图的表示：如何存储微博，微信等社交网络中的好友关系？

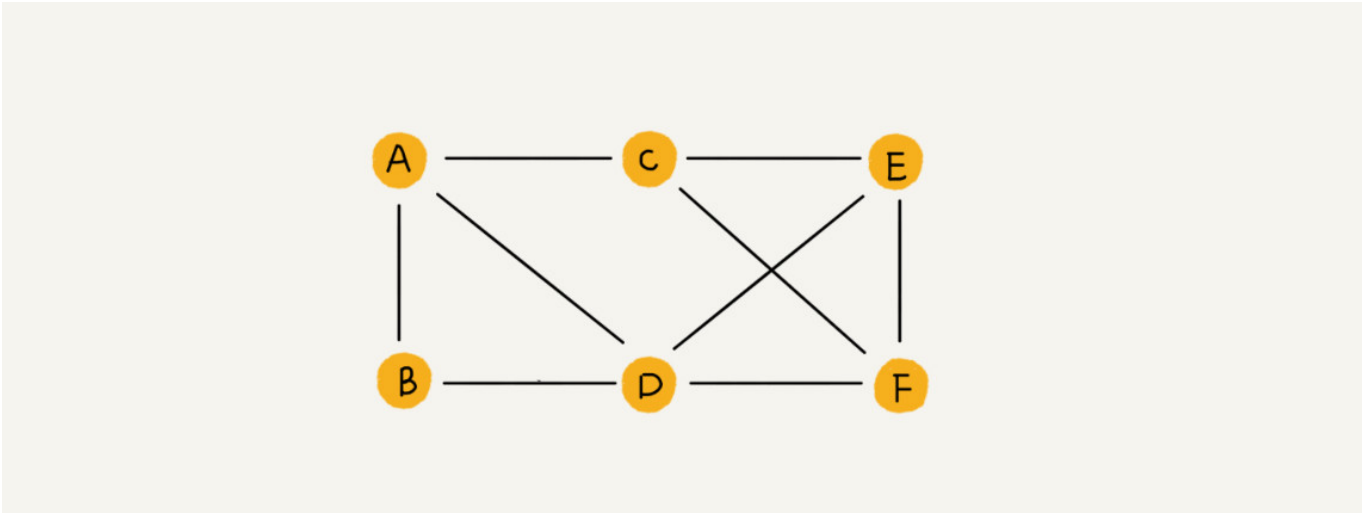
开篇问题，如何存储微博、微信等这些社交网络的好友关系？

图。实际上涉及图的算法有很多，也非常复杂，比如图的搜索、最短路径，最小生成树，二分图等等;

如何理解图呢？

图，和树比起来，是一种更加复杂的非线性表结构。

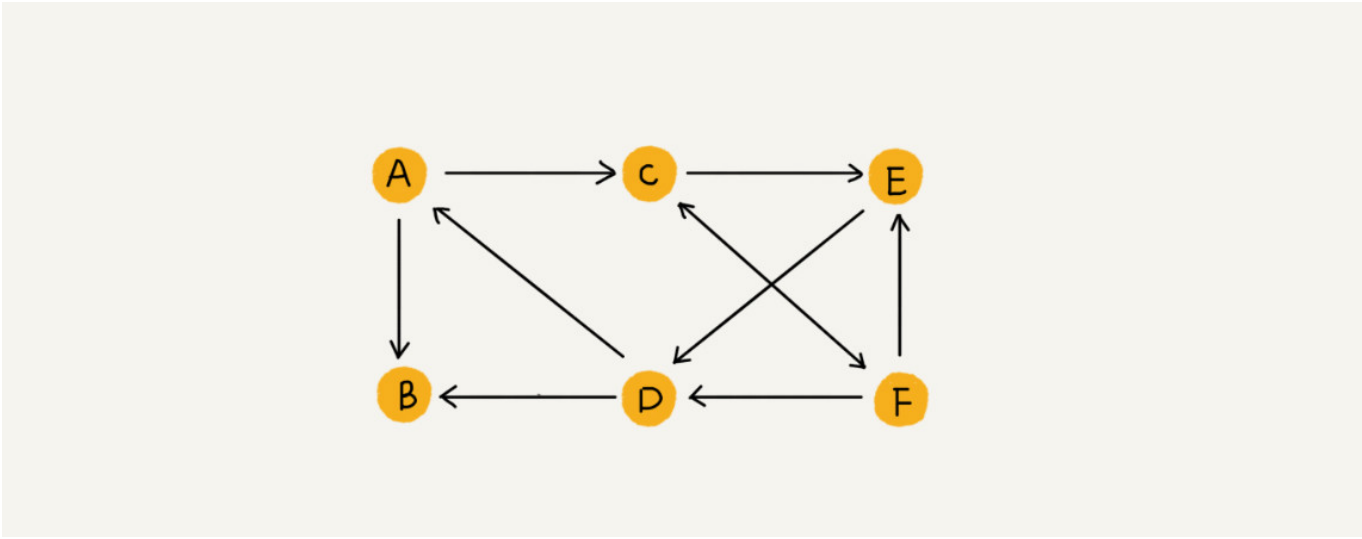
树中的元素我们称之为节点，图中的元素我们就叫做**顶点**,从下面的图中，可以看出来，图中的一个顶点可以和其他顶点建立连接关系，我们把这种建立的关系叫做**边**;



