me if you wish to meet

More announcements

- We do have LECTURE next week
  - A little more material
  - Last minute final-exam review
- Final exam is Monday 9 Dec, 8am

- I'll be posting some review materials soon – watch Learning Hub

Backtracking and Branch & Bound

(Chapter 12)

Backtracking and Branch & Bound

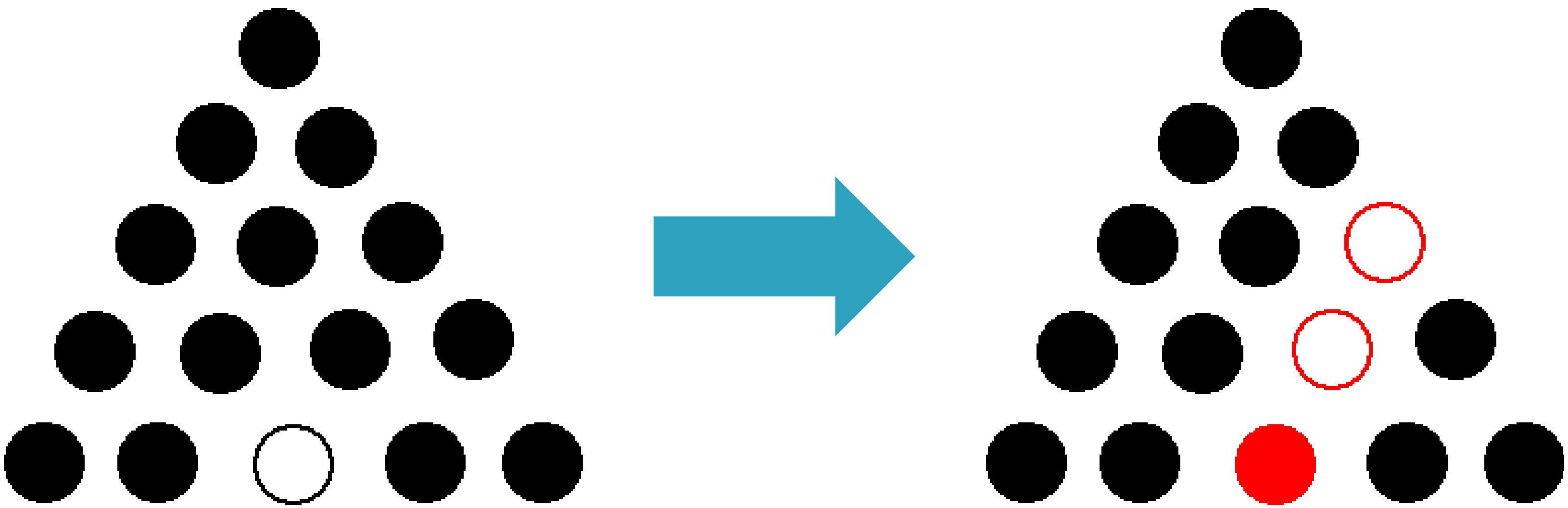
- Backtracking
  - n-Queens problem
- Branch & Bound
  - Assignment problem

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 methodical way of trying out various sequences of decisions, until you find one that "works"

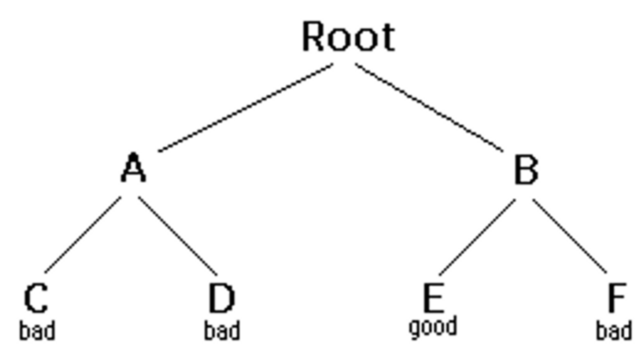
Golf-tee puzzle

Valid move

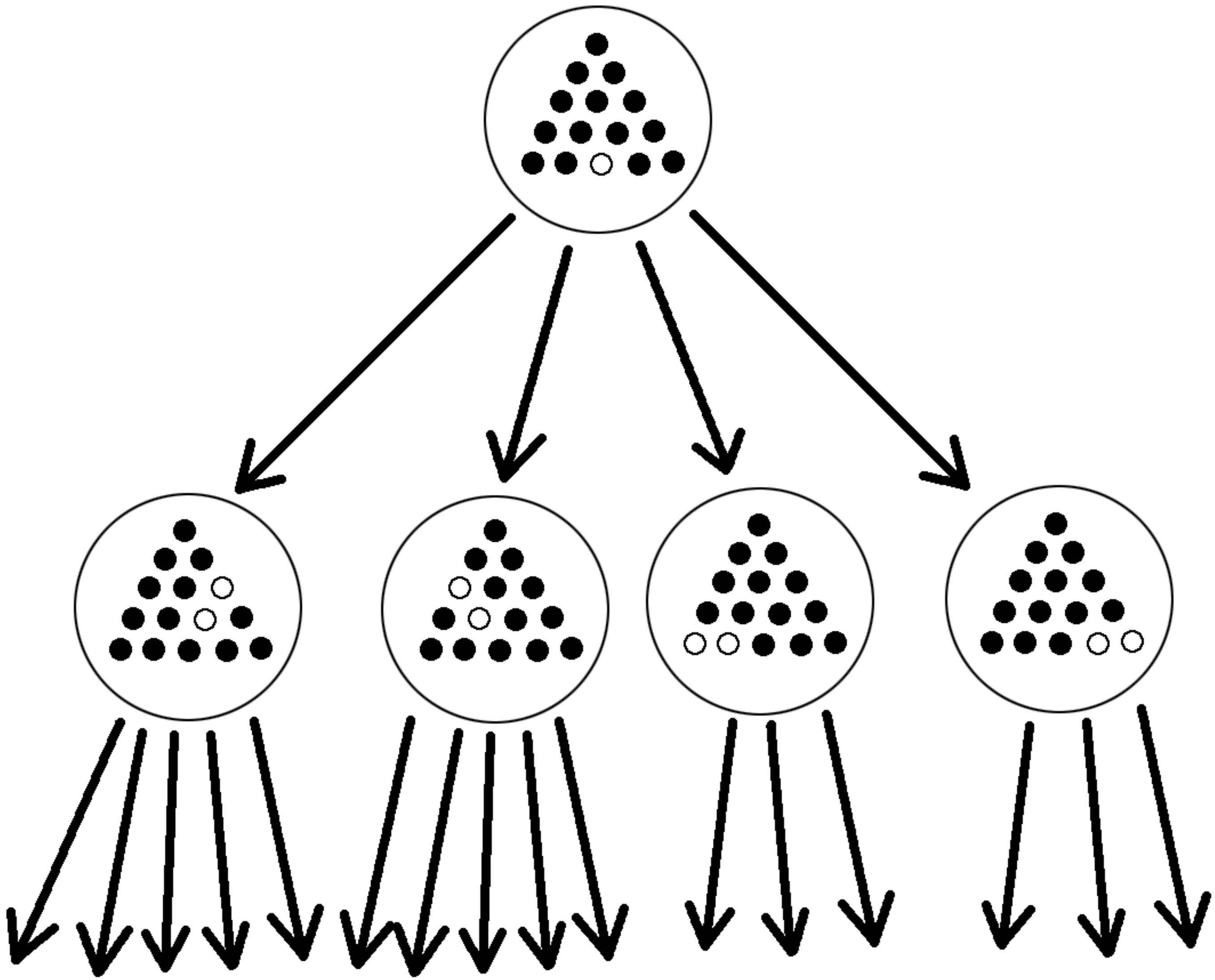


Backtracking

- Think of the solutions as being organized in a tree
  - The root represents initial state before the search begins
  - Nodes at first level represent first choice
  - Second… second choice..etc



