1 DP 基本

1.1 DP 的两种编程方法

以斐波那契数列计算为例: $F_1 = F_2 = 1, F_i = F_{i-1} + F_{i-1}, i >= 3$

简单递归

```
def classic_fib(n):
  if n == 1 or n == 2:
    return 1
  return classic_fib(n-1) + classic_fib(n-2)
```

1.1.1 自顶向下结合记忆化

依然采取递归的程序结构, 但是计算返回值前先检查待求结果是否已经被计算出. 解决每一个子问题之后存储结果, 需要时直接返回已经缓存的结果.

```
memo = dict()
  def memo_fib(n):
    global memo
    if n == 1 or n == 2:
      return 1
    elif n in memo.keys():
      return memo[n]
    memo[n] = memo_fib(n-1) + memo_fib(n-2)
    return memo[n]
```

1.1.2 自底向上结合制表

规避了递归编程. 在解决大问题时先解决小问题, 逐步递推到大问题. 递推过程一般需要填写多维表格 dp, 编码时会使用若干 for 循环体填表.

```
def dp_fib(n):
  dp = list()
  dp.append(1)
  dp.append(1)
  for i in range(2, n):
    dp.append(dp[i-2] + dp[i-1])
  return dp[-1]
```

```
n = 10
print(classic_fib(n))
print(memo_fib(n))
print(dp_fib(n))
>>
55
55
55
```

1.2 DP 的设计与实现