开篇问题中所说的社交网络，就是一个非常典型图结构。 微信网络：我们可以将每个用户看做是一个顶点。如果两个用户之间互加好友，那就再两者之间建立一条边。其中每个用户有多少个好友，对应到图中就是顶点的度。就是跟顶点相连的边的条数。

实际上微博的社交关系和微信之间还存在不一样，微博允许单向关注，这种单向关注表示在图中，我们就需要对图进行相应的改造。在图中引入方向的概念。

如果用户A关注了用户B，我们就再图中画一条A到B的带箭头的边，来表示方向。如果用户A和B相互关注了，那么我们就画一条从A指向B的边，再画一条从B指向A的边。我们把这种用方向的图叫做有向图。刚刚那种图，称之为无向图、



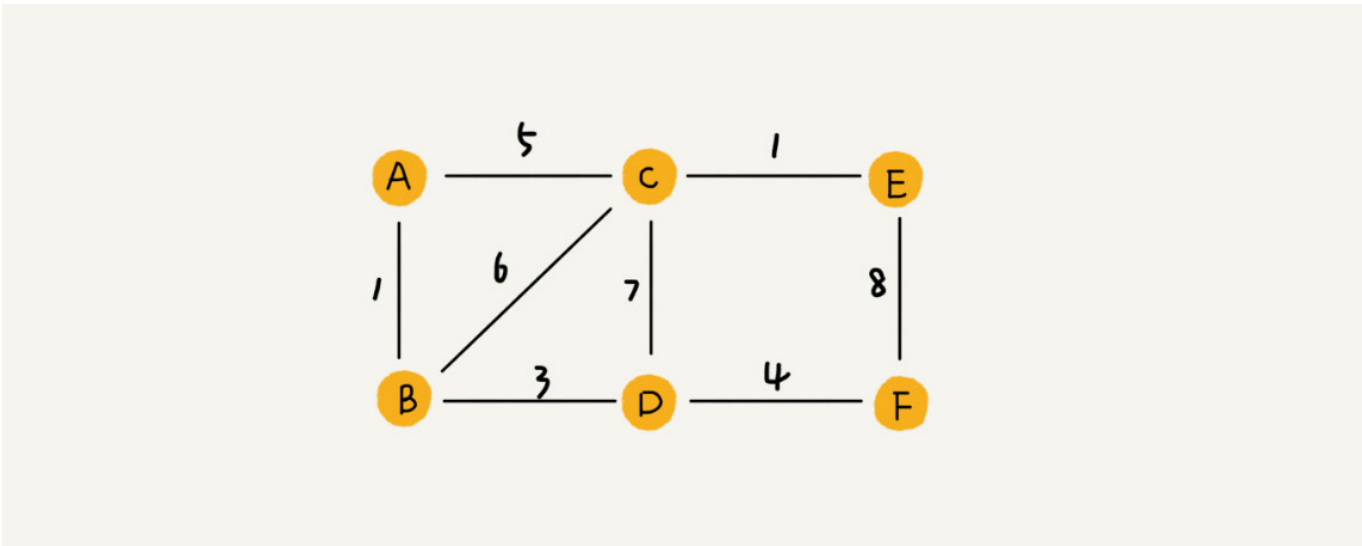
在无向图中我们讲了度的概念，表示一个顶点用多少边。在有向图中，我们把度分为**入度和出度**。

