动态规划(上)

九章官网下载最新课件

九章算法强化班 第5章

扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: ninechapter
微博: http://www.weibo.com/ninechapter
知乎: http://zhuanlan.zhihu.com/jiuzhang
官网: http://www.jiuzhang.com

Overview

- 滚动数组
  - House Robber I/II
  - Maximal Square
- 记忆化搜索
  - Longest Increasing Subsequence
  - Coin in a line I/II/III

动态规划的4点要素

1. 状态 State
   - 灵感，创造力，存储小规模问题的结果
     - 最优解/Maximum/Minimum
     - Yes/No
     - Count(*)
2. 方程 Function
   - 状态之间的联系，怎么通过小的状态，来求得大的状态
3. 初始化 Intialization
   - 最极限的小状态是什么，起点
4. 答案 Answer
   - 最大的那个状态是什么，终点

滚动数组优化

滚动数组优化

f[i] = max(f[i-1], f[i-2] + A[i]);

转换为

f[i%2] = max(f[(i-1)%2]和 f[(i-2)%2])

House Robber

http://www.lintcode.com/en/problem/house-robber/
http://www.jiuzhang.com/solutions/house-robber/

公主追王子
For循环 -----> DP

House Robber

- 序列型动态规划
- 状态 State
  - f[i] 表示前i个房子中，偷到的最大价值
- 方程 Function
  - f[i] = max(f[i-1], f[i-2] + A[i-1]);
- 初始化 Intialization
  - f[0] = 0;
  - f[1] = A[0];
- 答案 Answer
  - f[n]

House Robber

- 序列型动态规划
- 状态 State
  - f[i] 表示前i个房子中，偷到的最大价值
- 方程 Function
  - f[i] = 1. A[i-1] + max(f[i-2]....f[1]) 偷第i个房子
  - 2. f[i-1] 不偷第i个房子

  总结为
  - f[i] = max(f[i-1], f[i-2] + A[i-1]);
- 初始化 Intialization
  - f[0] = 0;
  - f[1] = A[0];
- 答案 Answer
  - f[n]

House Robber II

http://www.lintcode.com/en/problem/house-robber-ii/
http://www.jiuzhang.com/solutions/house-robber-ii/

滚动数组优化一维

- 这类题目特点
  - f[i] = max(f[i-1], f[i-2] + A[i]); 由 f[i-1],f[i-2] 来决定状态
- 可以转化为
  - f[i%2] = max(f[(i-1)%2]和 f[(i-2)%2]) 由f[(i-1)%2]和 f[(i-2)%2] 来决定状态
- 观察我们需要保留的状态来确定模数

- 其他一维滚动数组的题目
  - http://www.lintcode.com/en/problem/climbing-stairs/

Maximal Square

http://www.lintcode.com/en/problem/maximal-square/
http://www.jiuzhang.com/solutions/maximal-square/

小技巧

网格类的题目
正方形用右下角作为定位角
长方形可以用左上角和右下角作为定位角

Maximal Square

1. 状态 State
   f[i][j] 表示以i和j作为正方形右下角可以拓展的最大边长
2. 方程 Function
   if matrix[i][j] == 1
   　　f[i][j] = min(LEFT[i - 1][j], UP[i][j-1], f[i-1][j-1]) + 1;
   if matrix[i][j] == 0
   　　f[i][j] = 0
3. 初始化 Intialization
   f[i][0] = matrix[i][0];
   f[0][j] = matrix[0][j];
4. 答案 Answer
   max{f[i][j]}

Maximal Square

1. 状态 State
   f[i][j] 表示以i和j作为正方形右下角可以拓展的最大边长
2. 方程 Function
   if matrix[i][j] == 1
   　　f[i][j] = min(f[i - 1][j], f[i][j-1], f[i-1][j-1]) + 1;
   if matrix[i][j] == 0
   　　f[i][j] = 0
3. 初始化 Intialization
   f[i][0] = matrix[i][0];
   f[0][j] = matrix[0][j];
4. 答案 Answer
   max{f[i][j]}

