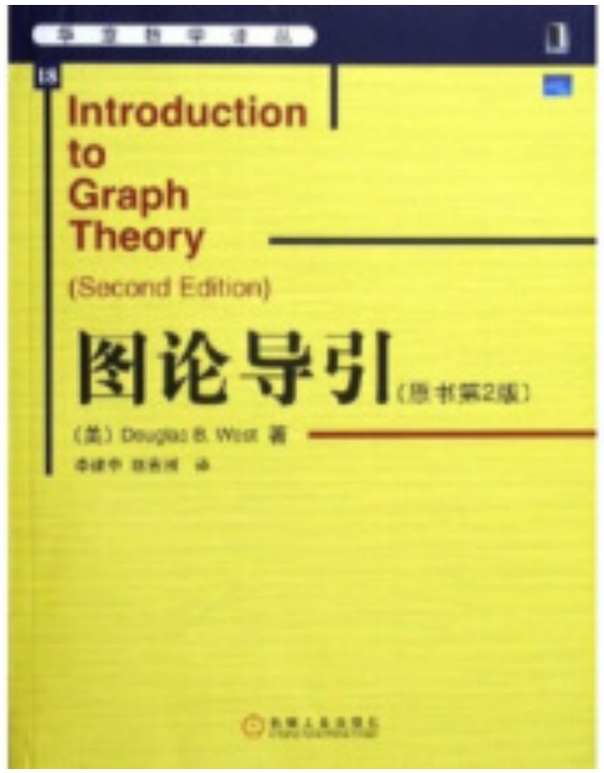
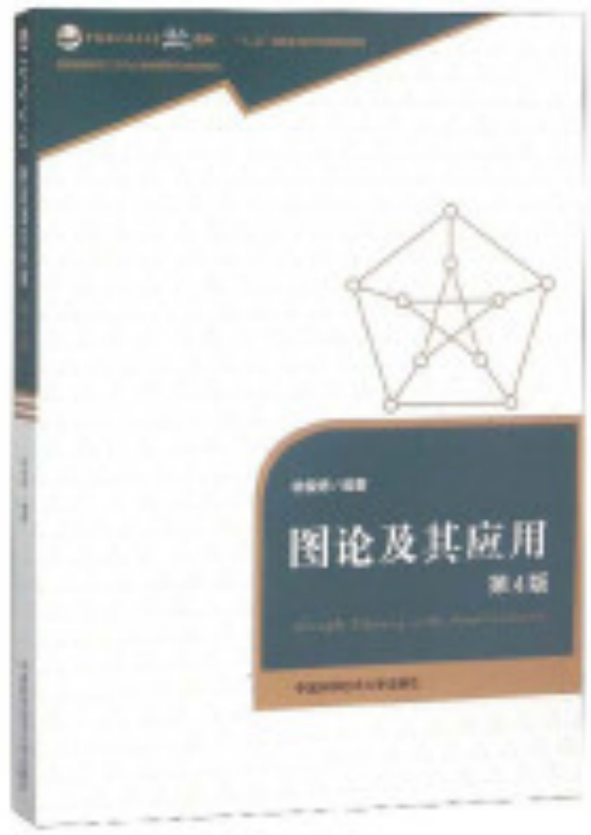
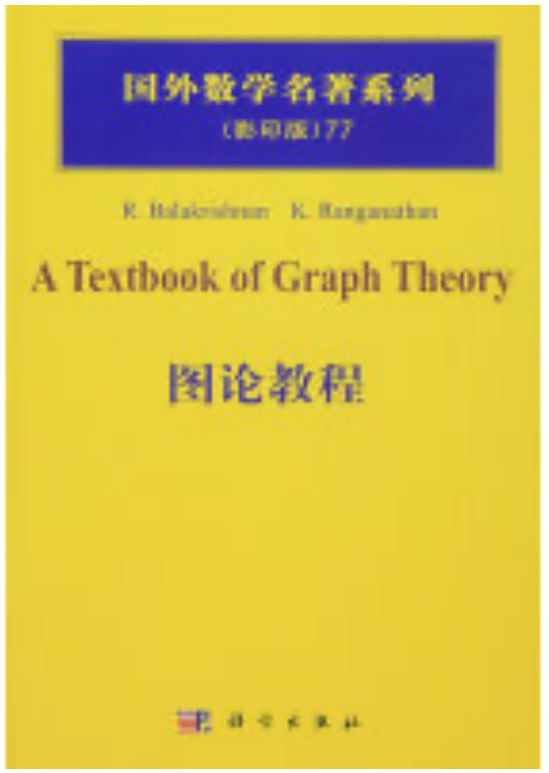
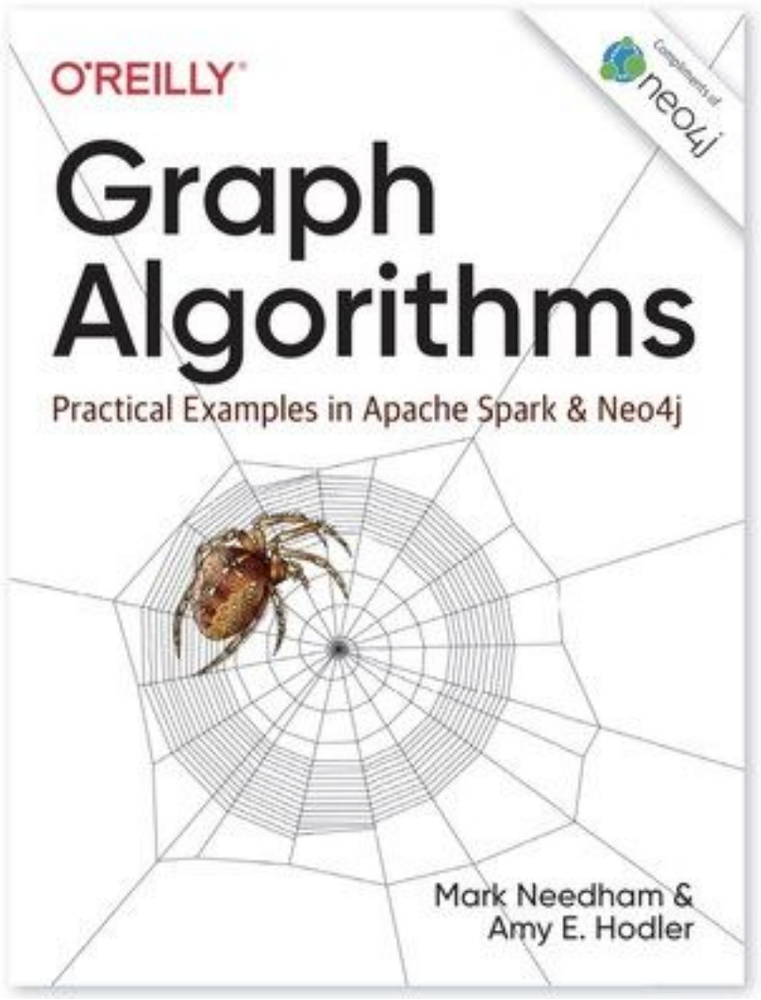
数据结构实验（8）

