西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

动态规划与贪心算法

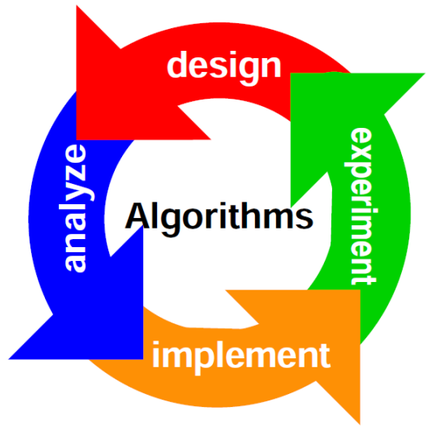
1、动态规划算法

2、贪心算法

11.3 常用算法

- 一、算法策略
- 枚举法、递推法、分治法、动态规划算法、贪心算法、回溯算法、分支限界法
- 二、数值计算
- 方程组求解、矩阵运算、函数积分、随机性模拟算法等
- 三、数据处理
- 数据拟合、参数估计、插值、线性规划、多元规划等

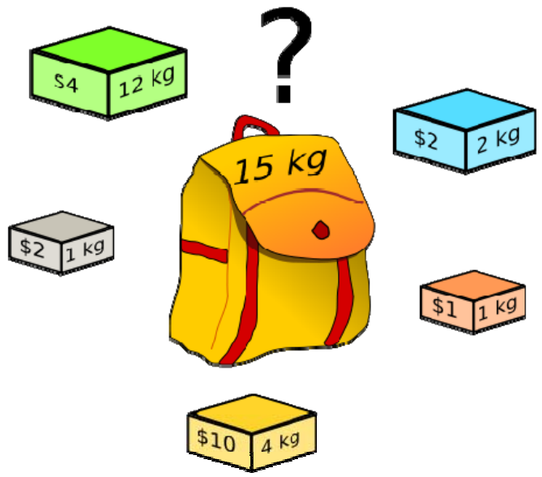
11.3 常用算法

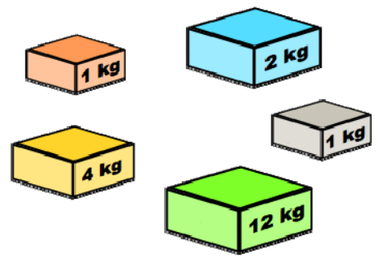


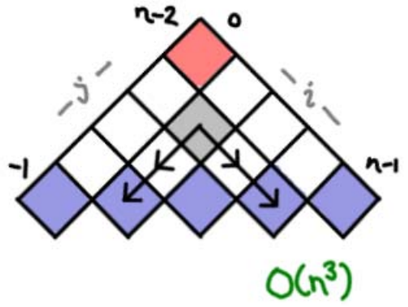
- 四、图论算法
- 搜索算法、最短路、网络流、二分图等算法
- 五、最优化理论
- 模拟退火算法、神经网络算法、遗传算法
- 六、图像处理算法

11.3.2 动态规划

- 动态规划（DP，dynamic programming）
- 动态规划是一种在数学、计算机科学和经济学中使用的，通过把原问题分解为相对简单的子问题的方式来求解复杂问题的方法。







11.3.2 动态规划

- 1. 基本思想
- 动态规划通常用于求解具有某种最优性质的问题。在这类问题中，可能会有许多可行解，每一个解都对应于一个值，因而希望找到具有最优值的解。

11.3.2 动态规划

- 动态规划算法与分治法类似，基本思想是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。

11.3.2 动态规划

- 但与分治法不同的是，动态规划经分解得到的子问题往往不是互相独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算了很多次。如果能够保存已解决的子问题的答案，在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。

11.3.2 动态规划

- 可以用一个表（数组）来记录所有已解的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中，这就是动态规划法的基本思路。具体的动态规划算法多种多样，但它们具有相同的填表格式。

