第五讲：图及其相关算法

王争

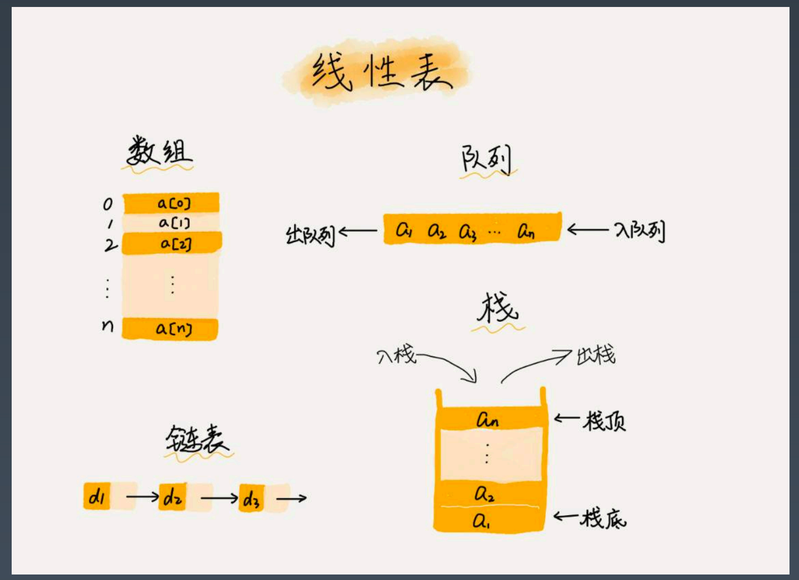
前 Google 工程师

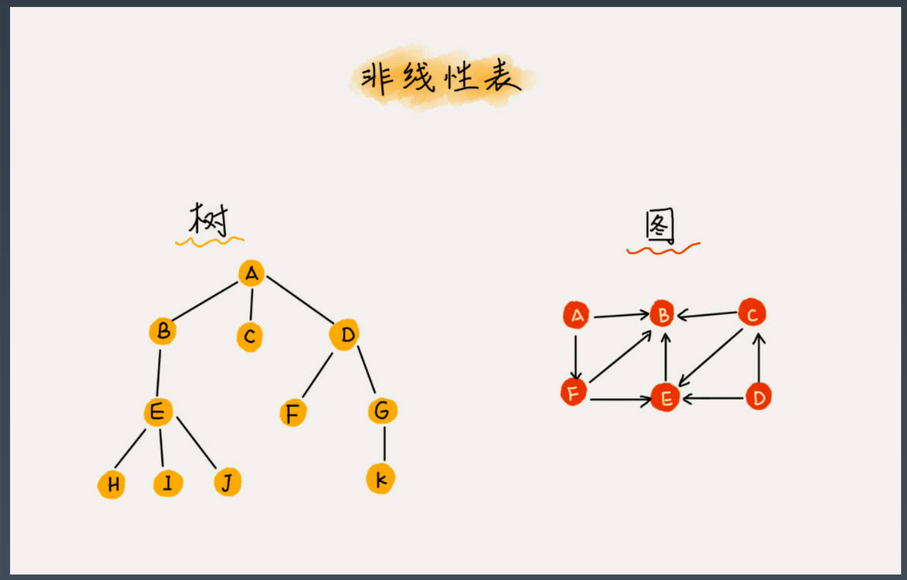
目录

图的概念与存储

图的定义

图是一种非线性数据结构，两个重要的概念：顶点，边





链表：结点，树：节点，图：顶点

图的应用场景

微博、微信等社交网络中的好友关系

地图导航、交通网络：最短路径

游戏地图、迷宫：A*寻路

计算机网络：最小生成树

人际关系、六度空间理论：推荐系统

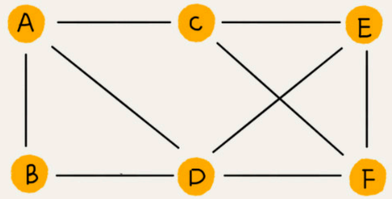
Gradle、Maven、Makefile 编译依赖关系：拓扑排序

互联网中的网页构建成一张大图：搜索引擎的爬虫

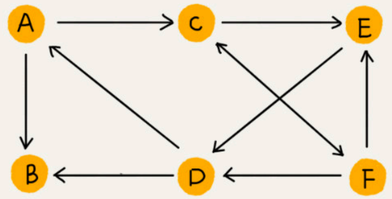
知识图谱（Knowledge Graph）：征信、搜索

图的分类

