1. 核心思想 Bruce force

DP 局部暴力搜索 | 大问题 (子问 / 子问 / 子问) 有多种技术

最优解 Dynamic Programming | Try everything

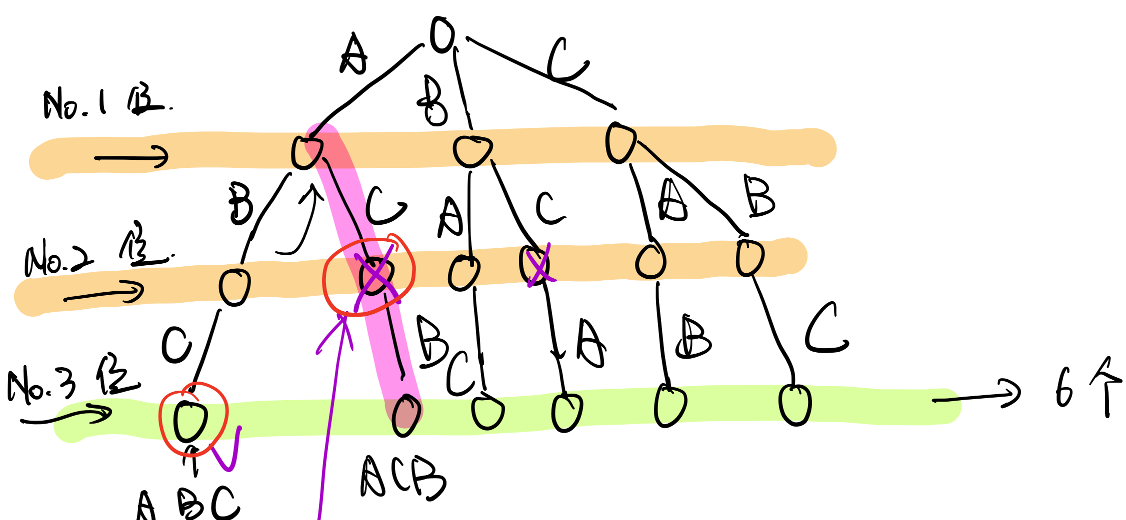
└ 基于数学归纳法

回溯 — find all of them { 排列问题 / 组合问题 }

2. Toy example

A. B. (C) — 在中间 | 1 2 3 | A | | | 3! = 6种

① 构造状态树



② 限制条件 Apply bounding function

③ 分支限界 vs. 回溯法
BFS DFS

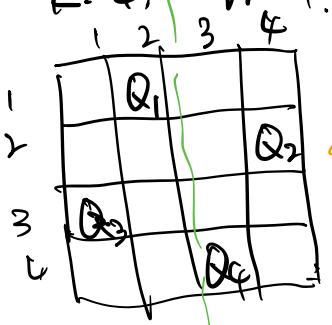
3. n-Queens 问题.

在 k * k 棋盘上，放 n 个 Queen，让它们不能彼此攻击.

求：有多少种放法？

不行 { 同一行 / 同一列 / 对角线.

例：k=4, n=4.

