**算法分析与设计实验报告**

**第 三 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 李平凡 | 学号 | 201907040102 | | 班级 | 计科1905 |
| 时间 | 5.20 | 地点 | 软件大楼 | | | |
| 实验名称 | 字符距离问题 | | | | | |
| 实验目的 | 通过搜索资料，要求掌握一些基本问题和典型例题的处理方式 | | | | | |
| 实验原理 | 采用动态规划的方法找到状态转移方程，来进行自底向上的求解。状态转移方程的寻找来源于对题目的理解，题目中的最短拓展距离满足最优子结构性质，所以这个题目的最短拓展距离在处理最后一个点的时候要么来源于两个字符串最后一个字母的拓展距离加上前面剩下的字母的最短拓展距离和，要么来源于字符串a最后一个字母和空格的拓展距离和剩下字母与字符串b的全部相比的拓展距离，或者来源于字符串b最后一个一个字母和空格的拓展距离和剩下字母与字符串a的全部相比的拓展距离 | | | | | |
| 实验步骤 | ①首先输入两个字符串，利用string容器保存方便操作  ②利用动态规划的思想状态转移方程为min（s[i-1][j]+k，s[i][j-1]+k，s[i-1][j-1]+dist(m[i-1],n[j-1])）自底向上求解得到最后最短拓展距离  ③ 将最短拓展距离输出 | | | | | |
| 关键代码 | void val(string m,string n,int k)  {  int ml,nl;  ml=m.size();  nl=n.size();  int s[ml+1][nl+1];  s[0][0]=0;  for(int i=0;i<=ml;i++)  {  for(int j=0;j<=nl;j++)  {  if(i+j) //m和n字符串至少一个长度不为0  {  s[i][j]=INF;  int temp=s[i-1][j]+k;  if(i&&temp<s[i][j])  {  s[i][j]=temp;  }  temp=s[i][j-1]+k;  if(j&&temp<s[i][j])  {  s[i][j]=temp;  }  temp=s[i-1][j-1]+dist(m[i-1],n[j-1]);  if(i&&j&&temp<s[i][j])  {  s[i][j]=temp;  }  }  }  }  cout<<s[ml][nl];  }  首先初始化存最优解的数组，初始化第一个s[0][0]元素为0，因为两个字符串长度为0，也就相当于两个空格，距离为0，然后对于不断增长的循环变量，不断自底向上求解，最后得到最后结果 | | | | | |
| 测试结果 | 100字符串长度测试结果耗时0ms    近7000字符串长度测试结果耗时为0ms      近200000字符串长度测试结果耗时为4ms | | | | | |
| 实验心得 | 因为设置k值为1，所以对于计算机而言计算就会非常简便，因为这个题就可以简化成用空格来扩展字符串最后除非是两个相同的字符，否则就只会用空格来扩展，这样它的计算量就会大大减少，那么时间的开销就在于二维数组的计算上，因为这个二维数组是与字符串的长度紧紧挂钩所以最后的时间开销是O(m\*n)m对应字符串a的长度，n对应字符串长度b的长度。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |