Lecture 8 Dynamic Programming(II)

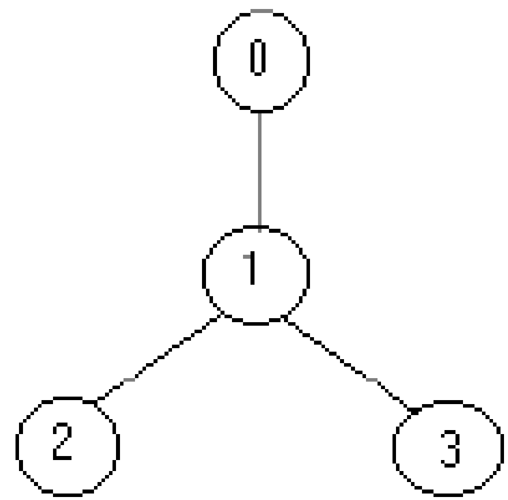
Lecturer: Han Lin

树型DP

- 树型DP：在树结构上的动态规划
  - 树本身就是一种递归结构
- 很多在一般图结构上是NP难的问题，在树结构上存在多项式时间的DP算法
  - 图着色问题
  - 最小点覆盖问题
  - etc.
- 算法实现通常借助DFS过程

例题：树的最小点覆盖

- 问题描述：给出一个n个结点的树，要求选出其中的一些顶点，使得对于树中的每条边(u, v)，u和v至少有一个被选中. 请给出选中顶点数最少的方案. (POJ 1463)
  - 例如：右图的最佳方案为选择顶点1.



- 如果给出的是一般的图，则不大可能存在多项式时间的算法.

思路

- 在树结构中，每个结点都是某棵子树的根
- 类似POJ1463，我们把n个结点分别用0~n-1编号
- 用ans[i][0]表示：在不选择结点i的情况下，以i为根的子树，最少需要选择的点数；

  用ans[i][1]表示：在选择结点i的情况下，以i为根的子树，最少需要选择的点数.

动态规划方程

- ans[i][0]表示：在不选择结点i的情况下，以i为根的子树，最少需要选择的点数；

  ans[i][1]表示：在选择结点i的情况下，以i为根的子树，最少需要选择的点数.

- 当i是叶子时，ans[i][0] = 0, ans[i][1] = 1;

  否则，

  ans[i][0] = Σans[j][1] (对于i的所有子结点j)

  ans[i][1] = 1+Σmin(ans[j][0], ans[j][1]) (对于i的所有子结点j)

算法实现及复杂度

- 算法实现：利用DFS的过程，参见程序P1463.c
- 算法复杂度：O(N)
- 小问题：有人认为上述算法并不能称之为动态规划，你知道是什么原因吗?
  - 答案：因为没有重复的子问题.

状态压缩DP

- 状态压缩DP，也被称为集合DP，因为将集合压缩为一个整数而得名.
- 用于很多经典问题，如TSP等.
- 通常不能得到多项式时间的算法，但由于避免了重复计算相同的子问题，而提高了算法的性能.

状态（子集）压缩的方法

对于集合S={0,1, 2, ..., N-1}，在N较小的情况下，S的子集A可以用一个N位二进制数x来表示.

表示方法：当i属于A时，x的第i位为1；否则为0.

例如：当N=16时，集合A={2，3，5}，可以表示为x=101100，即十进制整数44.

例题：图的最长路

- 问题描述：给出一个带权有向图G=(V, E)，求图中的一条最长简单路径. (Sicily 1123)
- 如果暴力穷举所有路径，复杂度不低于O(N!)

动态规划方程

- 状态表示：用ans[j][i]表示以j点为起点，并且经过点集i中的点恰好一次而不经过其它点的路径长度的最大值. 如果这个状态不存在，就是无穷小.
- 状态转移：

  如果i只包含一个点，ans[j][i] = 0;

  否则，ans[j][i] = max(graph[j][k] + ans[k][s])

  s表示i集合中去掉了j点的集合，k遍历集合s中的点，点j到点k有边存在，graph[j][k]表示边(j, k)的权值.
- 最后结果便是所有ans[j][i]的最大值.