顶点的入度，表示有多少条边指向这个顶点；顶点的出度，表示有多少条边是以这个顶点为起点指向其他的顶点。 对应在微博中，入度就是粉丝人数，出度就是关注人数。

刚刚我们讲到无向图和有向图，我们再来看看另一种社交网络：QQ；

QQ中更加复杂，QQ亲密度这样的功能。QQ不仅记录了用户之间的好友关系，还记录了两个用户之间的亲密度。那么如何在图中表示亲密度这种情况呢？

这里就用到了另一种图，**带权图**。在带权图中，每条边都有一个权重，我们可以用这个权重来表示QQ好友之间的亲密度。

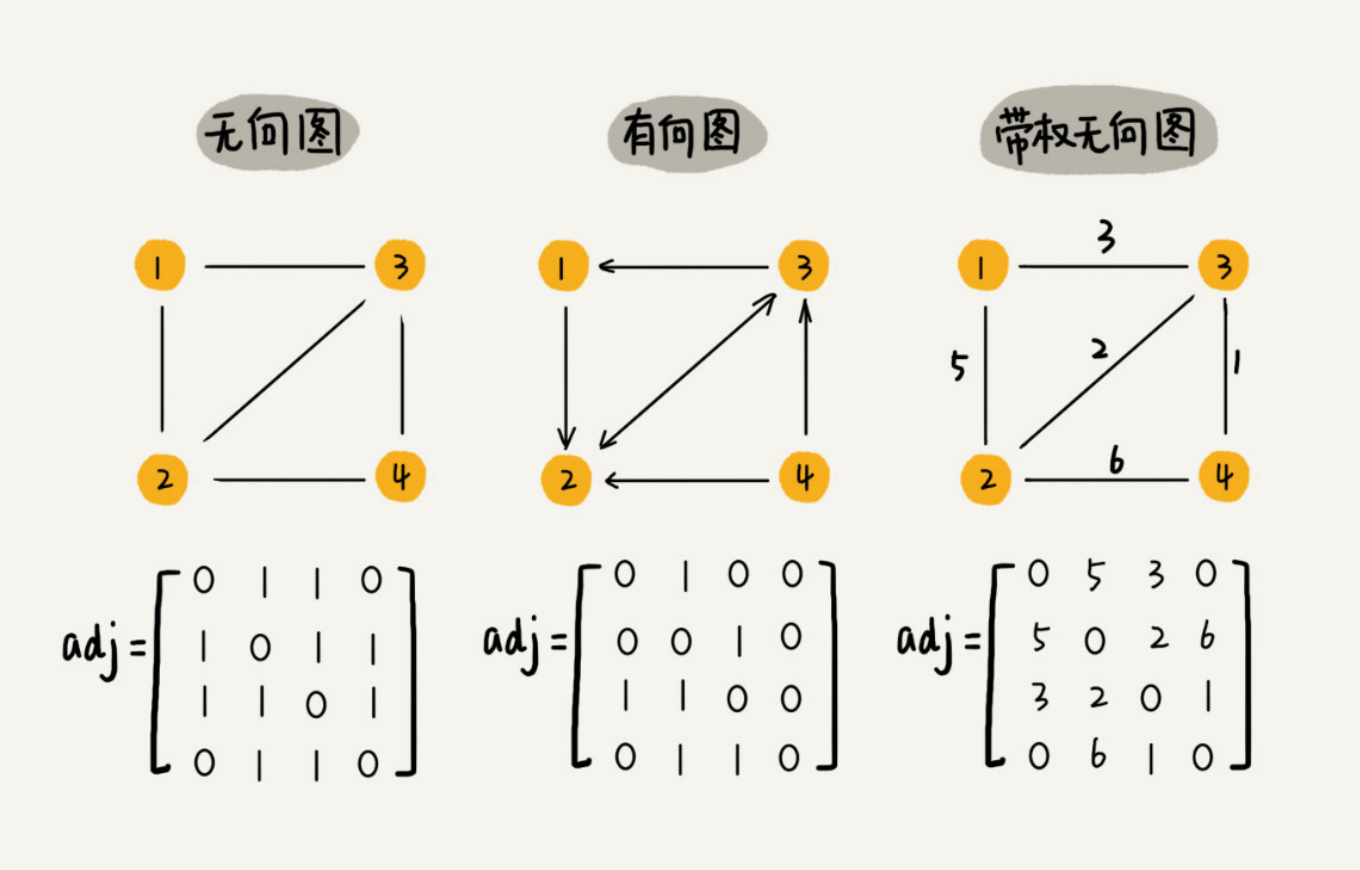


如何存储图这种数据结构呢？

邻接矩阵存储方法

图最直观的存储方法就是，**邻接矩阵**

邻接矩阵的底层依赖一个二维数组。对于无向图来说，如果顶点i和顶点j之间有边，我们就将A[i][j]和A[j][i]标记为1；对于有向图来说，如果顶点i到顶点j之间有一条箭头从i指向j的边，那我们就将A[i][j]标记为1。同理，如果有一条箭头从j指向i的边，我们就将A[j][i]标记为1。对于带权图，数组就存储相应的权重值。



邻接矩阵表示的图，虽然直观，简单，但是比较浪费存储空间。

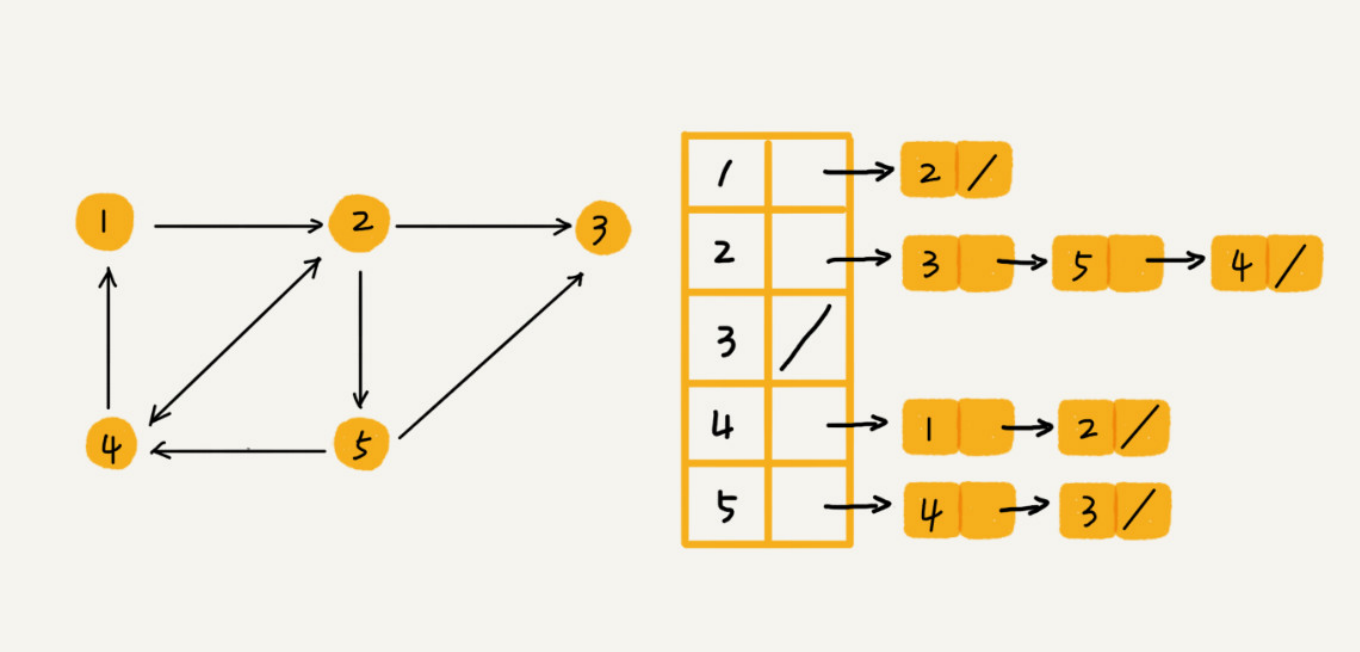
对于无向图来书，其实使用一半的矩阵就可表示出来，另外一半是白白浪费的。

如果我们存储的**稀疏图**，也就是顶点很多，每个节点上面的边并不多，那么这样的邻接矩阵的存储方法就更加浪费空间了。

但是邻接矩阵的优点是，邻接矩阵的存储方式简单，直接，在邻接矩阵中获取两个顶点的关系时，就非常的高效。其次用邻接矩阵存储的另一个好处是方便计算。

邻接表存储方法

邻接表乍一看，有点像散列表。每个顶点对应一个链表。链表中存储的是这个顶点对应连接的其他顶点。



邻接表这种存储方式比较节省空间，但是使用起来就比较耗费时间。

在上图中我们如果要在邻接表中确认一条从顶点2到顶点4的边，那么我们就要遍历顶点2对应的那条链表，查看链表中是否存在顶点4。而且链表的存储方式对缓存是不友好的。