State-space tree

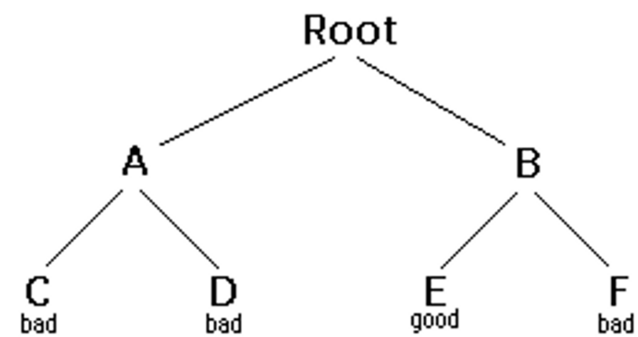


Backtracking in words

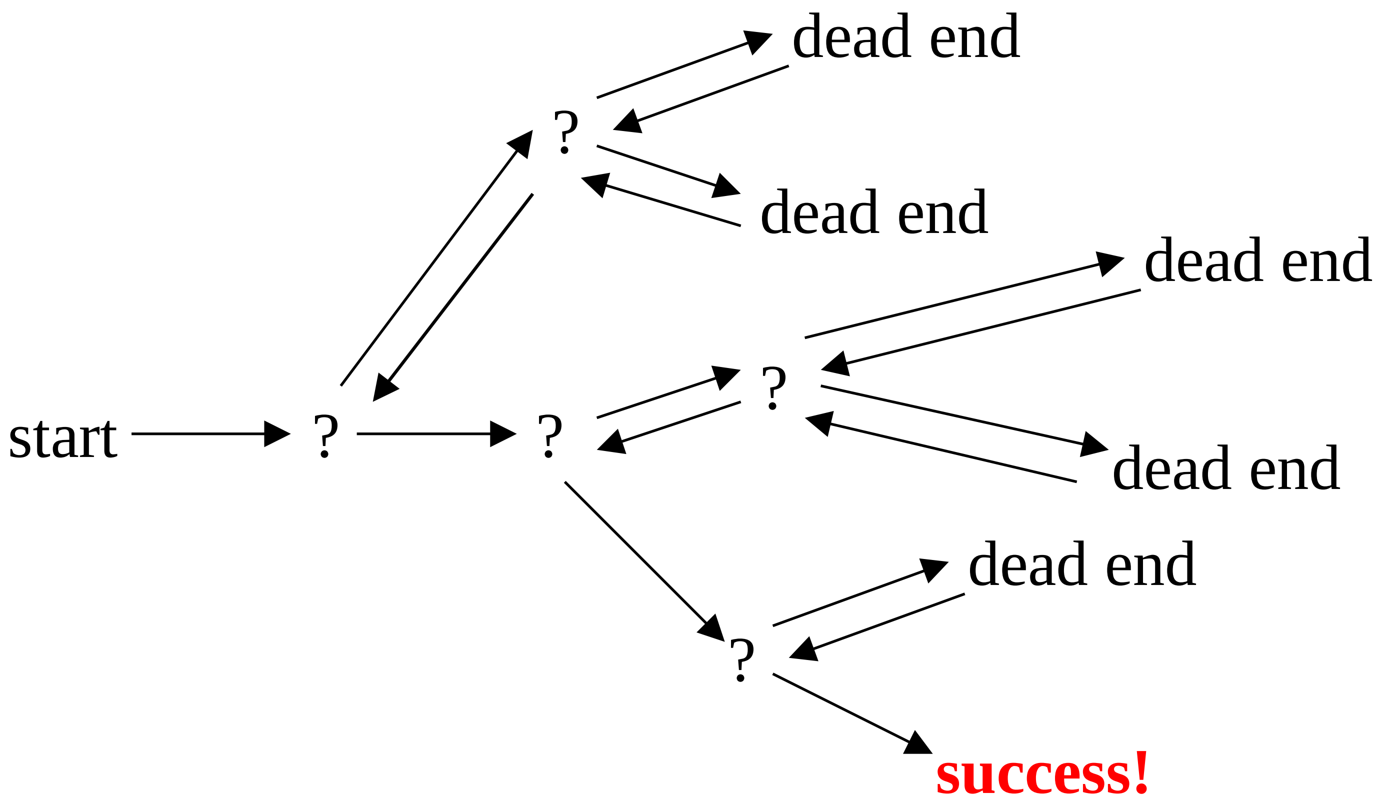
- <u>IDEA:</u>
  - Construct solutions one component at a time
  - If a partial solution can be developed further without violating constraints:
    - Choose first legitimate option for the next component
  - If there is *no option* for the next component
    - Backtrack to replace the last component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

Backtracking – 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!



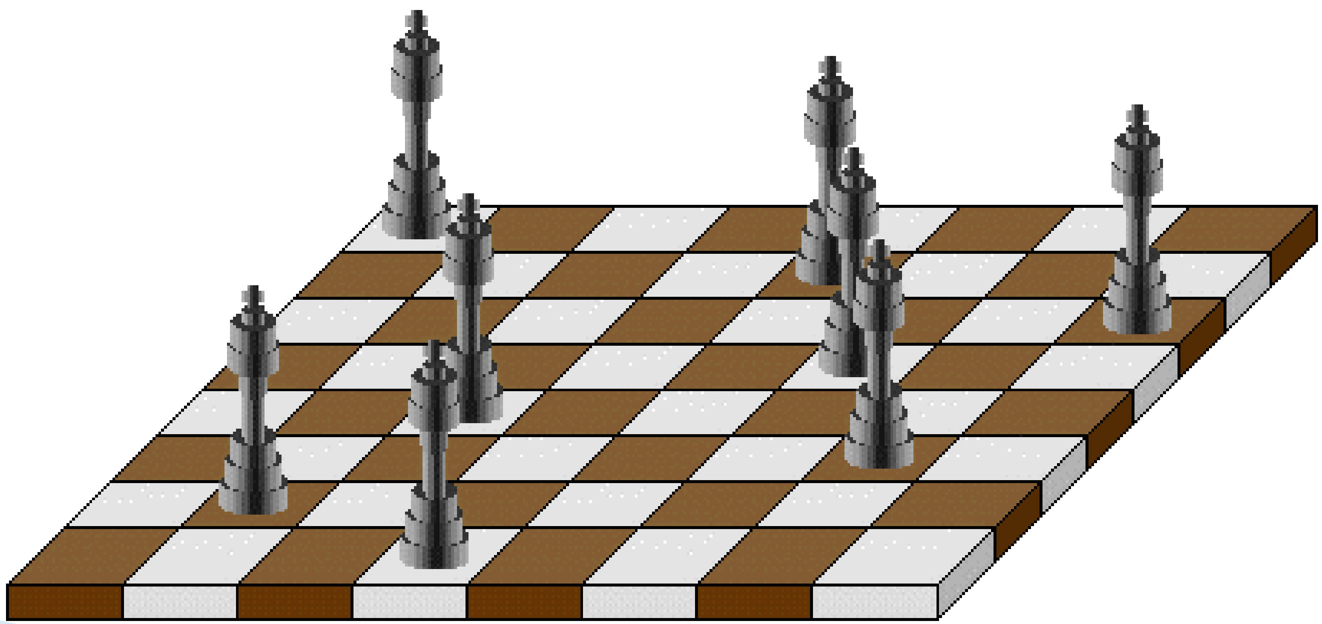
Backtracking (animation)