11.3.2 动态规划

- 动态规划算法的有效性依赖于问题本身所具有的三个重要性质
- （1）最优子结构性质
- 当问题的最优解包含了其子问题的最优解时，称该问题具有最优子结构性质。

- （2）无后效性
- 即某阶段状态一旦确定，就不受这个状态以后决策的影响。也就是说，某状态以后的过程不会影响以前的状态，只与当前状态有关。

11.3.2 动态规划

- （3）子问题重叠性质
- 在用递归法自顶向下解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。即子问题之间是不独立的，一个子问题在下一阶段决策中可能被多次使用到。
- 该性质并不是动态规划适用的必要条件，但是如果没有这条性质，动态规划算法同其他算法相比就不具备优势

11.3.2 动态规划

- 2．基本步骤
- ①找出最优解的性质，并刻画其结构特征。
- ②递归地定义最优值（写出动态规划方程、状态转移方程）。
- ③以自底向上的方式计算出最优值。
- ④根据计算最优值时得到的信息，构造一个最优解。

11.3.2 动态规划

- 3．算法实现
- （1）使用动态规划求解问题，最重要的就是确定动态规划三要素：
- ①问题的阶段。
- ②每个阶段的状态。
- ③从前一个阶段转化到后一个阶段之间的递推关系。

11.3.2 动态规划

- 3．算法实现
- （2）递推关系式和递归方程
- 递推关系必须是从次小的问题开始到较大的问题之间的转化，因此，动态规划往往可以用递归程序来实现，不过因为递推可以充分利用前面保存的子问题的解来减少重复计算，所以对于大规模问题来说，有递归不可比拟的优势，这也是动态规划算法的核心之处。

11.3.2 动态规划

- 3．算法实现
- （3）最优决策表
- 确定了动态规划三要素，整个求解过程可以用一个最优决策表来描述。
- 最优决策表是一个二维表，其中行表示决策的阶段，列表示问题状态，表格需要填写的数据一般对应此问题的在某个阶段某个状态下的最优值。

11.3.2 动态规划

【例11.20】

给定一个正整数n，找出仅由1、3、4累加之和等于n的式子的数目。

例如：对于n=5，结果为6，因为：

5 = 1 + 1 + 1 + 1 + 1
  = 1 + 1 + 3
  = 1 + 3 + 1
  = 3 + 1 + 1
  = 1 + 4
  = 4 + 1

11.3.2 动态规划

例题分析

（1）定义子问题：

令D_n=由1、3、4累加之和等于n的式子的数目。

（2）寻找（递推的）状态方程：

①考虑一个可行解$n=x_1+x_2+\ldots+x_m$

②如果$x_m=1$，则剩余项之和必定是n-1

③因此，以$x_m=1$结束的式子的数目等于D_{n-1}

④考虑其他情形（$x_m=3$，$x_m=4$）

11.3.2 动态规划

例题分析

得：$D_n = D_{n-1} + D_{n-3} + D_{n-4}$

其中：

$D_0=1$

$D_n=0$，当n为负数

另外，设置$D_0=D_1=D_2=1$，$D_3=2$

11.3.2 动态规划

例11.20

```
1  #include <stdio.h>
2  int main()
3  {
4    int D[1000],n,i;
5    scanf("%d",&n);
6    D[0]=D[1]=D[2]=1;
7    D[3]=2;
8    for(i=4; i<=n; i++)
9      D[i]=D[i-1]+D[i-3]+D[i-4];
10   printf("%d\n",D[n]);
11   return 0;
12 }
```

11.3.2 动态规划

【例11.21】

数字三角形：有一个数字三角形（如下图）。现有一只蚂蚁从顶层开始向下走，每走下一级时，可向左下方向或右下方向走。求走到底层后它所经过的数的最大值。

11.3.2 动态规划

- （1）求解分析
- ①阶段——把问题划分为几步，在动态规划中，称为“划分阶段”。数字三角形中，每一层可看作是一个阶段。
- ②状态——每一阶段有多种选择，不同的选择会有不同的结果，把每阶段的不同情形叫做“状态”。每一阶段包括多个状态。数字三角形中，表示走到第j个数时所经过的数字和的最大值的变量叫做状态。