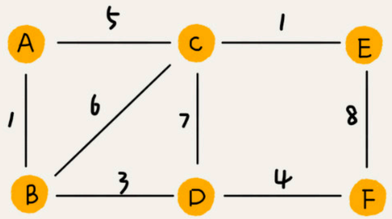
无向图



有向图



有权图



图的表示（存储）方法

1. 存储在内存中

2. 持久化存储在数据库中

3. 存储在专业图数据库中，比如 neo4j，存储超大图并且涉及大量图计算

图在内存中表示（存储）方法

1. 邻接矩阵
2. 邻接表
3. 逆邻接表
4. 十字交叉链表
5. 邻接多重表

图在内存中表示（存储）方法一邻接矩阵

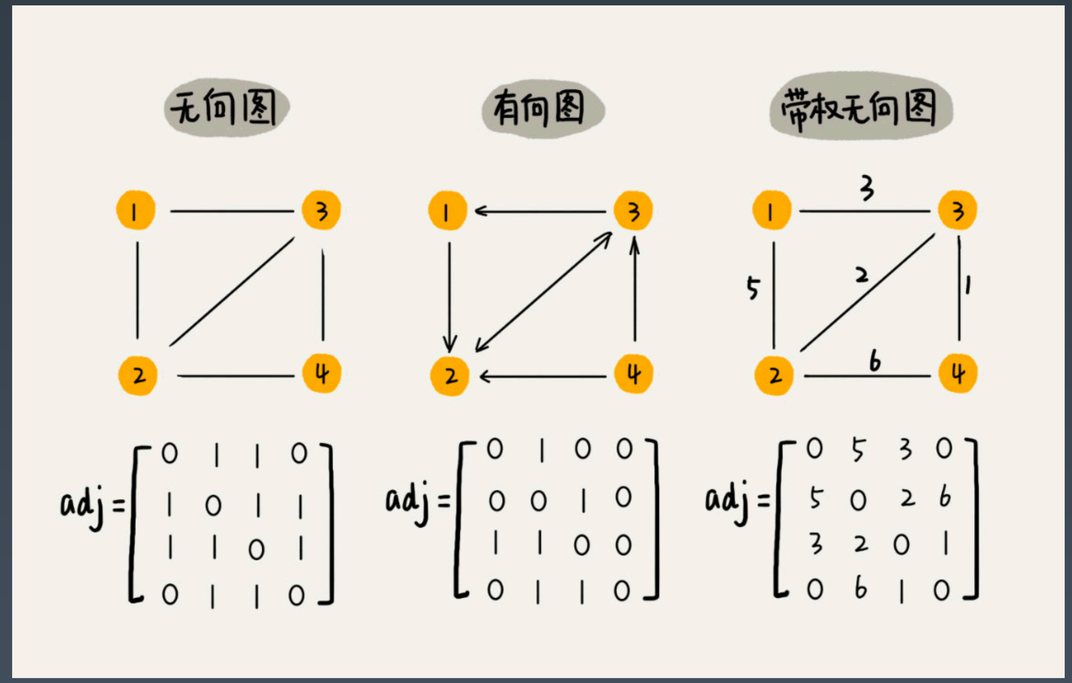
1. 邻接矩阵

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 与顶点 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，有一条箭头从顶点 i 指向顶点 j 的边，那我们就将 A[i][j] 标记为 1。同理，如果有一条箭头从顶点 j 指向顶点 i 的边，我们就将 A[j][i] 标记为 1。

