让面试官眼前一亮的算法
——记忆化搜索

主讲人 令狐冲
课程版本 v7.0

什么是记忆化搜索

在函数返回前，记录函数的返回结果

在下一次以**同样参数**访问函数时直接返回记录下的结果

记忆化搜索函数的三个特点

函数有返回值

函数返回结果之和输入参数相关，和其他全局状态无关

参数列表中传入哈希表或者其他用于记录计算结果的数据结构

记忆化搜索 vs 动态规划

记忆化搜索是动态规划的**一种实现方式**

动态规划的另外一种实现方式是多重循环（下节课）

所以记忆化搜索**就是**动态规划

动态规划的核心思想：由大化小

动态规划的算法思想：大规模问题的依赖于小规模问题的计算结果

类似思想算法的还有：递归，分治法

独孤九剑 —— 破箭式

三种适用动态规划的场景

三种不适用动态规划的场景

三种适用DP的场景

求最优值

求方案数

求可行性

三种适用动规的场景

- 求最值
  - dp[] 的值的类型是最优值的类型
  - dp[大问题] = max{dp[小问题1], dp[小问题2], ...}
  - dp[大问题] = min{dp[小问题1], dp[小问题2], ...}
- 求方案数
  - dp[] 的值的类型是方案数（整数）
  - dp[大问题] = ∑(dp[小问题1], dp[小问题2], ...)
  - ∑=sum
- 求可行性
  - dp[] 的值是 true / false
  - dp[大问题] = dp[小问题1] or dp[小问题2] or ...
  - 代码通常用 for 小问题 if dp[小问题] == true then break 的形式实现

三种不适用DP的场景

求所有的具体方案

输入数据是无序的

暴力算法时间复杂度已经是多项式级别

三种不适用 DP 的场景

- 求出所有的具体方案
  - http://www.lintcode.com/problem/palindrome-partitioning/
  - 只求出一个具体方案还是可以用 DP 来做的（下节课）
  - 该判断标准成功率 99%
- 输入数据是无序的
  - http://www.lintcode.com/problem/longest-consecutive-sequence/
  - 背包类动态规划不适用此判断条件
  - 除去背包问题后，该判断标准成功率 60-70%，有一些题可以先排序之后按序处理
- 暴力算法的复杂度已经是多项式级别
  - http://www.lintcode.com/problem/largest-rectangle-in-histogram/
  - 动态规划擅长与优化指数级别复杂度(2^n,n!)到多项式级别复杂度(n^2,n^3)
  - 不擅长优化n^3到n^2
  - 该判断标准成功率 80%
- 则 **极不可能** 使用动态规划求解

Wildcard Matching

http://www.lintcode.com/problem/wildcard-matching/

http://www.jiuzhang.com/solution/wildcard-matching/

类别：匹配型动态规划

适用场景：求可行性

代码


```java
public boolean isMatch(String s, String p) {
    if (s == null || p == null) {
        return false;
    }

    // boolean[][] memo = new boolean[s.length()][p.length()];
    // boolean[][] visited = new boolean[s.length()][p.length()];
    // return isMatchHelper(s, 0, p, 0, memo, visited);

    return isMatchHelper(s, 0, p, 0);
}

private boolean allStar(String p, int pIndex) {
    for (int i = pIndex; i < p.length(); i++) {
        if (p.charAt(i) != '*') {
            return false;
        }
    }
    return true;
}

private boolean charMatch(char sFirstChar, char pFirstChar) {
    return (sFirstChar == pFirstChar || pFirstChar == '?');
}
```

```java
private boolean isMatchHelper(String s, int sIndex,
                              String p, int pIndex) {
    if (pIndex == p.length()) {
        // 如果 p 从 pIndex 开始是空字符串了，那么 s 必须从 sIndex
            是空字符串才能匹配上
        return sIndex == s.length();
    }

    if (sIndex == s.length()) {
        // 如果 s 从 sIndex 是空，那么 p 必须全是 *
        return allStar(p, pIndex);
    }

    char sFirstChar = s.charAt(sIndex);
    char pFirstChar = p.charAt(pIndex);

    if (pFirstChar != '*') {
        return charMatch(sFirstChar, pFirstChar) && isMatchHelper(s, sIndex +
            1, p, pIndex + 1);
    }

    for (int i = sIndex; i <= s.length(); i++) {
        if (isMatchHelper(s, i, p, pIndex + 1)) {
            return true;
        }
    }

    return false;
}
```