11.3.2 动态规划

- （1）求解分析
- ③状态转移方程——可以用一个递推式表示某阶段到下一阶段的递推关系，称为状态转移方程。状态转移方程一般含有max{}或min{}之类。
- ④决策——对方法进行选择。每个阶段都有一个决策，这样的选择是有范围的，称为“决策允许集合”。
- ⑤策略——完整决策的组合，“最优策略”即最佳的决策策略。

11.3.2 动态规划

例题分析

设二维数组A[i][j]，表示走到第i行第j个数时所经过的数字和的最大值。
例如对图中三角形，A[3][2]=max{1+6+2,1+3+2}。

11.3.2 动态规划

例题分析

可以得到递推关系

A[i][j]=p[i][j]+max{A[i-1][j-1],A[i-1][j]}

（注意A[i-1][j-1]或A[i-1][j]不存在时的处理），其中p[i][j]表示第i行第j个数的数值。

设置初始值：A[1][1]=p[1][1]。

最终求出A[i][j]，结果是max{A[i][j]}

11.3.2 动态规划

例11.21

```
 1 #include <stdio.h>
 2 int main()
 3 {
 4   int a[100][100],i=0,j=0,n,max=0,ul=0,ur=0;
 5   scanf("%d",&n);
 6   for(i=0; i<n ;i++)
 7     for(j=0 ; j<=i ; j++) {
 8       scanf("%d",&a[i][j]);
 9       ul = (i-1>=0)&&(j-1>=0) ? a[i-1][j-1] : 0;
10       ur = (i-1>=0) ? a[i-1][j] : 0;
11       a[i][j] = ( ul>ur ? ul : ur )+ a[i][j];
12       max = a[i][j] > max ? a[i][j] : max;
13   }
14   printf("%d",max);
15   return 0;
```

