研究生算法课课堂笔记

上课日期：12.22　　第(三)节课

组长：高飞　　组员：李凡丁

组员：崔文　　组员：丁宇辰

注意：请提交 Word 格式文档。

一、　NP 与计算的难解性

a) 第七次作业为第八章书后习题，20、26、37。作业不需要提交，但是属于课程要求的范围，需要理解相关内容。

b) 归约证明要注意归约方向：

   i. $X \leq_p Y$，代表问题 X 可以在多项式时间内归约到问题 Y，或者说问题 X 至少和问题 Y 一样难。

   ii. 如果要证明问题 Y 是属于 NPC 的，那么需要寻找一个已知属于 NPC 的问题 X，证明：

$$X \leq_p Y \quad \&\& \quad Y \in NP$$

   iii. 如果证明了$Y \leq_p X$，已知 X 是 NPC 问题的情况下，并不能说明 Y 也是 NPC 问题，只能说明 Y 比 NPC 问题 X 简单，但 Y 可能是 P 问题。但如果此时 X 是 P 问题，可以得到 Y 也是 P 问题。这是证明 Y 是 P 问题的一种方式。

   iv. 在证明$X \leq_p Y$时，只需要证明一般性的问题 X 可以在多项式时间内被归约到问题 Y 的某种实例即可。因为，如果 Y 的某种实例是 NPC 问题，那么 Y 也是 NPC 问题。

   v. 在证明$X \leq_p Y$时，需要说明问题 X 的解和多项式时间归约后问题 Y 的解是等价的（将两个问题的解对应）。

二、　第五次作业第一题 ------- 数组拆分问题

*类似算法导论中的矩阵乘法加括号，和数组合并（沙子合并）问题，算法设计一书的习题中对这类题目没有涉及。

a) 正确的解法：

设$\mathrm{opt}(i,j)$代表切分第i个切分点（不包括第i个切分点对应的字符）到第j个切分点（包括第j个切分点对应的字符）之间的字符串所需要的最小代价，即左开右闭区间。定义第 0 个切分点为字符串开头的前一个位置，第 m+1 个切分点为字符串结尾的位置。

$$\mathrm{opt}(i,j) = \min_{i<k<j}\{opt(i,k) + opt(k,j)\} + L(j) - L(i), 0 \leq i < j \leq m+1$$

$$opt(i,i+1) = 0$$

$$\mathrm{return\ opt}(0\,,\mathrm{m}+1)$$

此算法时间复杂度为$O(m^3)$，空间复杂度为$O(m^2)$。

※※ 意外的错误 1，计算拆分字符串固定代价的时候加 1：$L(j) - L(i) + 1$

如果按照正确的解法设计状态转移方程，则$\mathrm{opt}(i,j)$处理的$(L(i), L(j)]$区间的字符串，这个字符串的长度为$L(j) - L(i)$。

如果理解$\mathrm{opt}(i,j)$代表处理$[i,j]$区间的字符串，则状态转移方程应变为：（假设字符串下标为 1~n）

$$\mathrm{opt}(i,j) = \min_{i<L(k)<j}\{opt(i,L(k)) + opt(L(k)+1,j)\} + j - i + 1, 1 \leq i < j \leq n$$

$$opt(i,j) = 0, \text{if} \nexists L(k) \in [i,j]$$
$$\text{return opt}(1,n)$$

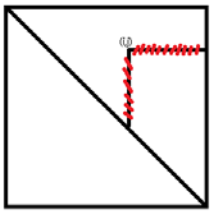
b) 反向追踪的时候需要记录什么信息？

需要记录每次取得最优解对应的切分点 k。空间复杂度增加$O(m^2)$，算法的时间复杂度增加$O(m)$。实际上，由于$opt(i,j)$中 i 一定小于 j，所以只需要半个矩阵就可以记录，而另外半个矩阵可以用来记录最优解对应的切分点 k，这样就省下了一半的空间。

