Lecture 11: Backtracking, Branch & Bound

Textbook: Chapter 12

第11讲：回溯法、分枝限界法

教材：第12章

Golf-tee puzzle

Jump
and
Remove

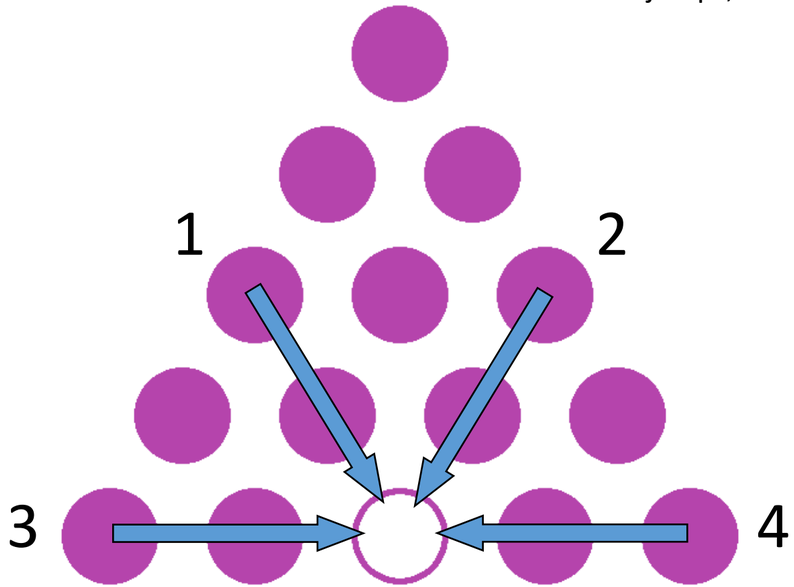
高尔夫球座谜题

跳跃
并移除

Valid moves

This position has four valid moves:

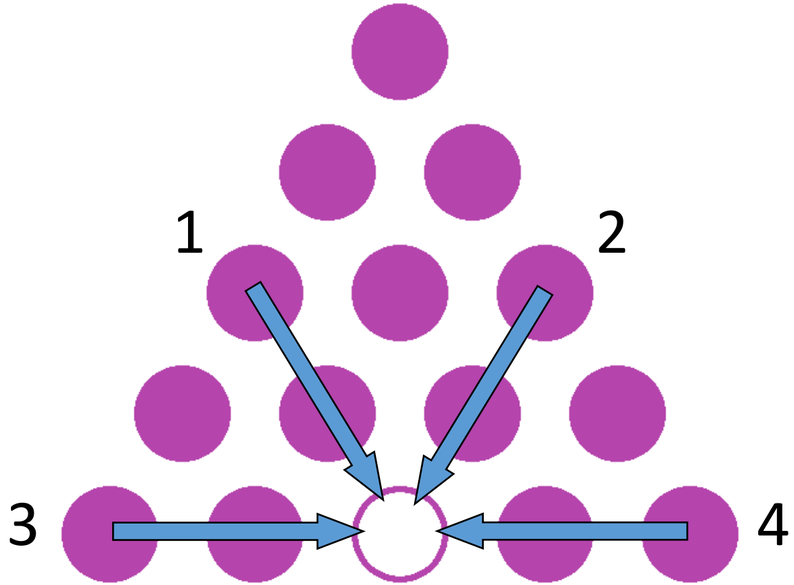
To win:
13 consecutive valid jumps; one peg left



有效移动

这个位置有四个有效的移动：

获胜条件：连续13次有效跳跃；只剩一个棋子



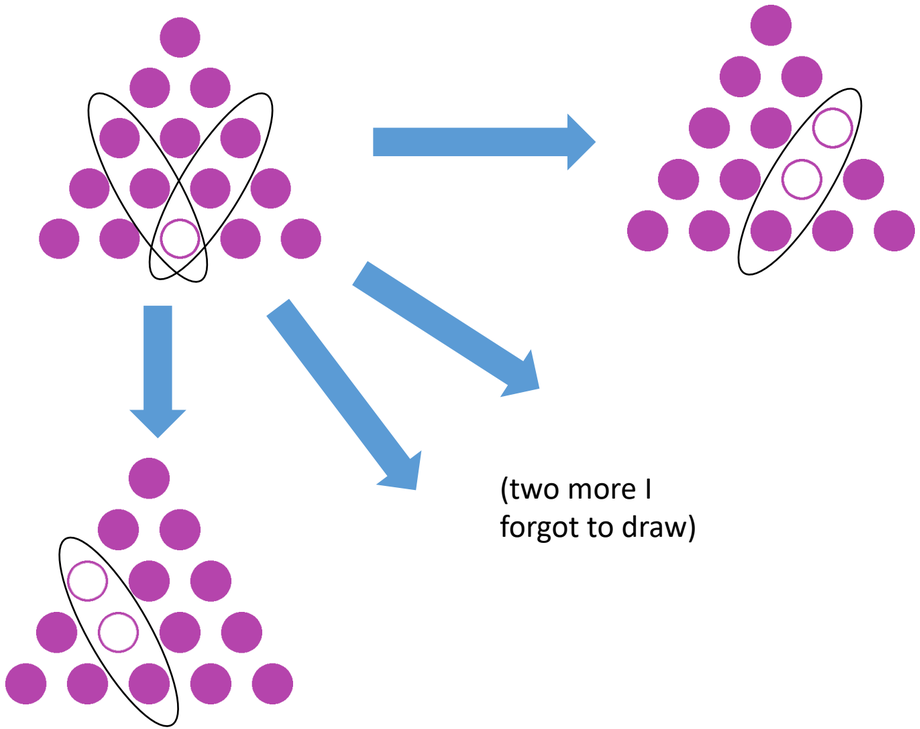
Backtracking

- 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"

回溯法

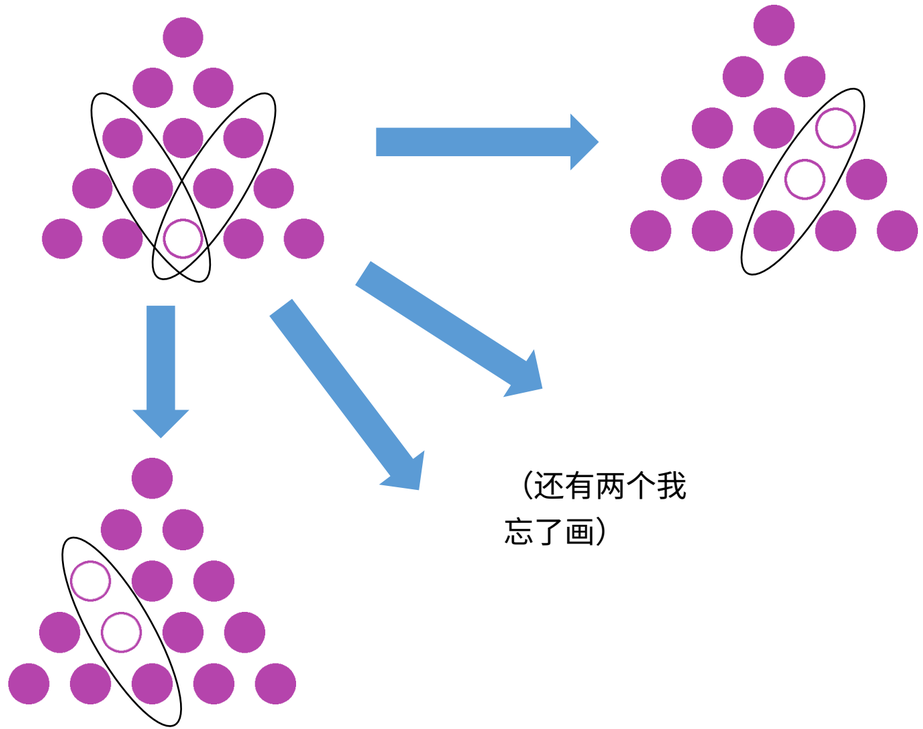
- 假设你必须在一系列 决策 中，从各种 选项 中进行选择，其中
  - 你没有足够的信息来确定该选择什么
  - 每个决策都会导致一组新的选择
  - 某些选择序列（可能不止一个）可能是你问题的解决方案
- 回溯法是一种系统地尝试不同决策序列的方法，直到你找到一个“可行”的方案

