**算法设计与分析 实验报告**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 王少博 | | | 院系 | | 软件学院 | | | 学号 | | 181110315 |
| 任课教师 | | 王金宝 | | | | | 指导教师 | | 王金宝 | | |
| 实验地点 | | 研究院1号楼5?7 | | | | | 实验时间 | | 2020年10月16日 | | |
| 实验名称 | | 使用动态规划技术的算法设计与分析 | | | | | | | | | |
| 同 组 人 | | 无 | | | | | | | | | |
| 实验内容（问题，思路，程序，结果） | | | | | | | | | 内容选择 | |  |
| **最长增长子序列问题定义如下：**  输入：由*n*个数组成的一个序列*S*：*a1,a2,…,an*  输出：*S*的子序列*S’=b1,b2,…,bk* ，满足：  (1) *b1≤b2 ≤ … ≤ bk* ，  (2) |*S’*|最大  使用动态规划技术设计算法求解最长增长子序列问题。请分析优化解的结构，递归地定义优化解的代价，给出算法的伪代码和实现过程中的主要代码，并分析算法的时间复杂性. 随机生成一个数组，并展示运行结果。  本次作业使用jupyter notebook编写。注：**如果有多组可行解，仅仅输出其中一个。**  由的定义可知，已经包含在最长递增子序列中，因此由定义可知，其一定不在中。反之，如果，我们证明其一定在中。如若不然 优化解结构 我们首先定义数组，表示以为结尾的最长递增子序列。定义集合。必然存在一个j，使得a[j]在中(否则，，加入之后依然满足最长递增子序列的定义，矛盾！)那么我们有：    j的选择不唯一。  我们证明:是以j为末尾的最长递增子序列。反之，存在其最长子序列为L，那么由于a[i]>a[j]，所以a[i]L，因此L+a[i]为L的一个递增子序列，且长度大于，矛盾！因此我们证明了该问题具有优化解结构。 重叠子问题   由于我们从大到小求解，所以求解的时候对于不同的j的选择，具有相同的子问题重叠。 递归求解  * dp[0:n-1]: dp[i]记录以第i个元素为结尾构成的最长递增序列长度 * pre\_node[0:n-1]：pre\_node[i]记录以第i个元素为结尾构成的序列前一个元素（用于构造最优解） * path[0:n-1]：path[i]记录以第i个元素为结尾构成的最长递增序列   我们经过上述的推导知，    当选择完j之后我们记录pre\_node[i]=j。 伪代码描述 getLength(a){   for i <- 0 to n-1:   dp[i] <- 1;   pre\_node[i] <- i;     for j <- 0 to n-1:   for i <- 0 to j-1:   if a[i]<=a[j] and dp[j]<dp[i]+1   dp[j]<-dp[i]+1; //更新dp[j]为最大的值   pre\_node[j]<-i; //记录选择的j  }  以i为尾的lis至少包含a[i]，所以dp[i]初始化为1；pre\_node[i]默认指向自己。  利用上述信息计算最长序列末尾元素。  getMaxLengthandPos(dp){ pos <- -1; length <- -1;  for i <- 0 to n-1:  if dp[i]>length:  pos <- i;  length <- dp[i]; return length, path[pos];  } 构造最优解 最优解的信息可以通过pre\_node直接求出，我们把最后的结果存储在了path中。  getPath(pre\_node, a){   for j in <- 0 to n-1:  if pre\_node[j]=j:  path[j] <- j;   else:  path[j] <- path[pre\_node[j]] + j;  } 可视化结果 我们随机采样一组30个点，然后做出lis的图像。其中x轴为序列元素的下标，y轴为每个序列元素的值。 | | | | | | | | | | | |
| 实验结论（结果分析、遇到的困难和解决方法等） | | | | | | | | | 备注 | |  |
| 复杂度分析：   1. 时间复杂度：计算代价时两层循环为O(n²)，构造最优解一层循环为O(n)，总体为O(n²)。经过调研发现利用二分法可以把复杂度降为O(nlogn)但是不是动态规划方法。 2. 空间复杂度：dp数组大小为n，pre\_node数组大小为n，path列表大小为O(n²)。注意到实际上path列表无需存在，直接在getPath()函数中打印信息即可。这里为了可视化结果所以存储了。如果不存path，我们的空间复杂度为O(n)。   遇到的困难主要就是一开始的思维和实际的求解时互逆的，我们通过当前一项考虑其优化子结构问题，但是最后求解的时候时自底向上求解，避免了重复子问题导致大量重复计算，这就是魅力所在吧。同时证明动态规划算法的适用性的两条缺一不可，这一点时我们常常忽略的。  **注：详细代码实现附在了jupyter notebok导出的pdf中。** | | | | | | | | | | | |
| 教师评价 | | | 备注 | |  | | | 得 分 | |  | |
|  | | | | | | | | | | | |