邻接表如果拉链过长也可以类似于散列表一样，使用红黑树等更加高效的数据结构。

实际开发中我们可以将邻接表中的链表改成平衡二叉查找树等高效查找的数据结构。

解答开篇

有了前面的理论知识，我们来看开篇的问题，如何存储微博，微信等社交网络的好友关系？

前面我们分析了，微博，微信是两种图，前者是有向图，后者是无向图。在这个问题上，两者的解决思路差不多。

以微博来讲解。

首先我们要知道我们的需求是怎样的，也就是我们需要怎样的操作。

假设我们需要支持下面这样几个操作：

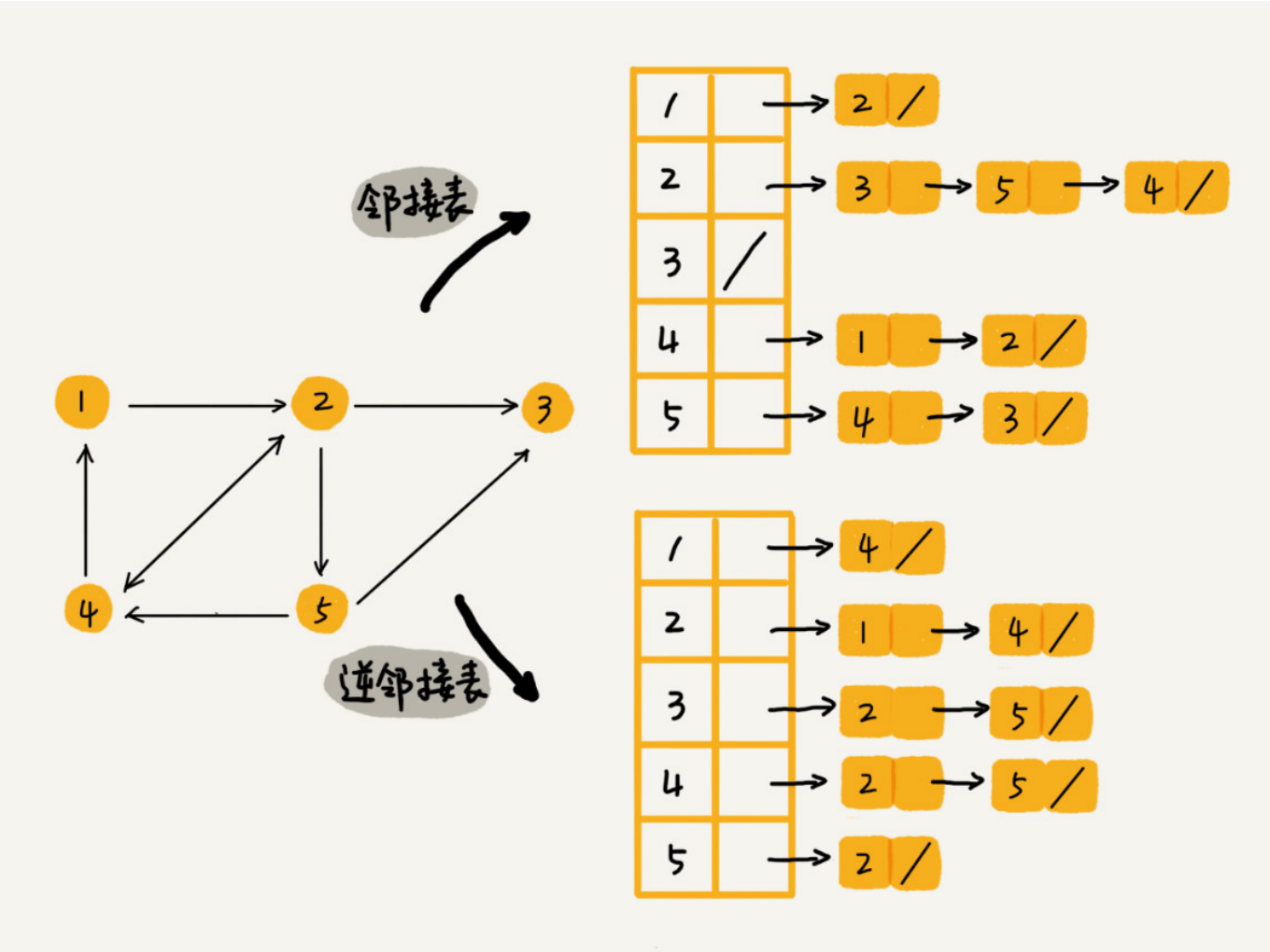
1. 判断用户A是否关注了B；
2. 判断用户A是否是用户B的粉丝；
3. 用户A关注用户B；
4. 用户A取消关注用户B；
5. 根据用户名称的首字母排序，分页获取用户的粉丝列表；
6. 根据用户名称的首字母排序，分页获取用户的关注列表；