11.3.2 动态规划

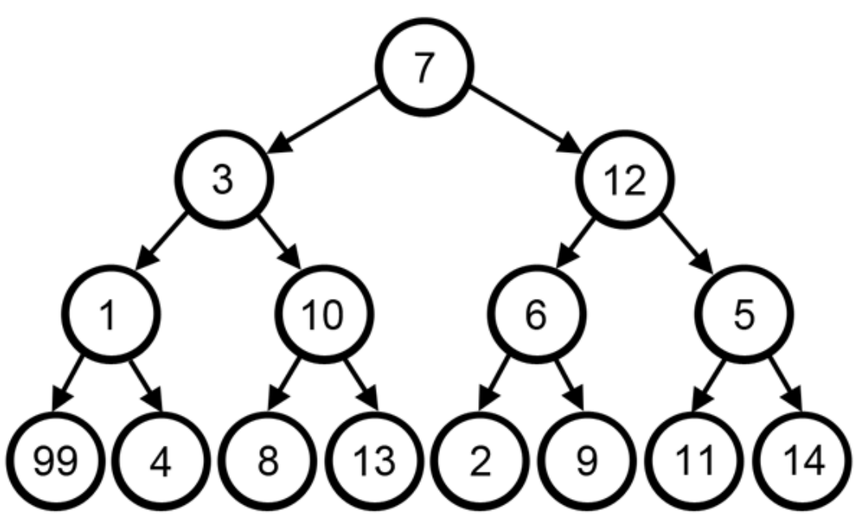
例11.21

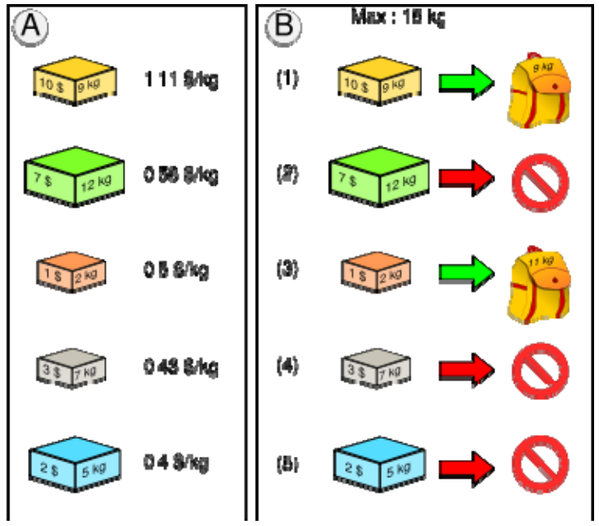
程序运行屏幕

```
23
5↙
1↙
6 3↙
8 2 6↙
2 1 6 5↙
3 2 4 7 6↙
```

11.3.3 贪心算法

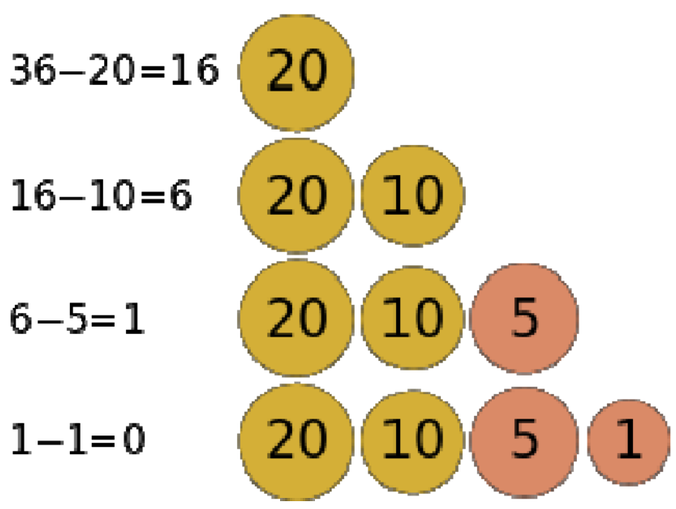
- 贪心算法（Greedy Algorithm）
- 在求最优解问题的过程中，依据某种贪心标准，从问题的初始状态出发，直接去求每一步的最优解，通过若干次的贪心选择，最终得出整个问题的最优解，这种求解方法就是贪心算法。





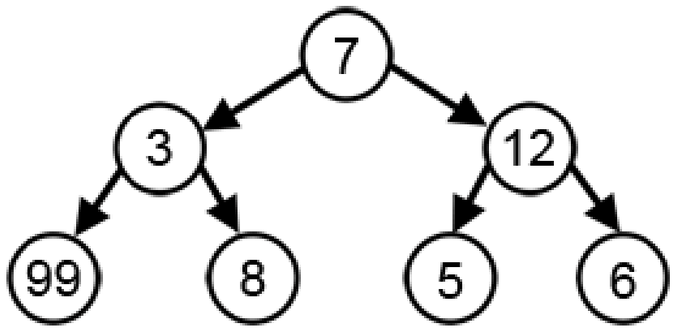
11.3.3 贪心算法

- 换零钱的时候会应用到贪心算法。例如把36元换成零钱：
- 20 > 10 > 5 > 1



11.3.3 贪心算法

- 1．基本思想
- 从贪心算法定义可以看出，贪心算法并不是从整体上考虑问题，它所做出的选择只是在某种意义上的局部最优解，而由问题自身的特性决定了运用贪心算法可以得到最优解。



11.3.3 贪心算法

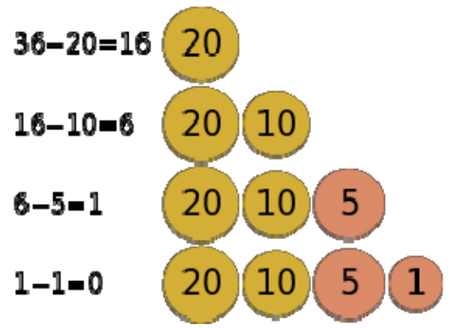
- 可以用贪心算法求解的问题一般具有两个重要的性质：
- （1）贪心选择性质
- 贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