图算法（2）


Introduction
to
Graph
Theory
(Second Edition)
图论导引(原书第2版)
(美) Douglas B. West 著


图论及其应用
第4版


国外数学名著系列
(影印版)77
R. Balakrishnan K. Ranganathan
A Textbook of Graph Theory
图论教程



Graph Algorithms

by Amy E. Hodler, Mark Needham

Publisher: O'Reilly Media, Inc.
Release Date: May 2019
ISBN: 9781492047674

TOC:

目录

- 并行计算及 Map-Reduce 框架
  - 以 Word Counting 问题为例

第一问 为官避事耻不耻？学校高层享数百万年薪，却无有作为，进不能守校园之平安，退不能保教学之秩序。暴徒公然漠视法律与校规，非法侵占校园，校方竟对此视而不见，甚至默许社会极端势力进入校园。谈判无能，放任校园被暴徒摧毁，却屡屡公开谴责港警履职行为。如今校方反求政府担责，实属推卸责任。

第二问 堂堂名校何以沦为“基地组织”？大批暴徒聚集校园，大批教授学生仓皇撤离。暴徒装扮与恐怖分子几无二致，本校师生进入校园却要接受暴徒“安检”，其行径何其荒谬！校长面对暴徒竟予以同情，简直辱没学者之名。中大乃先师钱穆等人苦心建立，中学西学在此交融。如今名誉受辱，举国哗然，莘莘学子流离失所，科研人才报国无门，学校管理层难辞其咎。