Maximal Square

1. 状态 State
   f[i][j] 表示以i和j作为正方形右下角可以拓展的最大边长
2. 方程 Function
   if matrix[i][j] == 1
   　　f[i%2][j] = min(f[(i - 1)%2][j], f[i%2][j-1], f[(i-1)%2][j-1]) + 1;
   if matrix[i][j] == 0
   　　f[i%2][j] = 0
3. 初始化 Intialization
   f[i%2][0] = matrix[i][0];
   f[0][j] = matrix[0][j];
4. 答案 Answer
   max{f[i%2][j]}

Follow up

01矩阵里面找一个，对角线全为1，其他为0的正方形

二维动态规划空间优化

这类题目特点
f[i][j] = 由f[i-1]行 来决定状态,
第i行跟 i-1行之前毫无关系,
所以状态转变为
f[i%2][j] = 由f[(i-1)%2]行来决定状态

二维滚动数组相关题目

Unique Paths
http://www.lintcode.com/en/problem/unique-paths/

Minimum Path Sum
http://www.lintcode.com/en/problem/minimum-path-sum/

Edit Distance
http://www.lintcode.com/en/problem/edit-distance/

记忆化搜索

记忆化搜索

- 本质上：动态规划
- 动态规划就是解决了重复计算的搜索

- 动态规划的实现方式：
  - 循环（从小到大递推）
  - 记忆化搜索(从大到小搜索)
    - 画搜索树
    - 万金油

Longest Increasing Subsequence

http://www.lintcode.com/en/problem/longest-increasing-continuous-subsequence/

http://www.jiuzhang.com/solutions/longest-increasing-continuous-subsequence/

[4, 2, 5, 4, 3, 9,8,10]

Longest Increasing continuous Subsequence 2D

http://www.lintcode.com/en/problem/longest-increasing-continuous-subsequence-ii/
http://www.jiuzhang.com/solutions/longest-increasing-continuous-subsequence-ii/

10	2	7
2	3	6
11	4	5

Longest Increasing continuous Subsequence 2D

- 多重循环DP遇到的困难：
  - 从上到下循环不能解决问题
  - 初始状态找不到

- 那我们有没有可以比较暴力解决的方法呢？
  - 有搜索，我们从大的往小的搜索

Longest Increasing continuous Subsequence 2D

- 普通搜索

```
// 循环求所有状态
For i = 1 -> n
  For j = 1 -> n
    search (i,j) // 直接求i,j最为结尾最长子序列


int search(int x, int y, int[][] A)   {
    for(int i = 0; i < 4; i++) {
        nx = x + dx[i];
        ny = y + dy[i];
        if( A[x][y] > A[nx][ny]) {
            // 通过 search( nx, ny, A) 更新最长子序列
        }
    }
    //返回答案
}
```

Longest Increasing continuous Subsequence 2D

- 普通搜索

```
// 循环求所有状态
For i = 1 -> n
  For j = 1 -> n
   search (i,j) // 直接求i,j最为结尾最长子序列


int search(int x, int y, int[][] A)   {
    for(int i = 0; i < 4; i++) {
        nx = x + dx[i];
        ny = y + dy[i];
        if( A[x][y] > A[nx][ny]) {
            // 通过 search( nx, ny, A) 更新最长子序列
        }
    }
    //返回答案
}
```

- 记忆化搜索

```
// 循环求所有状态
For i = 1 -> n
  For j = 1 -> n
   dp[i][j] = search (i,j) // 直接求i,j最为结尾最长子序列


int search(int x, int y, int[][] A)   {
    if(flag[x][y] != 0) // 遍历过直接返回
            return dp[x][y];

    for(int i = 0; i < 4; i++) {
        nx = x + dx[i];
        ny = y + dy[i];
        if( A[x][y] > A[nx][ny]) {
            // 通过 search( nx, ny, A) 更新最长子序列
        }
    }
    //返回答案
}
```