- 对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

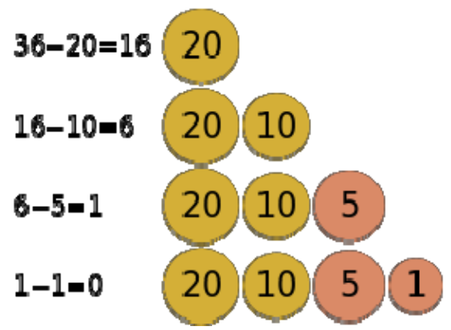
11.3.3 贪心算法

- （2）最优子结构性质
- 当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

- 2．求解过程
- ①候选集合C：为了构造问题的解决方案，有一个候选集合C作为问题的可能解，即问题的最终解均取自于候选集合C。
- 例如，在换零钱问题中，各种面值的货币｛20,10,5,1｝构成候选集合。



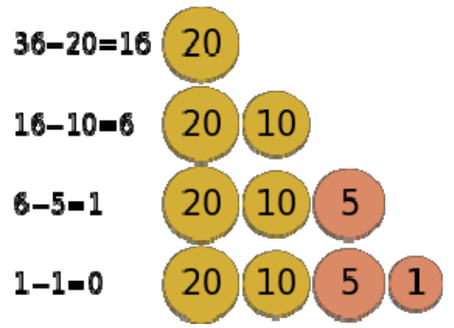
- 2．求解过程
- ②解集合S：随着贪心选择的推进，解集合S不断扩展，直到构成一个满足问题的完整解。
- 例如，已换的零钱构成解集合。



11.3.3 贪心算法

- 2．求解过程
- ③解决函数Solution：检查解集合S是否构成问题的完整解。

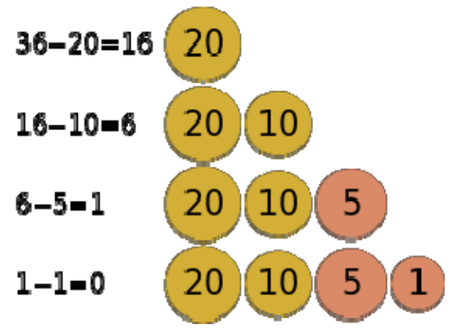
- 例如，解决函数是已换的零钱恰好等于总额。



11.3.3 贪心算法

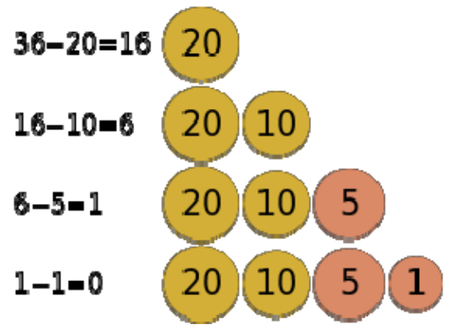
- 2．求解过程
- ④选择函数Select：即贪心策略，这是贪心算法的关键，它指出哪个候选对象最有希望构成问题的解，选择函数通常和目标函数有关。

- 例如，贪心策略就是在候选集合中选择面值最大的货币。



11.3.3 贪心算法

- 2．求解过程
- ⑤可行函数Feasible：检查解集合中加入一个候选对象是否可行，即解集合扩展后是否满足约束条件。
- 例如，可行函数是每一步选择的货币和已换的零钱相加不超过总额。