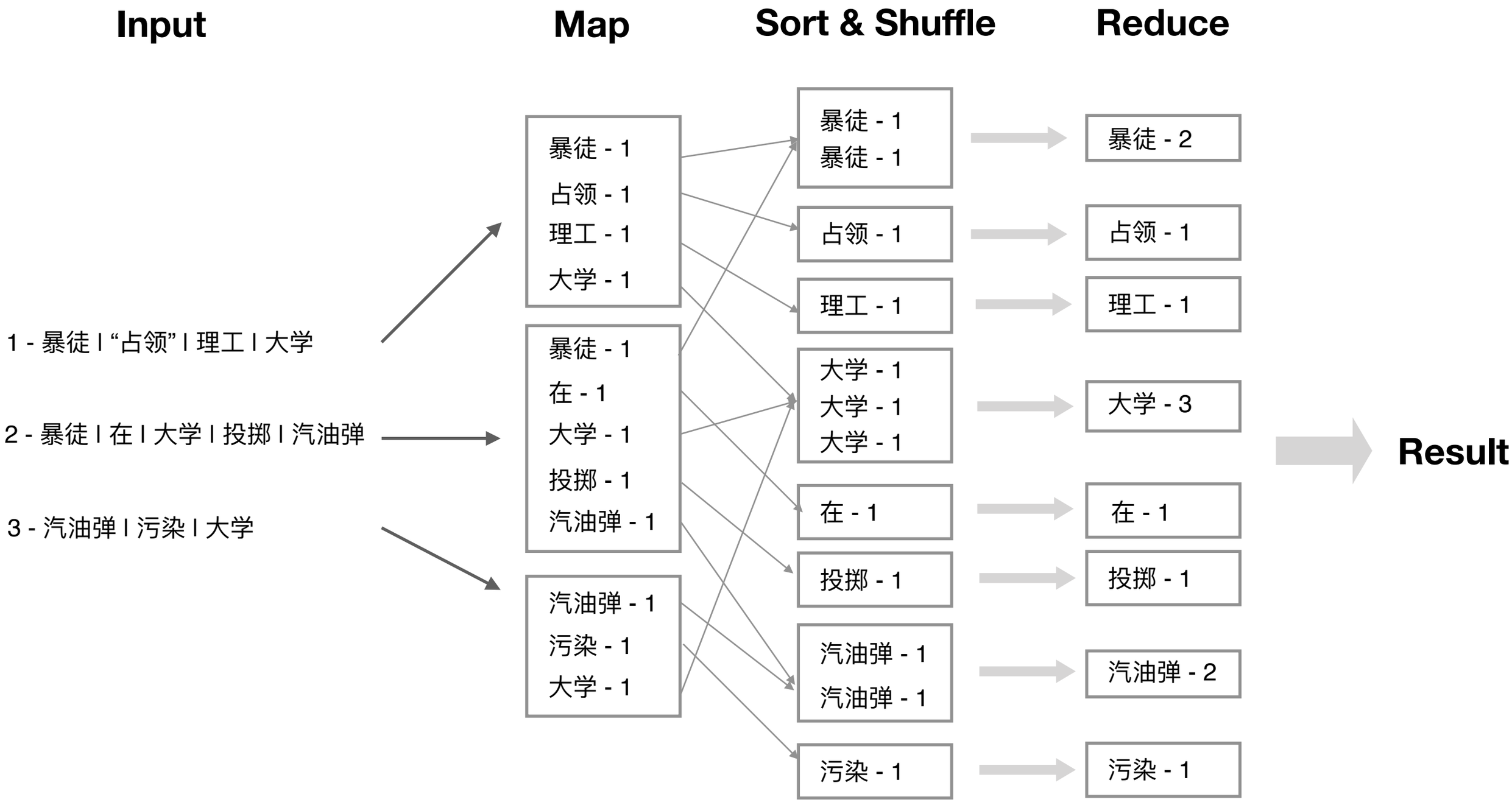
第三问 养虎为何？学校管理层无底线出卖学生利益，一味退缩致使美丽校园横遭蹂躏，沦为修罗战场，“成就”一批时代暴徒。副校长数月前已获悉校内存在危险化学品试验基地，此乃严重犯罪，为何隐瞒不报？为何不发布任何校园安全提示提醒在校学生？是毫不在意，还是包庇？直至今日举世皆知，中大校内被查获数千枚汽油弹，坐实“暴大”之名，岂能开脱责任？

- 单机算法：`words(doc.txt) | sort | uniq -c`
  - 传统单机处理能力无法应对 Web 时代爆炸的数据量
- 一个 通用 的并行计算框架

似曾相识？第一节课里提到的函数式编程！

- 并行计算及 Map-Reduce 框架
  - 一个 通用 的并行计算框架
    - Input：通用的数据模型 Key-Value
    - Map：
      - 每个 Key-Value 交给一个 Mapper 处理，Mapper 处理后输出一个或多个 Key-Value
  - GroupBy：
    - 将 Mapper 处理后的 Key-Value 按 Key 排序、分组
  - Reduce：
    - 每组一个 Reducer，进行聚合操作

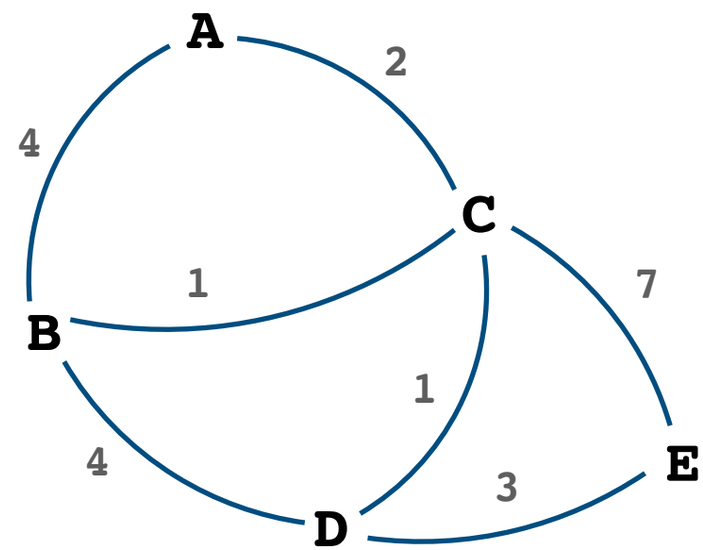
- 并行计算及 Map-Reduce 框架
  - 一个 通用 的并行计算框架



目录

- 路径搜索与规划问题
  - 研究图中节点的可达性和可达速度问题
  - 算法：
    - 广度优先搜索
    - 深度优先搜索
    - 单源最短（优）路径
    - 所有节点间的最短路径
    - 最小生成树
    - Random Walk
- 应用：
  - 地图导航类
  - 共享单车投放点优化
  - 外卖、网约车调度
  - 数据中心路由器算法
  - 微博社交关系推荐