对于带权图，数组中就存储相应的权重。



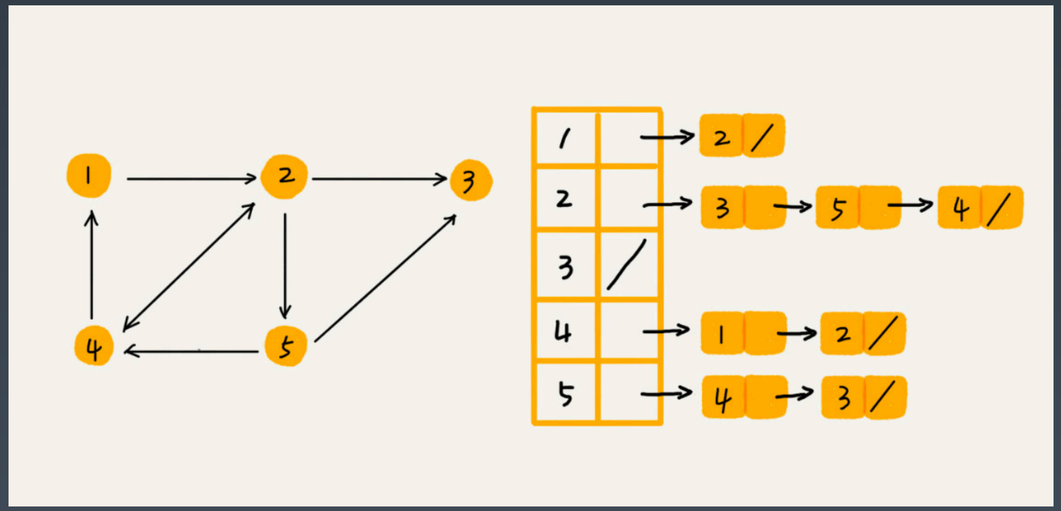
图在内存中表示（存储）方法一邻接表

2. 邻接表

```
public class Graph {
  private int v; // 顶点的个数
  private LinkedList<Integer> adj[]; // 邻接表

  public Graph(int v) {
    this.v = v;
    adj = new LinkedList[v];
    for (int i=0; i<v; ++i) {
      adj[i] = new LinkedList<>();
    }
  }

  public void addEdge(int s, int t) {
    adj[s].add(t);
  }
}
```



每个顶点对应一条链表，链表中存储的是与这个顶点相连接的其他顶点。

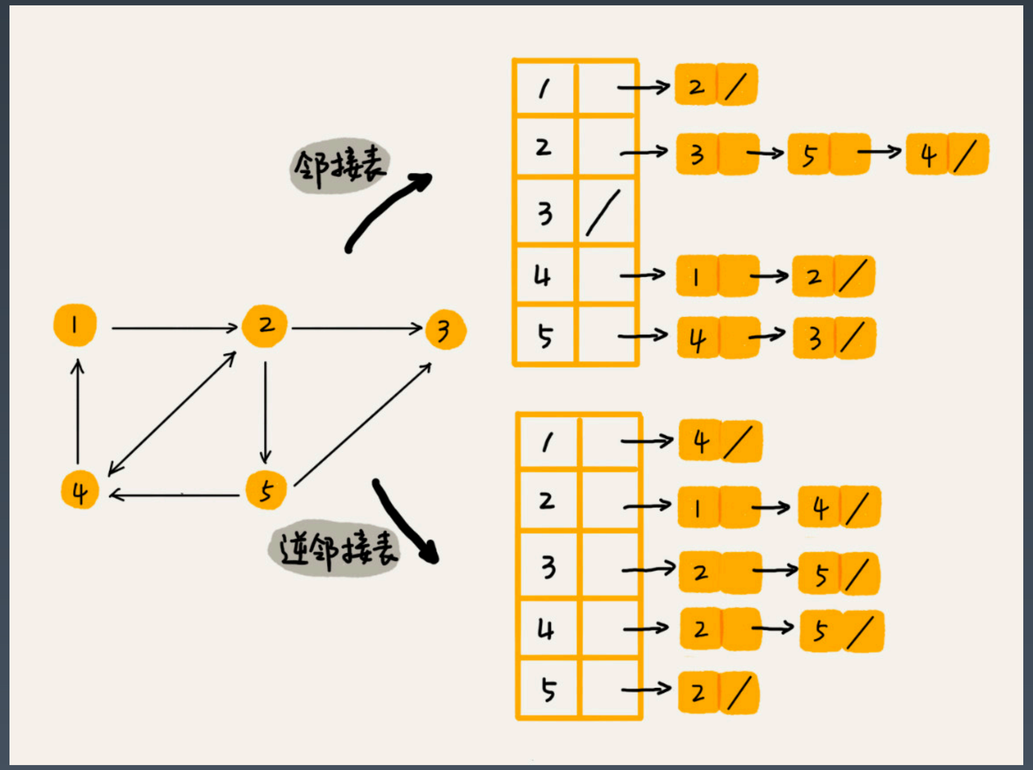
图在内存中表示（存储）方法—邻接表

3. 逆邻接表

```
public class Graph {
  private int v; // 顶点的个数
  private LinkedList<Integer> adj[]; //  逆邻接表

  public Graph(int v) {
    this.v = v;
    adj = new LinkedList[v];
    for (int i=0; i<v; ++i) {
      adj[i] = new LinkedList<>();
    }
  }

  public void addEdge(int s, int t) {
    adj[t].add(s);
  }
}
```



关于图的一个设计面试题分析

如何存储微博这种社交网络中的好友关系?

1. 需求分析与定义
挖掘、假设、估算：将开放问题转化成可解问题

2. 建模与算法
将复杂的场景抽象成具体的数据结构，构建对应的算法

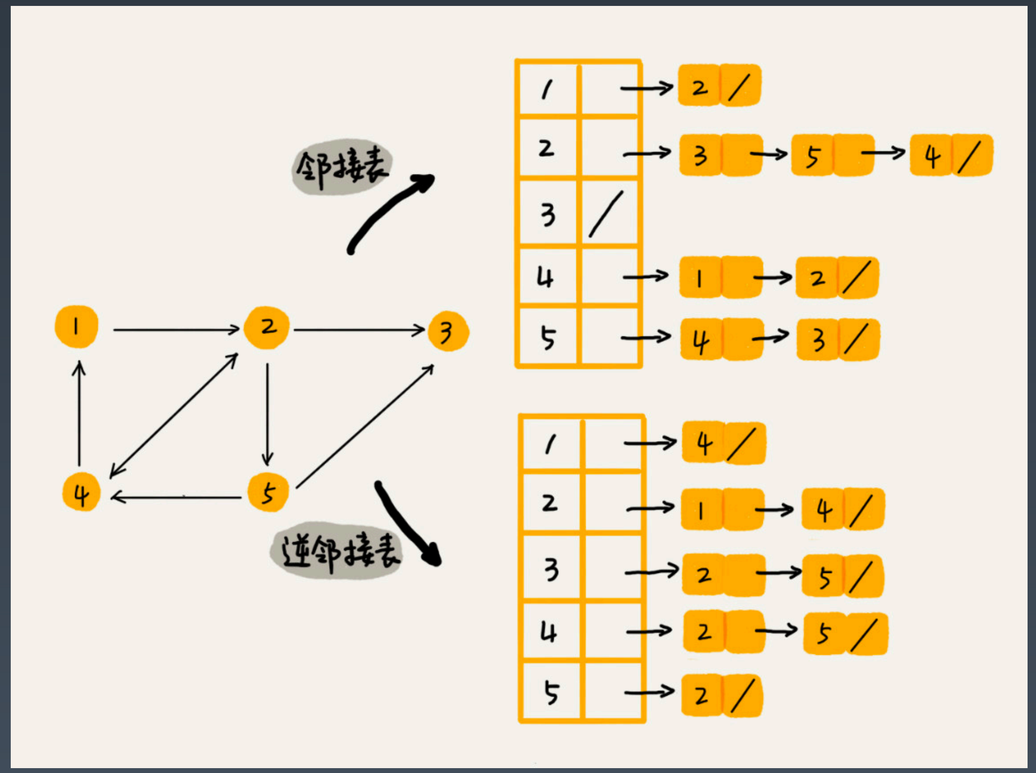
关于图的一个设计面试题分析

针对微博用户关系，假设我们需要支持下面这样几个操作：

- 用户 A 关注用户 B
- 用户 A 取消关注用户 B
- 判断用户 A 是否关注了用户 B
- 判断用户 A 是否被用户 B 关注
- 根据用户名称的首字母排序，分页获取用户的粉丝列表
- 根据用户名称的首字母排序，分页获取用户的关注列表

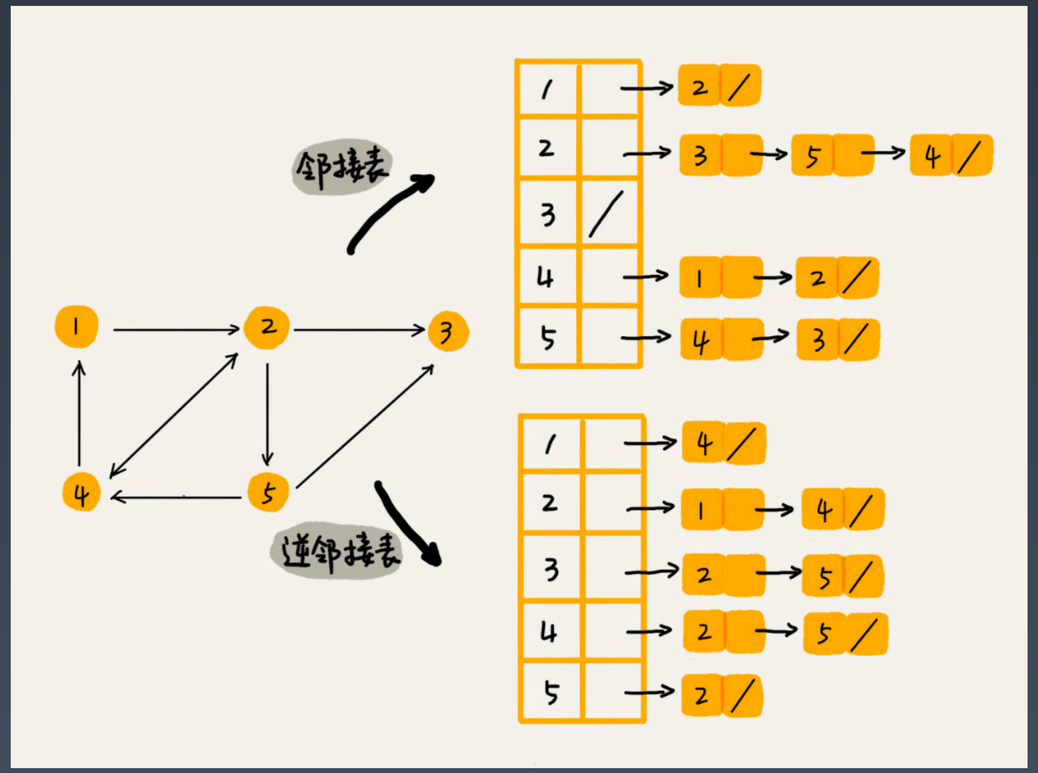
数据结构是为算法服务的，所以具体选择哪种存储方法，与期望支持的操作有关系。

关于图的一个设计面试题分析



- 用户 A 关注用户 B
- 用户 A 取消关注用户 B
- 判断用户 A 是否关注了用户 B
- 判断用户 A 是否被用户 B 关注
- 根据用户名称的首字母排序，分页获取用户的关注列表
- 根据用户名称的首字母排序，分页获取用户的粉丝列表

关于图的一个设计面试题分析



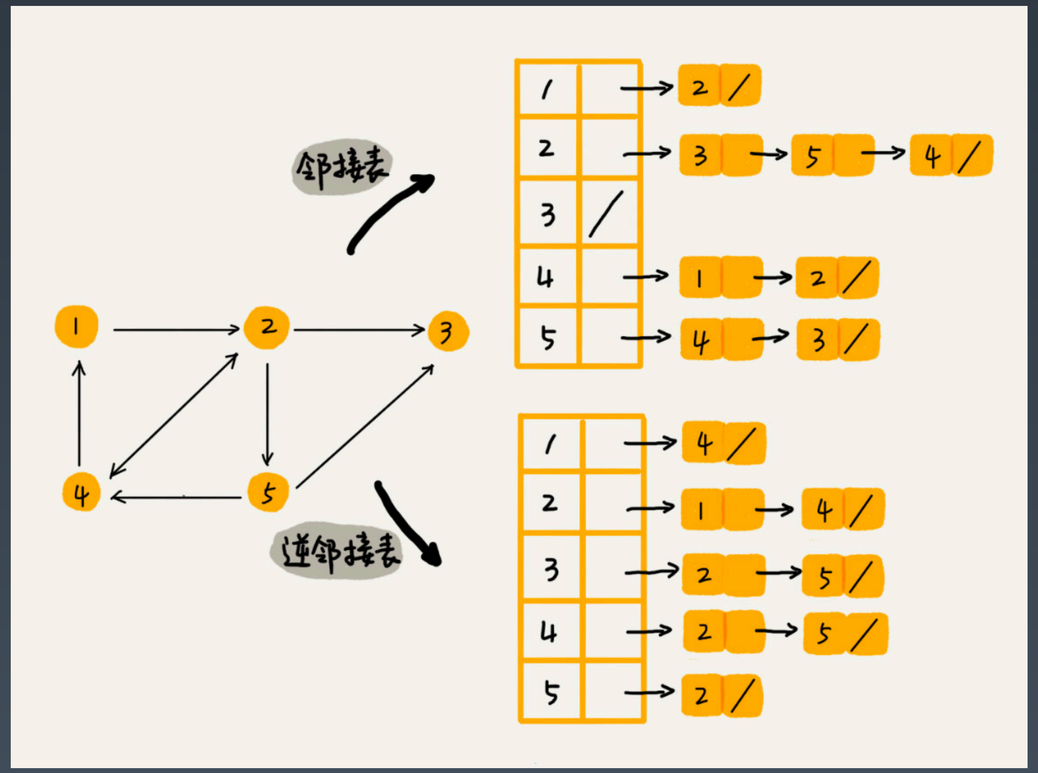
如何更加快速地判断两个用户之间是否存在关注与被关注的关系?

优化思路:

将邻接表中的链表改为支持快速查找的动态数据结构：红黑树、跳表、有序动态数组还是散列表呢?

跳表插入、删除、查找都非常高效，时间复杂度是 O(logn)，空间复杂度上稍高，是 O(n)。跳表中存储的数据本来就是有序的，分页获取粉丝列表或关注列表，就非常高效。

关于图的一个设计面试题分析



如何更加快速地判断两个用户之间是否存在关注与被关注的关系?

优化思路:

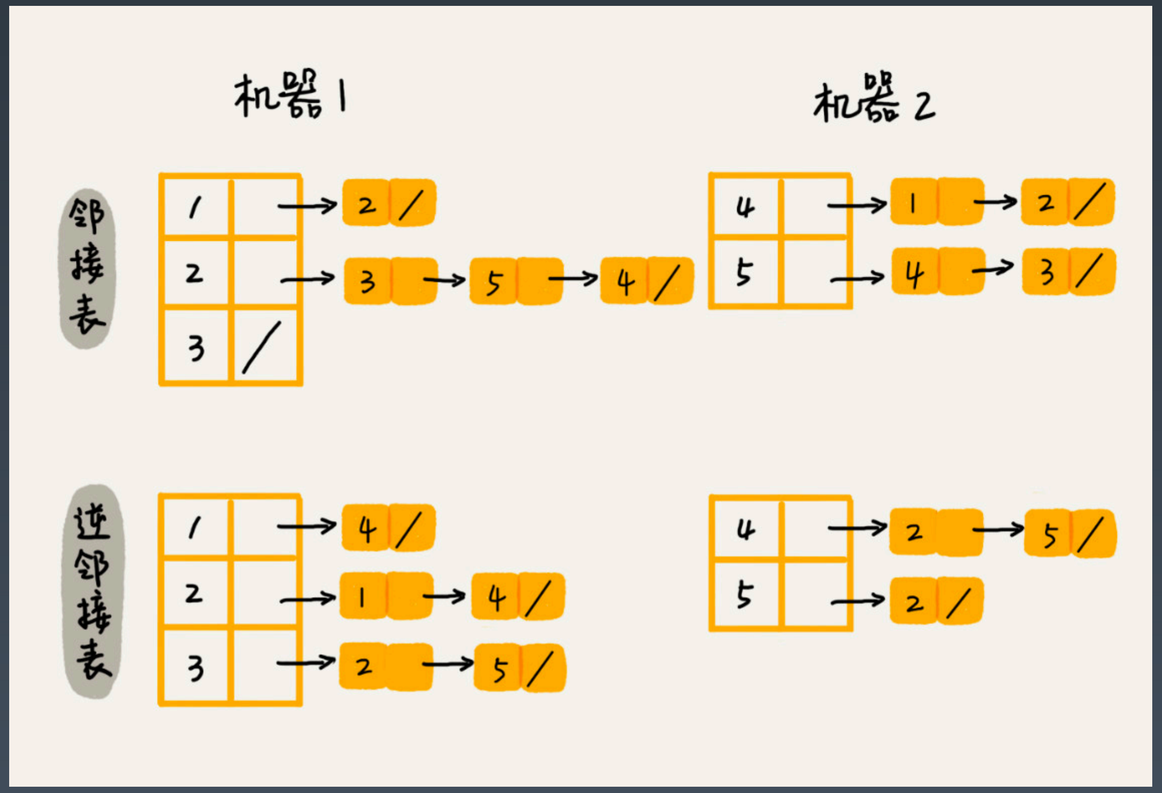
将邻接表中的链表改为支持快速查找的动态数据结构：红黑树、跳表、有序动态数组还是散列表呢?

跳表插入、删除、查找都非常高效，时间复杂度是 O(logn)，空间复杂度上稍高，是 O(n)。跳表中存储的数据本来就是有序的，分页获取粉丝列表或关注列表，就非常高效。

关于图的一个设计面试题分析

如果对于小规模的数据，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，上面的解决思路是没有问题的。但是如果像微博那样有上亿的用户，数据规模太大，我们就无法全部存储在内存中了。这个时候该怎么办呢?