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

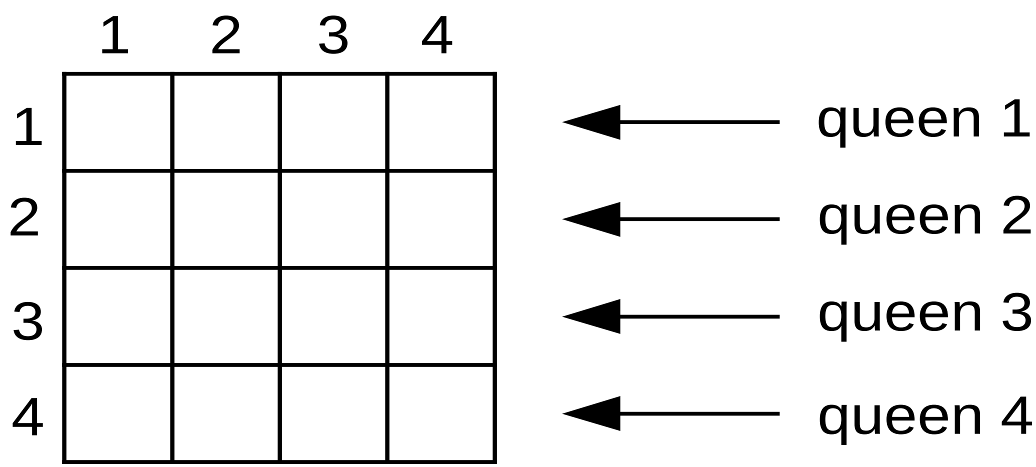
Example: n-Queens Problem

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



Example: 4-Queens

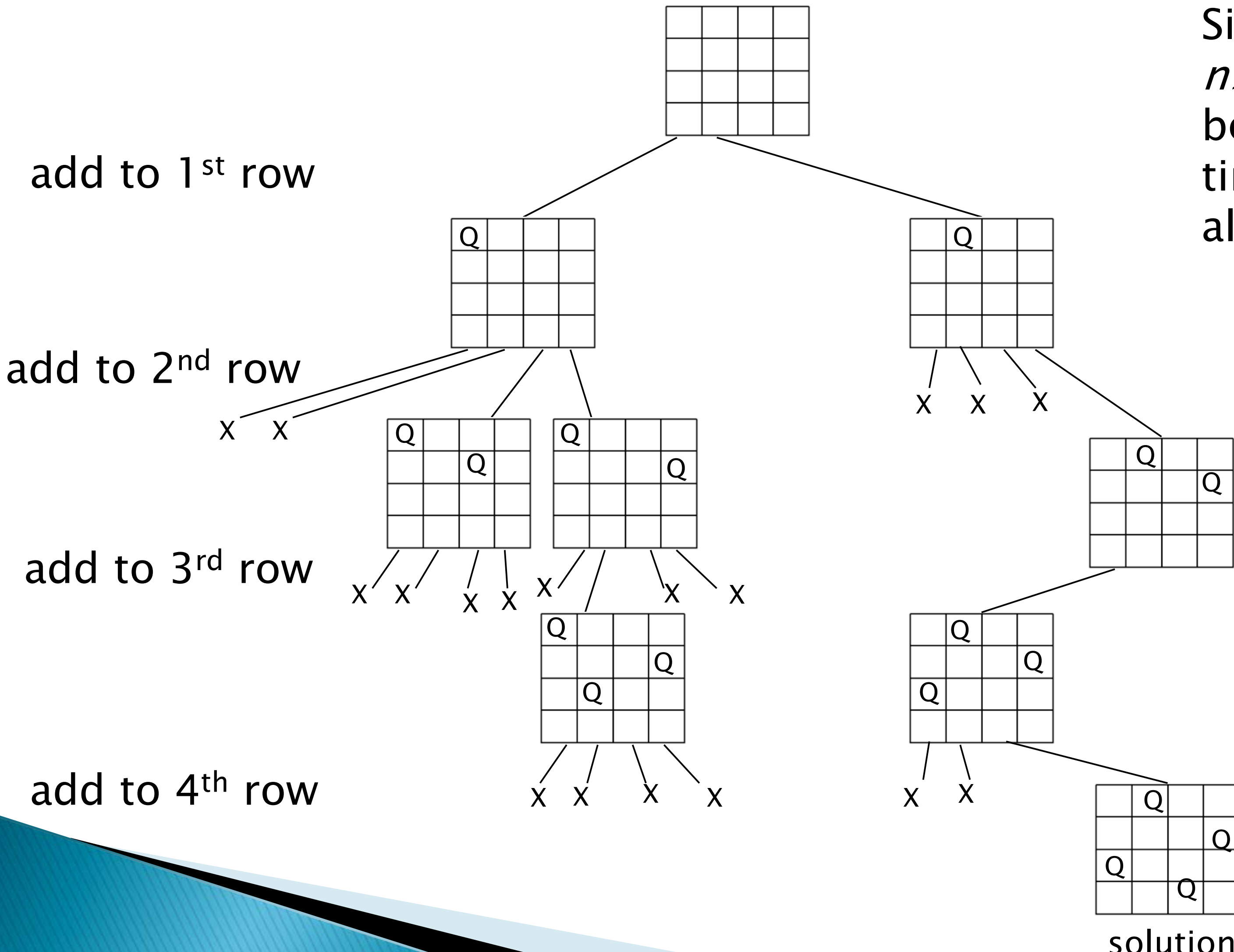
- n=4



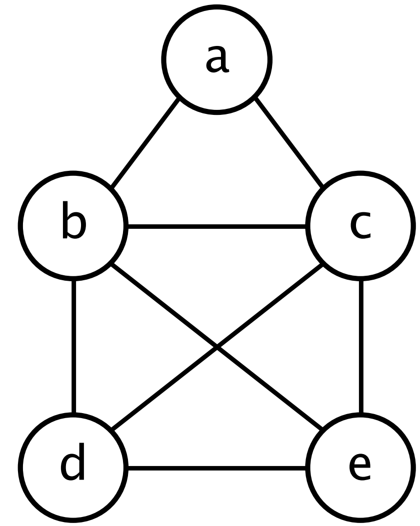
- We can solve it by backtracking
  - Root is empty board
  - At level i… put a queen in row i

