动态规划

作者：屠添翼

递归的重叠子问题

- 斐波那契序列：
- $f(1)=1$;
- $f(2)=1$;
- $f(n)=f(n-1)+f(n-2)$; $(n>2)$

递归的重叠子问题

- 斐波那契序列的递归实现：

```
int Fibonacci(int n)
{
    if(n==1||n==2) return 1;
    return Fibonacci(n-1)+Fibonacci(n-2);
}
```

递归的重叠子问题

- $f(6)=f(5)+f(4)$
- $=f(4)+f(3)+f(4)$
- $=f(3)+f(2)+f(3)+f(4)$
- $=f(2)+f(1)+f(2)+f(3)+f(4)$
- $=3+f(3)+f(4)$
- $=3+f(2)+f(1)+f(4)$
- $=5+f(4)$
- $=5+f(3)+f(2)$
- $=5+f(2)+f(1)+f(2)=8$

递归的重叠子问题

- 在上述递归过程中，计算f(6)和f(5)都需要去计算f(4)，导致对同一个值f(4)计算了两次，浪费了时间，我们把这种情况叫做递归的重叠子问题.
- 一种避免递归的重叠子问题的方法就是，当我们求出一个值的时候，就把它保存起来，当下次再用到这个值的时候，直接调用保存的值，而不是再计算一次.

递归的重叠子问题

```
int num[100];
for(i=0;i<100;i++) num[i]=-1;
int Fibonacci(int n)
{
    if(num[n]!=-1) return num[n];
    if(n==1||n==2) num[n]=1;
    else num[n]=Fibonacci(n-1)+Fibonacci(n-2);
    return num[n];
}
```

动态规划

- 实际上，动态规划的实质就是通过保存计算过的状态，来避免递归的重叠子问题．解决冗余，是动态规划的根本目的．
- 动态规划实质上是一种以空间换时间的技术，它在实现的过程中，不得不存储产生过程中的各种 状态，所以它的空间复杂度要大于其它的算法。选择动态规划算法是因为动态规划算法在空间上可以承受，而搜索算法在时间上却无法承受，所以我们舍空间而取时 间。

动态规划的适用条件

- 1．最优化原理（最优子结构性质）
  - 最优化原理可以这样阐述：一个最优化策略具有这样的性质，不论过去状态和决策如何，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。简而言之，一个最优化策略的子策略总是最优的。一个问题满足最优化原理又称其具有最优子结构性质。

动态规划的适用条件

- 2．无后向性
  将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，而只能通过当前的这个状态。换句话说，每个状态都是过去历史的一个完整总结。这就是无后向性，又称为无后效性。

状态和状态转移

- 状态的选择和状态转移方程是解决动态规划的关键所在.
- 所谓状态就是问题的实例，比如斐波那契序列中，每一个从1到n的数都代表一个状态．而状态转移就是由一些状态转移到另一个状态，比如斐波那契序列中，可以由状态n-1和n-2转移到状态n．因此，我们就说状态转移方程为：Fibonacci(n)＝Fibonacci(n-1)＋Fibonacci(n-2).