然后我们考虑用怎样的方式来存储一个图，目前来说就是邻接矩阵和邻接表。因为社交网络是一个稀疏图，所以我们采用邻接表来存储。

但是邻接表来存储有向图是不够的。我们去查找某个用户关注了那些用户非常容易，但是如果想知道某个用户都被那些用户关注了，也就是用户的粉丝列表，是非常困难的。

基于这个问题，我们需要一张逆邻接表。邻接表中存储了用户的关注关系。逆邻接表中存储的是用户的被关注关系。 对应于图中逆邻接表中存储的是指向这个顶点的顶点。

如果要查找某个用户关注了那些用户，我们可以在邻接表中查找；如果查找某个用户被哪些用户关注了，我们从逆邻接表中查找。



基础的邻接表中不适合快速判断两个用户之间是否是关注与关注的关系，所以我们选择改进版本；

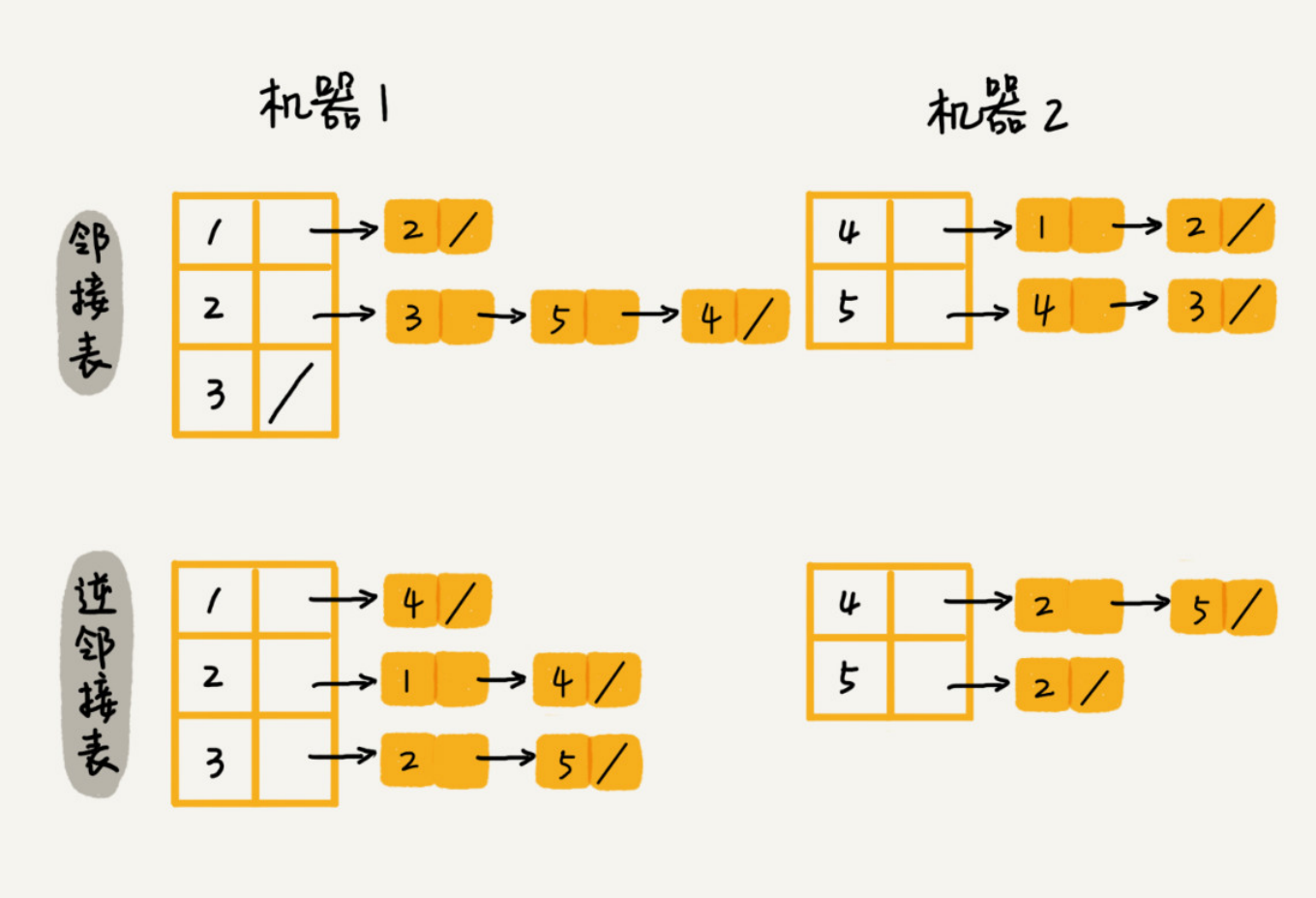
具体是选择哪种动态数据结构呢?

红黑树，跳表，有序动态数组还是散列表呢?

因为我们需要按照用户名称首字母来排序，分页来获取用户的粉丝列表或者关注列表，用调表这种结构在合适不过了。这是因为调表插入、删除、查找都非常高效。时间复杂度是O(nlogn),空间复杂度上稍高，是O(n);但是最重要的一点，在跳表中存储的数据本来就已经是有序的了，分页获取粉丝列表或者关注列表，就非常高效了。

