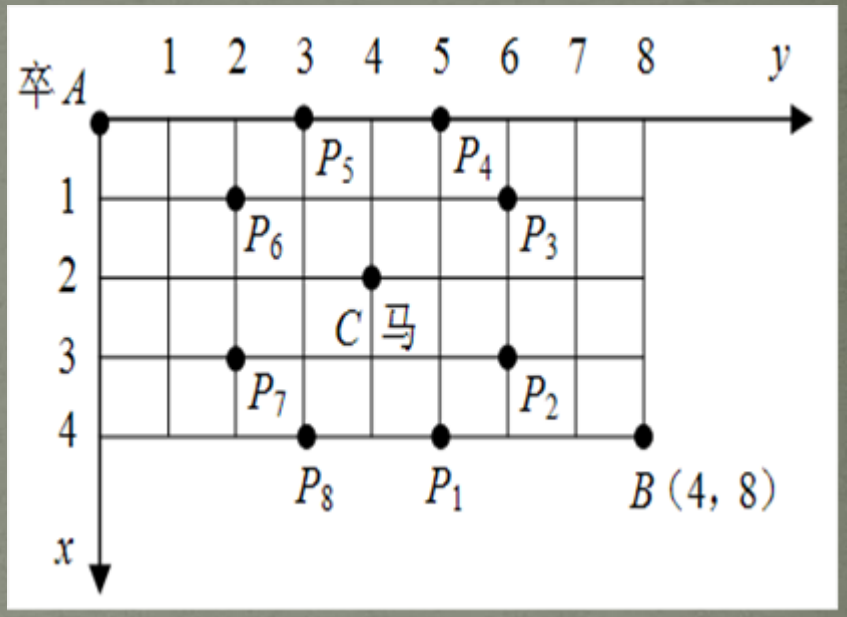
动态规划初步

李琛

lic.tiny@gmail.com

马拦过河卒

- 棋盘上A点有一个过河卒，需要走到目标B点。卒行走的规则：可以向下或向右。
- 同时在棋盘上的任一点有一个对方的马（如图的C点），该马所在的点和所有跳跃一步可达的点称为对方马的控制点。卒不能走到对方马的控制点。



问题分析

- 搜索？
- 递推？

问题分析

- 搜索？
- 卒可以往下或往右跳，所以一条路径的长度为n+m-1。
- 路径上每个点都要决定往下还是往右，所以每次搜索都有两种决策。
- 所以搜索的复杂度为$O(2^{n+m-1})$.
- 搜索在n，m=15时就会超时。
- 其实，对本题稍加分析就能发现，要到达棋盘上的一个点，只能从左边过来（称之为左点）或是从上面过来（称之为上点）。

问题分析

- 根据加法原理，到达某一点的路径数目，就等于到达其相邻的上点和左点的路径数目之和，因此我们可以使用逐行（或逐列）递推的方法来求出从起点到终点的路径数目。障碍点（马的控制点）也完全适用，只要将到达该点的路径数目设置为0即可。