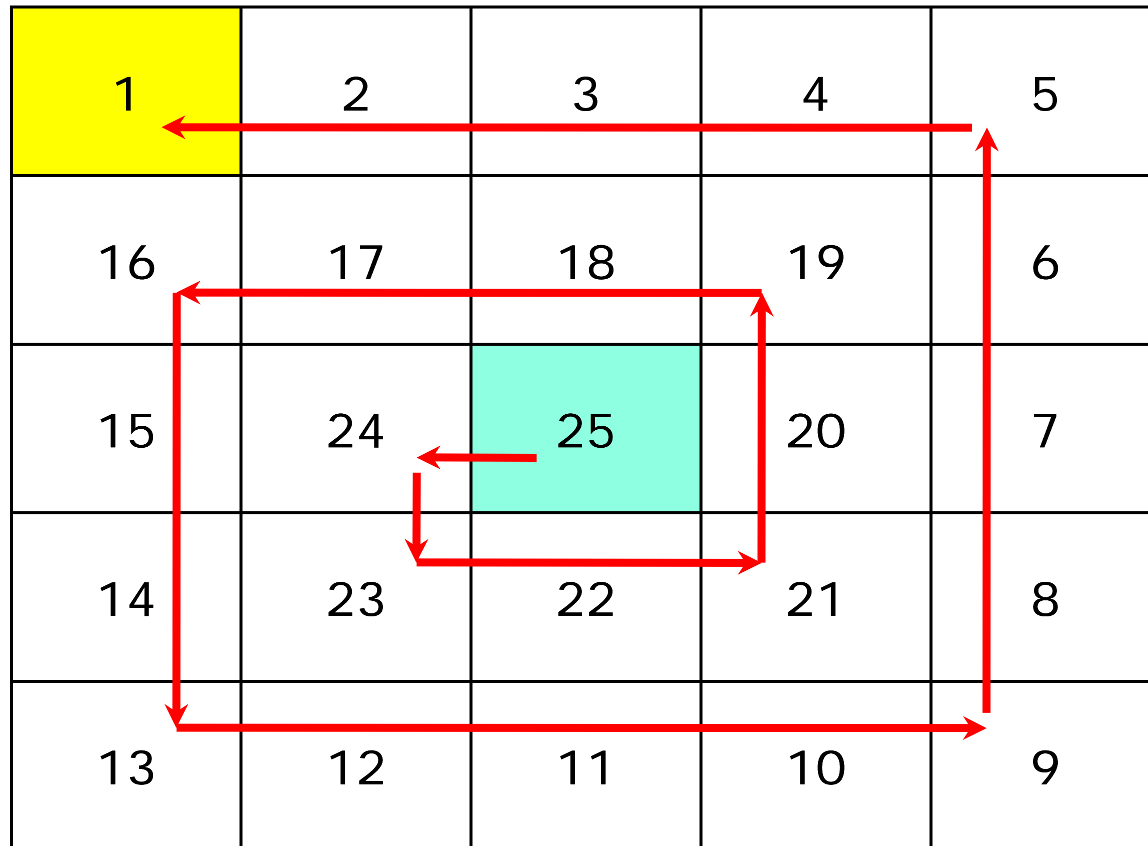
动态规划的实现

- 动态规划一般有以下两种实现方法：
- 1.记忆化搜索：当需要求一个状态时，首先看这个状态是否被计算过，如果计算过，直接使用计算过的值；否则，计算一次，并保存结果.
- 一般用递归实现，也就是刚才我们求斐波那契序列所使用的方法.
- 2.递推：所谓递推就是，按照一定顺序将一定范围内的所有状态都计算出来.
- 一般用非递归的形式实现.

滑雪【POJ1088】

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

滑雪【POJ1088】



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

思路

- 状态选择：
  - dp[x][y]表示以(x,y)为起点的最长滑坡的长度。
- 状态转移方程：
  - dp[x][y]=max{dp[x-1][y], dp[x+1][y], dp[x][y-1], dp[x][y+1]}
  - 所有dp[]值里最大的那个就是最长滑坡的长度。
- 初始条件：
  - dp[x][y]=0

典型问题一：矩阵乘法链

- 问题描述：
- 两个矩阵A1和A2相乘的时候，所执行的乘法的次数为a1*a2*a3(假设A1为a1* a2的矩阵，A2为a2*a3的矩阵).
- 当三个以上的矩阵相乘的时候，根据矩阵乘法的次序，所执行的乘法的次数也是不同的．比如：假设三个矩阵A1,A2,A3的大小分别为5*10,10*20,20*35.

矩阵乘法链

- 两种不同的乘法顺序所需要执行乘法的次数分别如下：
- (A1*A2)*A3
- 5*10*20=1000 5*20*35=3500
- 1000+3500=4500
- A1*(A2*A3)
- 10*20*35=7000 5*10*35=1750
- 7000+1750=8750

矩阵乘法链

- 可见，方法(A1*A2)*A3更好．现在给你n个矩阵大小，求怎样计算才能是执行的乘法次数最少．
- 假设n个矩阵分别为A1,A2,...,An,它们的大小存在数组a[n+1]中，其中(a[i],a[i+
- 1])代表矩阵Ai的大小.