Changing state



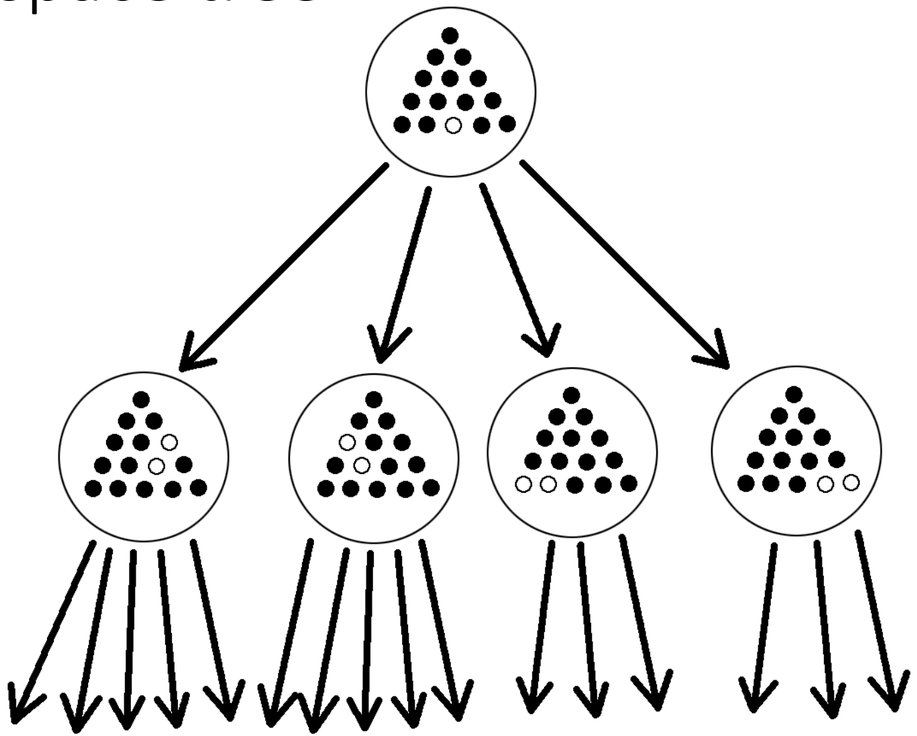
(two more I
forgot to draw)

更改状态



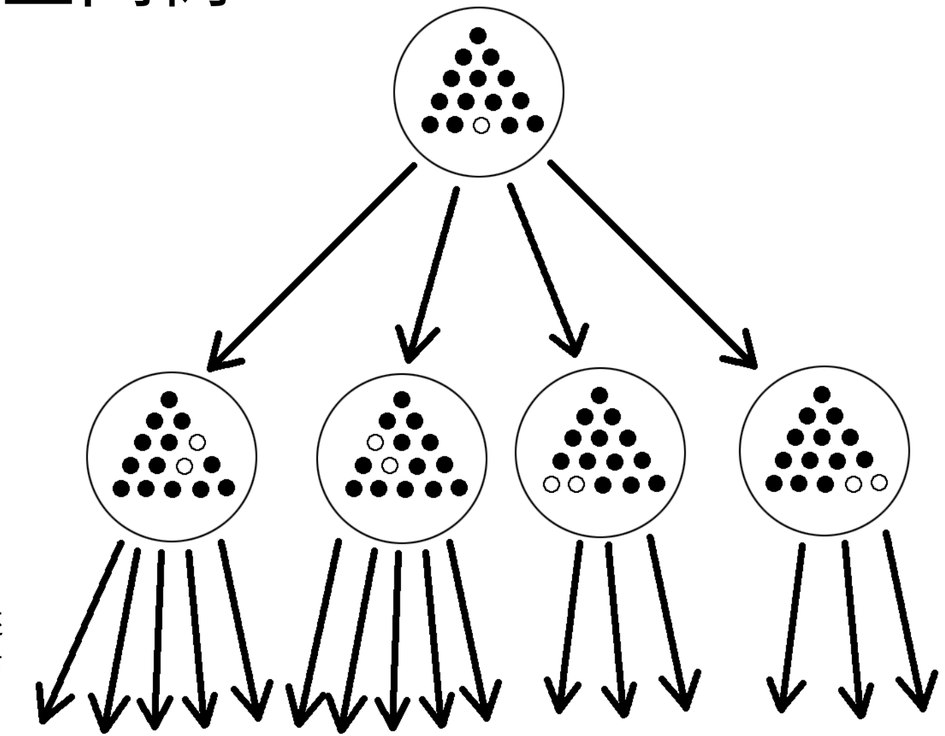
（还有两个我
忘了画）

State-space tree



So many more places to go from here!

状态空间树



从这里出发还有那么多地方可以去！

Backtracking in words

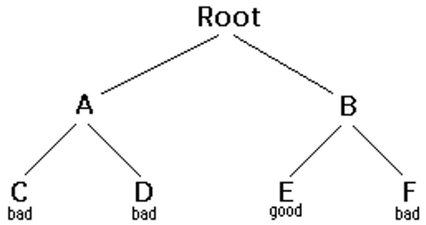
- IDEA:
  - Construct solutions one component at a time
  - If a partial solution can be developed further without violating constraints:
    - Choose first legitimate option for the next component
  - If there is *no option* for the next component
    - Backtrack to replace the last component of partial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

词语中的回溯

- 思路:
  - 一次构建一个组件来构造解决方案
  - 如果部分解决方案可以在不违反约束的情况下进一步开发:
    - 为下一个组件选择第一个合法选项
  - 如果下一个组件没有选项
    - 回溯以替换部分解决方案的最后一个组件

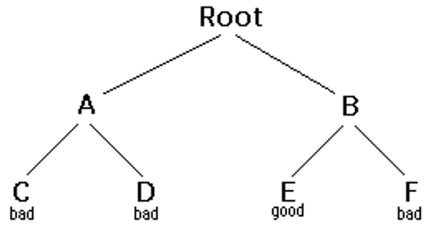
Backtracking

- 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.