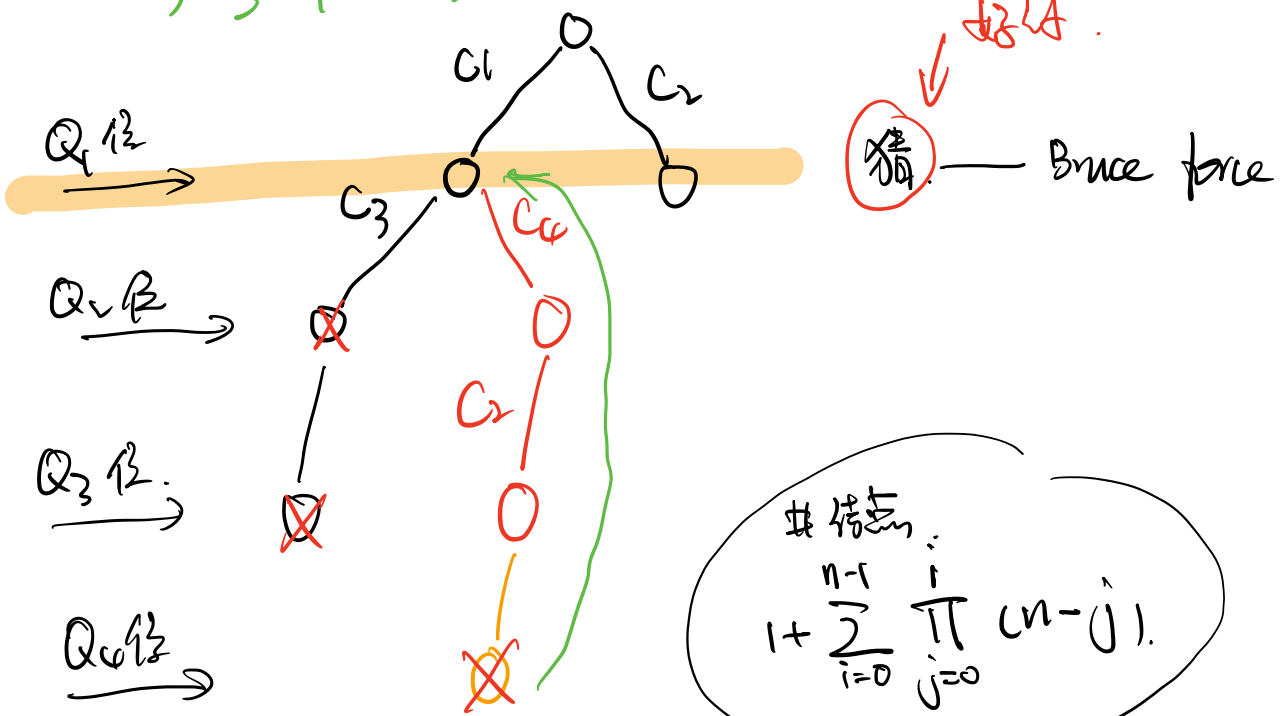


Q1 Q2 Q3 Q4 在哪一列？

→ (2, 4, 1, 3)

→ 3 1 4 2

选择：Q_k 放在哪一列？



* 解决回溯问题的范式.

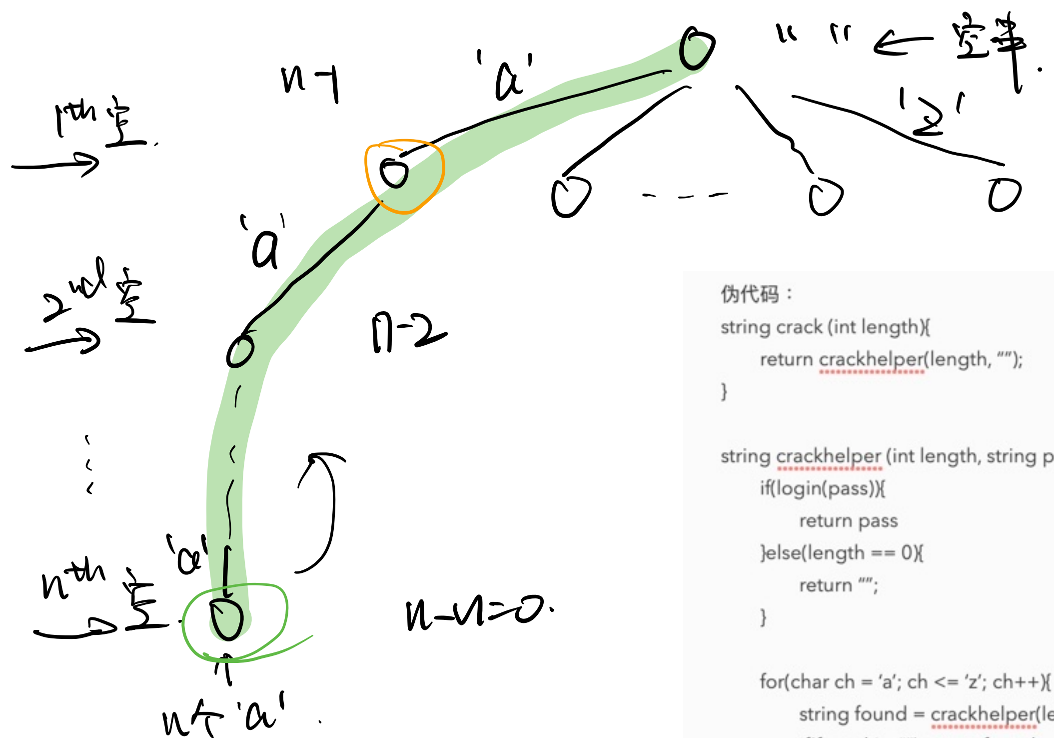
① base case. (停止节点) { 成功. / 失败.

② recursive case. {
a. 做选择 ← 穷举.
b. explore ← 新选择子问题.
c. 撤销

3. Cracker. (排序 + 找到答案就停下来).

• 构造一个什么样子字符串? → login(password) 为真

Assume: ① 不能为空. ② 52个英文字母. ③ 有最大长度 n.



伪代码：

```
string crack (int length){
    return crackhelper(length, "");
}

string crackhelper (int length, string pass){
    if(login(pass)){
        return pass
    }else(length == 0){
        return "";
    }

    for(char ch = 'a'; ch <= 'z'; ch++){
        string found = crackhelper(length -1, pass + ch);
        if(found != "") return found;
        found  = crackhelper(length -1, pass + CH);
        if(found != "") return found;
    }
    return "";
}
```

分析：

base case. ← 递归停止的地方.

- 成功. login(password) → True.
- 失败. 可用余数为0. → False

recursive case:

For 所有字母:

选择. ① add ch 到老密码 password → newpassword.