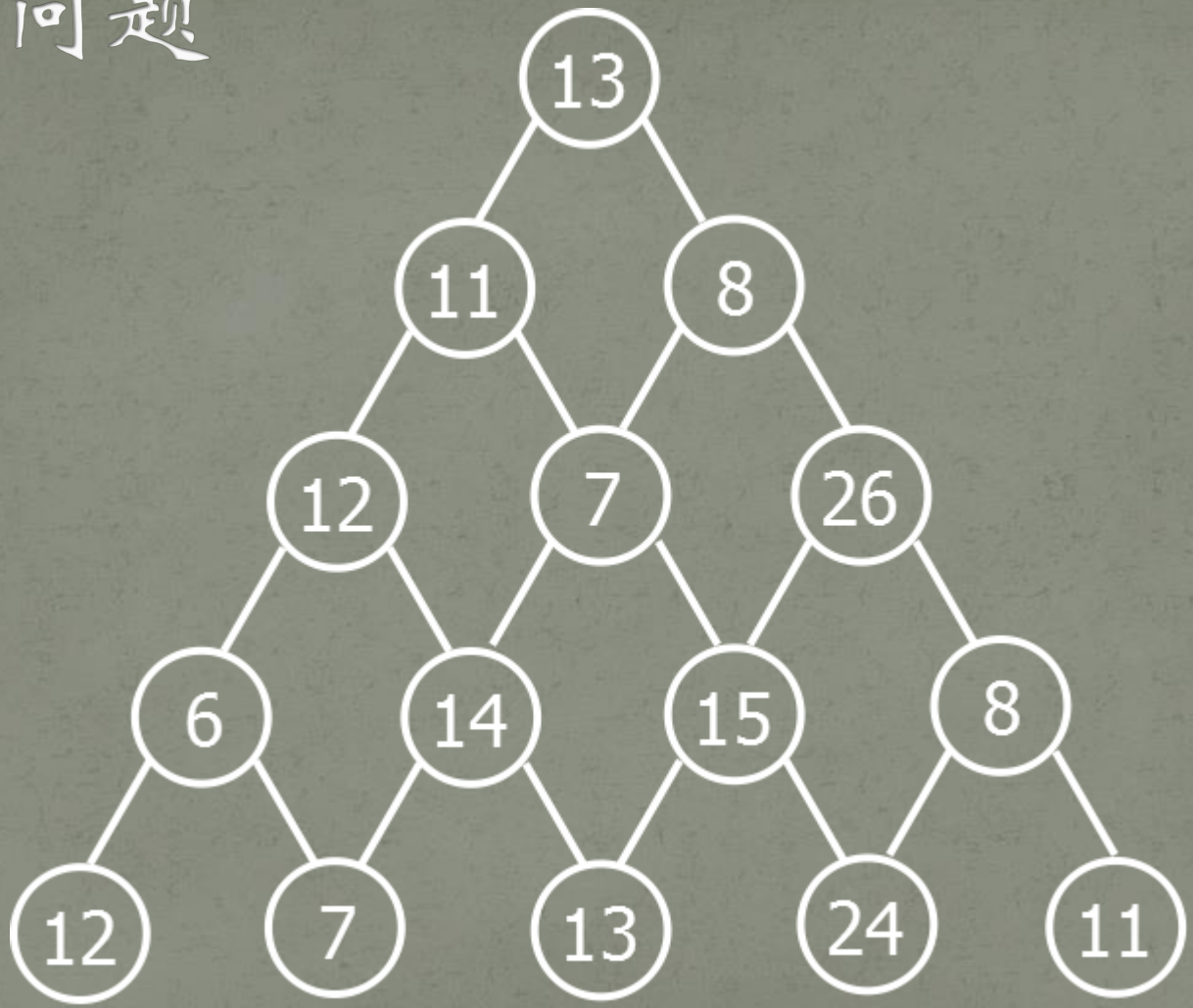
问题分析

- 假设用F[i,j]表示到达点（i,j）的路径数目，用g[i,j]表示点（i,j）是否是对方马的控制点，g[i,j]=0表示不是对方马的控制点，g[i,j]=1表示是对方马的控制点。则可得到如下的递推关系式：
- F[i,j]=0　　{ g[x,y] = 1 }
- F[0,j]=F[0,j-1]　　{ j>0, g[x,y] = 0 }
- F[i,0]=F[i-1,0]　　{ i>0, g[x,y] = 0 }
- F[i,j]=F[i-1,j]+F[i,j-1]　　{i>0, j>0, g[x,y] = 0}
- 这里的F[i-1,j]，F[i,j-1]均已事先求出。

数塔问题

- 设有一个三角形的数塔，顶点为根结点，每个结点有一个整数值。从顶点出发，可以向左走或向右走，若要求从根结点开始，请找出一条路径，使路径之和最大，只要输出路径的和。

数塔问题

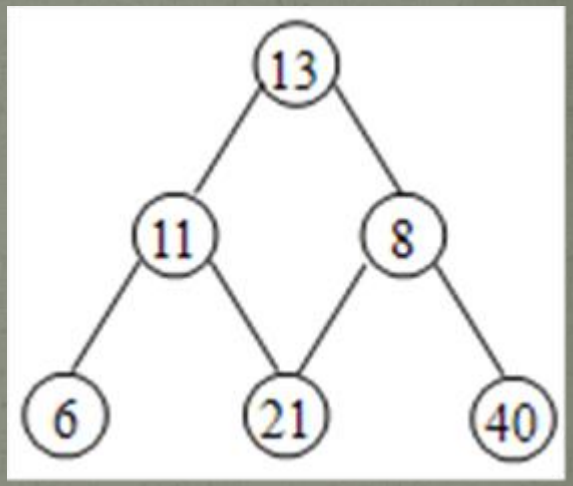


问题分析

- 贪心？
- 每次选择相邻的最大的数

- 如下图，根据贪心法，得出的解为：13-11-21；
- 而实际上最优解应为：13-8-40。

目光短浅



问题分析

- 搜索？
- 搜索出所有自顶向下的路径，对所有路径进行判断。
- 当N=1　P=1
- 　　N=2　P=2
- 　　N=3　P=4
- 　　……
- 　　N=K　P=2^(K-1)
- 例如：当N=50，P=2^49=500万亿条路径。