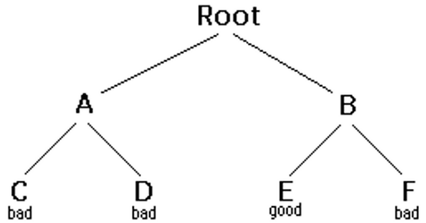
回溯

- 将这些解设想为以树状结构组织
  - 每个节点代表解在某一阶段的“状态”
  - 根节点表示搜索开始前的初始状态
  - 第一层的节点表示第一个选择
  - 第二层……第二个选择……依此类推。



Backtracking – Abstract Example

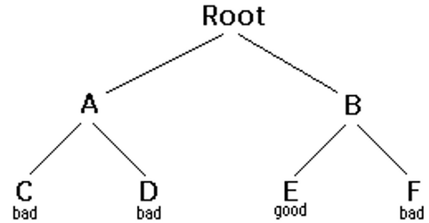
- Starting at Root, your options are A and B. You choose A.
- At A, your options are C and D. You choose C.
- C is bad. Go back to A.
- At A, you have already tried C, and it failed. Try D.
- D is bad. Go back to A.
- At A, you have no options left to try. Go back to Root.
- At Root, you have already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

回溯——抽象示例

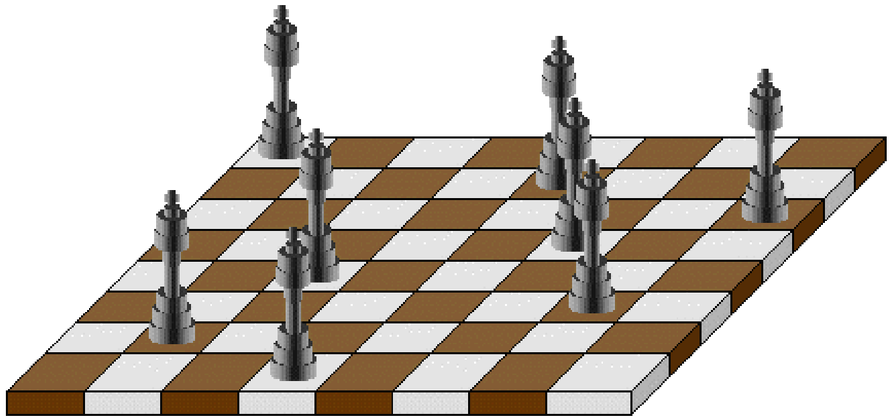
- 从根节点开始，你的选项是 A 和 B。你选择了 A。
- 在 A 节点，你的选项是 C 和 D。你选择了 C。
- C 是不可行的。返回到 A。
- 在 A 节点，你已经尝试过 C，但失败了。请尝试 D。
- D 是不可行的。返回到 A。
- 在 A 节点，你已没有其他可尝试的选项。返回到根节点。
- 在根节点，你已经尝试了 A。请尝试 B。
- 在 B 节点，你的选择是 E 和 F。请尝试 E。
- E 是可行的。恭喜！



