Memoization: 3150ms -> 130ms -> 44ms (Python)

StefanPochmann ★ 46834 October 16, 2015 3:29 PM 12.7K VIEWS

54

Without memoization:
~ 3150 ms

```
class Solution(object):
    def canWin(self, s):
        return any(s[i:i+2] == '++' and not self.canWin(s[:i] + '-' + s[i+2:])
                   for i in range(len(s)))
```

With memoization:
~ 130 ms

```
class Solution(object):
    _memo = {}
    def canWin(self, s):
        memo = self._memo
        if s not in memo:
            memo[s] = any(s[i:i+2] == '++' and not self.canWin(s[:i] + '-' + s[i+2:])
                          for i in range(len(s)))
        return memo[s]
```

With memoization:
~ 140 ms

The previous one reuses memoized results from previous test cases, but that's not why it's fast. It's almost as fast without that.

```
class Solution(object):
    def canWin(self, s):
        memo = {}
        def can(s):
            if s not in memo:
                memo[s] = any(s[i:i+2] == '++' and not can(s[:i] + '-' + s[i+2:])
                              for i in range(len(s)))
            return memo[s]
        return can(s)
```

With memoization and counts instead of a string:
~ 44 ms

Using tuples like `(2, 3)` to represent a state instead of strings like `"-+++---++--"`.

```
class Solution(object):
    def canWin(self, s):
        memo = {}
        def can(piles):
            piles = tuple(sorted(p for p in piles if p >= 2))
            if piles not in memo:
                memo[piles] = any(not can(piles[:i] + (j, pile-2-j) + piles[i+1:])
                                  for i, pile in enumerate(piles)
                                  for j in range(pile - 1))
            return memo[piles]
        return can(map(len, re.findall(r'\+\++', s)))
```

python fast memoization

Comments: 16 Best Most Votes Newest to Oldest Oldest to Newest

Type comment here... (Markdown is supported)

Post