代码


```python
def isMatch(self, s, p):
    if s == None or p == None:
        return False

    # memo = [[0] * len(p) for _ in range(len(s))]
    # visited = [[0] * len(p) for _ in range(len(s))]
    # return self.is_match_helper(s, 0, p, 0, memo, visited)

    return self.is_match_helper(s, 0, p, 0)

def all_star(self, p, p_index):
    for i in range(p_index, len(p)):
        if p[i] != '*':
            return False
    return True

def char_match(self, s_first_char, p_first_char):
    return s_first_char == p_first_char or p_first_char == '?'
```

```python
def is_match_helper(self, s, s_index, p, p_index):
    if p_index == len(p):
        # 如果 p 从 pIndex 开始是空字符串了，那么 s 必须从 sIndex
            是空字符串才能匹配上
        return s_index == len(s)

    if s_index == len(s):
        # 如果 s 从 sIndex 是空，那么 p 必须全是 *
        return self.all_star(p, p_index)

    s_first_char = s[s_index]
    p_first_char = p[p_index]

    if p_first_char != '*':
        return self.char_match(s_first_char, p_first_char) and self
            .is_match_helper(s, s_index + 1, p, p_index + 1)

    for i in range(s_index, len(s) + 1):
        if self.is_match_helper(s, i, p, p_index + 1):
            return True

    return False
```

代码


```java
private boolean isMatchHelper(String s, int sIndex,
                              String p, int pIndex) {
    if (pIndex == p.length()) {
        // 如果 p 从 pIndex 开始是空字符串了，那么 s 必须从 sIndex
            是空字符串才能匹配上
        return sIndex == s.length();
    }

    if (sIndex == s.length()) {
        // 如果 s 从 sIndex 是空，那么 p 必须全是 *
        return allStar(p, pIndex);
    }

    char sFirstChar = s.charAt(sIndex);
    char pFirstChar = p.charAt(pIndex);

    if (pFirstChar == '*') {
        return isMatchHelper(s, sIndex + 1, p, pIndex) ||
               isMatchHelper(s, sIndex, p, pIndex + 1);
    }

    return charMatch(sFirstChar, pFirstChar) &&
           isMatchHelper(s, sIndex + 1, p, pIndex + 1);
}
```

```python
def is_match_helper(self, s, s_index, p, p_index):
    if p_index == len(p):
        # 如果 p 从 pIndex 开始是空字符串了，那么 s 必须从 sIndex
            是空字符串才能匹配上
        return s_index == len(s)

    if s_index == len(s):
        # 如果 s 从 sIndex 是空，那么 p 必须全是 *
        return self.all_star(p, p_index)

    s_first_char = s[s_index]
    p_first_char = p[p_index]

    if p_first_char == '*':
        return self.is_match_helper(s, s_index + 1, p, p_index) or
            self.is_match_helper(s, s_index, p, p_index + 1)

    return self.char_match(s_first_char, p_first_char) and self
        .is_match_helper(s, s_index + 1, p, p_index + 1)
```

代码


```java
private boolean isMatchHelper(String s, int sIndex,
                              String p, int pIndex,
                              boolean[][] memo,
                              boolean[][] visited) {
    if (pIndex == p.length()) {
        // 如果 p 从 pIndex 开始是空字符串了，那么 s 必须从 sIndex 是空字符串才能匹配上
        return sIndex == s.length();
    }

    if (sIndex == s.length()) {
        // 如果 s 从 sIndex 是空，那么 p 必须全是 *
        return allStar(p, pIndex);
    }

    if (visited[sIndex][pIndex]) {
        return memo[sIndex][pIndex];
    }

    char sFirstChar = s.charAt(sIndex);
    char pFirstChar = p.charAt(pIndex);
    boolean match;

    if (pFirstChar == '*') {
        match = isMatchHelper(s, sIndex + 1, p, pIndex, memo, visited) ||
                isMatchHelper(s, sIndex, p, pIndex + 1, memo, visited);
    } else {
        match = charMatch(sFirstChar, pFirstChar) &&
                isMatchHelper(s, sIndex + 1, p, pIndex + 1, memo, visited);
    }

    memo[sIndex][pIndex] = match;
    visited[sIndex][pIndex] = true;
    return match;
}
```

```python
def is_match_helper(self, s, s_index, p, p_index, memo, visited):
    if p_index == len(p):
        # 如果 p 从 pIndex 开始是空字符串了，那么 s 必须从 sIndex 是空字符串才能匹配上
        return s_index == len(s)

    if s_index == len(s):
        # 如果 s 从 sIndex 是空，那么 p 必须全是 *
        return self.all_star(p, p_index)

    if visited[s_index][p_index]:
        return memo[s_index][p_index]

    s_first_char = s[s_index]
    p_first_char = p[p_index]
    match = False

    if p_first_char == '*':
        match = self.is_match_helper(s, s_index + 1, p, p_index, memo, visited) or \
        self.is_match_helper(s, s_index, p, p_index + 1, memo, visited)
    else:
        match = self.char_match(s_first_char, p_first_char) and \
        self.is_match_helper(s, s_index + 1, p, p_index + 1, memo, visited)

    memo[s_index][p_index] = match
    visited[s_index][p_index] = True
    return match
```