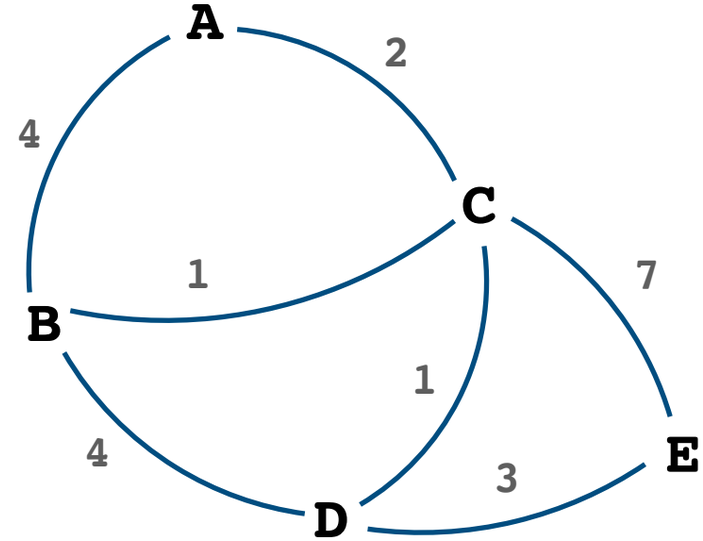
- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS
      - 数据模型：
        - Key：节点编号
        - Value：节点信息：<与起点的距离，邻接表>
      - Map：
        - 输入 K-V，获取当前节点与起点的距离，遍历 V 中的邻接表，输出新的 K-V
      - Sort & Shuffle：
        - 按 Key 节点编号分组
      - Reduce：
        - 读入所有 K-V，包含各个前序节点到当前节点 K 的所有可能的距离，选取最短距离并输出



- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS

```
func mapper(key n, value N)
    Emit(key n, value N)        // 为什么?
    if N.distance != INF then
        for all edge m in N.adjacencyList do:
            Emit(key m.node, value <N.distance + m.weight, NULL>)
```

```
func reducer(key n, value N1, N2, N3…)
    d_min = INF
    M = <>
    for all value N in [N1, N2, N3…] do
       if N.adjacencyList is not NULL then
           M.adjacencyList = N.adjacencyList
       if N.distance < d_min then
           d_min = N.distance
    M.distance = d_min
    Emit(key n, value M)
```



- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS

```
func mapper(key n, value N)
    Emit(key n, value N)
    if N.distance != INF then
        for all edge m in N.adjacencyList do:
            Emit(key m.node, value <N.distance + m.weight, NULL>)
```

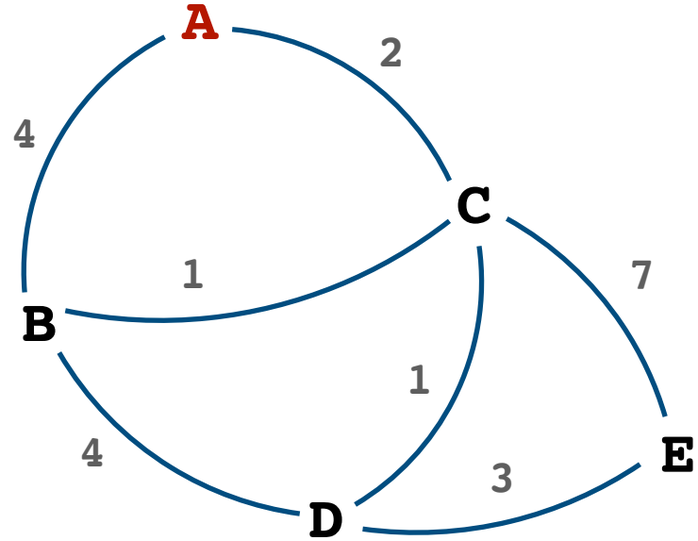
- Mapper Input:

```
<A, (0, [(B, 4), (C, 2)])>
<B, (inf, [(A, 4), (C, 1), (D, 4)])>
<C, (inf, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (inf, [(B, 4), (C, 1), (E, 3)])>
<E, (inf, [(C, 7), (D, 3)])>
```

- Mapper Output:

```
<B, (4, NULL)>
<C, (2, NULL)>
```

```
<A, (0, [(B, 4), (C, 2)])>
<B, (inf, [(A, 4), (C, 1), (D, 4)])>
<C, (inf, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (inf, [(B, 4), (C, 1), (E, 3)])>
<E, (inf, [(C, 7), (D, 3)])>
```



- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS

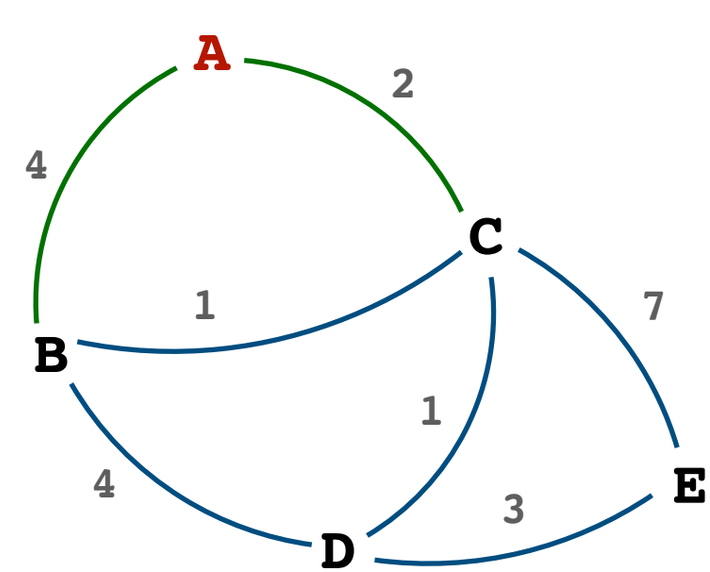
```
func reducer(key n, value N1, N2, N3…)
    d_min = INF
    M = <>
    for all value N in [N1, N2, N3…] do
       if N.adjacencyList is not NULL then
           M.adjacencyList = N.adjacencyList
       if N.distance < d_min then
           d_min = N.distance
M.distance = d_min
Emit(key n, value M)
```