Longest Increasing continuous Subsequence 2D

- 那怎么根据DP四要素转化为记忆化搜索呢？

- State: dp[x][y] 以x,y作为结尾的最长子序列
- Function:
  - 遍历x,y 上下左右四个格子
  - dp[x][y] = dp[nx][ny] + 1
    - (if a[x][y] > a[nx][ny])
- Intialize:
  - dp[x][y] 是极小值时，初始化为1
- Answer: dp[x][y]中最大值

```
1  // 循环求所有状态
2  For i = 1 -> n
3    For j = 1 -> n
4     dp[i][j] = search (i,j) // state定义
5
6
7  int search(int x, int y, int[][] A)   {
8      if(flag[x][y] != 0)
9              return dp[x][y];
10
11
12     dp[x][y] = 0;  // Intialize
13     for(int i = 0; i < 4; i++) { // Intialize
14         nx = x + dx[i];
15         ny = y + dy[i];
16         if( A[x][y] > A[nx][ny]) { // Function
17             dp[x][y] = Math.max(dp[i][j],  search( nx, ny, A) + 1);
18         }
19     }
20
21     return dp[x][y]; //Answer
22 }
```

什么时候用记忆化搜索？

1. 状态转移特别麻烦，不是顺序性。
2. 初始化状态不是很容易找到。

博弈类DP

Coins in a line

http://www.lintcode.com/en/problem/coins-in-a-line/
http://www.jiuzhang.com/solutions/coins-in-a-line/

博弈类DP

博弈有先后手

- State:
  - 定义一个人的状态
- Function:
  - 考虑两个人的状态做状态更新
- Intialize:
- Answer:

先思考最小状态

然后思考大的状态-> 往小的递推，那么非常适合记忆化搜索

Coins in a line

- 方法一
- State:
  - dp[i] 现在还剩i个硬币，现在先手取硬币的人最后输赢状况
- Function:
  - dp[i] = (dp[n-2]&& dp[n-3]) || (dp[n-3]&& dp[n-4])
- Intialize:
  - dp[0] = false
  - dp[1] = true
  - dp[2] = true
  - dp[3] = false
- Answer:
  - dp[n]

Coins in a line

- 方法二
- State:
  - dp[i] 现在还剩i个硬币，现在当前取硬币的人最后输赢状况
- Function:
  - dp[i] = true (dp[i-1]==false 或者 dp[i-2]==false)
- Intialize:
  - dp[0] = false
  - dp[1] = true
  - dp[2] = true
- Answer:
  - dp[n]

Coins in a Line II

http://www.lintcode.com/en/problem/coins-in-a-line-ii/
http://www.jiuzhang.com/solutions/coins-in-a-line-ii/

[5,1,2,10]

Coins in a Line II

- 方法一
- State:
  - dp[i] 现在还剩i个硬币，现在先手取硬币的人最后最多取硬币价值
- Function:
  - i 是所有硬币数目
  - coin[n-i] 表示倒数第i个硬币
  - dp[i] = max(min(dp[i-2], dp[i-3])+coin[n-i]) , (min(dp[i-3],dp[i-4])+coin[n-i]+coin[n-i+1])
- Intialize:
  - dp[0] = 0
  - dp[1] = coin[i-1]
  - dp[2] = coin[i-2] + coin[i-1]
  - dp[3] = coin[i-2] + coin[i-3]
- Answer:
  - dp[n]

Coins in a Line II

九章算法

- 方法二
- State:
  - dp[i] 现在还剩i个硬币，现在当前取硬币的人最后最多取硬币价值
- Function:
  - i 是所有硬币数目
  - sum[i] 是后i个硬币的总和
  - dp[i] = max(sum[i]-dp[i-1], sum[i] - dp[i-2])
- Intialize:
  - dp[0] = 0
  - dp[1] = coin[i-1]
  - dp[2] = coin[i-2] + coin[i-1]
- Answer:
  - dp[n]