State-Space Tree of 4-Queens

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



Hamiltonian cycle example



Backtracking and Branch & Bound

- Backtracking
  - n-Queens problem
- Branch and Bound
  - Assignment problem

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

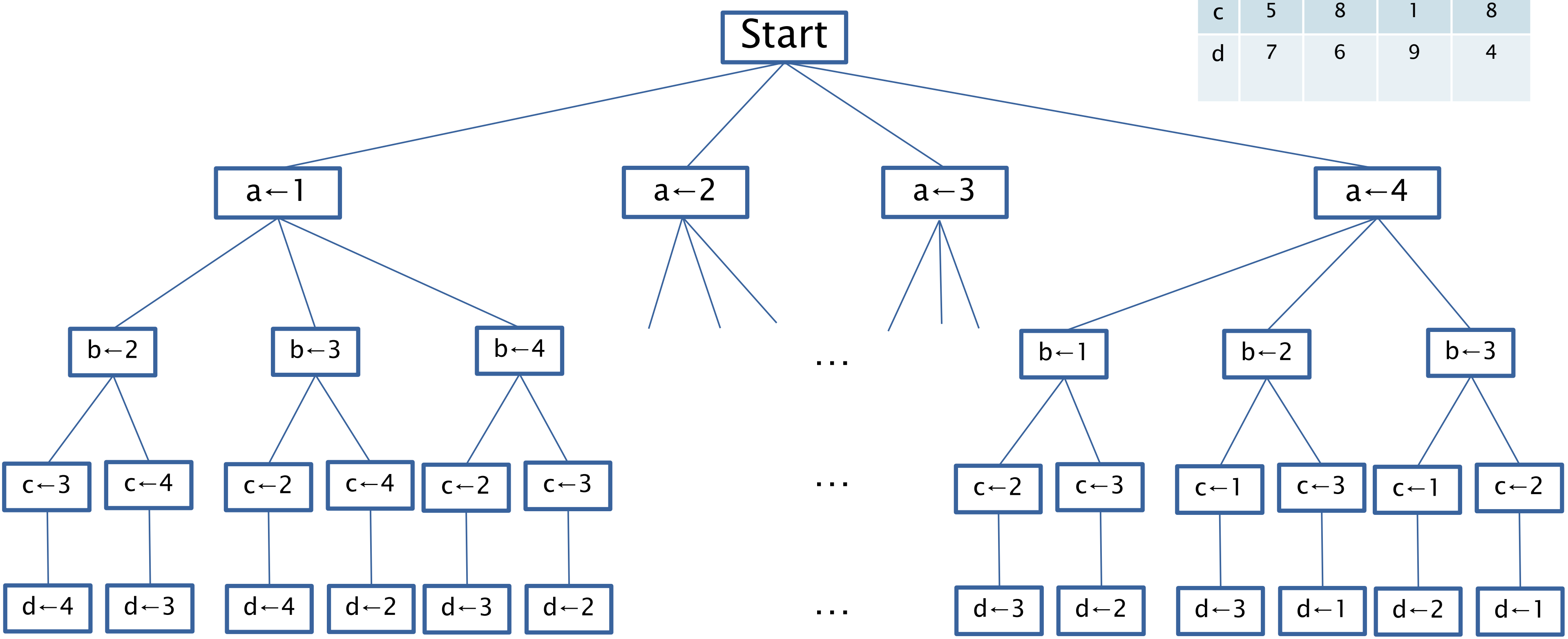
Select one element in each row of the cost matrix C so that:

- 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

lb = 2+3+1+4 =10

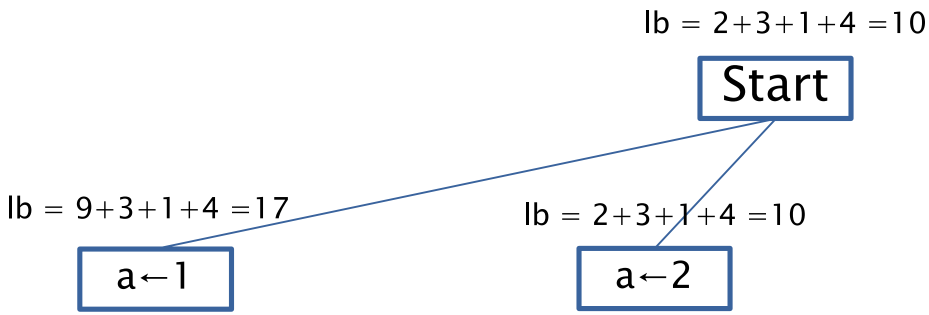
Start

lb = 9+3+1+4 =17

a←1

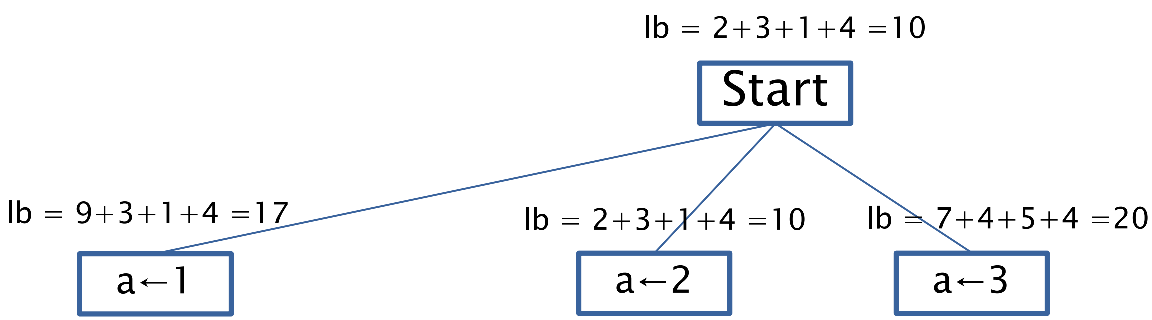
Assignment Problem (Branch & Bound)

