动态规划入门

数学科学学院
School of Mathematical Sciences

acm International Collegiate Programming Contest

数字三角形

- 7
- 3 8
- 8 1 0
- 2 7 4 4
- 4 5 2 6 5
- 从第一层走到最后一层，每次向左下或右下走，求路径的最大权值和。

解法

- 设f[i][j]表示从第i行第j个点到底部的路径权值和的最大值

```
f[i][j]= max(f[i+1][j],f[i+1][j+1])
         +a[i][j]
```

如何求解

- 如果转移方程？
- 观察方程的性质
- 因为第i行的解要由第i+1行的解得到，所以必须从下向上转移！！

如何求解

- 最后一行怎么处理？
- `f[i][j]=max(f[i+1][j],f[i+1][j+1])+a[i][j]`
- i=n时，i+1行是非法状态

- 边界条件，特殊处理即可。
- i=n: f[i][j]=a[i][j]

代码展示

```
for (int i=1; i<=n; i++)// 初始化边界
    f[n][i]=a[n][i];
// 转移顺序很重要
for (int i=n-1; i>0; i--)
    for (int j=1; j<=i; j++)
        dp[i][j] =
max(dp[i+1][j+1],dp[i+1][j]) + a[i][j];
```

要点

- 状态定义：如何描述一个子问题？
- 定义要明确。

- 状态转移方程：如何由子问题构造出原问题的解？
- 边界条件、初始条件
- 递推顺序

数字三角形II

1

3　2

4　10　1

4　3　2　20

- 从第一层走到最后一层，每次向左下或右下走，求路径的最大权值和。
- 某一层可以随意跳

解法

- 加一维状态
- 记f[i][j][k]为以(i,j)为顶点的子问题的解，k=1表示可以随意跳，k=0表示不能随意跳。
- 原问题的解即为f[1][1][1]
- 转移方程？

数字三角形III

1

3 2

4 10 1

4 3 2 20

- 从第一层走到最后一层，每次向左下或右下走，求路径的权值和的个位数的最大值

解法

- 之前的定义是否可行？
- 如何转移？

- `f[i][j]=max{f[i+1][j],f[i+1][j+1]}+a[i][j]`
- `f[i][j]=`
- `(max{f[i+1][j],f[i+1][j+1]}+a[i][j])mod 10`
- `f[i][j]=max{(f[i+1][j]+a[i][j]) mod 10,`
  `(f[i+1][j+1]+a[i][j]) mod 10}`

解法

- 之前的定义是否可行？
- 记f[i][j][k]为，以(i,j)为起点走到底部，路径权值和的个位数是否有可能等于k

- 如何转移？
- 对于每一个(i,j)，枚举k，遍历所有状态即可

总结

- 算法设计
  - 状态定义
  - 状态转移（递推顺序有时很重要）
  - 边界条件、初始条件

- 条件
  - 无后效性
  - 最优子结构

滑雪

为了获得速度，滑雪的路径必须向下倾斜，每次可以向上下左右4个方向滑行。区域由一个二维数组给出，每个数字代表该点的高度。求滑行的最长距离。

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

困难？

- 能用刚才数字三角形的方法做么？
  - 哪里不能？
  - 没有明确的依赖顺序，无法直接用递推进行转移
- 解决办法
  - 记忆化搜索

Fibonacci数列

- 把递归过程中的值记录下来，这样就能保证每个值只算一次。

```
F[0] = F[1] = 1;
for (int i = 2; i <= n; ++i) F[i] = -1;
int Fib(int n)
{
      if (F[n] != -1) return F[n];
      F[n] = Fib(n-1) + Fib(n-2);
      return F[n];
}
```

滑雪的记忆化搜索实现

```
int n, m;
int dp[N][N], a[N][N];
int dx[4]={1, -1, 0, 0}, dy[4]={0, 0, 1, -1};

int dfs(int x, int y){
    if(dp[x][y]!=-1)
        return dp[x][y];
    dp[x][y] = 1;
    for(int k=0; k<4; k++){
        int tx = x+dx[k], ty = y+dy[k];
        if(tx>=1 && tx<=n && ty>=1 && ty<=m && a[x][y]>a[tx][ty])
            dp[x][y] = max(dp[x][y], dfs(tx,ty)+1);
    }
    return dp[x][y];
}
```

再次总结

- 如何利用转移方程求解
  - 递推
  - 递归
  - 求解通项公式
- 如何看待记忆化
  - 避免大量重复计算
  - 简洁明了，方便理解
  - 递推比较繁琐，或没有明确的依赖顺序（图）

经典模型

01背包

- 在n件物品取出若干件放在空间为V的背包里，每件物品的体积为w1，w2……wn，与之相对应的价值为p1,p2……pn。

01背包

- 定义状态f[i][j]：
- 考虑前i件物品，选的物品总体积=j的情况下价值和的最大值
- 转移方程： O(n*V)

```
f[i][j]=max(f[i-1][j],
            f[i-1][j-wi]+pi)
```