用于构建解的树称为 状态空间树 ，节点是 部分解，边是 选择

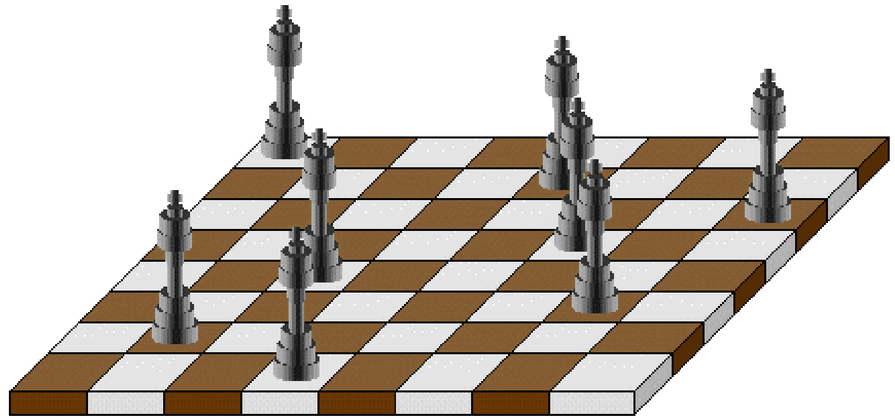
Example: n-Queens Problem

- 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



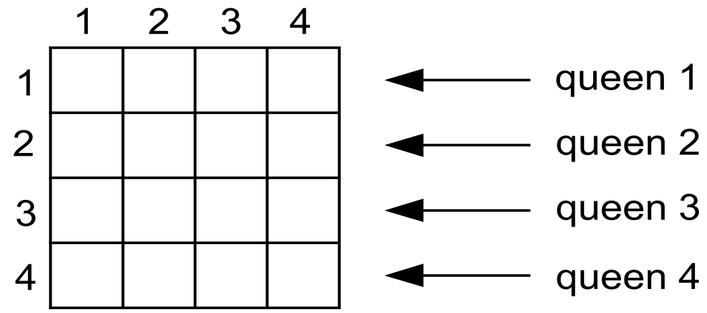
示例：n 皇后问题

- 在 *n*×*n* 的棋盘上放置 *n* 个皇后，使得任意两个皇后都不在同一行、同一列或同一对角线上
  - 即没有任何一个皇后会攻击到其他皇后



Example: 4-Queens

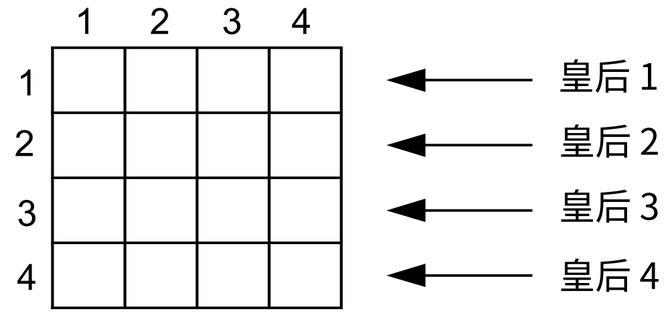
- n=4



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

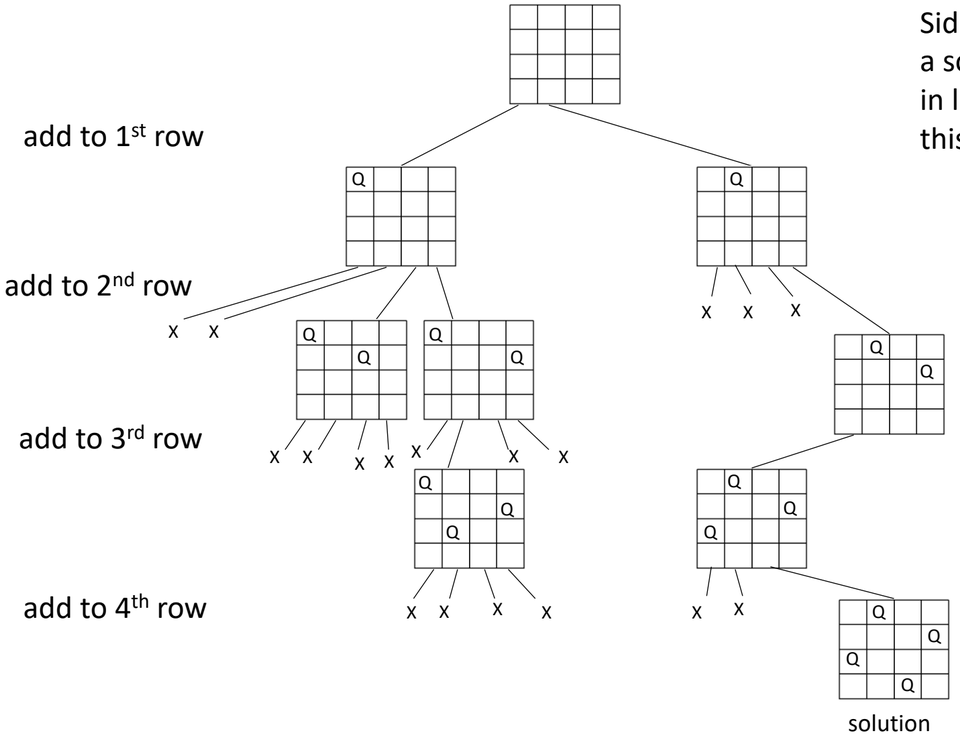
示例：4 皇后问题

- n=4



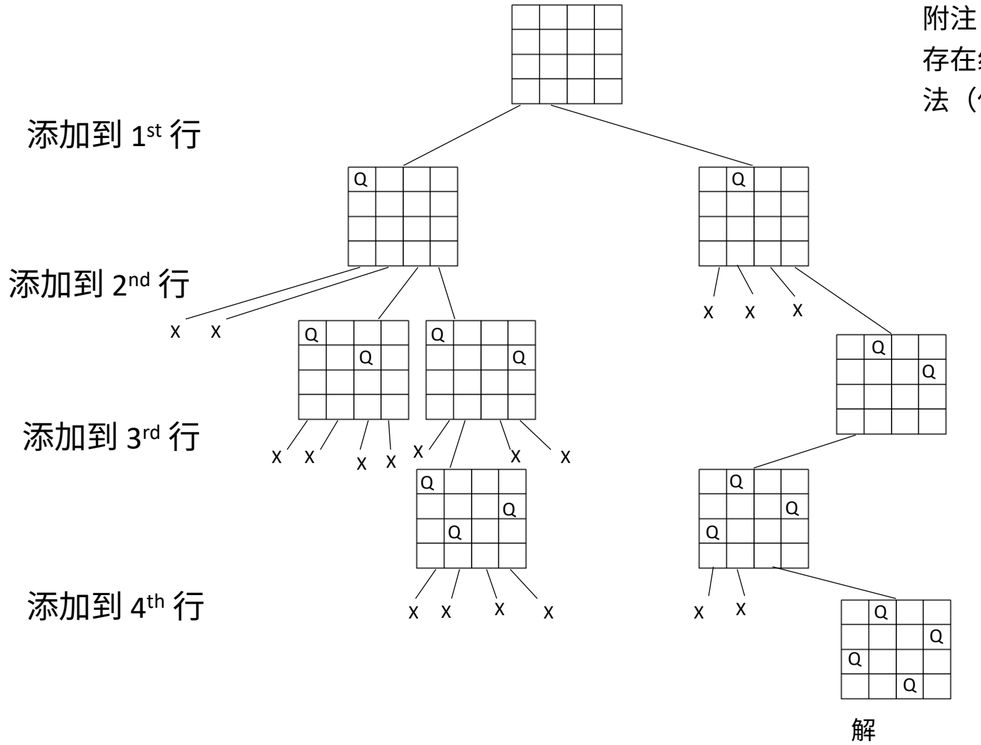
- 我们可以通过回溯法来解决
  - 根节点为空棋盘
  - 在第 i 步（第 i 层）……在第 i 行放置一个皇后

State-Space Tree of 4-Queens



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

4-皇后问题的状态空间树



附注：对于任意 *n>3*，都存在线性时间内的求解方法（但不是使用此算法）

Takeaways from the N- queens demonstration

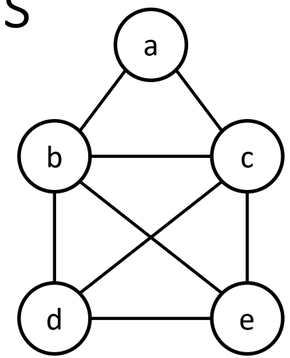
- Moving around in a DFS-like way through the State Space Tree
- This is the essence of a backtracking algorithm
- Proceed to the next possible choice; examine the choice; if "promising", we continue; if "non-promising", we backtrack (go back up the tree)
- At each LEVEL of the tree we have partial solutions 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

N皇后演示的要点

- 以类似深度优先搜索（DFS）的方式在状态空间树中移动
- 这正是回溯算法的核心思想
- 前进到下一个可能的选择；检查该选择；如果“有希望”，则继续；如果“无希望”，则回溯（返回树的上层）

- 在树的每一层，我们都有规模逐渐增大的部分解——逐步趋向一个完整解
- 树的叶子节点可能是死胡同，或者（如果足够深入树中）是解

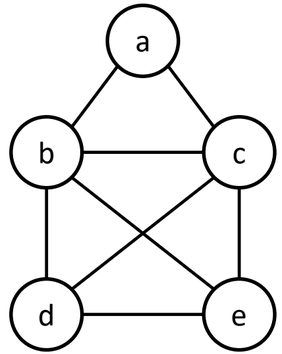
Example: Hamiltonian cycles



- 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?

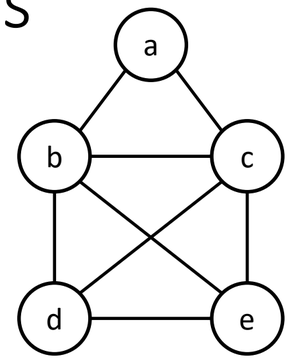
示例：哈密顿环



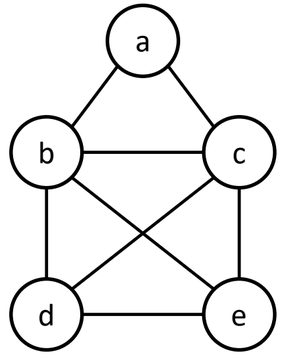
- 从任意顶点开始
- 逐步构建一条路径
- 在每一“层”，尝试添加每一个剩余的邻接顶点
- 在死胡同时回溯

- 状态空间是什么?