超时

问题分析

- 考虑和马拦过河卒的联系？
- 如果我已经知道了第i-1行所有格子的最优值，如何得出第i行的答案？

问题分析

- 令F[i][j]为处于i行j列的最优值。
- 递推式：
- f[i,j]=max{f[i-1,j-1],f[i-1,j]} + a[i,j]

- f[i,j]=max{f[i-1,j-1],f[i-1,j]} + a[i,j]
- 这是一个类似于递归的公式，意思为：要求f[n,n]就要先求f[n-1,n-1]和f[n-1,n]，要求f[n-1,n-1]就要先求f[n-2,n-2]和 f[n-2,n-1]，……而要求f[i,j]，就要先求f[i-1,j-1]和f[i-1,j]，……，而f[1,1]=a[1,1]。在具体计算的时候，只要从f[1,1]开始“顺推”下去，先推出每一行的答案，然后一列一列推到f[n,n]即可。

- 注意边界条件！
- 第一列的格子只能从上面走，f[i,1]=f[i-1,1]+a[I,1]
- 对角线上的格子只能从左上走，f[I,i]=f[i-1,i-1]+a[I,i]
- 由于第n层所有的格子均符合条件，所以答案为f[n,i]的最大值

核心代码

```
f[1,1]:=a[1,1];
for i:=2 to n do
  begin
    f[i,1]:=f[i-1,1]+a[i,1]; // 第一行
    f[i,i]:=f[i-1,i-1]+a[i,i]; // 斜边
    for j:=2 to i-1 do // 中间部分
      if f[i-1,j-1]>f[i-1,j] then
           f[i,j]:=f[i-1,j-1]+a[i,j]
      else f[i,j]:=f[i-1,j]+a[i,j];
  end;
```

分析

- 与马拦过河卒的异同？
- 有边界条件
- 有阶段性
- 无后效性
- 求最优值VS统计个数

动态规划简介

- 动态规划是运筹学的一个分支。它是解决多阶段决策过程最优化问题的一种方法。1951年，美国数学家R.Bellman提出了解决这类问题的“最优化原则”，1957年发表了他的名著《动态规划》，该书是动态规划方面的第一本著作。动态规划问世以来，在工农业生产、经济、军事、工程技术等许多方面都得到了广泛的应用，取得了显著的效果。
- 动态规划运用于信息学竞赛是在90年代初期，它以独特的优点获得了出题者的青睐。此后，它就成为了信息学竞赛中必不可少的一个重要方法，几乎在所有的国内和国际信息学竞赛中，都至少有一道动态规划的题目。所以，掌握好动态规划，是非常重要的。

动态规划中的概念

- 阶段
- 状态
- 决策
- 策略与最优策略
- 状态转移方程
- 最优化概念

拦截导弹（NOIP1999）

- 问题描述：
- 某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭，由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。
- 输入导弹的枚数和导弹依次飞来的高度（雷达给出的高度数据是不大于30000的正整数，每个数据之间有一个空格），计算这套系统最多能拦截多少导弹？如果要拦截所有导弹最少要配备多少套这种导弹拦截系统？

- 样例输入：
- 8
- 389 207 155 300 299 170 158 65
- 样例输出：
- 6（最多能拦截的导弹数）
- 2（要拦截所有导弹最少要配备的系统数）

问题分析

- 先考虑第一问。
- 类似数塔，我们假设一个数组F。
- F[i]表示拦截的最后一枚导弹是i时，能拦截的最多导弹数。
- 很明显，F[i]的最大值即为第一问的答案。

问题分析

- 如果已知F[1]...F[i-1]，如何求F[i]?