如果不记录每次取得最优解对应的切分点，则可以通过再次运行算法得到切分序列。空间复杂度不会增加，时间复杂度最坏情况下增加$O(m^2)$，因为 OPT 数组是已经计算好的，每次需从 m 个拆分点选出最小的，一共选 m 次。但是就像快速排序一样，算法实际上不会总进入最坏情况。如果每次最小值都在一半处取到，T(m)=2T(m/2)+O(m)，时间复杂度为$O(m * \log m)$

c) 如果不要求得到拆分序列，只要一个最优拆分值，；能否运用滚动数组将记录子问题解得矩阵压缩到 1 维？

不可以。因为计算$opt(i,j)$的时候需要用到$opt(i,k)$和$opt(k,j)$，从保存子问题解的矩阵看来， (i,j)位置的解不仅仅依赖其后一行或者一列，而依赖如图所有位置的解，所以不可以压缩到 1 维。



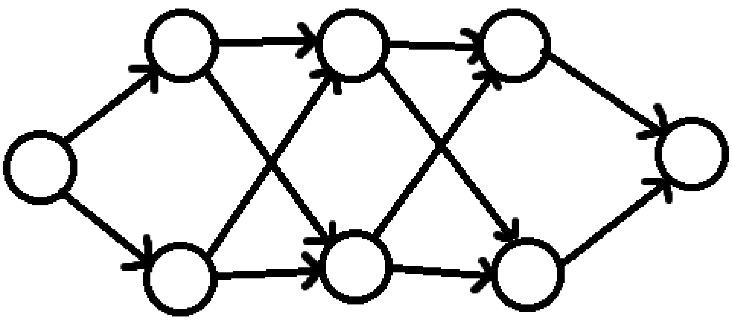
三、 第五次作业第二题 ------ 声音序列问题

a) 错误的解法 1

使用 DFS 或者 BFS 进行搜索，不进行判重操作。

算法流程大致为：定义 search(Vi , j)，对于一个节点 Vi 和对应的音素 j，找到所有 Vi 到达的节点 Vt，并且边对应的音素为 j，调用 search(Vt , j+1)。整个算法从 search(V0,1)开始，search(V1,2)search(V2,2)……直到访问到音素为 k 终止。

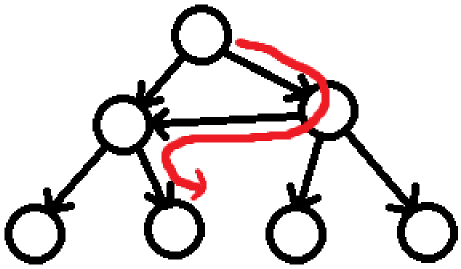
这个算法求得的结果是正确的，但是算法复杂度是指数级别的，因为同一节点可能被访问多次。比如下图，虽然使用搜索算法可以获得结果，但是显然这种网络结构更加适合动态规划求解。



b) 错误的解法 2

直接使用 DFS 或者 BFS，这两种搜索方法是不会重复访问相同的节点的。这就导致这两种方法无法获得正确的解。因为事实上，正确的解中可以存在重复节点，即有环，而直接使用 DFS 或者 BFS 是无法得到这样的解的。

即使正确的解中不存在重复的节点，如下图中的情况，假设红色的路线是正确的访问路线， DFS 或者 BFS 依然无法获得正确的解。



c) 正确的递归求解方法

在不判断重复的递归方法中加入一个哈希表，(Vi , j) 代表从 Vi 出发，找到 $\sigma_j, \sigma_{j+1}, \ldots \sigma_k$的最大概率。在递归调用 search 函数的时候查询、记录所得的结果即可。

这种方法虽然和动态规划很像，但是其求解子问题的顺序是不一样的。一般的动态规划需要求解所有子问题，而这种递归的方法只求解需要的子问题，即求解的问题数量少。但是递归程序本身有递归开销，所以整体算法的效率和动归之间应当是差不多的。

d) 预知后事如何，且听下回分解。