Follow up:
Regular Expression Matching

http://www.lintcode.com/problem/regular-expression-matching/

http://www.jiuzhang.com/solution/regular-expression-matching/

面试是一定不会让你做完整版的 Regular Expression 的

所以一定是阉割版的

面试评分标准

Strong Hire: 两个都答出来，且写出来，Bug Free or Bug 很少

Hire / Weak Hire: 两个都答出来，写完第一个，第二个能基本在第一个的基础上改完，允许有一些提示和少量 Bug

No Hire: 没写完，或者需要很多提示

Strong No: 第一个都没写完

记忆化搜索时间复杂度

=动态规划的时间复杂度

=O(状态总数 * 计算每个状态的时间耗费)

休息 5 分钟

Take a break

Word Pattern II

http://www.lintcode.com/problem/word-pattern-ii/

http://www.jiuzhang.com/solutions/word-pattern-ii/

这个题是否可以记忆化？

代码


```java
public boolean wordPatternMatch(String pattern, String string) {
    Map<Character, String> mapping = new HashMap<>();
    Set<String> used = new HashSet<>();
    return isMatch(pattern, string, mapping, used);
}
```

```java
private boolean isMatch(String pattern, String string, Map<Character, String> mapping, Set<String> used) {
    if (pattern == null || pattern.length() == 0) {
        return string.length() == 0;
    }

    char ch = pattern.charAt(0);
    if (mapping.containsKey(ch)) {
        String word = mapping.get(ch);
        if (!string.startsWith(word)) {
            return false;
        }
        return isMatch(pattern.substring(1), string.substring(word.length()), mapping, used);
    }

    for (int i = 0; i < string.length(); i++) {
        String word = string.substring(0, i + 1);
        if (used.contains(word)) {
            continue;
        }

        used.add(word);
        mapping.put(ch, word);
        if (string.length() < word.length()) {
            return false;
        }
        if (isMatch(pattern.substring(1), string.substring(i + 1), mapping, used)) {
            return true;
        }
        mapping.remove(ch);
        used.remove(word);
    }

    return false;
}
```

代码


```python
def wordPatternMatch(self, pattern, string):
    return self.is_match(pattern, string, {}, set())

def is_match(self, pattern, string, mapping, used):
    if not pattern:
        return not string

    char = pattern[0]
    if char in mapping:
        word = mapping[char]
        if not string.startswith(word):
            return False
        return self.is_match(pattern[1:], string[len(word):], mapping, used)

    for i in range(len(string)):
        word = string[:i + 1]
        if word in used:
            continue

        used.add(word)
        mapping[char] = word

        if self.is_match(pattern[1:], string[i + 1:], mapping, used):
            return True

        del mapping[char]
        used.remove(word)

    return False
```

Word Break

右边的代码正确性没有问题

但是存在一个问题导致其无法通过测试

这个问题是什么？

```python
def is_possible(self, s, index, max_length, word_set, memo):
    if index in memo:
        return memo[index]

    if index == len(s):
        return True

    memo[index] = False
    for i in range(index, len(s)):
        if i + 1 - index > max_length:
            break
        word = s[index : i + 1]
        if word not in word_set:
            continue
        if self.is_possible(s, i + 1, max_length, word_set, memo):
            memo[index] = True
            break

    return memo[index]

def get_max_length(self, word_set):
    max_length = 0
    for word in word_set:
        max_length = max(max_length, len(word))
    return max_length
```

Word Break

右边的代码正确性没有问题

但是存在一个问题导致其无法通过测试

这个问题是什么？

```
private boolean isPossible(String s, int index, int maxLength, Set<String> wordSet, Map<Integer, Boolean> memo) {
    if (memo.containsKey(index)) {
        return memo.get(index);
    }

    if (index == s.length()) {
        return true;
    }

    memo.put(index, false);
    for (int i = index; i < s.length(); i++) {
        if (i + 1 - index > maxLength) {
            break;
        }
        String word = s.substring(index, i + 1);
        if (!wordSet.contains(word)) {
            continue;
        }
        if (isPossible(s, i + 1, maxLength, wordSet, memo)) {
            memo.put(index, true);
            break;
        }
    }

    return memo.get(index);
}

private int getMaxLength(Set<String> wordSet) {
    int maxLength = 0;
    for (String word : wordSet) {
        maxLength = Math.max(maxLength, word.length());
    }
    return maxLength;
}
```