- 样例：
- 8
- 389 207 155 300 299 170 158 65
- 如何求F[7]?
- 满足要求的有389 207 300 299
- 即F[7]可以从F[1] F[2] F[4] F[5]转移
- F[1] = 1 F[2] = 2 F[4] = 2 F[5] = 3
- 因为F[7]表示能拦截的最多导弹数
- 所以F[7]应取其中的最大值+1
- 即F[7] = 3 + 1 = 4

- F[i]的其余值也类似
- 于是我们可以归纳出以下式子：
- $F[i] = \max\{F[j] + 1\}$
- $(1 \le j < i$ 且 $A[j] \le A[i])$
- 边界条件 $F[1] = 1$

比如

- 一开始

	1	2	3	4	5	6	7	8
H[i]	13	7	9	16	38	24	37	18
F[i]	1	1	1	1	1	1	1	1

- 转移

	1	2	3	4	5	6	7	8
H[i]	13	7	9	16	38	24	37	18
F[i]	1	1	2	3	4	4	5	4

核心代码

```
For i:=1 to n do F[i]:=1;
for i:=2 to n do                    {分阶段求出每步的最优值}
  for j:=1 to i-1 do
        if h[i]<=h[j] then
          F[i]:=max(F[i],F[j]+1);
max:=F[1];      {以下打擂台求出最大值}
for i:=2 to n do
  if F[i]>max then max:=F[i];
writeln('max=',max);
```

- 动态规划的经典问题：
- 最长不上升子序列

- 接下来看第二问

- 多次求最长不下降子序列？
- **7 5 4 1 6 3 2**
- “系统数最少”和“每套系统打导弹最多”没有关系
- 如果已有多个系统可以拦截该导弹，选哪一个？

- 当前瞄准高度较高的系统的“潜力”较大
- 而较低的系统不同，它不能打下更多的导弹
- 正解：贪心
- 1.当已有的系统均无法拦截该导弹时，启用新系统
- 2.如果有多个系统可以拦截，选择瞄准高度最低的

```
sys[1]:=h[1]; tail:=1;
{sys[]为当前已有系统的瞄准高度}
for i:=2 to n do
begin
  minheight:=maxlongint;
  for j:=1 to tail do  {找一套最适合的系统}
    if sys[j]>h[i] then
       if sys[j]<minheight then
         begin minheight:=sys[j]; select:=j end;
    if minheight=maxlongint   {开一套新系统}
      then begin tail:=tail+1; sys[tail]:=h[i] end
      else sys[select]:=h[i]
end;
writeln(tail);
```

运用动态规划的条件

- 最优化原理
- 无后效性原则

最优化原理

- 作为整个过程的最优策略具有：无论过去的状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略的性质。也可以通俗地理解为：子问题的局部最优将导致整个问题的全局最优，即问题具有最优子结构的性质。也就是说一个问题的最优解只取决于其子问题的最优解，非最优解对问题的求解没有影响。

数塔问题拓展

- 将三角形中的数字改为-100～100之间的整数，求从顶至底的某处的一条路径，使该路径所经过的数字的总和对K取模最接近于零。
- 满足最优化原理吗？

- 这个问题就不具有最优子结构性质了，因为子问题最优，即最接近于零，反而可能造成问题的解远离零，这样的反例是不难构造的，本问题就不能用动态规划的方法解决了。

只有当一个问题呈现出最优子结构时，动态规划才可能是一个合适的候选方法。

无后效性原则

- 所谓无后效性原则，指的是这样一种性质：某一阶段的状态一旦确定，则此后过程的演变不再受此前各状态及决策的影响。也就是说“未来与过去无关”。当前的状态是此前历史的一个完整总结，此前的历史只能通过当前的状态去影响过程未来的演变。
- 比如，如果一个问题被划分各个阶段之后，阶段I中的状态只能由阶段I-1中的状态转移方程得来，与其他没有关系，特别是与未发生的状态没有关系，这就是无后效性。

- 例如：有n个城市，编号1~n，有些城市之间有路相连，有些则没有，有路则当然有一个距离。现在规定只能从编号小的城市走到编号大的城市，问你从编号为1的城市走到编号为n的城市要花费的最短距离是多少？
- $F[i] = \min(F[j] + G[i,j])(1 \le j < i)$
- 去掉“只能从编号小的城市走到编号大的城市”？
- F[i]可以从任意的F[j]转移

不满足无后效性

BOOKSHELF

- 一条笔直的书架上放有N本书，每本书都有一个整数编号，并且竖立摆放在书架上，就象你平时在图书馆里所见的那样。现在所有的书次序是乱的，要求你用最少的插入次数将所有的书以按编号从小到大的次序摆放好，一次插入操作的过程是：将一本书从书架上抽出来，然后重新插入到书架上，你可以将书插在最前面，也可以将书插在最后，当然你也可以将抽出来的书插在某两本书中间。