- Reducer Input:

```
<A, (0, [(B, 4), (C, 2)])>
<B, (4, NULL)>  <B, (inf, [(A, 4), (C, 1), (D, 4)])>
<C, (2, NULL)>  <C, (inf, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (inf, [(B, 4), (C, 1), (E, 3)])>
<E, (inf, [(C, 7), (D, 3)])>
```

- Reduce Output

```
<A, (0, [(B, 4), (C, 2)])>
<B, (4, [(A, 4), (C, 1), (D, 4)])>
<C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (inf, [(B, 4), (C, 1), (E, 3)])>
<E, (inf, [(C, 7), (D, 3)])>
```

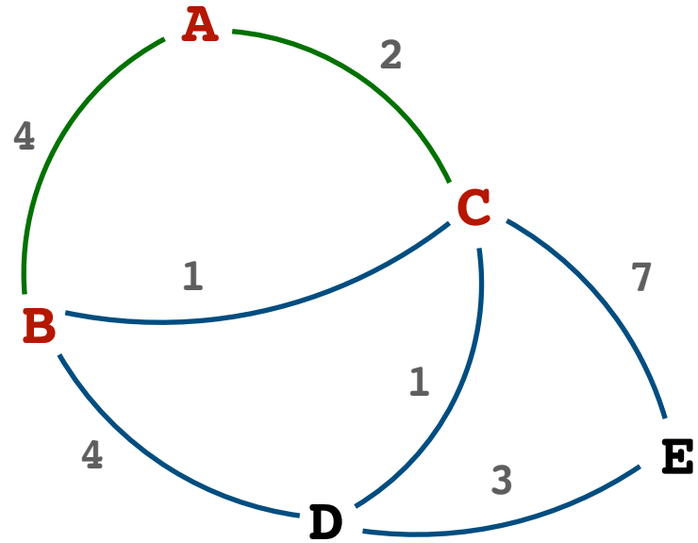


- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS

```
func mapper(key n, value N)
    Emit(key n, value N)
    if N.distance != INF then
        for all edge m in N.adjacencyList do:
            Emit(key m.node, value <N.distance + m.weight, NULL>)
```

- Mapper Input:

```
<A, (0, [(B, 4), (C, 2)])>
<B, (4, [(A, 4), (C, 1), (D, 4)])>
<C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (inf, [(B, 4), (C, 1), (E, 3)])>
<E, (inf, [(C, 7), (D, 3)])>
```



- Mapper Output:

```
<B, (4, NULL)>  <C, (2, NULL)>
<A, (8, NULL)>  <C, (5, NULL)>  <D, (8, NULL)>
<A, (4, NULL)>  <B, (3, NULL)>  <D, (3, NULL)>  <E, (9, NULL)>
```

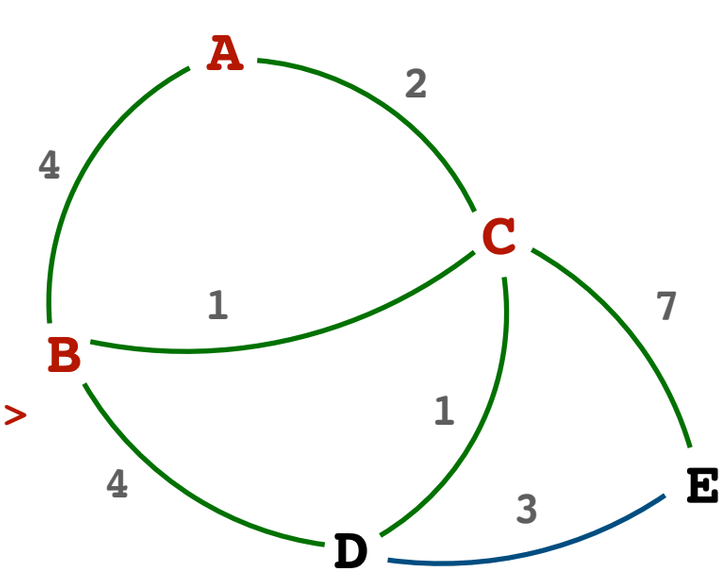
```
<A, (0, [(B, 4), (C, 2)])>
<B, (4, [(A, 4), (C, 1), (D, 4)])>
<C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (inf, [(B, 4), (C, 1), (E, 3)])>
<E, (inf, [(C, 7), (D, 3)])>
```

- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS

```
func reducer(key n, value N1, N2, N3…)
    d_min = INF
    M = <>
    for all value N in [N1, N2, N3…] do
       if N.adjacencyList is not NULL then
           M.adjacencyList = N.adjacencyList
       if N.distance < d_min then
           d_min = N.distance
       M.distance = d_min
       Emit(key n, value M)
```

- Reducer Input:

```
<A, (8, NULL)>  <A, (4, NULL)>  <A, (0, [(B, 4), (C, 2)])>
<B, (4, NULL)>  <B, (3, NULL)>  <B, (4, [(A, 4), (C, 1), (D, 4)])>
<C, (2, NULL)>  <C, (5, NULL)>  <C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (8, NULL)>  <D, (3, NULL)>  <D, (inf, [(B, 4), (C, 1), (E, 3)])>
<E, (9, NULL)>  <E, (inf, [(C, 7), (D, 3)])>
```