记忆化搜索的缺陷

递归深度太深，导致 StackOverflow

Word Break II

http://www.lintcode.com/problem/word-break-ii/

http://www.jiuzhang.com/solution/word-break-ii/

不适用场景：求出所有具体方案而非方案总数

但是可以使用动态规划进行优化

优化方案1

用 Word Break 这个题的思路

使用 memo[i] 代表从 i 开始的后缀是否能够被 break

在 DFS 找所有方案的时候，通过 memo 可以进行**可行性剪枝**

代码


```python
def wordBreak(self, s, wordDict):
    max_length = self.get_max_length(wordDict)
    # use dfs to find all break path
    results = []
    self.dfs(s, 0, max_length, wordDict, {}, [], results)
    return results

def is_possible(self, s, index, max_length, word_set, memo):
    if index in memo:
        return memo[index]
    if index == len(s):
        memo[index] = True
        return True

    memo[index] = False
    for i in range(index, len(s)):
        if i + 1 - index > max_length:
            break
        word = s[index: i + 1]
        if word not in word_set:
            continue
        if self.is_possible(s, i + 1, max_length, word_set, memo):
            memo[index] = True
            break

    return memo[index]
```

```python
def get_max_length(self, word_set):
    max_length = 0
    for word in word_set:
        max_length = max(max_length, len(word))
    return max_length

def dfs(self, s, index, max_length, word_set, memo, path, results):
    if index == len(s):
        results.append(" ".join(path))
        return
    # prunning
    if not self.is_possible(s, index, max_length, word_set, memo):
        return

    for i in range(index, len(s)):
        if i + 1 - index > max_length:
            break
        word = s[index: i + 1]
        if word not in word_set:
            continue
        path.append(word)
        self.dfs(s, index + len(word), max_length, word_set, memo,
            path, results)
        path.pop()
```

代码


```java
public List<String> wordBreak(String s, Set<String> wordSet) {
    Map<Integer, Boolean> memo = new HashMap<>();
    int maxLength = getMaxLength(wordSet);
    // use dfs to find all break path
    List<String> results = new ArrayList<>();
    dfsGetResults(s, 0, maxLength, wordSet, memo, new ArrayList<String>(), results);
    return results;
}

private boolean isPossible(String s, int index, int maxLength, Set<String> wordSet, Map<Integer, Boolean> memo) {
    if (memo.containsKey(index)) {
        return memo.get(index);
    }

    if (index == s.length()) {
        return true;
    }

    memo.put(index, false);
    for (int i = index; i < s.length(); i++) {
        if (i + 1 - index > maxLength) {
            break;
        }
        String word = s.substring(index, i + 1);
        if (!wordSet.contains(word)) {
            continue;
        }
        if (isPossible(s, i + 1, maxLength, wordSet, memo)) {
            memo.put(index, true);
            break;
        }
    }

    return memo.get(index);
}
```

```java
private int getMaxLength(Set<String> wordSet) {
    int maxLength = 0;
    for (String word : wordSet) {
        maxLength = Math.max(maxLength, word.length());
    }
    return maxLength;
}

private void dfsGetResults(String s, int index, int maxLength, Set<String> wordSet, Map<Integer, Boolean> memo,
        ArrayList<String> used, List<String> results) {
    if (index == s.length()) {
        results.add(join(" ", used));
        return;
    }
    // prunning
    if (!isPossible(s, index, maxLength, wordSet, memo)) {
        return;
    }

    for (int i = index; i < s.length(); i++) {
        if (i + 1 - index > maxLength) {
            break;
        }
        String word = s.substring(index, i + 1);
        if (!wordSet.contains(word)) {
            continue;
        }
        used.add(word);
        dfsGetResults(s, index + word.length(), maxLength, wordSet, memo, used, results);
        used.remove(used.size() - 1);
    }
}
```

代码


```
private String join(String str, ArrayList<String> used) {
    StringBuilder sb = new StringBuilder();
    boolean isFirst = true;
    for (String word : used) {
        if (isFirst) {
            isFirst = false;
        } else {
            sb.append(str);
        }
        sb.append(word);
    }
    return sb.toString();
}
```

