长的像递归，只不过子递归已经算完了

例一：钢条切割

> 每段长度都有一个价格，怎样切割总收益最大

2^n-1种切割方案都要考虑，但可利用小结果

算法1：自顶向下

> DAG

```
CutRodRec(prices,n):
    if (n==0)
      return 0
    r = -INF
    for (i=1 to n)
      r = Max(r, prices[i]+CutRodRec(prices,n-i))
    return r
```

- 如果要存储每个n的最优方案r[i]

```
CutRodRecMemAux(prices,r,n):
    for (i=0 to n)
      r[i] = -INF
    if (r[n]>0)
      return r[n]
    if (n==0)
      q = 0
    else
      q = -INF
      for (i=1 to n)
        q = Max(q, prices[i]+CutRodRecAux(prices,r,n-i))
    r[n] = q//只有记录作用，未重复调用
    return q
```

- Θ（nn）

算法2：自底向上

> Floyd-Warshall也是自底向上

```
PrintOpt(cuts,n):
while (n>0)
    Print cuts[n]
    n = n - cuts[n]
```

```
CutRodIter(prices,n):
    r[0] = 0
    for (i=1 to n)
        q = -INF
        for (j=1 to i)
            q = Max(q, prices[j] + r[i-j])//调用之前已赋值
            cuts[i]=j
        r[i] = q
    return r[n]
```

- Θ（nn）

例二：矩阵乘法

> n个矩阵相乘找到最小代价
> Ai of size pi−1×pi

- 最优子结构

> 只有这个例子是中分方法

Let $m[i,j]$ be min cost for computing $A_iA_{i+1}\cdots A_j$

$$m[i,j]=\min_{i\le k<j}\left(m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\right)$$

```
MatrixChainDP(A1, A2,…,An):
for (i=1 to n)
  m[i,i] = 0
for (l=2 to n)
```

```
  for (i=1 to n-l+1)
    j = i+l-1
    m[i,j] = INF
    for (k=i to j-1)
       cost = m[i,k] + m[k+1,j] + pi-1*pk*pj//从小规模开始，重复利用
      if (cost < m[i,j])
         m[i,j] = cost
         s[i,j] = k
return <m,s>  //m是代价，s是拆分点
```

- O(n^3)
- 空间O（nn）

动态规划原理

步骤

1、解析问题结构

> Characterize the structure of solution

2、递归算式

> Recursively define the value of an optimal solution.

3、解决子问题

> Compute the value of an optimal solution.

4、最优解

> Construct an optimal solution.

时间

- 子问题总数*每种子问题选择=时间
- 自底向上

> 先找到子问题最优解，然后求原问题最优解

贪心&动态规划

- 贪心先做选择
- 动态规划先计算所有子问题最优解，然后以递归式形式合并子问题

重叠子问题

- 问题的递归算法总是重复处理相同的子问题

> 处理的不同，那是分治法

- 所以要对每个子问题的解决方法进行记录

例三：字符串相似度

> 防错误输入法
> 给定字符串A[1…m] B[1…n]

- 按最后一列情况分三类

- $(-, B[n])$: edit distance of $A[1\cdots m]$ and $B[1\cdots(n-1)]$
- $(A[m], B[n])$: edit distance of $A[1\cdots(m-1)]$ and $B[1\cdots(n-1)]$
- $(A[m], -)$: edit distance of $A[1\cdots(m-1)]$ and $B[1\cdots n]$

$$dist(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} dist(i,j-1)+1 \\ dist(i-1,j)+1 \\ dist(i-1,j-1)+I[A[i]=B[j]] \end{cases} & \text{otherwise} \end{cases}$$

```
1  EditDistDP(A[1…m],B[1…n]):
2  for (i=0 to m)
3     dist[i,0] = i
4  for (j=0 to n)
5     dist[0,j] = j
6  for (i=1 to m)
7     for (j=1 to n)
8        delDist = dist[i-1,j] + 1
9        insDist = dist[i,j-1] + 1
10       subDist = dist[i-1,j-1] + Diff(A[i],B[j])
11       dist[i,j] = Min(delDist,insDist,subDist)
12 return dist
```

