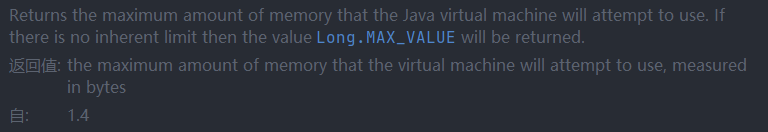
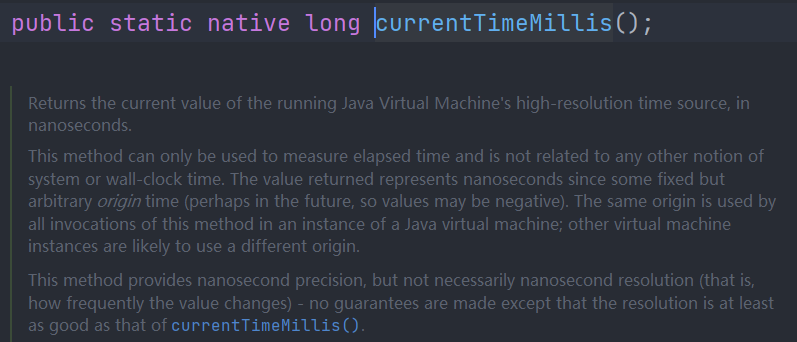
**实验对象：**

本次实验采用了三分图和五分图进行测试。目标是利用这两种图结构，从数据中抽取某些点，分别测量模型训练，一定数量embedding抽取所需的时间（单位纳秒）和空间代价（单位比特）

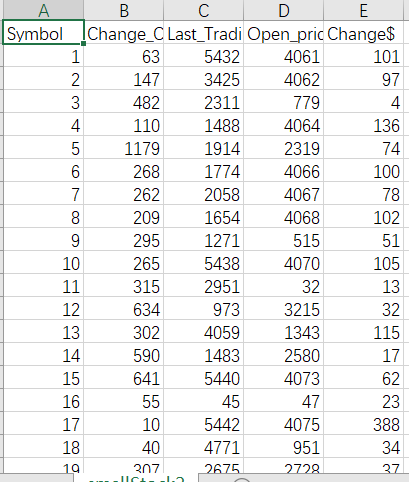
对于内存测量的说明：



对于时间测量的说明：只与同一主机内java虚拟环境中的时间相关，和系统时间无关，单位是纳秒。

  
**测试数据**：

测试所用数据为经过删减的stock数据集。包含了两个数据源，每个数据源的情况如下图所示：包含了50个元组，其中每个元组包含了5条属性：



具体实验数据情况请参看 ：\ICDE2021\gugu\log\newStock。内部有4个smallStock数据集，此处采用了前两个。这两个在原stock数据集中为数据源2和数据源3 所采集到的数据。

实验测试方式：

在graph embedding和word2vec的衔接代码部分，加入能随机抽取点，进行embedding获取的embedding函数。具体思路是：

首先获取生成的图的所有点的集合。然后在点集范围内，利用随机数生成下标，然后索引对应的点。获取到点的内容后，复用原始的get embedding函数，获取到随机抽到的点的embedding。其中抽取数目控制利用循环实现。注意，由于生成的随机数可能有相等的情况，要利用并查集结构，保证抽取的下标是互不相同的，然后整体size增加，直到等于带抽取元素数目。最后进行size检验，如果返回的size和目标size不一致，控制台有对应的警告信息。

存在的问题：即使用while循环约束维持了已抽取的索引下标的集合大小，但是最终获取到的embedding集合，还存在个数和预期不匹配的情形。代码如下所示：  


实验数据：

三分图原始数据存放在log/newStock/SourceThreeEMBDI\*.txt

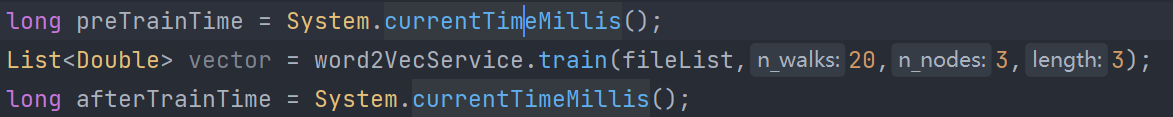
五分图原始数据存放在log/newStock/FifthGraphEMBDI/RandomFifthGraphEMBDI\*.txt

三分图总结：

一共进行6次抽样，每次抽样20个embedding。整体模型训练时间分别为：

70049，70085，71052，68577，71123（单位ns）。平均训练时间为：70177.2。

Ps：这里训练时间采用的是train函数的占用时间



实际运行时由于控制台有很多显示进度的辅助信息，实际运行时间可能在1分钟到2分钟之间。

获取随机抽取的20个embedding的时间分别为：

7,6,6,6,6,6.平均时间为：6.17。

此时间和控制台的表现相符：一旦结束训练，后续的工作即刻完成。

训练内存占用分别为：

508.8M，508.2M，500.7M，501.0M，505.2M，500.4M。平均内存占用为：504.05M。

**爆栈分析：**

8G电脑中推荐给java虚拟环境分配的内存是1024M，上限是1520M.初始化架构需要590M，IDE稳定运行时，常规上限是980M。对于本文介绍的数据集，能成功跑通，并且ide无异常，但是控制台初始化构建图时有明显卡顿，此时内存占用超出了IDE内存，但是在jdk要求范围内，所以实验能得到正确结果。但是之前的数据集中，对于4个数据源，100元组，13种属性的数据，会产生jdk VM的数组长度溢出，超过了int类型上届。将数据集缩小到2个时，会产生heap overflow的报错，即：爆栈。经过适当裁剪数据集，现在模型中的数据在可容忍的范围内能得到正确结果。

**提升方案：**

提升电脑内存，分配更多的虚拟内存。

**对于内存占用过多的猜想：**

每个数据集的个体大小较大，单次I/O代价很高，需要很大的内存占用。考虑将数据分小块进行读取是否能提高速率？（但是同时要保证对于数据集划分成不同的数据源，这对于用户输入的文件顺序有较高要求）

对于20个embedding的抽取的内存占用：平均占用为1.2M。占用空间很小。  
ps抽取方式见第一页思路，主要调用了medallia库的getRawVector（String s）方法。

对距离测量的说明：

这里采取的是计算了（余弦）相似度。

里面还是有负数！看和1的偏离程度就可以了（上次采用的softmax技术不知道为啥又失效了）

相似度测完怎么用emmm？大部分分布在0.2到0.4之间，少部分较高的达到0.6以上，少部分0.05左右表示基本无关。