- Reduce Output

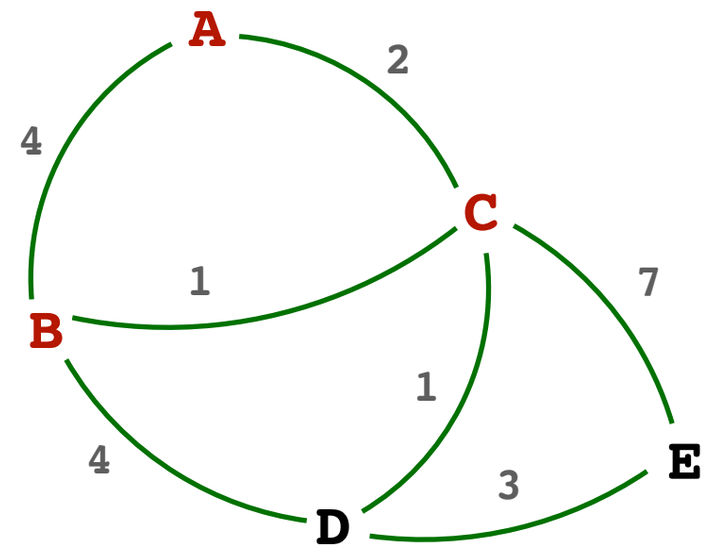
```
<A, (0, [(B, 4), (C, 2)])>
<B, (3, [(A, 4), (C, 1), (D, 4)])>
<C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (3, [(B, 4), (C, 1), (E, 3)])>
<E, (9, [(C, 7), (D, 3)])>
```

- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS

```
func mapper(key n, value N)
    Emit(key n, value N)
    if N.distance != INF then
        for all edge m in N.adjacencyList do:
            Emit(key m.node, value <N.distance + m.weight, NULL>)
```

- Mapper Input:

```
<A, (0, [(B, 4), (C, 2)])>
<B, (3, [(A, 4), (C, 1), (D, 4)])>
<C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (3, [(B, 4), (C, 1), (E, 3)])>
<E, (9, [(C, 7), (D, 3)])>
```



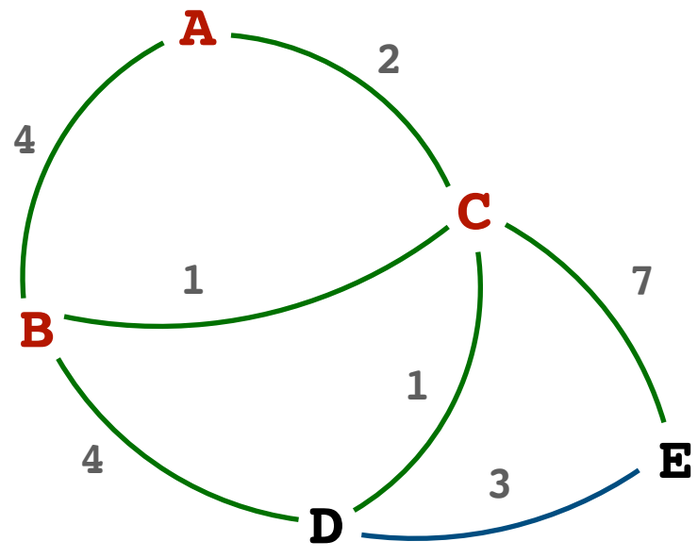
- Mapper Output:

```
<B, (4, NULL)>  <C, (2, NULL)>
<A, (7, NULL)>  <C, (4, NULL)>  <D, (7, NULL)>
<A, (4, NULL)>  <B, (3, NULL)>  <D, (3, NULL)>  <E, (9, NULL)>
<B, (7, NULL)>  <C, (4, NULL)>  <E, (6, NULL)>
<C, (16, NULL)>  <D, (12, NULL)>
```

```
<A, (0, [(B, 4), (C, 2)])>
<B, (3, [(A, 4), (C, 1), (D, 4)])>
<C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (3, [(B, 4), (C, 1), (E, 3)])>
<E, (9, [(C, 7), (D, 3)])>
```

- 路径搜索与规划问题
  - 广度优先搜索
    - 单源最短路径问题的 Map-Reduce 版 BFS

```
func reducer(key n, value N1, N2, N3…)
    d_min = INF
    M = <>
    for all value N in [N1, N2, N3…] do
       if N.adjacencyList is not NULL then
           M.adjacencyList = N.adjacencyList
       if N.distance < d_min then
           d_min = N.distance
    M.distance = d_min
    Emit(key n, value M)
```



- Reducer Input:

```
<A, (7, NULL)>  <A, (4, NULL)>  <A, (0, [(B, 4), (C, 2)])>
<B, (4, NULL)>  <B, (3, NULL)>  <B, (7, NULL)>  <B, (3, [(A, 4), (C, 1), (D, 4)])>
<C, (2, NULL)>  <C, (4, NULL)>  <C, (4, NULL)>  <C, (16, NULL)>
    <C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (7, NULL)>  <D, (3, NULL)>  <D, (12, NULL)>  <D, (3, [(B, 4), (C, 1), (E, 3)])>
<E, (9, NULL)>  <E, (6, NULL)>  <E, (9, [(C, 7), (D, 3)])>
```

- Reduce Output

```
<A, (0, [(B, 4), (C, 2)])>
<B, (3, [(A, 4), (C, 1), (D, 4)])>
<C, (2, [(A, 2), (B, 1), (D, 1), (E, 7)])>
<D, (3, [(B, 4), (C, 1), (E, 3)])>
<E, (6, [(C, 7), (D, 3)])>
```

那么，多少轮 Map-Reduce 后可以停止 BFS?