例四：最大独立集MaxIS

> 已知树的根，求包含最多不相邻的点
>
> 树：直系亲属不同时存在

- 最优子结构：大图的MIS包含子图的MIS

> 如果要了父亲，那么爷爷和儿子都不能要了

Let $mis(T_u, 1)$ be size of MaxIS of T_u, s.t. u in the MaxIS.
Let $mis(T_u, 0)$ be size of MaxIS of T_u, s.t. u NOT in the MaxIS.

$$mis(T_u, 1) = 1 + \sum_{v \text{ is a child of } u} mis(T_v, 0)$$
$$mis(T_u, 0) = \sum_{v \text{ is a child of } u} mis(T_v)$$
$$mis(T_u) = \max\{mis(T_u, 0), mis(T_u, 1)\}$$

```
MaxIsDP(u):
mis1 = 1,mis0 = 0
for (each child v of u)
  mis1 = mis1 + MaxISDP(v).mis0
  mis0 = mis0 + MaxISDP(v).mis
mis = Max(mis0,mis1)
return <mis,mis0,mis1>
```

- O(V+E)=O(V)

例五：子集合求和

> 在X[1,n]中找任意数的和为T

- 原始递归O(2^n)
- 按包含第一个元素分类

If there is a solution S, either $X[1]$ is in it or not.
If $X[1] \in S$, then there is a solution to instance "$X[2 \cdots n], T - X[1]$
If $X[1] \notin S$, then there is a solution to instance "$X[2 \cdots n], T$".

Let $ss(i, t) = true$ iff instance "$X[i \cdots n], t$" has a solution.

$$ss(i,t) = \begin{cases} true & \text{if } t = 0 \\ ss(i+1,t) & \text{if } t < X[i] \\ false & \text{if } i > n \\ ss(i+1,t) \vee ss(i+1,t-X[i]) & \text{otherwise} \end{cases}$$

- n行T列都要遍历到

```
SubsetSumDP(X,T):
ss[n,0] = True
for (t=1 to T)//先筛一遍X[n]的具体值
  ss[n,t] = (X[n]==t)?True:False
for (i=n-1 downto 1)
  ss[i,0] = True//t=0初始化
  for (t=1 to X[i]-1)
    ss[i,t] = ss[i+1,t]
  for (t=X[i] to T)
    ss[i,t] = Or( ss[i+1,t], ss[i+1,t-X[i]] )
return ss[1,T]
```

- O(nT)

> 实际未必比暴力算法快
> 都不是多项式问题

证无最优子结构

> 最优子结构也可能是贪心

- 无权最短路径：有最优子结构

> 原因：子问题具有无关性，每个子问题之间不共享资源

- 无权最长路径：无最优子结构

15.3-6 假定你希望兑换外汇，你意识到与其直接兑换，不如进行多种外币的一系列兑换，最后兑换到你想要的那种外币，可能会获得更大收益。假定你可以交易 n 种不同的货币，编号为 1，2，…，n，兑换从 1 号货币开始，最终兑换为 n 号货币。对每两种货币 i 和 j，给定汇率 r_{ij}，意味着你如果有 d 个单位的货币 i，可以兑换 dr_{ij} 个单位的货币 j。进行一系列的交易需要支付一定的佣金，金额取决于交易的次数。令 c_k 表示 k 次交易需要支付的佣金。证明：如果对所有 $k=1, 2, \cdots, n$，$c_k=0$，那么寻找最优兑换序列的问题具有最优子结构性质。然后请证明：如果佣金 c_k 为任意值，那么问题不一定具有最优子结构性质。

> 关键：分解出两段后，ck变化了

步骤

- 把最优的一段分解，分出的两个小的非最优
- 最好举反例