动 态 规 划 3

常州市第一中学　林厚从

经典问题4——区间DP

乘积最大（NOIP2000，MAXMIN）

- 问题描述：
  - 设有一个长度为n的数字字符串，要分成k+1个部分，使得k+1个部分的乘积最大。
  - 例如：n=6，k=3，数字字符串为“310143”，可能情况如下：
  - 3*1*0*143 = 0
  - 3*1*01*43=129
  - 3*1*014*3=126
  - 3*10*1*43=1290
  - 3*10*14*3=1260
  - 3*101*4*3=3636
  - 31*0*1*43=0
  - 31*01*4*3=372
  - 310*1*4*3=3720
  - 从上面的结果可以看出，最大乘积为：310*1*4*3=3720。
  - 数据范围：n<=50，k<=10。

经典问题4——区间DP

乘积最大（NOIP2000，MAXMIN）

- 问题分析：
  - 本题就是常说的“区间DP”，类似题目还有添加括号、石子合并、能量项链等问题。
  - 方法是穷举最后一个“*”的位置，DP的思路如下：
  - 设f[i][j]表示前i个数字中插入j个“*”的最大乘积。
  - 显然：j=0时，f[i][j]=S1..Si。
  - 而j>=i时，没有意义，所以f[i][j]=0
  - 否则：我们可以假设最后一个“*”插在第k个数字之后，这样前k个数字之间就要插入j-1个“*”，而$S_{k+1}..S_i$组成最后一个数。

转移方程为：

- $f[i][j]=\max\{ f[k][j-1] * S_{k+1}..S_i \}$
- 其中：
- $2<=i<=n,1<=j<=i-1,j<=k<=i-1$

经典问题4——区间DP

乘积最大（NOIP2000，MAXMIN）

- 算法思路：
- 1、先求出第i个数字到第j个数字组成的一个数a[i][j]

- 2、初始化：for(i=1；i<=n；i++) f[i][0]=a[1][i];

- 3、dp求值：
- for(i=2；i<=n；i++)
- for(j=1；j<=i-1；j++)
- for(k=j；k<=i-1；k++)
- f[i][j]=max{ f[i][j],f[k][j-1]*a[k+1][i] }

- 4、输出：f[n][k]

经典问题4——区间DP

乘积最大（NOIP2000，MAXMIN）

- 跟踪演示：

初始化：所有的f[i][0]=a[1..i]

计算时：if i<=j then f[i][j]=0

f[2][1]=f[1][0]*a[2][2]=3*1=3

f[3][1]=max{f[1][0]*a[2][3],f[2][0]*a[3][3]}={3*10,31*0}=30

f[3][2]=max{f[1][1]*a[2][3],f[2][1]*a[3][3]}={0,3*0}=0

f[4][1]=max{f[1][0]*a[2][4],f[2][0]*a[3][4],f[3][0]*a[4][4]}={3*101,31*1,310*1}=310

f[4][2]=max{f[1][1]*a[2][4],f[2][1]*a[3][4],f[3][1]*a[4][4]}={0,3*01,30*1}=30

f[4][3]=……

f[5][1]=……

f[5][2]=max{f[1][1]*a[2][5],f[2][1]*a[3][5],f[3][1]*a[4][5],f[4][1]*a[5][5]}

={0,3*014,30*14,310*4}=1240

f[5][3]=……

f[5][4]=……

f[6][1]=……

f[6][2]=……

f[6][3]=max{f[1][2]*a[2][6],f[2][2]*a[3][6],f[3][2]*a[4][6],f[4][2]*a[5][6],f[5][2]*a[6][6]}

={0,0,0*143,30*43,1240*3}=3270

经典问题4——区间DP

乘积最大（NOIP2000，MAXMIN）

- 思考讨论：

  根据前面的讨论，本题也可以乘号的个数k划分阶段，进行DP（参考标程），因为k<=10。

- 友情提醒：

  根据本题的数据范围，n<=50，本题还要高精度运算，所以有一定难度。

添加括号（VIJOS1038）

- 问题描述：

给定一个正整数序列a(1)，a(2)，…，a(n),(1<=n<=20)

不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。

例如:

给出序列是4，1，2，3。

第一种添括号方法:

((4+1)+(2+3))=((5)+(5))=(10)

有三个中间和是5，5，10，它们之和为:5+5+10=20

第二种添括号方法

(4+((1+2)+3))=(4+((3)+3))=(4+(6))=(10)