算法实现和复杂度

- 算法实现：见程序S1123.c，注意其中集合运算如何用位运算来实现.
  - 判断点j是否属于集合i：i & (1<<j)
  - 在集合i中去除点j：i – (1<<j) 或者i & (~(1 << j))
  - 在集合i中加入点j：i | (1<<j)
- 时间复杂度
  - 状态数O(poly(N)*2^N)，其中poly(N)表示关于N的一个多项式.

动态规划优化概述

- 动态规划算法的复杂度通常取决于两方面：
  - 状态数
  - 转移的花费
- 因此，动态规划算法的优化也在于两方面：
  - 减少状态数，例如在记忆化搜索中，我们不会计算无用的子问题
  - 减少转移的花费，很常见

例题1：硬币问题

- 问题描述：给出N种硬币，每种硬币的面值为A[i]，个数为C[i]（1<=i<=N），问可以取到1~M中的多少个值. (POJ 1742)

基本算法：

用f[i][v] 表示价值v能否由前i种硬币取到，能则为true，否则为false.

状态转移方程：

$$f[i][v]=||\{f[i-1][v-k*A[i]] \mid 0<=k<=C[i]\}$$

复杂度是O(NM*max(C[i])).

多余的计算

我们可以发现上述过程有大量多余的计算.

例如，当A[i]=5，我们要计算f[i][100]，我们会去询问 f[i-1][95]，f[i-1][90]，f[i-1][85]，...

当我们要计算f[i][105]时，我们又会询问f[i-1][100]，f[i-1][95]，f[i-1][90]，f[i-1][85]，...

这种计算转移的方式导致大量重复的询问

新的计算方式

```
For i = 1 .. N
    For j = 1 .. M
        If F[i - 1, j] is true, T[i, j] = Ci. Else T[i, j] = -1
    For j = 1 .. M - Ai
        If T[i, j + Ai] = -1 and T[i, j] > 0
            F[i, j + Ai] = true
            T[i, j + Ai] = T[i, j] – 1
```

复杂度O(NM)!

例题2：最长上升子序列

- 问题描述：给出一个含有n个数的序列，求其中最长的单调递增子序列.

O(n^2)算法：

设 A[t]表示序列中的第t个数，F[t]表示从1到t这一段中以t结尾的最长上升子序列的长度，则有动态规划方程：

F[1] = 1,

F[t] = max{1, F[j] + 1} (j = 2, ..., t - 1, 且A[j] < A[t]).

优化思路

假设有两个元素A[x]和A[y]，满足

(1)A[x] < A[y]

(2)F[x] = F[y]

则F[y]在后面的计算中是肯定不会用到的！

启示：根据F的值进行分类. 对于F的每一个取值k，我们只需要保留满足F[t] = k的所有A[t]中的最小值. 设D[k]记录这个值，即D[k] = min{A[t]} (F[t] = k).

优化后算法的主要部分

```
D[0]=-1;
D[1]=A[1];
F[1]=1;
for (i=2;i<=n;i++)
 {
        j=find(D, A[i]); //找到最小的j，使得D[j] > A[i]
        D[j]=A[i];
        F[i]=j;
}
```

算法复杂度O(nlogn)

例题3：最优二叉搜索树

问题描述：见课本15.5节

O(n^3)算法：

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l .$$

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 , \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j . \end{cases}$$

$O(n\hat{}2)$算法

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1\,, \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j\,. \end{cases}$$

We define *root*[*i*, *j*], for $1 \le i \le j \le n$, to be the index *r* for which k_r is the root of an optimal binary search tree containing keys k_i, ..., k_j.

Ex15.5-4

Knuth show that $root[i, j-1] \le root[i, j] \le root[i+1, j]$ for all $1 \le i < j \le n$.

四边形不等式

这一结论的详细证明见《动态规划加速原理之四边形不等式》

作业

- 编程作业：

  POJ3342, Sicily1019： 树型DP

  POJ2817, 2288： 状态压缩DP

  POJ1742： 硬币问题

  POJ2533： 最长上升子序列

  POJ3709： 利用单调性进行优化的DP

See you next time!