11.3.3 贪心算法

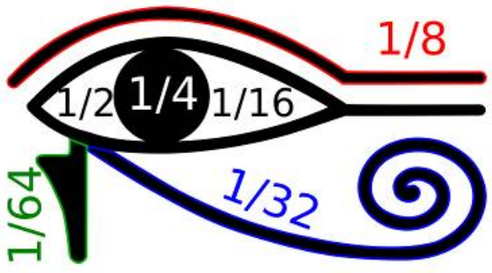
- 3. 编程模式

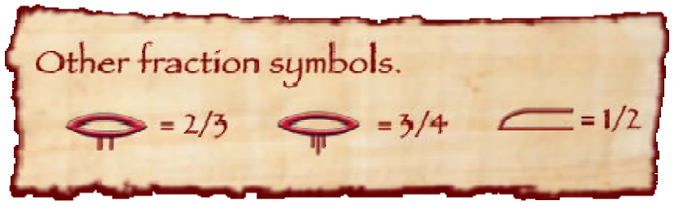
```
//贪心算法求解伪代码
Greedy(C) //①C是问题的输入集合即候选集合
{
  S={ }; //②初始解集合为空集
  while (! Solution(S)) { //③集合S没有构成问题的一个解
    x=Select(C); //④在候选集合C中做贪心选择
    if Feasible(S, x)  //⑤判断集合S中加入x后的解是否可行
      S=S+{x}; //②
    C=C-{x}; //①
  }
  return S;
}
```

11.3.3 贪心算法

【例11.8】

将一个真分数表示为埃及分数之和的形式。所谓埃及分数，是指分子为1的分数形式，如7/8=1/2+1/3+1/24。





11.3.3 贪心算法

例题分析

分析

（1）贪心算法设计

对于给定的分数，如何快速寻求其埃及分数式？应用贪心选择，每次选择分母最小的最大埃及分数是可行的。

例如要寻求分数7/8的埃及分数式，作以下贪心选择：

$$\frac{7}{8}>\frac{1}{2}\quad,\quad \frac{7}{8}-\frac{1}{2}=\frac{3}{8}>\frac{1}{3}\quad,\quad \frac{7}{8}-\frac{1}{2}-\frac{1}{3}=\frac{1}{24}$$

11.3.3 贪心算法

例题分析

一般地，对于给定的真分数 $a/b(a \neq 1)$ ，设 $d=[b/a]$ （表示取 b/a 的整数），考虑到

$$d < \frac{b}{a} < d+1$$

有

$$\frac{a}{b} = \frac{1}{d+1} + \frac{a(d+1)-b}{b(d+1)}$$

这个公式就是贪心选择最大埃及分数的依据。即取埃及分数的分母为c=d+1，真分数(ac-b)/bc去除公因数后，同以上a/b考虑。

11.3.3 贪心算法

例题分析

（2）算法设计步骤

①对给定的真分数a/b(a≠1)，求得c=[b/a]+1。

②若分母不大于上限，则存储各埃及分数的分母：f[k]=c。否则退出循环。

③给a和b实施迭代：a=a*c-b，b=b*c，为探索下一个埃及分数的分母作准备。

④去除a，b的公因数。

⑤若a≠1，继续循环；否则a=1，f[k]=b，退出循环输出结果。

11.3.3 贪心算法

例11.8

```
 1 #include <stdio.h>
 2 int EgyptianFraction(int a,int b,int D[20])
 3 { //贪心算法求a/b的埃及分数，求解结果放在D中
 4   int n=0,j,u,c;
 5   j=b;
 6   for(;;) { //反复循环求解
 7     c=b/a+1;
 8     if (c>1000000000||c<0) return 0; //分母超过上限无解
 9     if (c==b) c++; //分母不与给定分数的分母相同
10     D[++n]=c; //得到第n个分母
11     a=a*c-b; //准备下一次计算
12     b=b*c;
13     for (u=2;u<=a;u++) //试商去除a、b公因数
14       while (a%u==0&&b%u==0)
15         a=a/u , b=b/u;
```

11.3.3 贪心算法

例11.8

```
16      if (a==1 && b!=j) { //结束求解
17        D[++n]=b;
18        break;
19      }
20    }
21    return n; //有解返回分式项数
22 }
23 int main()
24 {
25    int a,b,n,i,D[20];
26    scanf("%d%d",&a,&b); //输入分子、分母
27    n=EgyptianFraction(a,b,D);
28    if (n>0) { //输出埃及分数式
29      printf("%d/%d=1/%d",a,b,D[1]);
30      for (i=2;i<=n;i++)
```

