研究生算法课课堂笔记

上课日期：2016.10.31　　第(1)节课

组长学号及姓名：1601214442 王金予

组员学号及姓名：1601214439 李宁　1601214428 付天怡

动态规划-sequence alignment，树形动规

内容概要：

1. sequence alignment 问题
2. 背包问题（完全背包和 0-1 背包）的空间压缩和反向追踪
3. 图的独立集和树形动态规划

详细内容：

一. sequence alignment

1.Alignment

假设有两个字符串 x1, x2, x3, …, xm 与 y1, y2, y3, …, yn.需要对字符逐个配对，进行匹配，以判断这两个字符串的相似性。在不同的匹配模式下，可能会得到不同的结果。以字符串 x=”ocurrance”与 y=”occurrence”为例，有如下 3 种匹配模式：

①不插入任何空格，逐个匹配，如图 1 所示，由字符不一致产生 6 个错配（mismatch），由字符串长度不一致产生 1 个 gap（以空格与字符想配对的情形）；

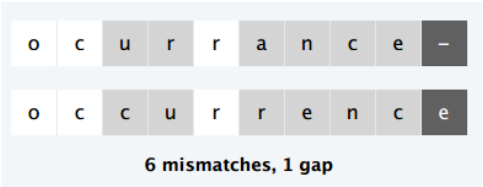

图 1

②在 x 的第 2 个和第 3 个字符间插入一个通配符空格，再逐个匹配，如图 2 所示，此时有 1 个由字符不一致产生的 mismatch 和 1 个由主动插入的空格产生的 gap;

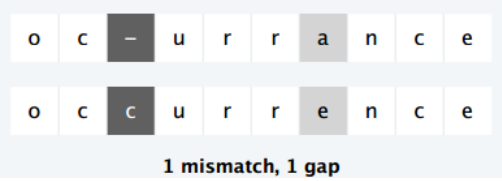

图 2

③一旦字符不一致，即插入通配符空格，如图 3 所示，此时全部 mismatches 被消除，但产生了 3 个 gaps.

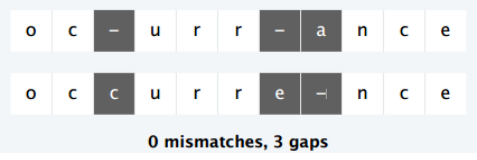

图 3

2.Edit Distance（编辑距离）

如何定量地表示两个串之间的相似性？如何衡量不同匹配模式下得到的结果的优劣性？这里引入一个编辑距离（Edit Distance）的概念。编辑距离，就是指将字符串 x 转换为字符串 y 所需的最少的编辑操作的次数。这里允许的编辑操作包括：

①将 x 中的某一个字符，替换成另一个字符（对应于 1 个 mismatch）;

②删除 x 中的一个字符（对应于在 y 中添加一个通配符空格，即 1 个 gap）；

③在 x 中插入一个字符（对应于在 x 中添加一个通配符空格，即 1 个 gap）。

一般来说，编辑距离越小，两个串的相似性越高。当然，为了表示不同编辑操作之间的差异性，对它们赋与不同的惩罚因子：

$\alpha_{x_i y_j}$：用 xi 与 yj 进行配对时产生的惩罚因子（mismatch/match）；

δ ：在 x 中删除或插入 1 个字符产生的惩罚因子（gap，对插入与删除不做区分）；

于是有两个串间的编辑距离（亦即可以理解成将串 x 转换成串 y 的代价 cost）：

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

公式 1

对于模式③下的匹配方式而言，如下图所示，有 Cost= $\alpha_{CG}+\alpha_{TA}+\delta$

图 4

3.求最小编辑距离的二维动规

（1）定义：OPT(i,j)表示对字符串 x 的前 i 个字符和字符串 y 的前 j 个字符进行匹配的最小代价。即 x 的前 i 个字符构成的子串与 y 的前 j 个字符构成的子串的最小编辑距离。

（2）递推公式：

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

公式 2

i=0：表示字符串 x 为空，所以字符串 y 全部和通配符进行匹配，代价为 jδ

j=0：表示字符串 y 为空，所以字符串 x 全部和通配符进行匹配，代价为 iδ

otherwise：有 3 种编辑操作供选择：

① 用 xi 与 yj 进行匹配，产生一个 match 或一个 mismatch，有惩罚因子$\alpha_{x_i y_j}$，加上子串剩余部分的最小编辑距离 OPT(i-1,j-1)；