优化方案 2

直接使用 memo[i] 记录从位置 i 开始的后缀
能够被 break 出来的所有方案

代码


```java
public ArrayList<String> wordBreak(String s, Set<String> wordDict) {
    // Note: The Solution object is instantiated only once and is reused by each test case.
    Map<String, ArrayList<String>> memo = new HashMap<String, ArrayList<String>>();
    return wordBreakHelper(s, wordDict, memo);
}

public ArrayList<String> wordBreakHelper(String s,
                                         Set<String> wordDict,
                                         Map<String, ArrayList<String>> memo){
    if (memo.containsKey(s)) {
        return memo.get(s);
    }

    ArrayList<String> partitions = new ArrayList<String>();

    if (s.length() == 0) {
        return partitions;
    }

    if (wordDict.contains(s)) {
        partitions.add(s);
    }

    for (int i = 1; i < s.length(); i++){
        String prefix = s.substring(0, i);
        if (!wordDict.contains(prefix)) {
            continue;
        }

        String suffix = s.substring(i);
        ArrayList<String> subPartitions = wordBreakHelper(suffix, wordDict, memo);

        for (String partition : subPartitions){
            partitions.add(prefix + " " + partition);
        }
    }

    memo.put(s, partitions);
    return partitions;
}
```

代码


```python
def wordBreak(self, s, wordDict):
    return self.dfs(s, wordDict, {})

def dfs(self, s, wordDict, memo):
    if s in memo:
        return memo[s]

    partitions = []

    if len(s) == 0:
        return partitions

    for i in range(1, len(s)):
        prefix = s[:i]
        if prefix not in wordDict:
            continue

        suffix = s[i:];
        sub_partitions = self.dfs(suffix, wordDict, memo)
        for partition in sub_partitions:
            partitions.append(prefix + " " + partition)

    if s in wordDict:
        partitions.append(s)

    memo[s] = partitions
    return partitions
```

极端情况

以上两种方法在极端情况下是否能有优化效果呢？

s = “aaaaaaaaaa...”

dict = {“a”, “aa”, “aaa”, ...}

Word Break II 的面试评分标准

Strong Hire: DFS+DP优化

Hire / Weak Hire: DFS 能写完，且 Bug free or Bug 不多，不需要提示 or 需要少量提示

No Hire: DFS 写不完，或者需要很多提示

Strong No: 啥都想不出

* Palindrome Partitioning

http://www.lintcode.com/problem/palindrome-partitioning/

http://www.jiuzhang.com/solutions/palindrome-partitioning/

一个类似 Word Break II 的题

但是使用记忆化搜索优化效果甚微

Word Break III

https://www.lintcode.com/problem/word-break-iii

https://www.jiuzhang.com/solution/word-break-iii

类别：前缀型/划分型动态规划

适用场景：求方案总数

代码


```java
public int wordBreak3(String s, Set<String> dict) {
    int maxLength;
    Set<String> lowerDict = new HashSet<>();
    maxLength = initialize(dict, lowerDict);
    return memoSearch(s.toLowerCase(), 0, maxLength, lowerDict, new HashMap<Integer, Integer>());
}

private int memoSearch(String s, int index, int maxLength, Set<String> lowerDict, Map<Integer, Integer> memo) {
    if (index == s.length()) {
        return 1;
    }

    if (memo.containsKey(index)) {
        return memo.get(index);
    }

    memo.put(index, 0);
    for (int i = index; i < s.length(); i++) {
        if (i + 1 - index > maxLength) {
            break;
        }
        String word = s.substring(index, i + 1);
        if (!lowerDict.contains(word)) {
            continue;
        }
        memo.put(index, memo.get(index) + memoSearch(s, i + 1, maxLength, lowerDict, memo));
    }

    return memo.get(index);
}

private int initialize(Set<String> dict, Set<String> lowerDict) {
    int maxLength = 0;
    for (String word : dict) {
        maxLength = Math.max(maxLength, word.length());
        lowerDict.add(word.toLowerCase());
    }
    return maxLength;
}
```

代码


```python
def wordBreak3(self, s, dict):
    max_length, lower_dict = self.initialize(dict)
    return self.memo_search(s.lower(), 0, max_length, lower_dict, {})

def memo_search(self, s, index, max_length, lower_dict, memo):
    if index == len(s):
        return 1

    if index in memo:
        return memo[index]

    memo[index] = 0
    for i in range(index, len(s)):
        if i + 1 - index > max_length:
            break
        word = s[index: i + 1]
        if word not in lower_dict:
            continue
        memo[index] += self.memo_search(s, i + 1, max_length, lower_dict, memo
            )

    return memo[index]

def initialize(self, dict):
    max_length = 0
    lower_dict = set()
    for word in dict:
        max_length = max(max_length, len(word))
        lower_dict.add(word.lower())
    return max_length, lower_dict
```