Coins in a Line III

http://www.lintcode.com/en/problem/coins-in-a-line-iii

www.jiuzhang.com/solutions/coins-in-a-line-iii

Coins in a Line III

九章算法

- 方法一:
- State:
  - dp[i][j] 现在还第i到第j的硬币，现在先手取硬币的人最后最多取硬币价值
- Function:
  - left = min(dp[i+2][j], dp[i+1][j-1])+coin[i]
  - right = min(dp[i][j-2], dp[i+1][j-1])+coin[j]
  - dp[i][j] = max(left, right).
- Intialize:
  - dp[i][i] = coin[i],
  - dp[i][i+1] = max(coin[i],coin[i+1]),
- Answer:
  - dp[0][n-1]

Coins in a Line III

- 方法二:
- State:
  - dp[i][j] 现在还第i到第j的硬币，现在当前取硬币的人最后最多取硬币价值
- Function:
  - sum[i][j]第i到第j的硬币价值总和
  - dp[i][j] = max(sum[i][j] - dp[i+1][j], sum[i][j] - dp[i][j-1]);
- Intialize:
  - dp[i][i] = coin[i],
- Answer:
  - dp[0][n-1]

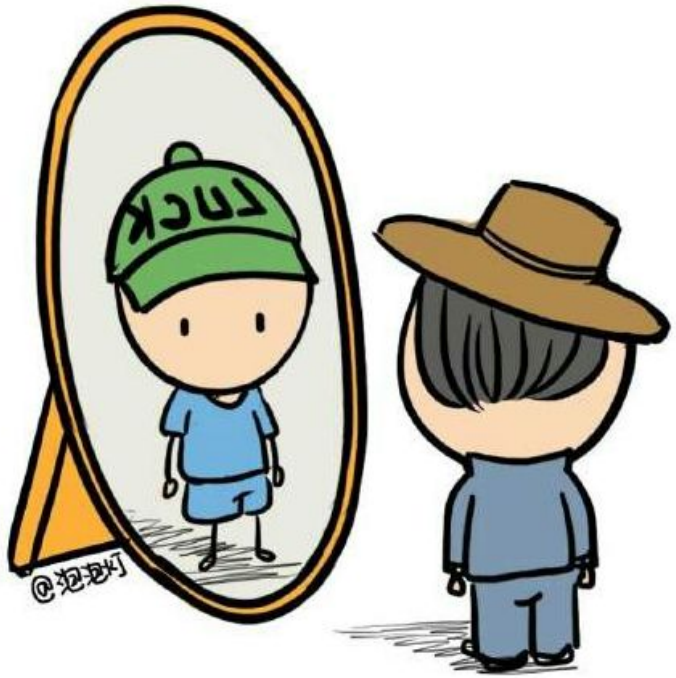
什么时候用记忆化搜索？

- 状态转移特别麻烦，不是顺序性。
  - **Longest Increasing continuous Subsequence 2D**
    - 遍历x,y 上下左右四个格子 dp[x][y] = dp[nx][ny]
  - **Coins in a Line III**
    - dp[i][j] = sum[i][j] - min(dp[i+1][j], dp[i][j-1]);

- 初始化状态不是很容易找到
  - **Stone Game**
    - **初始化dp[i][i] = 0**
  - **Longest Increasing continuous Subsequence 2D**
    - **初始化极小值**
- **从大到小**

今日重点三题

- House Robber
  - 滚动数组优化最简单的入门。
- Longest Increasing continuous Subsequence 2D
  - 记忆化搜索的经典题，此题只有记忆化搜索才能最优。
- Coins in a Line III
  - **博弈问题和记忆化搜索的结合**



The only person you should compare yourself to,

is the person you were yesterday.

唯一能够和你相比较的，就是那个曾经的自己。

Thank You