- 初始条件：f[0][0]=0, f[0][j>0]=-INF
- 答案：max(f[n][j<=V])

01背包

- 定义状态f[i][j]：
- 考虑前i件物品，选的物品总体积<=j的情况下价值和的最大值
- 转移方程：O(n*V)

`f[i][j]=max(f[i-1][j],f[i-1][j-wi]+pi)`

- 初始条件：f[0][j<=V]=0
- 答案：f[n][V]

数学科学学院
School of Mathematical Sciences
acm International Collegiate Programming Contest

完全背包

- 有n种物品和一个容量为V的背包，每种物品都有无限件可用。第i种物品的体积是w，价值是p。求解将哪些物品装入背包可使这些物品的体积总和不超过背包容量，且价值总和最大。

完全背包

- 定义状态dp[i][j]：
- 考虑前i件物品，选的物品总体积=j的情况下价值和的最大值
- 转移方程： O(n*V*V/W)

```
dp[i][j]=max(dp[i-1][j-wi*k]+pi*k)(k>=0)
```

- 初始条件：dp[0][0]=0, dp[0][j>0]=-INF
- 答案：max(dp[n][j<=V])

多重背包

- 有n种物品和一个容量为V的背包，每种物品都有一个数量限制l。第i种物品的体积是w，价值是p。求解将哪些物品装入背包可使这些物品的体积总和不超过背包容量，且价值总和最大。

背包问题

- 分组背包
- 二维背包
- 树上的背包

- 注：完全背包和多重背包都是可以优化到O(n*V)的，参考《背包问题九讲》

LIS

- 一个数列S如果分别是已知数列的单调上升子序列，且是所有符合此条件序列中最长的，则S称为已知序列的最长上升子序列。求S。

LIS

- 定义状态dp[i]：
- 表示以S[i]结尾的最长上升子序列的长度
- 转移方程：O(n*n)

```
dp[i] = max(dp[j]+1)
          (j<i && S[j]<S[i])
```

LIS

- 定义状态dp[S[i]]：
- 表示以S[i]结尾的最长上升子序列的长度
- 转移方程：

```
dp[S[i]]=max(dp[v]+1)(v<S[i])
```

- 复杂度？O(n*n)
- 用线段树或者树状数组优化？O(n*logn)

LCS

- 一个数列S如果分别是两个已知数列A，B的子序列，且是所有符合此条件序列中最长的，则S称为已知序列的最长公共子序列。求S。

LCS

- 定义状态dp[i][j]：
- 表示考虑A的前i位和B的前j位，最长公共子序列的长度
- 转移方程：O(|A|*|B|)

```
if(A[i]==B[j])
  dp[i][j] = dp[i-1][j-1]+1;
else
  dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
```

例题选讲

数字三角形IV

CDOJ 597

1

-3 2

4 -10 1

4 3 2 20

- 给一个数字三角形，可以在里面拿走一些数，拿走数的条件是，它左上和右上的数已经被拿走了，问最多能拿到多少数

dp[i][j] 代表，在确定要选i, j这个位置的情况下，在前j-1列和第j列的前i行的范围内，最多能选多少。

1

2 3

4 5 6

7 8 9 1

1 2 3 4 5

例如dp[3][3] 代表一定选6，在绿色区域范围内能选到的最大和有多少

1

2 3

4 5 6

7 8 9 1

1 2 3 4 5

转移：

dp[3][3] = max(dp[2][2], dp[3][2], dp[4][2], dp[5][2]) + sum[3][3]

sum代表i,j这个位置上面的数的和

转移：

$dp[i][j]=max\{dp[k][j-1],i-1<=k<=n\}+sum[i][j]$

n^3?

DNA Sequences

SPOJ SAMER08D

- 给两个串，求最长公共子序列
- 有一个额外的要求，子序列由若干个长度不小于K的子串组成
- K=3 A=lovxxelyxxxxx B=xxxxxxxlovely
- Ans=lovely
- |A|,|B|<=10^3， K<=100

DNA Sequences

- dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
- if(f[i][j]==k)
- dp[i][j]=max(dp[i-k][j-k]+k,dp[i][j]);
- else if(f[i][j]>k)
- dp[i][j]=max(dp[i][j], dp[i-1][j-1]+1,dp[i-k][j-k]+k);

数学科学学院
School of Mathematical Sciences
acm International Collegiate Programming Contest

练习

P1004 - 【IOI1994】数塔

P1008 - 【NOIP2005】采药

P1267 - 跳马问题

P1189 - 最长上升子序列

P1009 - 【NOIP2000】乘积最大

P1039 - 【NOIP2013】花匠

P2037 - 【一本通】完全背包

P1352 - [NOIP2006] 金明的预算方案

P1011 - 【NOIP2004】合唱队形

P2498 - 【DP合集】最长上升子序列LIS

P2038 - 【一本通】庆功会-多重背包

P1246 - 最长公共子序列的长度

Thank you