PrefixESpan 算法实现报告

问题背景

本算法所需要解决的是挖掘树上的频繁嵌入子树问题：

对于N个有序、带标签、有根的树$\{T_i\}$，需要找出其中的频繁嵌入子树T'，即T'是其中超过min_sup棵树的嵌入子树。

嵌入子树指在若嵌入子树中v_1是v_2的父节点，则在原树中v_1是v_2的祖先。

PrefixESpan算法介绍

首先需要定义前缀树。对于一棵树的先序遍历中的前N个节点构成的节点就是这个树的N-前缀树。

PrefixESpan算法由以下三步构成。

第一步：首先找到所有频繁的标签。这些标签同时也是一个大小为1的频繁子树。

第二步：以这些大小为1的频繁子树为基础划分搜索空间，分别搜索以这些频繁子树为1-前缀树的频繁子树。

第三步：对每个已经找到的频繁子树，建立对应的投影数据库（projected database），递归搜索以该频繁子树为前缀树的更大的频繁子树。

以上的重点是投影数据库的建立。

一棵树T对一棵模式子树T'的投影数据库可以用以下方法建立：对于模式子树的一个节点T'^j，对T作先序遍历找到所有标签匹配的节点，该节点及其所有子树就构成一个投影实例。所有投影实例就构成了一个投影数据库。

在投影数据库中搜索频繁模式，不仅要满足标签相同，同时还需要满足连接位置相同两个条件。

实现细节

接下来使用C++语言实现这个算法。

树的数据结构

树的数据结构围绕节点构建，一个节点类（struct Node）由两部分组成，一部分是节点本身的性质，包括所在树的编号，在树中的位置（用先序遍历编号），标签；另一部分是树的结构信息，包指向其父节点的指针和一组指向其子节点的指针，另有一个辅助成员变量记录子树节点数量。

模式子树和投影数据库的数据结构

对于一个模式子树，可以用增量方法建立以该模式子树的投影数据库。模式子树本身是一棵树，投影数据库由若干个投影实例（struct ProjectInstance）构成。

一个投影实例类（struct ProjectInstance）带有原树编号，以及一个从模式子树上节点到原树上可连接的子树的节点的映射（`map<Node *, vector<Node *>>`。

起始时，模式子树都为1。对于数据库中的每一棵树，找到所有标签一致的节点，每一个节点都可以构建一个投影实例，编号为节点所在树的编号，在投影实例中记录下该节点到所有子节点的边。

当模式子树增长时，新的投影实例从原来的投影实例中构建，首先找到投影实例中连接位置与标签一致的后辈节点，该节点的所有子树连接到增长节点，而该节点到其连接点的所有祖先节点本身无法再被添加，但这些节点的兄弟节点，仍然可以连接到增长节点的连接点上。但为了避免重复搜索，应当排除所有位置在这些节点之前的节点。

实验结果

以下为分别在CSlog、D10、F5、T1M四个数据集上的测试结果，每格两个数字分别表示发现的频繁模式数量（个）和运行时间（毫秒）。（测试系统为MacOS，CPU为1.4GHz Intel Core i5，内存8GB 2133 MHz）

	CSlog	**D10**	**F5**	**T1M**
行数	59,691	99,999	100,000	1,000,000
sup%=3%	27/1818	38/751	182/1471	24/4437
sup%=2.5%	50/2111	63/965	93/1101	29/4635
sup%=2%	76/2280	105/1250	116/1197	38/4871
sup%=1.5%	118/23079	217/1746	182/1471	59/5779

注意到尽管CSlog的树的数量没有其他三个数据集多，但挖掘时间却更长，通过观察源数据，可以发现CSlog的树比其他数据集中的树更大，结构更加复杂。