② 在 y 的第 j 个字符后插入 1 个通配符空格，使 xi 与空格匹配，产生 1 个 gap，有惩罚因子 δ，再加上子串剩余部分的最小编辑距离 OPT(i-1,j)；

③ 在 x 的第 i 个字符后插入 1 个通配符空格，使 yj 与空格匹配，产生 1 个 gap，有惩罚因子 δ，再加上子串剩余部分的最小编辑距离 OPT(i,j-1)；

①②③分别对应公式中 min 后面的三项，选取①②③中代价最小的一项，即选择当前最佳编辑操作。

（3）初始值和返回值

初始值：OPT(i, 0)= iδ, OPT(0,j)= jδ;

返回值：假定字符串 x 的长度为 m, 字符串 y 的长度为 n , 则返回值为 OPT(m,n)。

（4）依赖关系

OPT(i, j)依赖于 OPT(i-1, j)，OPT(i-1, j-1)，OPT(i, j-1)，即向下，向左，向左下方依赖相邻的 OPT 值，如图 5：

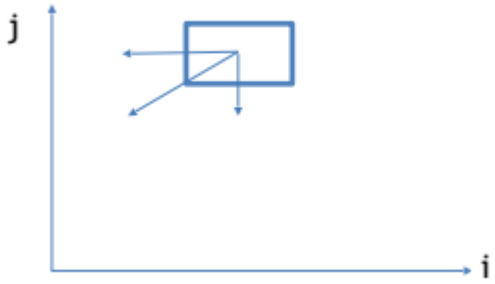

图 5

（4）时间复杂度 O(m*n)

这里的 O(m*n)指的是计算各 OPT(i, j)的时间复杂度。

若已知当前最优解 OPT(m, n)，反向追踪 match 过程，则时间复杂度为 O(m+n)。

（5）空间复杂度

由于每个 OPT(i, j)依赖于 OPT(i-1, j)，OPT(i-1, j-1)，OPT(i, j-1)三个值，空间可以压缩到 1 行多 1 个元素，如图 6 所示：

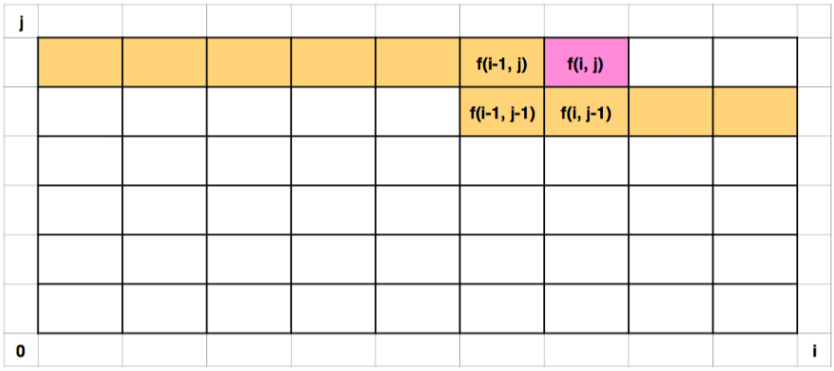

图 6

经空间压缩后如何实现反向追踪？Hirschberg’s algorithm 提出了一种反向追踪的方案。但是这种算法并不能解决其它很多问题中空间压缩后的反向追踪问题，普适性较差，适用情况有限，不做介绍。

二. 背包问题的反向追踪

1.完全背包问题（不要求恰好装满）

在空间限制为 O(W)的情况下，如何实现反向追踪，输出最优解的内容？

这里，空间复杂度为 O(W)的含义是：另外开一个一维数组来记录相关信息也合法。

策略：维护一个长度为 W 的一维数组，假设为 N[W]，用于记录对应重量 w 下，取得最优解时，最后放入背包的物品 N[w]。

追踪过程：不断地寻找当前重量下最优解所包含的最后一个物品。以图 7 为例：

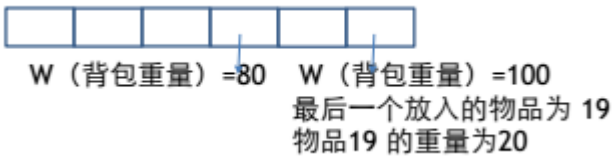

图 7

若当前追踪到 w = 100 时的最优解，且有数组 N 记录的信息 N[100] = 19。于是知，第 19 号物品被包含于此时的最优解中，假定第 19 号物品的重量为 20，于是继续搜索 w = 100 - 20 = 80 的最优解。由 N[80]又可知此时背包中包含的 1 个物品... 如此，反复地用当前重量 w 减去 N[w] 指示的物品的重量，继续追踪过程，即可找到完整路径。