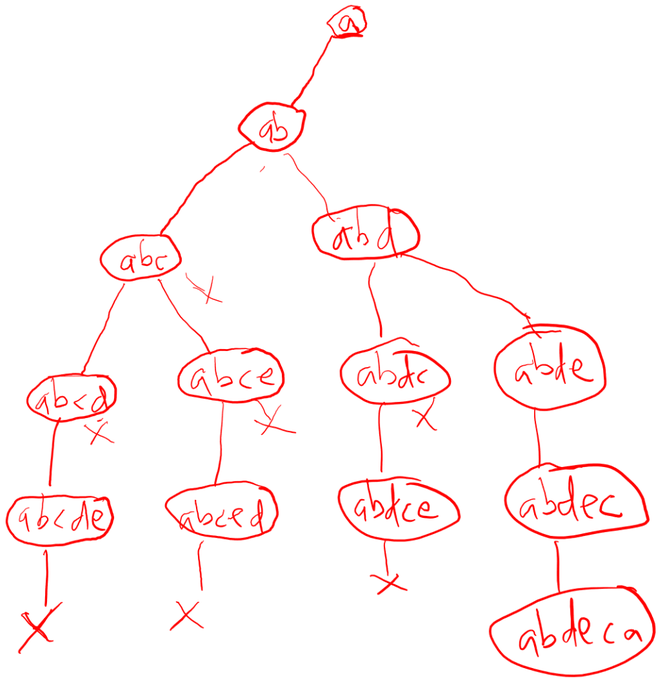
Example: Hamiltonian cycles

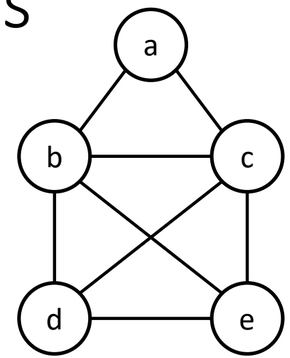


示例：哈密顿环

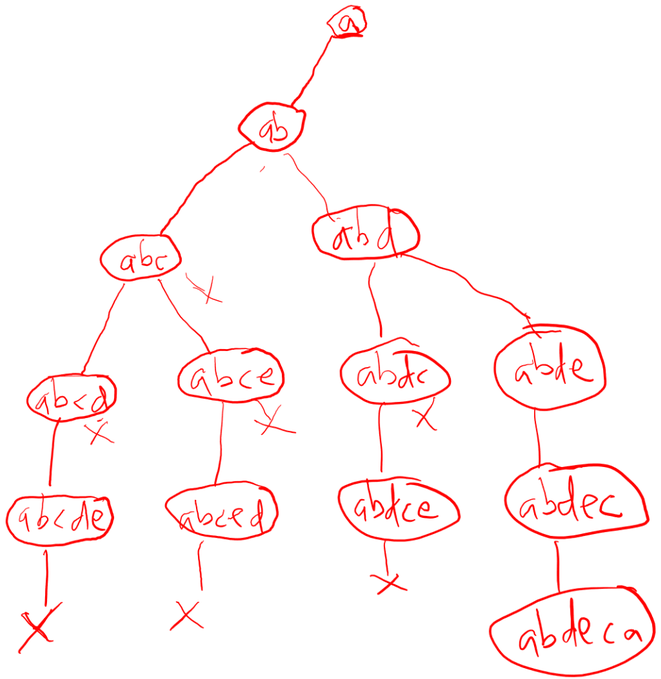


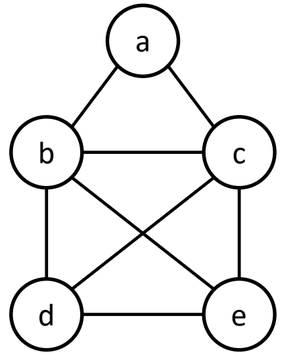
Example: Hamiltonian cycles





示例：哈密顿环





Branch and Bound 分支定界

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).

分支限界

- 思路：

  建立一个**限界函数**，用于计算状态空间树中某个节点处目标函数值的**界限**（目标函数值），**并判断该节点是否有希望**

  - **有希望的**（如果界限优于当前最优解的值）：继续扩展该节点。
  - **无希望的**（如果界限不优于当前最优解的值）：不再扩展该节点（剪枝状态空间树）。

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

指派问题

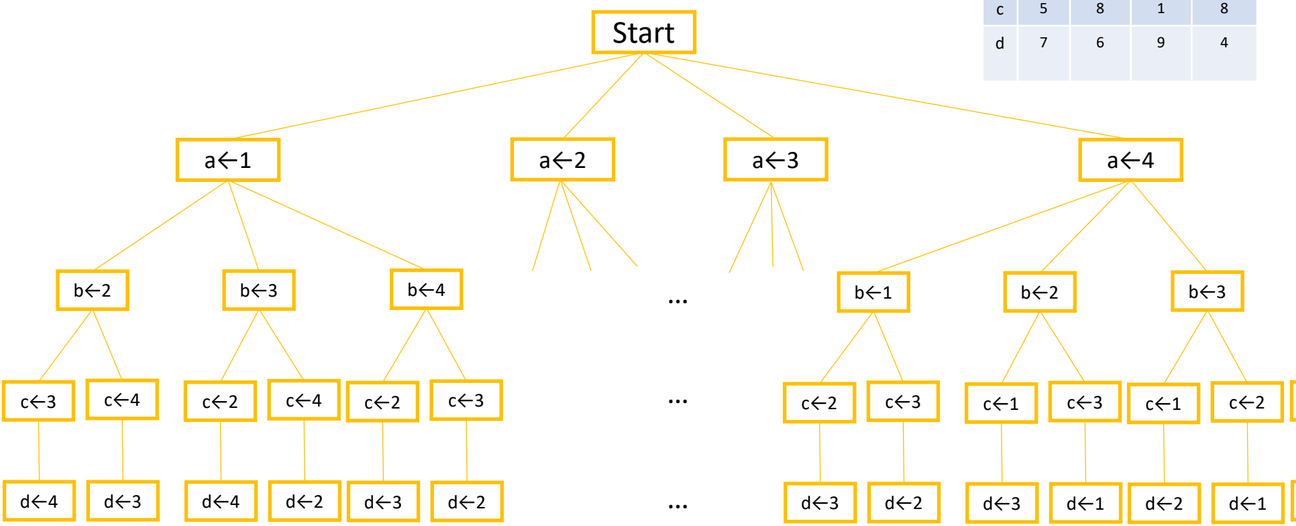
从成本矩阵 C 的每一行中选择一个元素，使得：

- 没有两个被选中的元素位于同一列
- 总和最小化

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

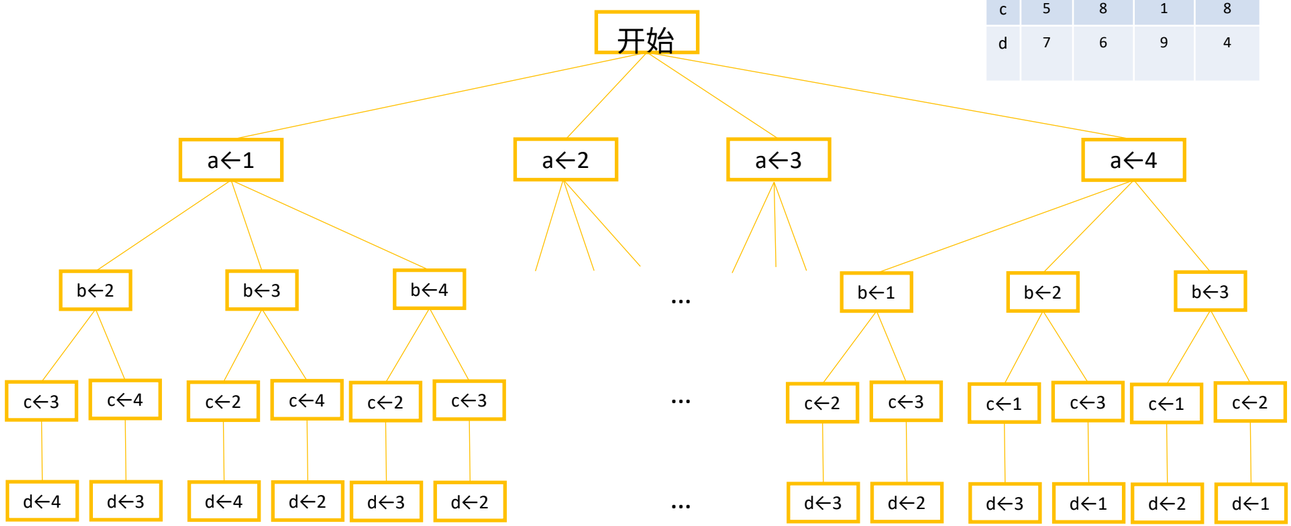
Assignment Problem (Brute Force)

	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



分配问题（暴力求解）

	Job 1	工作 2	工作 3	工作 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

分配问题 （分支与界限）

	任务 1	任务 2	任务 3	作业 4
a	9	(2)	7	8
b	6	4	(3)	7
c	5	8	(1)	8
d	7	6	9	(4)

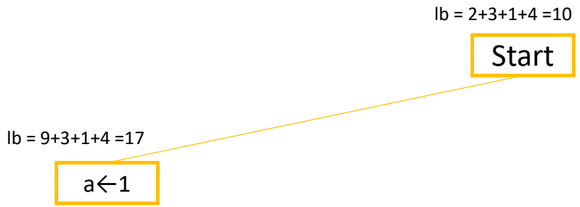
下界：此问题的任何解决方案的总成本至少为 10

lb = 2+3+1+4 =10

开始

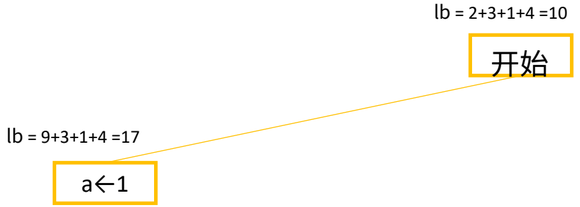
Assignment Problem (Branch & Bound)