	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



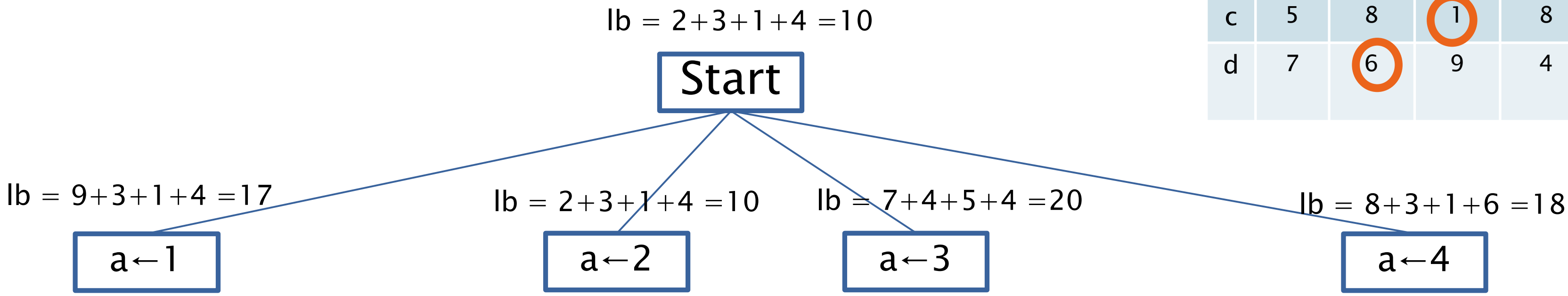
Assignment Problem (Branch & Bound)

	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



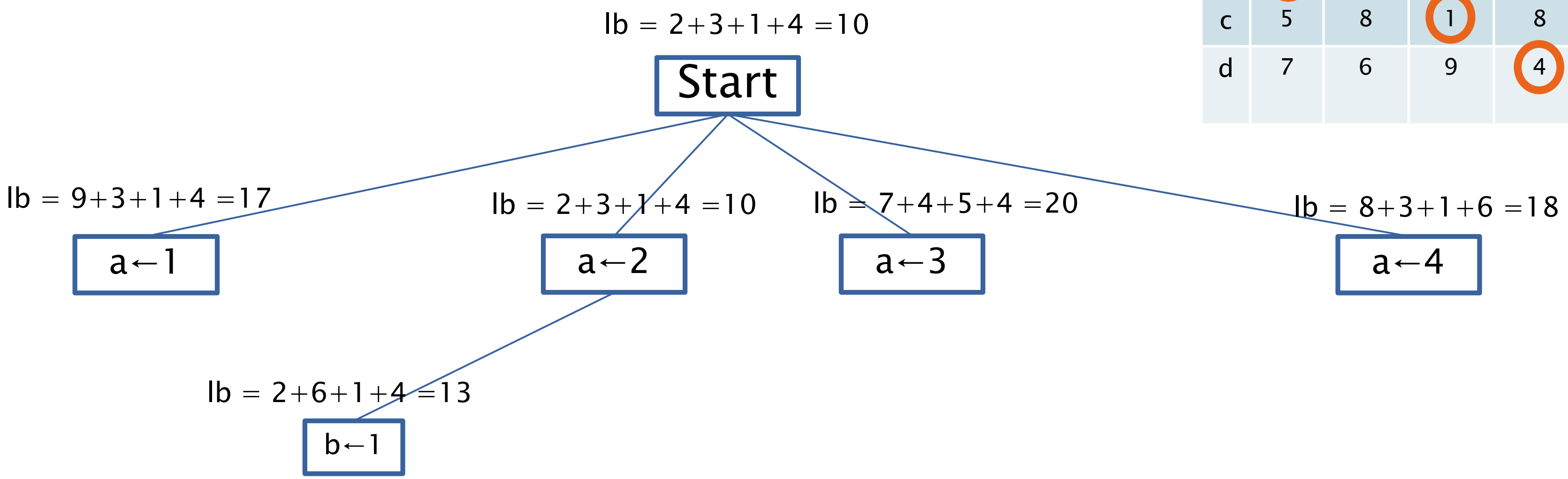
Assignment Problem (Branch & Bound)

	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



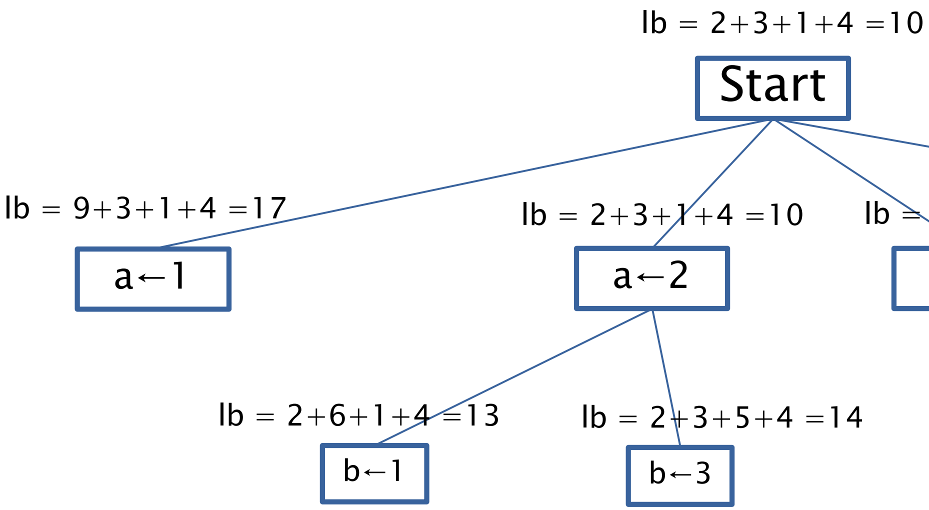
Assignment Problem (Branch & Bound)

	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



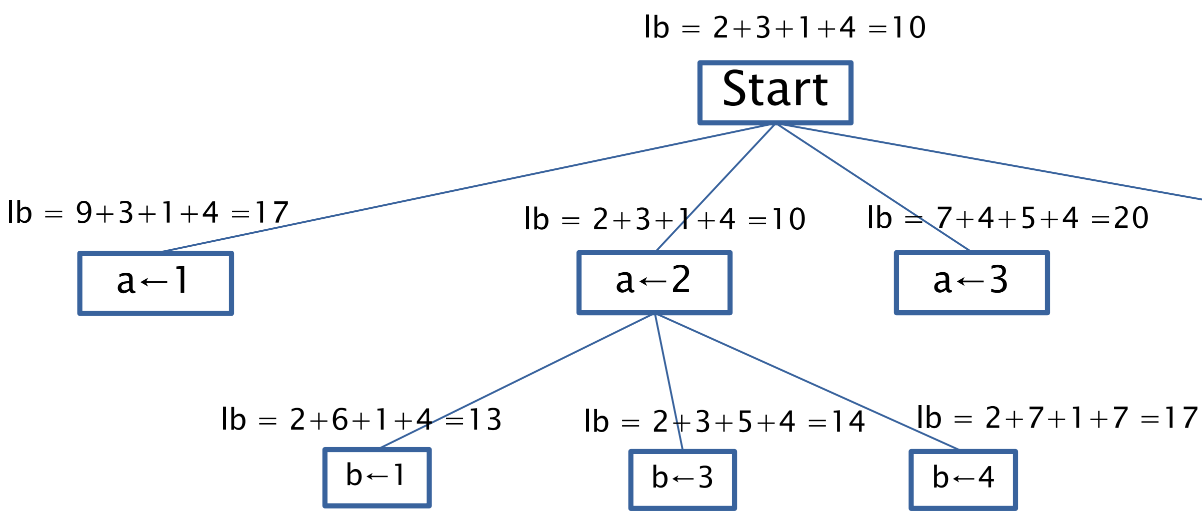
Assignment Problem (Branch & Bound)