注：

① 由于是完全背包问题，数组 N[ ]并非一定要记录最后一个物品，只需记录最优解中包含的任意一种物品，其意义不变；

② 在完全背包问题中，还涉及到物品 i 不装入背包的情况。此时，由于 f(w)的值保持不变，N[w]亦无需变动；

2.0-1 背包问题（不要求恰好装满）

如果是二维动态规划解决 0-1 背包问题，是不需要辅助结构来实现反向追踪的，只需用 f(i,w)和 f(i-1,w)进行比较，如果两个一样说明最后一件物品 i 没装进去，如果不一样，说明包含第 i 件物品，则找 f(i-1,w-w[i])，再用上述方法继续追踪。

在 0-1 背包问题中，当存储空间被压缩至一维后无法实现反向追踪，即使利用上述辅助数组也无法实现。原因在于 0-1 背包问题中，每件物品只能装 1 次或者不装，若存在性价比很高的物品，可能有多个 w 对应的最优解都包含有这个物品，于是他们的 N[w]记录的信息都相同，如下图所示：

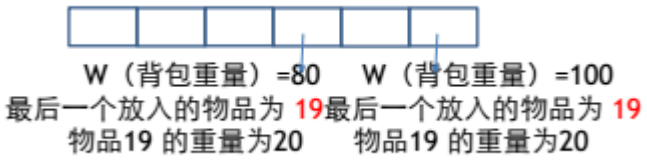

图 8

假定 19 号物品的性价比较高，重量为 20.当 w=80 和 w=100 时取得的最优解所包含的最后一件物品都为 19。则反追踪时输出第一个物品为 N[100]=19 号物品。接着取 w=100-20=80，看重量为 80 时的最优解。而 N[80]=19，违反了 0-1 背包的物品取用规则。因此，在这种情况下，无法实现 0-1 背包的反向追踪。

三. 树型动规

1.独立集（independent set）

（1）定义

给定图 G(V, E)，其独立集 V’满足：V’ ∈ V，对任意 u, v ∈ V’,一定有(u, v) ∉ E.

（2）求解树的最大独立集

对于给定一张图 G(V, E)，求解最大独立集是一个 NPC 问题，但当 G 为一棵树时，可以利用此时树的最大独立集的一些特性来简化问题：

① 树 G 的叶节点，一定可以包含于它的最大独立集中。

证明：假定树 G 有叶子节点 a 不在最大独立集 V’中，叶子节点 a 有父节点 b。于是可知一定有 b ∈ V’（否则，可以将 a 加入到 V’中，增大独立集 V’），此时，将 b 从 V’中取出，将 a 加入，不改变独立集 V’的大小|V’|.

②树 G 的最大独立集的求解实现：

类似于后序遍历（post-order traversal），寻找树 G 的所有叶子节点，加入到 V’中，在 G 中去除其父节点，得到新树，再重复上述过程，直至树空，这是一种贪心的策略。

（3）推广到一般的图

思考：对于如图 9 所示的一般的图，是否仍有叶子节点一定在最大独立集中？

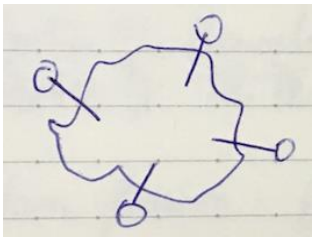

图 9

答案是肯定的。证明与（2）①类似。

对于贪心策略，要证明短时局部最优解为全局最优，常采用称之为“exchange argument”的证明方法：假定此时贪心策略选取的节点不在最优解中，对其进行一步步的变换，证明其不比全局最优更差。

但在求解一般图 G 的最大独立集时，贪心策略可能不能最终输出结果（在去除全部叶子节点后，无法继续贪心过程），但此时也已经将原问题转换为一个较简单的问题。

（4）求解权重和最大的独立集

叶节点：仅与其父节点相关联，对整体影响小；

父节点：除了叶节点外，还可能与其他节点相关联。

于是，① 当叶节点权重不小于父节点时，将叶节点添加至独立集中；② 当叶节点权重小于父节点时，需要采用动态规划做进一步的选取。

2. Anniversary Party

树型动规与一般动规相比，最显著的特点是：动规的过程按照树的拓扑结构进行，一般采取自底向上的方式进行。

（1）定义：f(u):以 u 为根的子树所产生的独立集的最大权重和。