	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



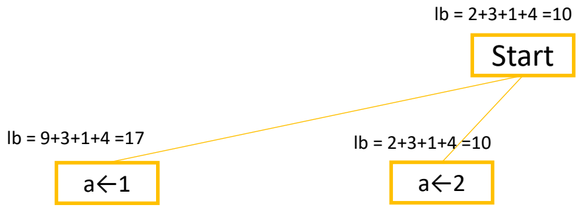
分配问题 （分支与界限）

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



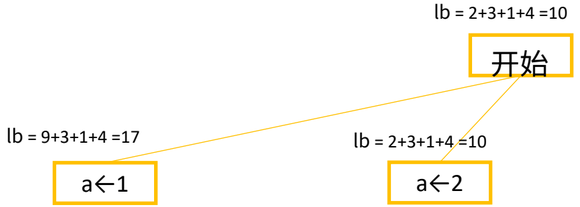
Assignment Problem (Branch & Bound)

	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



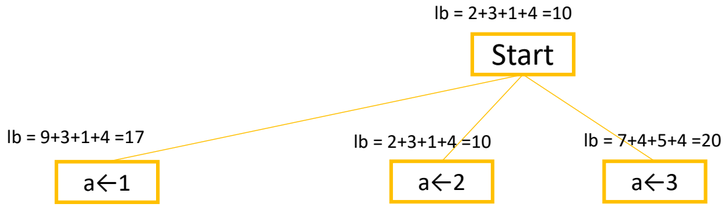
分配问题 （分支与界限）

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



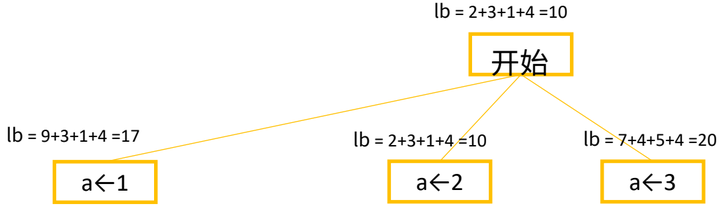
Assignment Problem (Branch & Bound)

	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



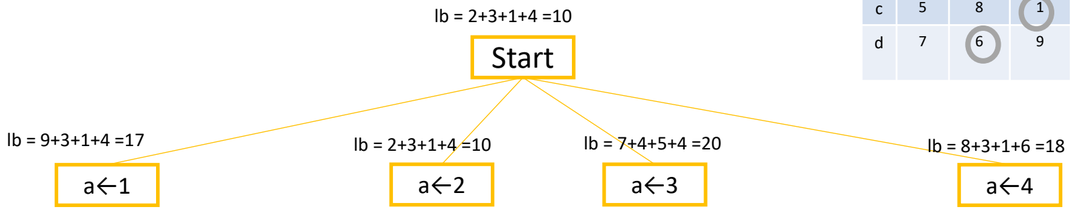
分配问题 （分支与界限）

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



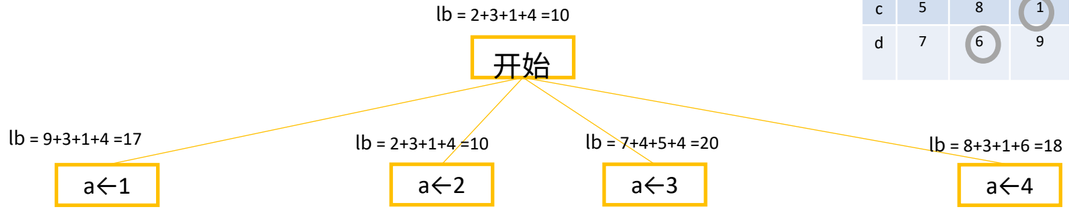
Assignment Problem (Branch & Bound)

	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



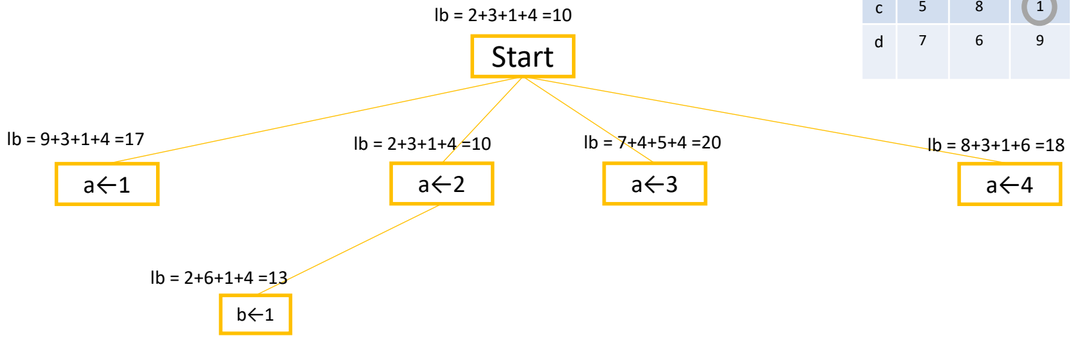
分配问题 (分支与界限)

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



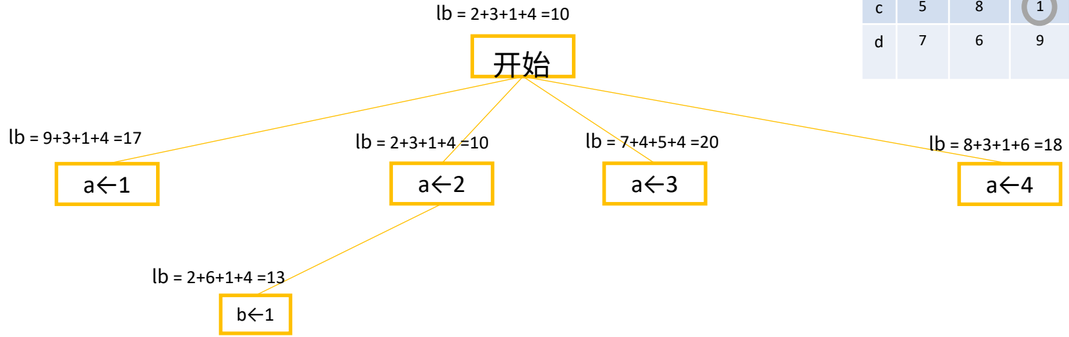
Assignment Problem (Branch & Bound)

	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)



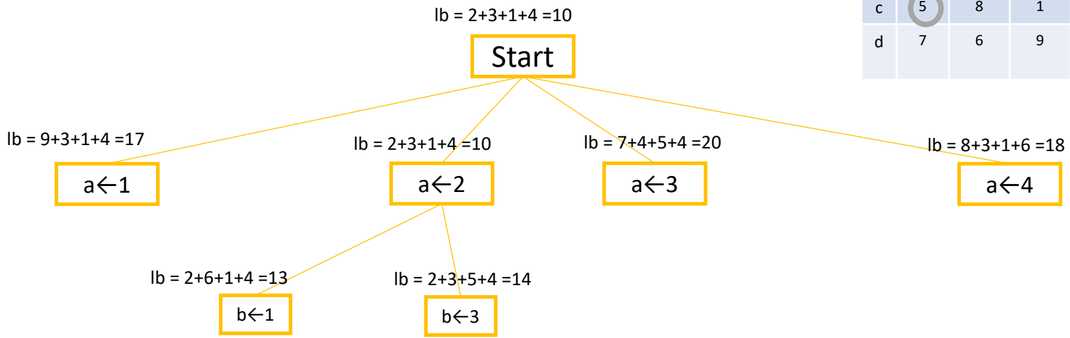
分配问题 (分支与界限)