图的直径

- 路径搜索与规划问题
  - 广度优先搜索
    - 扩展练习：
      - 题目：给定一个用图描述的社交关系网，求其其中每个人的二度人脉（朋友的朋友）。
      - 问：
        - 使用 BFS 解决该问题
        - 使用 Map-Reduce 框架下的 BFS 解决该问题

目录

- 中心性问题
  - 研究节点的重要性（中心性）及其对整个图的影响。
  - 算法：
    - Degree Centrality
    - PageRank
    - Personalized PageRank
- 应用：产出各种推荐算法、广告算法的重要特征

- 中心性问题
  - Degree Centrality

Degree Centrality

Degree Centrality is the simplest of the algorithms that we'll cover in this book. It counts the number of incoming and outgoing relationships from a node, and is used to find popular nodes in a graph. Degree Centrality was proposed by Linton C. Freeman in his 1979 paper "Centrality in Social Networks: Conceptual Clarification".

Reach

Understanding the reach of a node is a fair measure of importance. How many other nodes can it touch right now? The *degree* of a node is the number of direct relationships it has, calculated for in-degree and out-degree. You can think of this as the immediate reach of node. For example, a person with a high degree in an active social network would have a lot of immediate contacts and be more likely to catch a cold circulating in their network.

The *average degree* of a network is simply the total number of relationships divided by the total number of nodes; it can be heavily skewed by high degree nodes. The *degree distribution* is the probability that a randomly selected node will have a certain number of relationships.

Figure 5-3 illustrates the difference looking at the actual distribution of connections among subreddit topics. If you simply took the average, you'd assume most topics have 10 connections, whereas in fact most topics only have 2 connections.

- 中心性问题
  - Degree Centrality

When Should I Use Degree Centrality?

Use Degree Centrality if you're attempting to analyze influence by looking at the number of incoming and outgoing relationships, or find the "popularity" of individual nodes. It works well when you're concerned with immediate connectedness or near-term probabilities. However, Degree Centrality is also applied to global analysis when you want to evaluate the minimum degree, maximum degree, mean degree, and standard deviation across the entire graph.

Example use cases include:

- Identifying powerful individuals though their relationships, such as connections of people in a social network. For example, in BrandWatch's "Most Influential Men and Women on Twitter 2017", the top 5 people in each category have over 40 million followers each.

- 中心性问题
  - Page Rank
    - 思想：假设一个网页的入链越多，则该网页越重要
    - 定义：

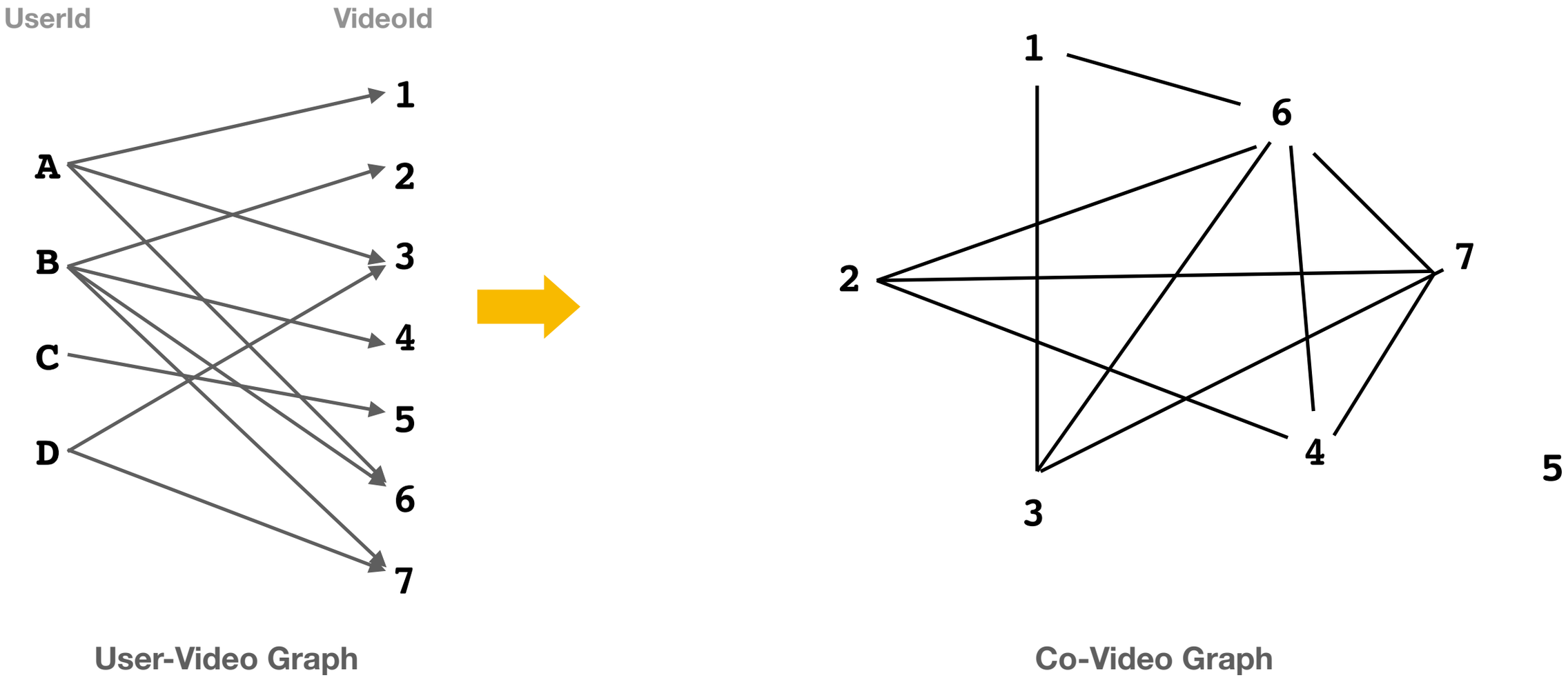
$$PR(u) = (1 - d) + d\left(\frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn)}\right)$$