关于图的一个设计面试题分析



我们可以通过哈希算法等数据分片方式，将邻接表存储在不同的机器上。

我们在机器 1 上存储顶点 1，2，3 的邻接表，在机器 2 上，存储顶点 4，5 的邻接表。

逆邻接表的处理方式也一样。

当要查询顶点与顶点关系的时候，我们就利用同样的哈希算法，先定位顶点所在的机器，然后再在相应的机器上查找。

关于图的一个设计面试题分析

user_id	follower_id
1	4
2	1
2	4
3	2
3	5
4	2
4	5
5	2

另外一种解决思路：

就是利用外部存储（比如硬盘），因为外部存储的存储空间要比内存会宽裕很多。

数据库是我们经常用来持久化存储关系数据的。

为了高效地支持前面定义的操作，我们可以在表上建立多个索引，比如第一列、第二列，给这两列都建立索引。

有关图的算法介绍

图相关算法引申介绍

广度、深度优先：图遍历算法、无权图的搜索算法

单源最短路径：Dijkstra算法

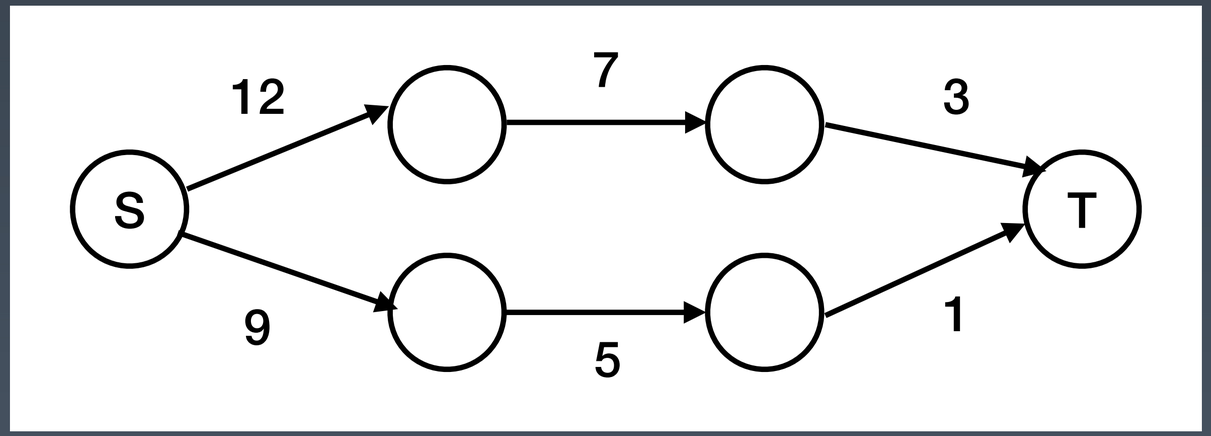
全源最短路径：Bellman-Ford、Floyd-Warshall

最下生成树算法：Prim、Kruskal，铺设网络

拓扑排序

网络流：货运

二分图匹配



BFS & DFS

什么是搜索算法?

搜索的定义：简单点理解，就是在地图中找从一个点到另一个点的路径

数据结构：算法是作用于具体数据结构之上的，图这种数据结构的表达能力很强，大部分涉及搜索的场景都可以抽象成“图”。

搜索算法：在图中找出从一个顶点出发到另一个顶点的路径。

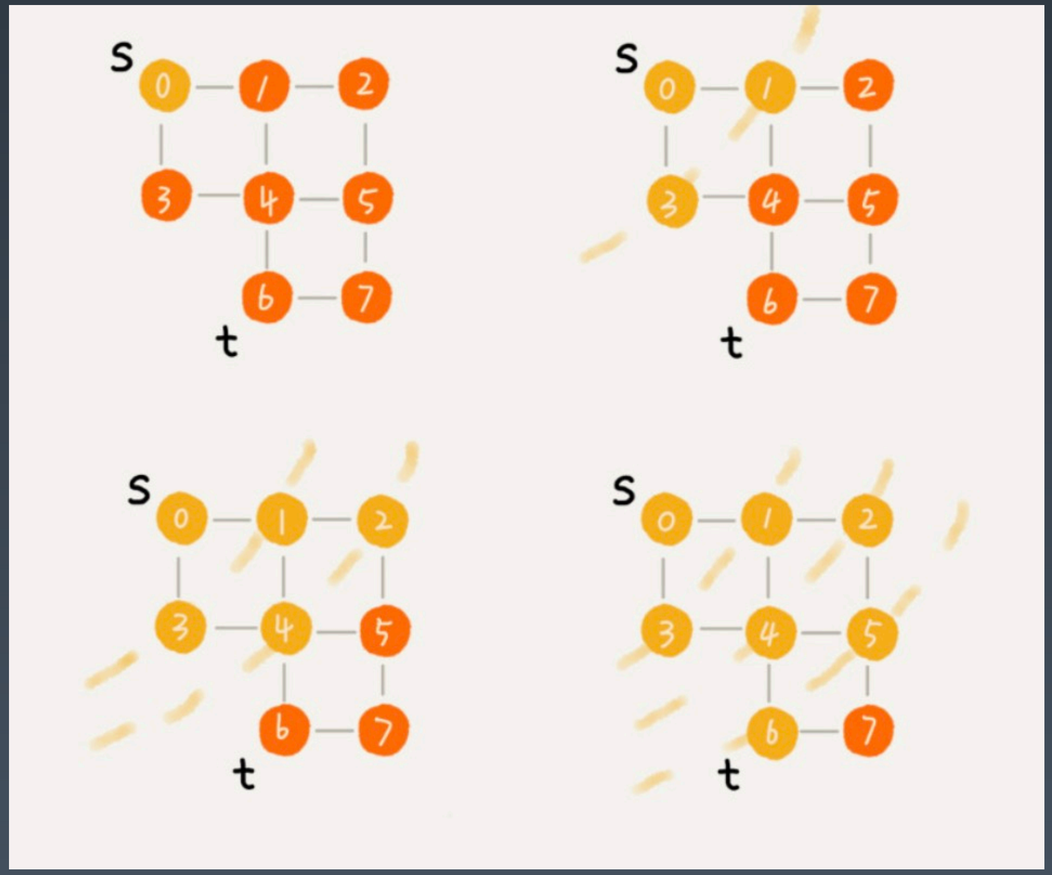
具体方法有很多，最简单、最“暴力”的深度优先、广度优先搜索，还有 A*、IDA* 等启发式搜索算法。