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



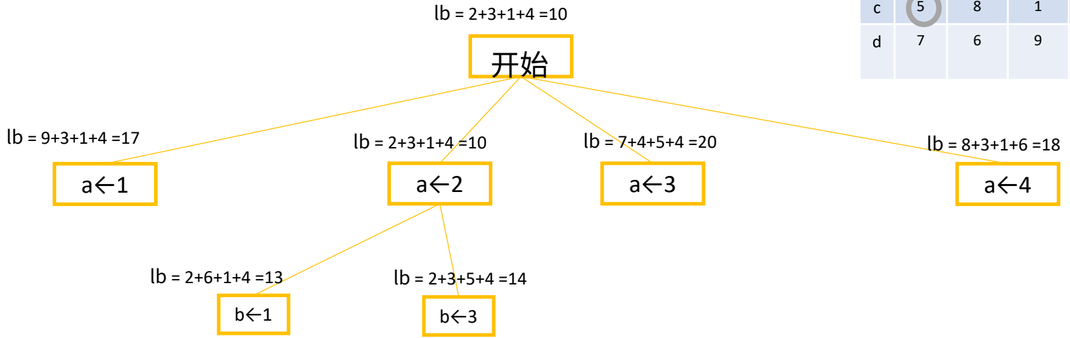
Assignment Problem (Branch & Bound)

	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)



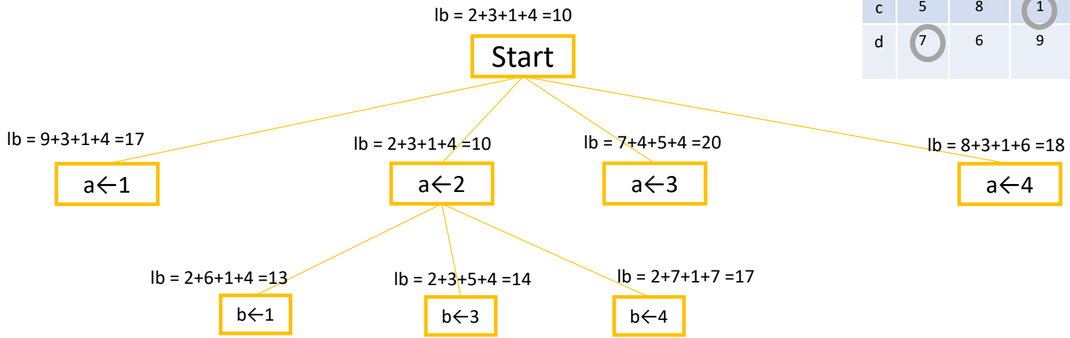
分配问题 (分支与界限)

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



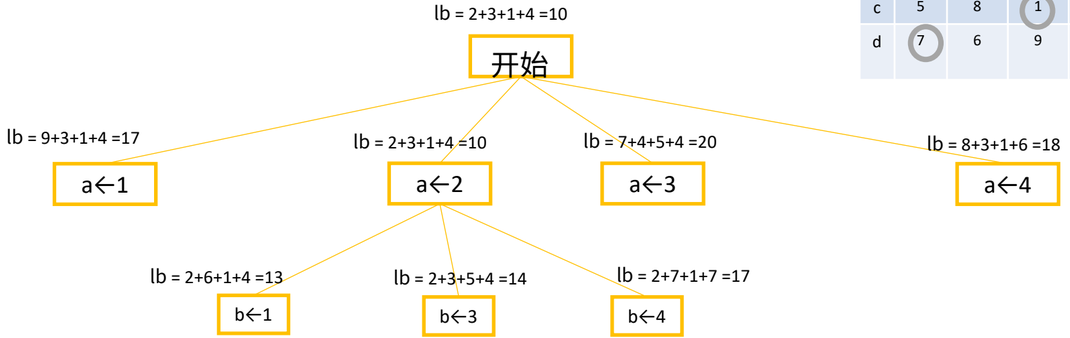
Assignment Problem (Branch & Bound)

	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



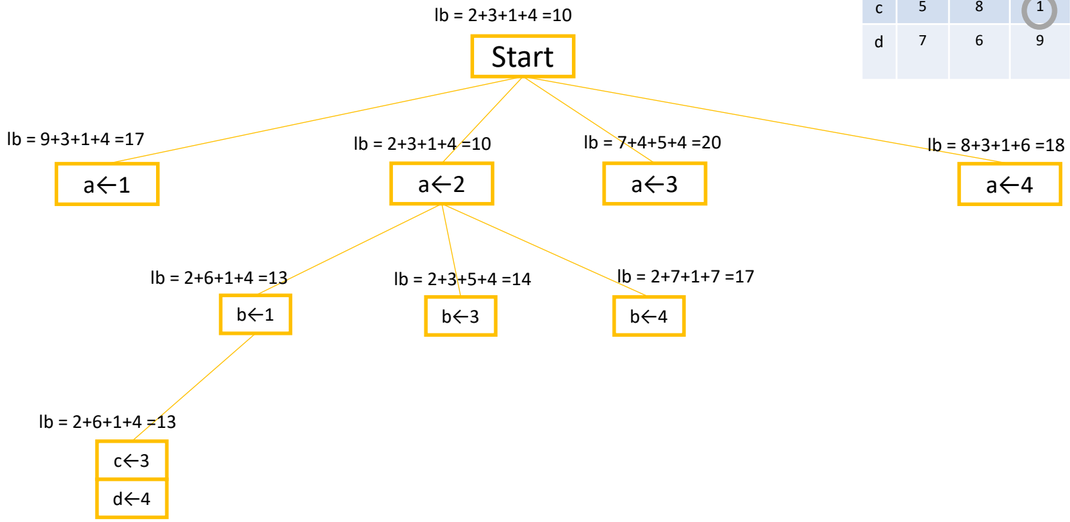
分配问题 (分支与界限)

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



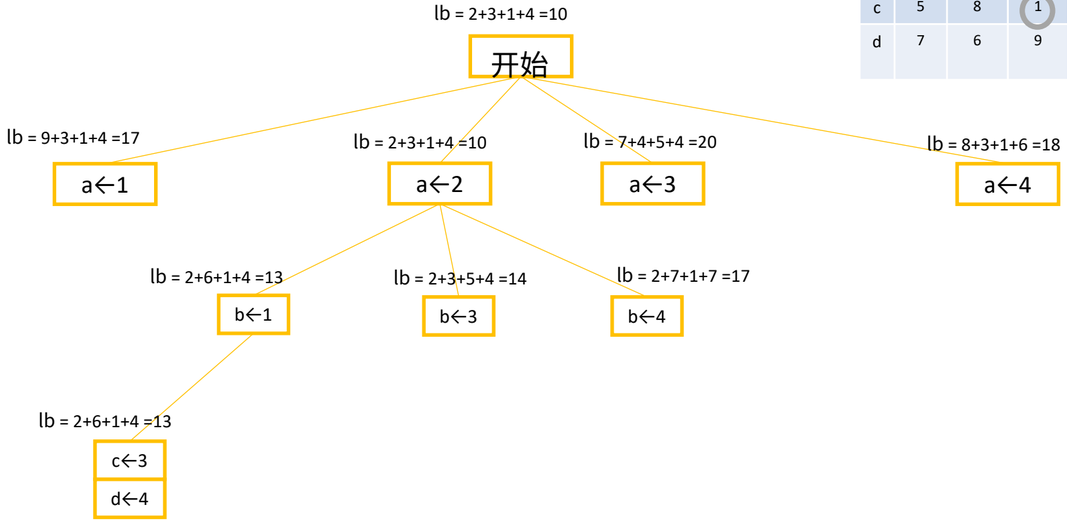
Assignment Problem (Branch & Bound)