中间和是3，6，10，它们之和为19。

现在要添上n-1对括号，加法运算依括号顺序进行，得到n-1个中间和，求出使中间和之和最小的添括号方法。

添加括号（VIJOS1038）

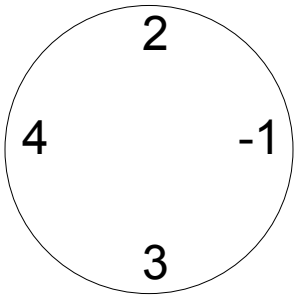
- 问题分析：
- 定义f[i][j]表示从第i个数到第j个数能取到的最小中间和之和。
- s[i]表示前i个数的和，那么s[i]-s[j-1]就是第i个数到第j个数的和。
- 状态转移：
- f[i][j] = min{ f[i][k] + f[k+1][j] + s[i] - s[j-1] }
- 可以枚举k的位置，求出f[i][j]。
- 具体实现可以采用记忆化搜索的方式来实现。
- 参考程序：vijos1038.cpp。

添加括号（VIJOS1038）

- 伪代码：从长度小的区间开始DP
  - for i=1 to n do f[i][i]=0;
  - for L=2 to n do //穷举区间的长度
  - for i=1 to n do //穷举左端点
  - begin
  - j=i+L-1; min=maxlongint; //计算右端点
  - if j>n then continue;
  - for k=i to j-1 do //穷举k的位置
  - if min>=f[i][k]+f[k+1][j] then
  - begin
  - min=f[i][k]+f[k+1][j];
  - num[i][j]=k;//num数组记录[i][j]达到最小值时的k值
  - end;
  - f[i][j]=min+a[i][j];//a[i][j]为预处理好的第i个数到第j个数的和
  - end;

经典问题5——环形结构
数字游戏（GAME,NOIP2003）

- 问题描述：
  - 丁丁最近沉迷于一个数字游戏之中。游戏是这样的，在你面前有一圈整数（一共n个），你要按顺序将其分为m个部分，各部分内的数字相加，相加所得的m个结果对10取模后再相乘，最终得到一个数k。游戏的要求是使你所得的k最大或者最小。
  - 例如，对于下面这圈数字（n=4，m=2）：



  - 当要求最小值时，((2-1) %10)×((4+3) %10)=1×7=7，要求最大值时，为((2+4+3) %10)×(-1 %10)=9×9=81。
  - 特别值得注意的是，无论是负数还是正数，对10取模的结果均为非负值。
  - 丁丁请你编写程序帮他赢得这个游戏。

经典问题5——环形结构
数字游戏（GAME,NOIP2003）

- 输入格式:
  - 第一行有两个整数，n（1≤n≤50）和m（1≤m≤9）。
  - 以下n行，每行一个整数，其绝对值不大于10^4，按顺序给出圈中的数字，首尾相接。
- 输出格式:
  - 输出两行，各包含一个非负整数。
  - 第一行是你程序得到的最小值，第二行是最大值。
- 输入样例:
  - 4 2
  - 4
  - 3
  - -1
  - 2
- 输出样例:
  - 7
  - 81

经典问题5——环形结构
数字游戏（GAME,NOIP2003）

- 问题分析：
  - 题意是给定一圈整数，将其按顺序分成m个部分，使得每个部分的和对10取模后再相乘的积最大或最小。
  - 但要注意的是，这里的取模运算不同于我们平时的取模运算，无论是负数还是正数，对10取模的结果均为非负值。
  - 按照模运算规则：$(a_1+a_2+\cdots+a_k)\%10=(a_1\%10+a_2\%10+a_k\%10)\%10$。
  - 设$g[i][j]=(a_i+a_{i+1}+\cdots+a_j)\%10$
  - 显然有：
  - $g[i][i]=(a_i\%10+10)\%10$
  - $g[i][j]=(g[i][j-1]+g[j][j])\%10$　$(1\leqslant i\leqslant n,\ i+1\leqslant j\leqslant n)$
  - 本题不论是数据规模还是解的最优性上都来有强烈的动态规划气息。但由于给的是一个环，让人有些无从下手。