11.3.3 贪心算法

例11.8

```
31          printf("+1/%d",D[i]);
32        printf("\n");
33    }
34    return 0;
35 }
```

11.3.3 贪心算法

例11.8

程序运行屏幕

```
5/11=1/3+1/9+1/99
```

```
5 11↙
```

11.3.3 贪心算法

- 4．动态规划算法和贪心算法的比较
- （1）利用动态规划求解最优问题的步骤：
  - ①证明该问题具有最优子结构性质；
  - ②根据最优子结构性质，写出最优值的递归表达式；
  - ③根据递归式，说明该问题具有重叠子结构性质；
  - ④采用自底向上的方式计算，写出求解最优值的非递归算法，同时构造最优解的解空间树；
  - ⑤遍历解空间树，求得最优解。

- 4．动态规划算法和贪心算法的比较
- （2）利用贪心算法求解最优问题的步骤：
  - ①选定合适的贪心选择的标准；
  - ②证明在此标准下该问题具有贪心选择性质；
  - ③证明该问题具有最优子结构性质；
  - ④根据贪心选择的标准，写出贪心选择的算法，求得最优解。

11.3.3 贪心算法

- 4．动态规划算法和贪心算法的比较
- （3）动态规划算法和贪心算法的共同点：
  - ①都属于递推算法
  - ②算法适用的问题都具有最优子结构，都利用局部最优解来推导全局最优解。

11.3.3 贪心算法

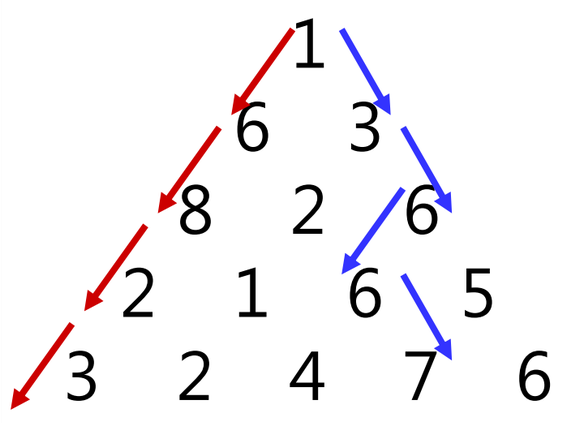
- 4．动态规划算法和贪心算法的比较
- （4）动态规划算法和贪心算法显著区别：
  - ①在动态规划算法中，以自底向上的方式来利用最优子结构，也就是，首先找到子问题的最优解，解决子问题，然后找到问题的一个最优解。
  - ②在贪心算法中，以自顶向下的方式使用最优子结构，也就是，贪心算法会先做出选择，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题。

- 4．动态规划算法和贪心算法的比较
- （5）动态规划算法和贪心算法的不同点：
  - ①贪心算法作出的每步贪心决策都无法改变，不能回退。因为贪心策略是由上一步的最优解推导下一步的最优解，而上一部之前的最优解不作保留。
  - ②动态规划算法的全局最优解中一定包含某个局部最优解，但不一定包含前一个局部最优解，因此需要记录之前的所有局部最优解。有回退功能。

11.3.3 贪心算法

例题分析

数字三角形：有一个数字三角形（如下图）。现有一只蚂蚁从顶层开始向下走，每走下一级时，可向左下方向或右下方向走。求走到底层后它所经过的数的最大值。



用贪心算法：每次朝最大值方向走，得到结果为1+6+8+2+3=20。可是明明还有另一条路，1+3+6+6+7=23。问题出在哪？每次的选择对后面的步骤会有影响！第3级选了8，就选不到第4、5级较大的数了。

贪心算法得不到最优解。


程序设计