	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



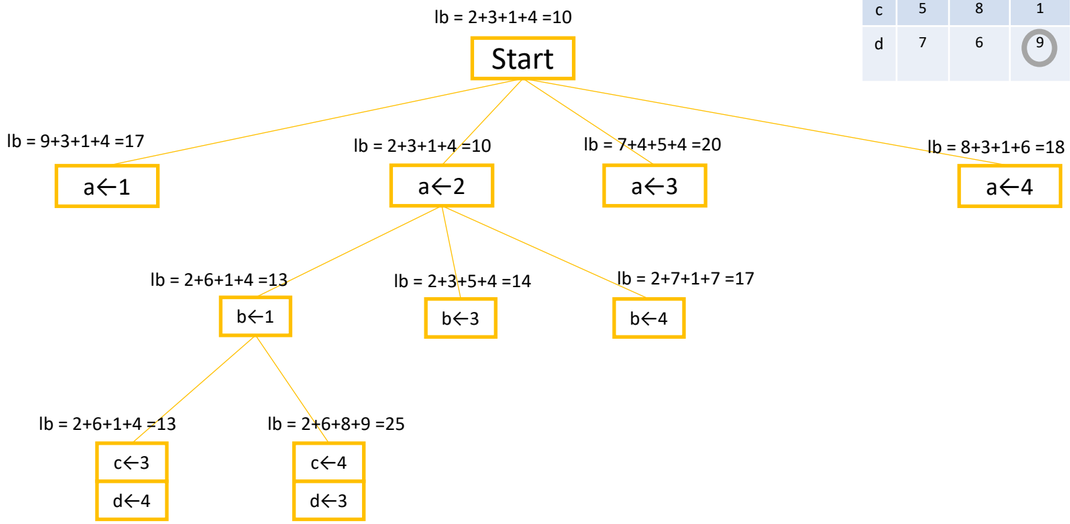
分配问题 (分支与界限)

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



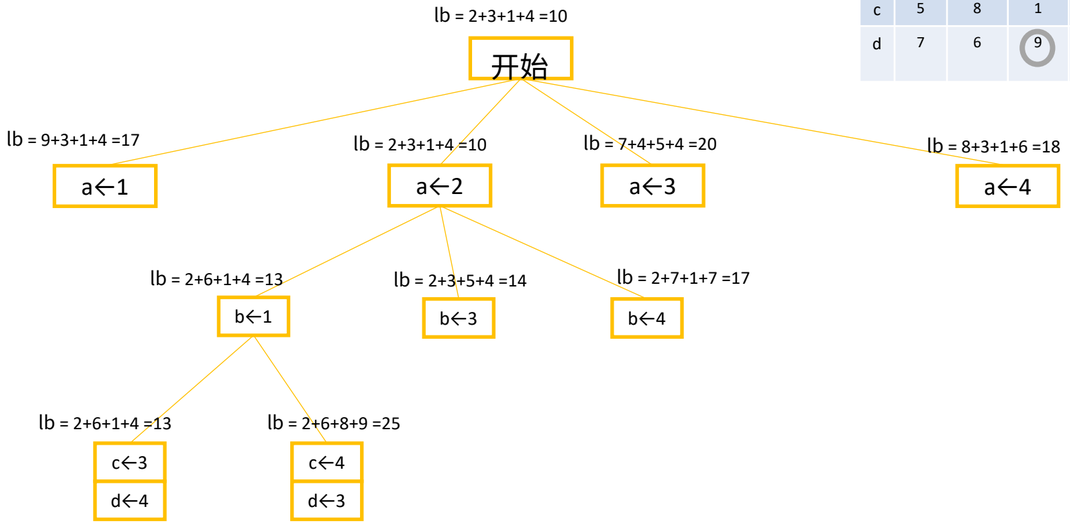
Assignment Problem (Branch & Bound)

	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



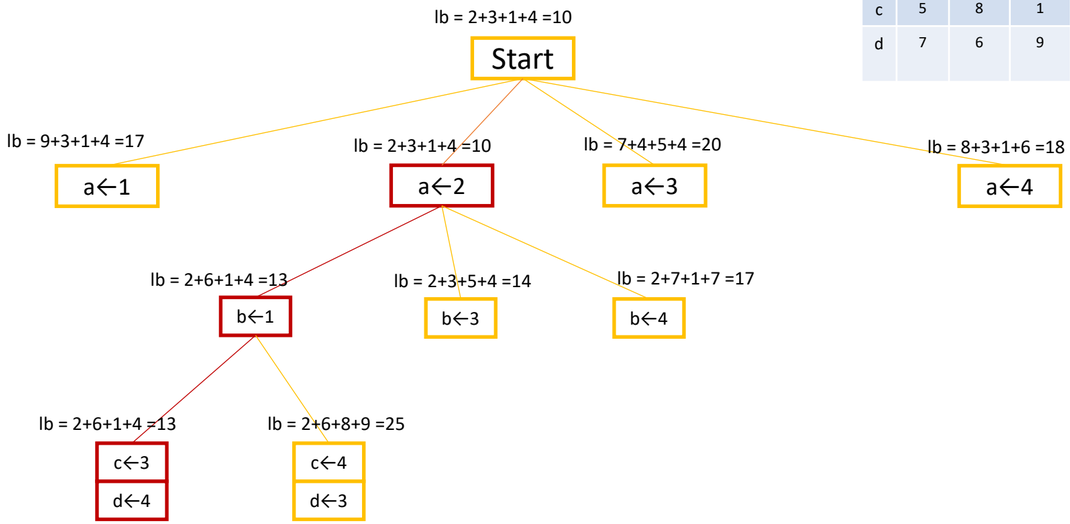
分配问题 (分支与界限)

	Job 1	作业 2	作业 3	作业 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)

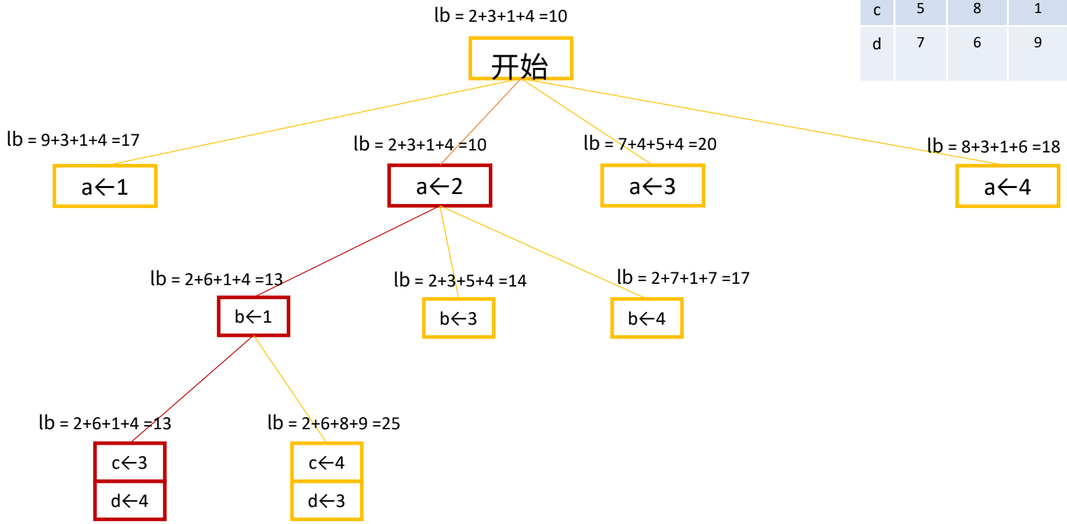
	Job 1	Job 2	Job 3	Job 4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4



Solution

分配问题 （分支与界限）

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



解决方案

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

分支限界法示例 2

	作业 1	作业 2	作业 3	作业 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

分支限界法示例 2

	作业 1	作业 2	作业 3	作业 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

分支限界法示例 2

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