	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



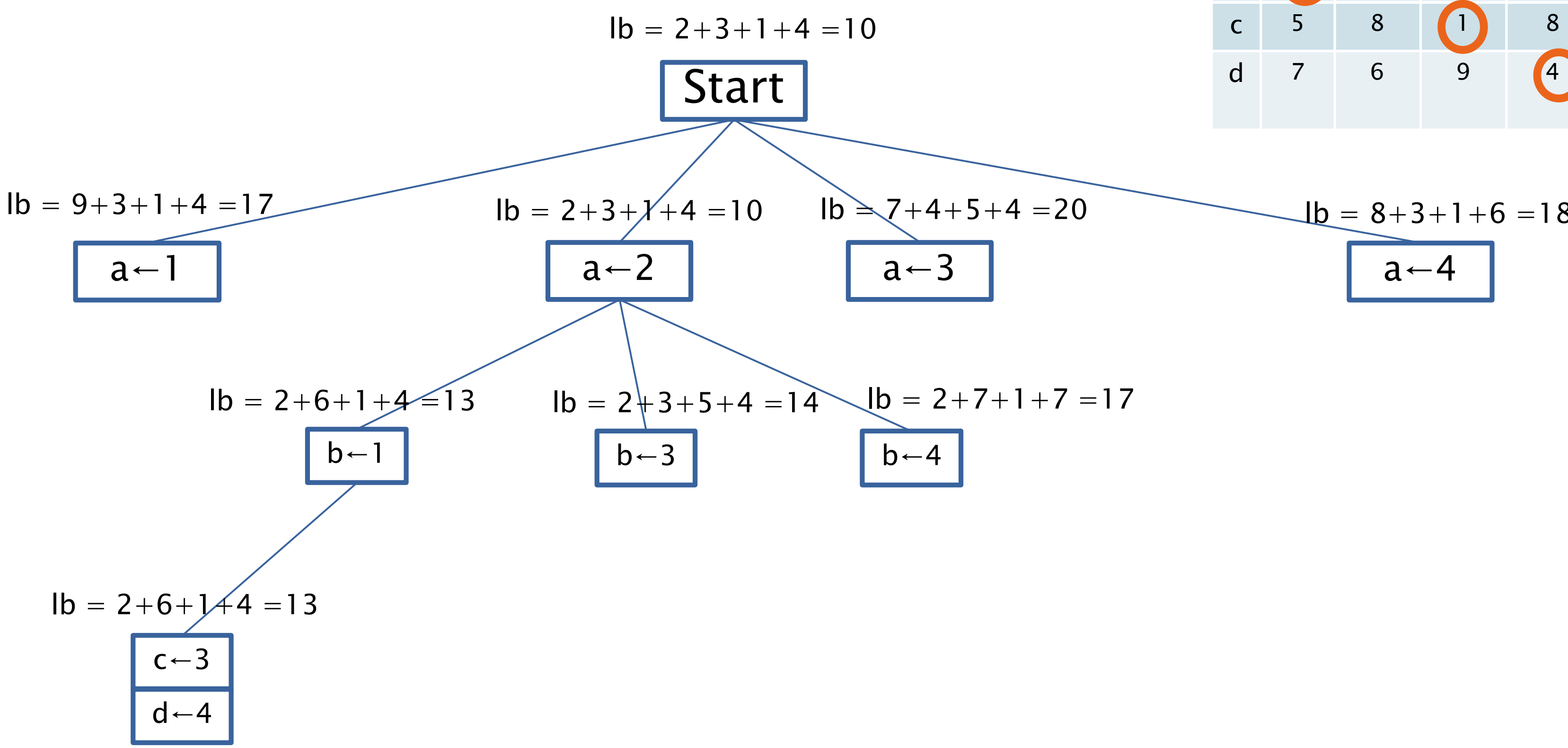
Assignment Problem (Branch & Bound)

	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



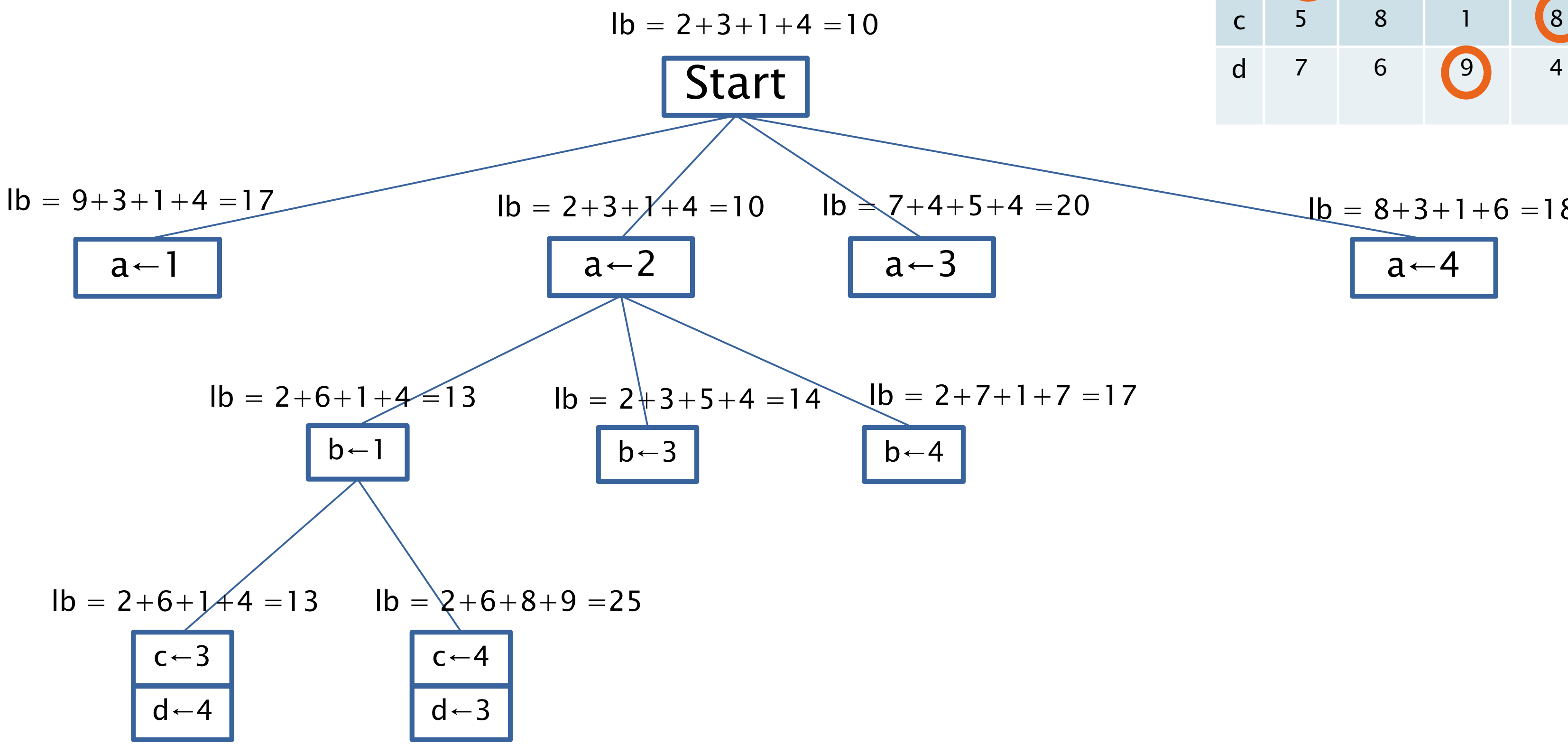
Assignment Problem (Branch & Bound)