探索. ② recursive call cracker(newpassword)

改变. ③ remove ch

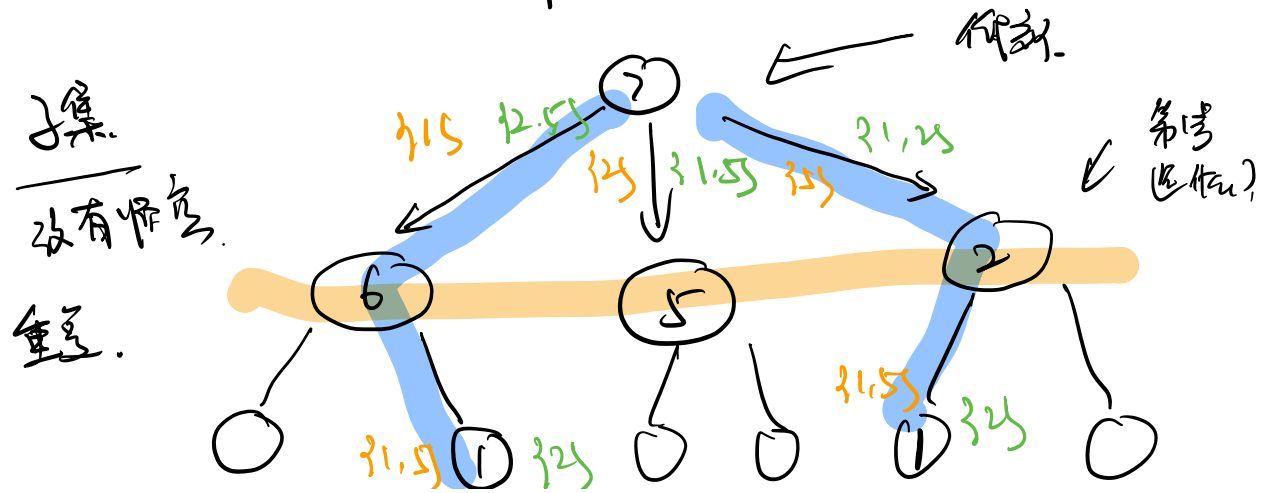
4. 24点 — sumup.

target value.

例. {1, 2, 5}. 7. → true.

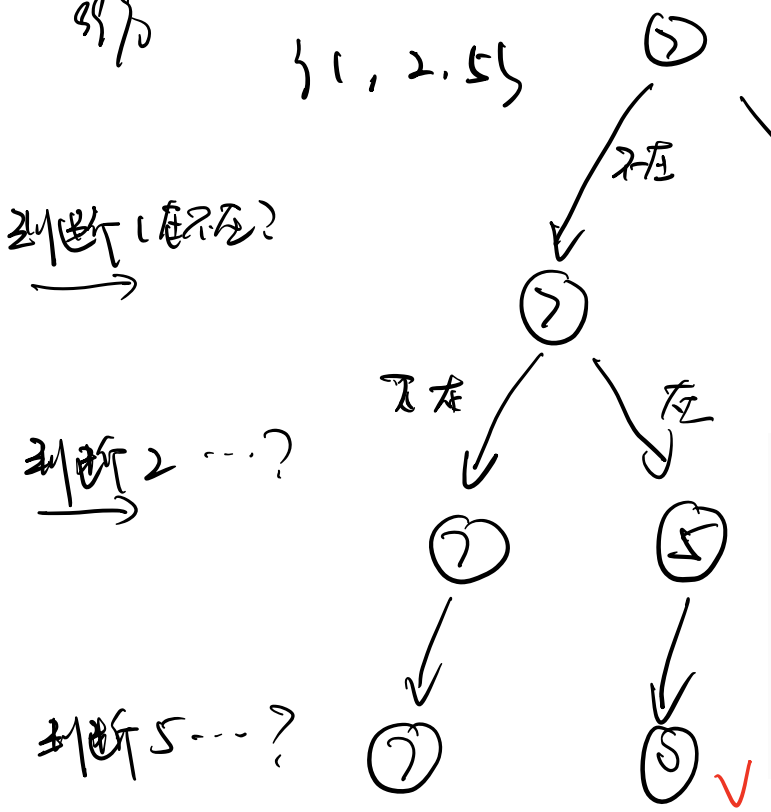
{1, 1, 3}. 6 → false.

已选中 { }. 可选择 { }.



→ 所有组合. ⇒ 判断元素是否在正确答案里?

{1, 2, 5}



2+4+6+1

=13

```
def sumup:
        if target is 0:
            return true;
        else if choices is empty or if target <0:
            return false;
    int temp = choice[0];
    choices.remove[0];
    if (sumup(target, choices) ||sumup(target - temp, choices))
    choices.insert(0,temp);
    return true;
    }
choices.insert(0,temp);
return false;
```

分析.

base case: { 成功 节点值为0.

失败. 可选list为0. & 节点值≠0.

recursive case:

① 选择: 任选可选元素. val

② explore: sump(target-val, choices)

③ backtrack: val 加回待选集

6. 分支限界 —— 组合优化.

最优解.
↑

C 代价函数: 以节点T为根节点的子树所有可行解的上界 ← 极大值.

判断 ⇐ 性质: 父节点代价 ≥ 子节点.

界: 当前找到的最优解

→ 停止分支
- ① 不满足约束条件.
- ② 代价函数 < 当前界.