对于小规模的数据，比如社交网络中只有几万，几十万个用户我们可以将整个社交关系存储在内存中，但是微博上亿的用户，无法全部存储在内存中，这个应该怎么办呢?

我们可以通过哈希算法，将数据分片，将邻接表存储在不同的机器上，如图所示，我们再机器1上面存储顶点1，2，3的邻接表，在机器2上存储顶点4，5的邻接表；同理逆邻接表处理方式一样。



除此之外 另一种解决思路，就是利用外部存储，因为外部存储的存储空间要比内存会宽裕很多。

内容小结

需要理解的几个概念：无向图，有向图，带权图，顶点，边，度，入度，出度。

除此之外还要理解：邻接矩阵和邻接表。

课后思考

1. 关于开篇思考题，我们只讲了微博这种有向图的解决思路，那像微信这种无向图，应该怎么存储呢？你可以照着我的思路，自己做一下练习。

无向图，其实只需要维护一张邻接表就可以实现；相比微博来说更加简单；

2. 除了我今天举的社交网络可以用图来表示之外，符合图这种结构特点的例子还有很多，比如知识图谱（Knowledge Graph）。关于图这种数据结构，你还能想到其他生活或者工作中的例子吗？

复杂网络中也是这样一张张图构成的，全国的交通路线图等等；