	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



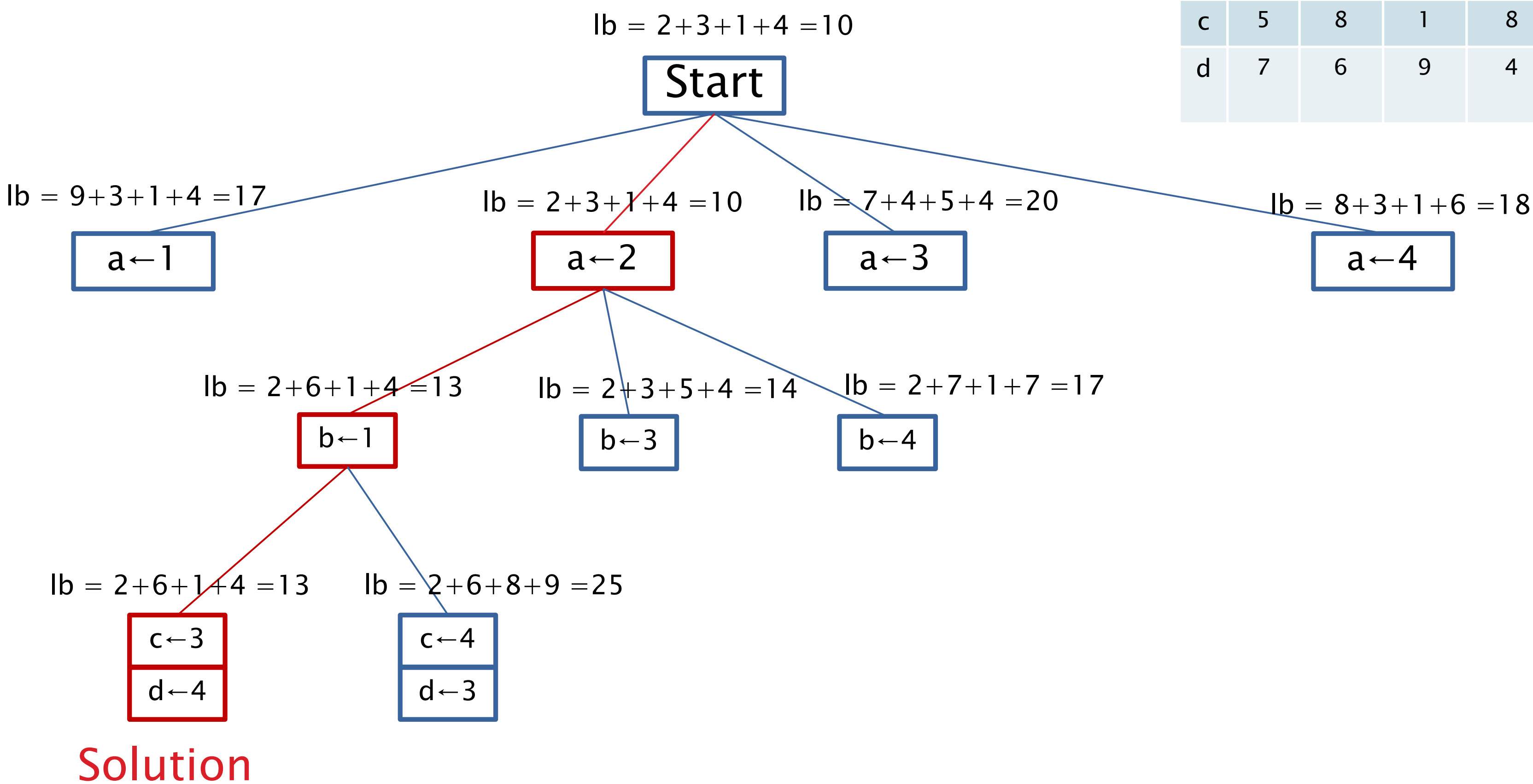
Assignment Problem (Branch & Bound)

	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



Limitations of Algorithms

(Chapter 11)

What We Have Seen So Far

- Lots of algorithms
  - Different kinds
    - (e.g. divide and conquer/dyn. programming)
  - Different applications
    - (e.g sorting/graph probs)

- Focus on efficient problem solving

- Today:
  - What are the *limits* of algorithmic problem solving?

Types of Problems

- *Decision problems*
  - Problems that have a yes/no answer
  - Example:
    - Does this graph have a Hamiltonian cycle?

- *Optimization problems*
  - Problems that involve maximizing or minimizing some parameter
  - Example:
    - What is the shortest path from A to B?

Types of Problems

- Optimization problems can be framed as decision problems…

- Example
  - How many colors are needed to color this map?

  Vs.
  - Can I color this map with 4 colors?

- This is often called the *decision problem version* of an optimization problem

A Basic Question

- Can every decision problem be solved by an algorithm?
- In other words:
  - Can we always write down a step by step process that will give the right yes/no answer to a question?

Program Termination

- We can write a program that never stops

```
X ← true
while (X)
{
}
```

- This can happen by accident too
  - And it is irritating to debug
  - It would be nice if we could know in advance if our program was going to run forever

The Halting Problem

- Here is a important theoretical question:

Can we write a program that tests whether or not an arbitrary program will terminate?

