从搜索到动规——记忆化搜索入门

主讲人 令狐冲

数字三角形 Triangle

http://www.lintcode.com/problem/triangle/

http://www.jiuzhang.com/solutions/triangle/

找出数字三角形中从上到下的一条最短路径

数字三角形 vs 二叉树

有 n 层的数字三角形和有 n 层的满二叉树
分别有多少个节点?
这里 n 代表层数

数字三角形 vs 二叉树

数字三角形 - O(n^2) 个节点

二叉树 - O(2^n) 个节点

DFS: Traverse

- 时间复杂度?
- A - O(n^2)
- B - O(2^n)
- C - O(n!)
- D - I don't know

```python
def minimumTotal(self, triangle):
    self.minimum = sys.maxsize
    self.traverse(triangle, 0, 0, 0)
    return self.minimum

def traverse(self, triangle, x, y, path_sum):
    if x == len(triangle):
        self.minimum = min(path_sum, self.minimum)
        return

    self.traverse(triangle, x + 1, y, path_sum + triangle[x][y])
    self.traverse(triangle, x + 1, y + 1, path_sum + triangle[x][y])
```

DFS: Divide & Conquer

- 时间复杂度?
- A - O(n^2)
- B - O(2^n)
- C - O(n!)
- D - I don’t know

```python
def minimumTotal(self, triangle):
    return self.divide_conquer(triangle, 0, 0)

def divide_conquer(self, triangle, x, y):
    if x == len(triangle):
        return 0

    left = self.divide_conquer(triangle, x + 1, y)
    right = self.divide_conquer(triangle, x + 1, y + 1)
    return min(left, right) + triangle[x][y]
```

DFS: Divide & Conquer + HashMap

- 时间复杂度?
- A - O(n^2)
- B - O(2^n)
- C - O(n!)
- D - I don’t know

```python
def minimumTotal(self, triangle):
    return self.divide_conquer(triangle, 0, 0, {})

# 函数返回从 x, y 出发, 走到最底层的最短路径值
# memo 中 key 为二元组 (x, y)
# memo 中 value 为从 x, y 走到最底层的最短路径值
def divide_conquer(self, triangle, x, y, memo):
    if x == len(triangle):
        return 0

    # 如果找过了, 就不要再找了, 直接把之前找到的值返回
    if (x, y) in memo:
        return memo[(x, y)]

    left = self.divide_conquer(triangle, x + 1, y, memo)
    right = self.divide_conquer(triangle, x + 1, y + 1, memo)

    # 在 return 之前先把这次找到的最短路径值记录下来
    # 避免之后重复搜索
    memo[(x, y)] = min(left, right) + triangle[x][y]
    return memo[(x, y)]
```

记忆化搜索 Memoization Search

注意不是 Memorization

使用 HashMap 记录搜索的中间结果从而避免重复计算的算法

就叫做记忆化搜索

什么记忆化搜索?

将函数的计算结果保存下来，下次通过同样的参数访问时，直接返回保存下来的结果

问:

1. 对这个函数有什么限制条件没有?
2. 和系统设计中的什么很像?

记忆化搜索通常能够将指数级别的时间复杂度降低到多项式级别。

记忆化搜索的本质：动态规划

动态规划为什么会快?

动态规划与分治的区别?

重复计算!

记忆化搜索 = 动态规划(DP)

记忆化搜索是动态规划的一种实现方式
记忆化搜索是用搜索的方式实现了动态规划
因此记忆化搜索，就是动态规划

Bash 游戏

https://www.lintcode.com/problem/bash-game/

https://www.jiuzhang.com/solutions/bash-game/

记忆化搜索非常适合博弈型动态规划

代码

```python
def canWinBash(self, n):
    return self.memo_search(n, {})

def memo_search(self, n, memo):
    if n <= 3:
        return True
    if n in memo:
        return memo[n]

    for i in range(1, 4):
        if not self.memo_search(n - i, memo):
            memo[n] = True
            return True
    memo[n] = False
    return False
```

```java
public boolean canWinBash(int n) {
    return memoSearch(n, new HashMap<Integer, Boolean>());
}

private boolean memoSearch(int n, Map<Integer, Boolean> memo) {
    if (n <= 3) {
        return true;
    }
    if (memo.containsKey(n)) {
        return memo.get(n);
    }

    for (int i = 1; i <= 3; i++) {
        if (!memoSearch(n - i, memo)) {
            memo.put(n, true);
            return true;
        }
    }
    return false;
}
```

这份代码有什么问题?

StackOverflow

为什么？n 可能很大，深度达到 O(n) 级别

如果时间复杂度和递归深度都是 O(n) 级别会栈溢出

如果时间复杂度是 O(n^2) 级别，递归深度是 O(n) 级别就不会溢出

为什么?

记忆化搜索的缺点

不适合解决 O(n) 时间复杂度的 DP 问题

因为会有 StackOverflow 的风险

Bash 游戏的解决思路

通过记忆化搜索的代码得到 n=1,2,3,4,... 的小数据结果

找规律，发现 n % 4 == 0 是 false，其他是 true

进一步想到 n % 4 == 0 时，先手取 x，后手取 4-x 先手必败

反之先手取 n % 4 个石子就让对手面对必败局面