矩阵乘法链

- 状态选择：
- dp[i][j]代表矩阵Ai到A[j]的最优解，则我们最终求的是dp[1][n];
- 状态转移方程：
- dp[i][j]=min{ dp[i][k]+dp[k+1][j] +a[i]*a[k+1]*a[j+1],i<=k<j }
- 初始条件：
- dp[i][i]=0,1<=i<=n.

典型问题二：最长上升子序列

- 问题描述：
- 给出一个序列，求它最长的上升子序列.
- 例如，序列{1,7,3,5,9,4,8},其中{1,7}, {3,4,8},{1,3,5,8},都是它的上升子序列，而{1,3,5,8}是它的最长的上升子序列.

最长上升子序列

- 状态选择：
- best[i]代表以序列中第i个数为最后一个数时，最长上升子序列的长度.
- 状态转移方程：
- $best[i]=\max\{1,\max\{best[j]+1, num[j]<num[i] \&\& j<i\}\}$
- 所有best[]值里最大的那个就是最优值.
- 初始条件：
- $best[i]=1, 1<=i<=n$.

经典问题三：最长公共子串

- 问题叙述：
- 给出两个字符串，求它们最长的公共子字符串.
- 比如，两个字符串abcfbc和abfcab，其中abb,abcb,abfc都是它们的公共子串，而abcb,abfc是它们的最长公共子串.

最长公共子串

- 状态选择：
- dp[i][j]代表第一个字符串前i个字符组成的字符串与第二个字符串前j个字符组成的字符串的最长公共子串的长度.
- 状态转移方程：
- 如果str1[i]==str2[j],dp[i][j]=dp[i-1] [j-1]+1;
- 否则dp[i][j]=max{ dp[i-1][j],dp[i] [j-1]}.
- 初始条件：
- dp[0][j]=0,0<=j<=len2;
- dp[i][0]=0,0<=i<=len1.

经典问题四：找硬币

- 问题描述：
- 有n种不同面值的硬币，每种硬币都有无穷多个，给定一个钱数w,问w是否可以用这些硬币组成.
- 例如：有面值为2,5,8三种面值的硬币，则，4,6,7都可以由这三种硬币组成，3却不可以.

找硬币

- 状态选择：
- dp[i][j]=1代表用前i种硬币可以组成价格j,相反dp[i][j]=0代表不能组成.
- 状态转移方程：
- 因为我们可以用第i种硬币也可以不用第i种硬币，所以
- dp[i][j]=dp[i-1][j]||dp[i-1][j-d[i]].
- 初始条件：
- dp[0][j]=0,1<=j<=w.
- dp[i][0]=1,0<=i<=n.

经典问题五：整数划分

- 问题描述：
- 正整数n表示成一系列将正整数之和. n=n1+n2+n3+...+nk(n1>=n2>=n3>=nk>=1, k>=1)这就是正整数n的一个划分，正整数n不同的划分个数称为正整数n的划分数, 记作p(n)例如：6 有如下11种划分:
- p(6)=6;5+1;4+2, 4+1+1;3+3, 3+2+1, 3+1+1+1;2+2+2, 2+2+1+1, 2+1+1+1+1;1+1+1+1+1+1;
- 求任意正整数的划分数p(n).

整数划分

- 状态选择：
- dp[n][m]代表在加数不大于m的情况下，n的划分数.
- 状态转移：
- 1.当m=1,dp[n][1]=1;
- 2.当m>n,dp[n][m]=dp[n][n];
- 3.当m=n,dp[n][n]=1+dp[n][n-1];
- 4.当m<n,dp[n][m]=dp[n][m-1]+dp[n-m][m].
- 练习题九：How many sums(toj1746).