- This is *The Halting Problem*
  - Originally posed by Alan Turing
  - (The actual question is a bit more precise)

The Halting Problem

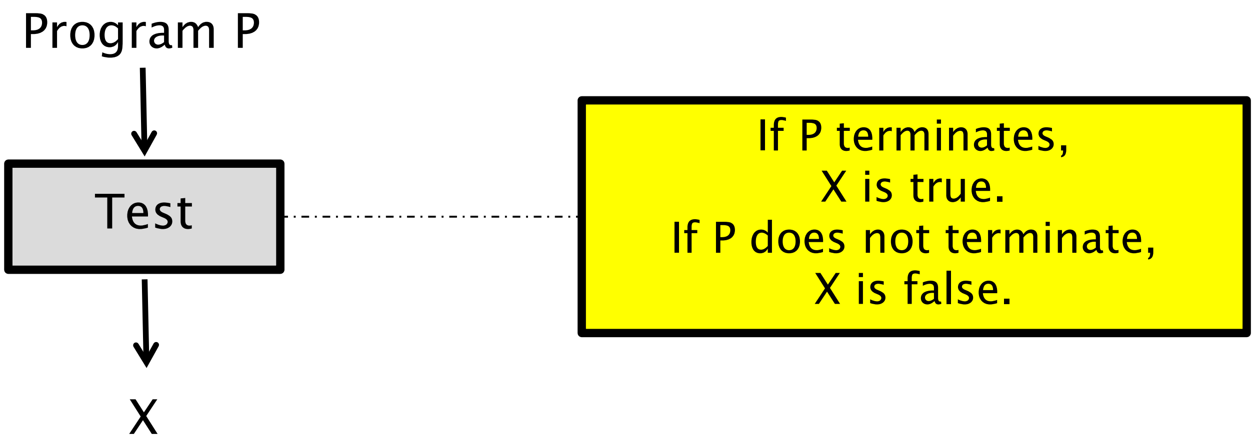
- We will *prove* that no such program exists

- The proof is by *contradiction*
  - So we assume that the program does exist…
  - Then we show that assumption leads to nonsense

- This is a standard method of proof in math

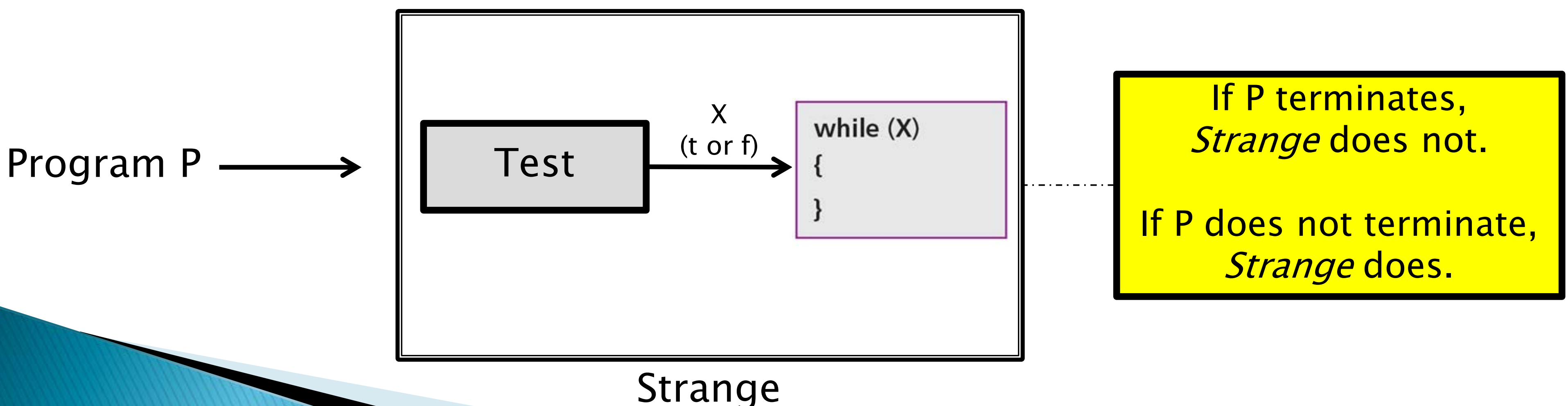
Proof – Step 1

- Suppose we have a program called *Test* such that:
  - It takes a program *P* as input
    - It returns *true* if the program P terminates
    - It returns *false* if the program P runs forever



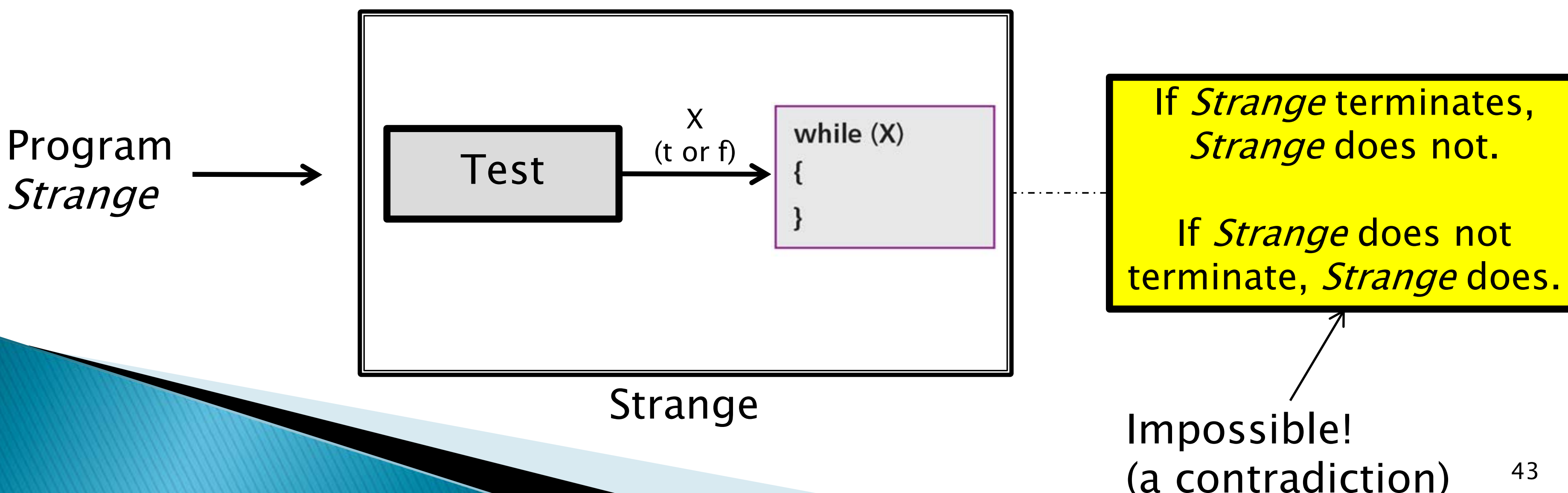
Proof – Step 2

- Now create another program called *Strange* made of two parts:
  - A copy of *Test* at the beginning
  - An empty loop—a loop with an empty body—at the end.
  - The loop uses X as the testing variable, which is actually the output of the Test program.
  - This program also uses P as the input.



Proof – Step 3

- So now we have written the program *Strange*
  - It takes any program as input
  - So we can put *Strange* as the input for itself



What We Showed

- If we assume *Test* exists, then:
  - The program *Strange* also exists
- But the program *Strange* can not exist
  - Because it terminates AND does not terminate at the same time

- Therefore… the program *Test* does not exist

There is no program that tests whether or not an arbitrary program will terminate.

What We Showed (cont.)

- We started with a simple decision problem:
  - Does an input program ever terminate?
- We showed that there is no program that solves this problem

What This Means

- There are decision problems that can not be solved ***by any algorithm***
  - The Halting Problem is an example

We say a decision problem is **undecidable** if it can not be solved by any algorithm.

- So what we normally say:
  - The Halting Problem is undecidable