广度优先搜索的基本思想

广度优先搜索（Breadth-First-Search），我们平常都把简称为 BFS。

直观地讲，它其实就是一种“地毯式”层层推进的搜索策略，即先查找离起始顶点最近的，然后是次近的，依次往外搜索。

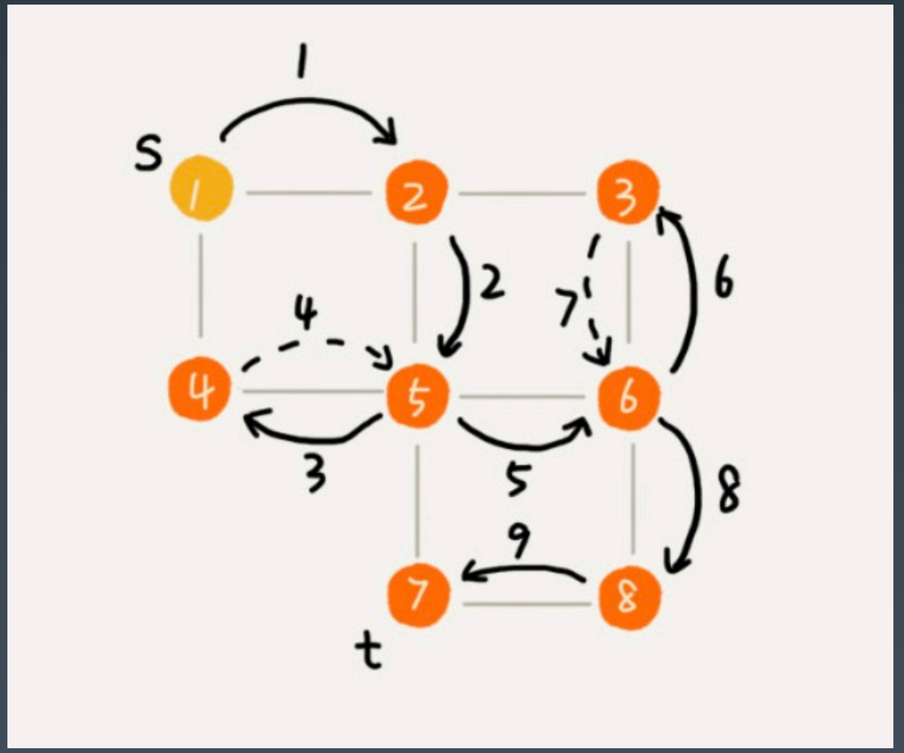
广度优先搜索可以找到最短路径。



深度优先搜索算法基本思想

深度优先搜索（Depth-First-Search），简称 DFS。最直观的例子就是“走迷宫”。

假设你站在迷宫的某个岔路口，然后想找到出口。你随意选择一个岔路口来走，走着走着发现走不通的时候，你就回退到上一个岔路口，重新选择一条路继续走，直到最终找到出口。这种走法就是一种深度优先搜索策略。



深度优先搜索算法代码实现

```
boolean found = false;

public void dfs(int s, int t) {
  found = false;
  boolean[] visited = new boolean[v];
  recurDfs(s, t, visited);
}
```

```
private void recurDfs(int w, int t, boolean[] visited) {
  if (found == true) return;
  visited[w] = true;
  if (w == t) {
    found = true;
    return;
  }
  for (int i = 0; i < adj[w].size(); ++i) {
    int q = adj[w].get(i);
    if (!visited[q]) {
      recurDfs(q, t, visited);
    }
  }
}
```

THANKS!