经典问题5——环形结构
数字游戏（GAME,NOIP2003）

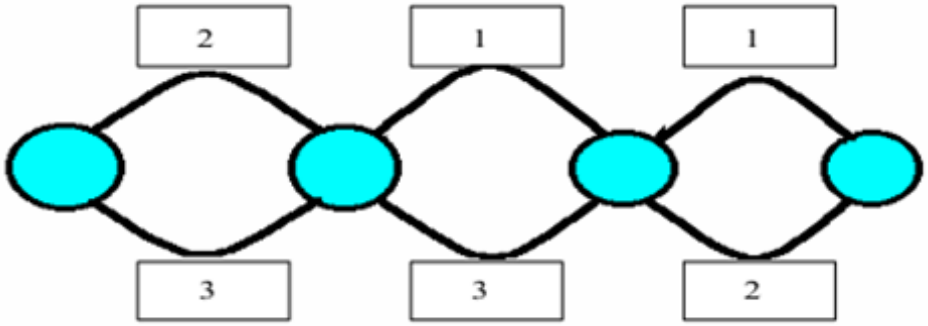
- 方法：破环为链。
- 先穷举“第一刀”，把这个环切成链(长度为2*N-1)，然后再分别使用动态规划的方法求解这个链。
- 切成链后，可以用类似于“乘积最大”的动态规划来解决，其状态转移方程为：

$$Opt(i,j)=\underset{k=i-1}{\overset{j-1}{Max}}(Min)\{((Sum[j]-Sum[k])\ \bmod\ 10+10)\bmod\ 10\times Opt(i-1,k)\}$$

- 其中：SUM[i]表示前i个整数的和，OPT(i，j) 表示前j个数分成i个部分所能取到的最值。
- 动态规划的时间复杂度是O(M*N^2)，穷举的时间复杂度是O(N)，所以，总的算法复杂度是O(M*N^3)。
- 类似题目：能量项链（ENERGY）。

问题讨论
——MOD 4 余数最小问题

如图，已知一个有向图，求一条从最左边的点走到最右边点的方案（只能从左往右走），使得所经过的权值和除以4的余数最小。



问题讨论
——MOD 4 余数最小问题

- 问题讨论：

  状态：F[i]表示到点i的最小值

  转移：F[i]=min((F[j]+num[i]) mod 4),j到i有边

  样例就出现bug ☹

- 本题不具有最优子结构，局部最优不能保证全局最优，所以不能用一维的dp写，以上算法实质上是一种贪心。

问题讨论
——MOD 4 余数最小问题

- 正确做法：二维的dp
  - f[i][j]表示到i点路径之和mod 4为j，是否可行(存在)
  - 初始i=1时：f[1][0]=1,f[1][1]=f[1][2]=f[1][3]=0
  - 当i=2时：f[2][0]=0,f[2][1]=0,f[2][2]=1,f[2][3]=1
  - 然后推i=3,4,5,……
  - 转移方程还是比较难写！

- dp的另外一种推法(倒推法)：
  - 从当前点去更新后继点的最优值；
  - 由f[i-1][j]=1去更新后面的点：
  - f[i][(j+way1[i-1]) % 4] = 1
  - f[i][(j+way2[i-1]) % 4] = 1

总结

- 状态→到某一种状态会有相应的最优值
- 决策→通过选择状态来进行转移
- 边界→开始时可以确定的极小/极大/特殊状态
- 无后效性→后面的决策对于前面的决策没有影响
- 最优子结构→每个状态都要取最优

总结

- 动态规划的时间复杂度？
  - 阶段数*每个阶段状态转移的状态数*每次状态转移的时间
  - 或者说：状态总数*每次状态转移的时间
  - 所以，要尽量减少状态总数（阶段数、维数）
- 动态规划只是求最优化问题的一种方法。
  - 贪心、搜索等
  - 搜索是对状态的穷举
  - 动态规划是对状态的选择与转移
- 应用动态规划要满足一定的条件。
  - 最优子结构
  - 无后效性

总结

注意点

- 一定要符合最优化原理，即满足最优子结构
- 可按照某一顺序，从小到大求解
- 从大到小求解可用记忆化搜索
- 注意边界条件
- 转移方程一定要清晰，不要漏情况
- 状态的意义一定要清晰

上机题

- 马棚问题(stable)
- 潜水员(submariner)
- 乌龟棋(noip2010提高组，tortoise)
- 取数字问题(number)
- 乘法游戏(mul)
- 回文词(palin)
- 能量项链(energy)
- 单词的划分(word)
- 乘积最大(noip2000提高组，maxmin)
- 数字游戏(noip2003提高组，game)
- 采药(noip2005初中组，medic)

后续知识展望

- 树形动态规划
- 动态规划的优化
  - 状态压缩
  - 单调队列
  - 四边形不等式
- ……

谢谢大家！