- **输入**
- 输入文件第一行包含一个整数n，其中1≤n≤100，表示书架上的书的总数，第二行包含n个用空格隔开的正整数，表示n本书的编号。
- **输出**
- 输出文件仅有一行，包含一个整数k，表示最少要插入的次数。
- **样例**
- BOOKSHELF.IN
- 4
- 4 3 2 1
- BOOKSHELF.OUT
- 3

分析

- 书可以插入任意位置，不好把握。
- 正难则反：插入次数尽量少等价于不变的书尽量多
- 哪些书可以不用变换位置？

分析

- 不用交换位置的书构成递增序列
- 问题转化为求最长上升子序列

核心代码

```
readln(n);
for i:=1 to n do read(a[i]);
for i:=1 to n do F[i]:=1;
for i:=2 to n do
  for j:=1 to i-1 do
    if a[i]>a[j] then
      F[i]:=max(F[i],F[j]+1);
maxlen:=1;
for i:=1 to n do if F[i]>maxlen then maxlen:=F[i];
writeln(n-maxlen);
```

书的复制

- 现在要把maxn本有顺序的书分给n个人复制（抄写），每一个人的抄写速度都一样，一本书不允许给两个（或以上）的人抄写，分给每一个人的书，必须是连续的，比如不能把第一、第三、第四本书给同一个人抄写。现在请你设计一种方案，使得复制时间最短。复制时间为抄写页数最多的人用去的时间。每个人必须有书抄。
- 输入
- 第一行两个整数maxn, n；（n<=maxn<=100）
- 第二行maxn个整数，第i个整数表示第i本书的页数。
- 输出
- 共n行，每行两个正整数，第i行表示第i个人抄写的书的起始编号和终止编号。n行的起始编号应该从小到大排列，如果有多解，则尽可能让前面的人少抄写。

- [样例]
- BOOK.IN
- 9 3
- 1 2 3 4 5 6 7 8 9
- BOOK.OUT
- 17

- 动态规划的关键：
- 状态表示与状态转移

状态的表示

- 这个问题的关键在于划分好书的分配方式。
- 由于一个人抄的书必须是连续的，因此不难发现这个问题是符合动态规划的条件的（即无后效性和最有子结构）。
- 我们用F[i, j]表示
- 前i个人抄到第j本书所需要消耗的最小时间。

状态的转移

- 穷举第i个人抄写的页数
- 那么抄写的代价为max(F[i-1,k],value[k+1..j])
- 则F[i,j]=min{max (F[i-1,k], value[k+1..j])}
- (i-1≤k≤j-1)

- 如何记录每个人抄写的页数？

记录决策

- 令G[i,j]表示F[i,j]取得最优值时候的k。
- G[i,j]=k表示第i个人抄k+1..j页时F[i,j]取最优值。
- 同时可以推出F[i,j]取最优值时，
- 第i-1个人抄G[i-1,G[i-1,k]]..k页。
- 以此类推，可以推出所有人的抄写方案。
- 最优值为F[n,maxn]。

核心代码

```
for i:=1 to n do
  for j:=i to maxn do
begin
  F[i,j]:=value[1..j];
  for k:=i-1 to j-1 do
    if F[i,j]>max(F[i-1,k],value[k+1..j])
    then begin
      F[i,j]:=max(F[i-1,k],value[k+1..j]);
      G[i,j]:=k;
    end;
end;
```

总结

- 上面所有的问题都可以用穷举和搜索来解决，但是比较之下动态规划却有很明显的时间优势，采取的是牺牲空间换取时间的方法。
- 在整个思考过程中，我们通常是采取换一个角度来考虑问题的方法，巧妙地把其中的某个参数转化成维度，从而将原本需要大量重复计算的东西存下来，随时方便调用。
- 因此，想要用好动态规划，必须得学会转化思考角度，充分利用空间来节省时间。

- 同时，我们也必须看到，在DP问题的思考过程中，最关键的两个东西就是
- 状态的表示和状态的转移。

- 这也是在运用动态规划的过程中的难点之一。

- 当然，动态规划也不是万能的。比如之前提到的数塔问题中，如果是要求一条路径使得最终结果最接近0的话，那么DP也无能为力了。

- 只有在满足前面所提到的最优子结构性质和不具有后效性的两个前提下，动态规划才能发挥它的作用。

Thanks for listening!