- 其中：
  - T1, T2, … , Tn 是指向网页 u 的所有网页
  - C(T) 是顶点的出度（out-degree）
  - d 是摩擦系数，通常设为 0.85

目录

- 社群发现问题
  - 研究节点之间的连接关系，找到拥有某些共性的节点群体，并分析他们的特性。
  - 算法：
    - 概率模型：
      - Adsorption Algorithm
      - 主题模型（Topic Model）
    - 深度学习：
      - NLP 领域各种词、句建模方法
      - 用户行为序列的建模方法
- 应用：产出各种推荐算法、广告算法的重要特征

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - 先从用户行为的描述方式讲起



- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - 背景：
      - 古老的基于 标签 的视频推荐算法：给用户推荐曾经看过的视频的同类视频（拥有相同的标签）
      - 关键在于标签的生成
        - 显性的标签：导演、演员、视频类型、年代 等
        - 隐性的标签：视频与视频在 Co-Video Graph 中的“相似性”

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - “近朱者赤，近墨者黑”
      - 让 Video-Video Graph 中，拥有同邻居的节点的相似性更高
    - 为相似性达到一定程度的视频打上一类 标签
      - 一共有多少标签：不知道
      - 一个视频会被打上多少标签：不知道

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - “近朱者赤，近墨者黑”
      - Adsorption Algorithm：过在特定步数内对从一个节点开始平均游走 (Averaging Walk)，到达另一个节点的概率
        - 我们在游走的过程中，将标签信息被平均传递

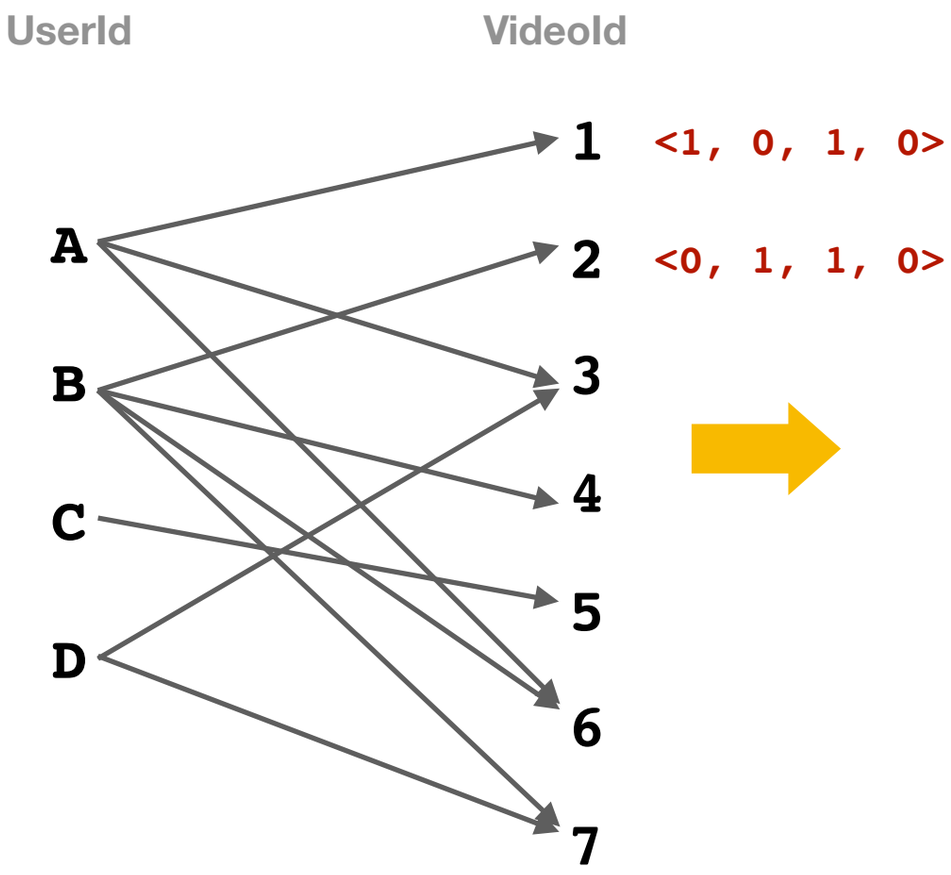
Algorithm Adsorption:

Input: $G = (V, E, w)$, L, V_L.
repeat
  for each $v \in V \cup \widetilde{V}$ do:
    Let $L_v = \sum_u w(u, v) L_u$
  end-for
  Normalize L_v to have unit L_1 norm
until convergence
Output: Distributions $\{L_v \mid v \in V\}$

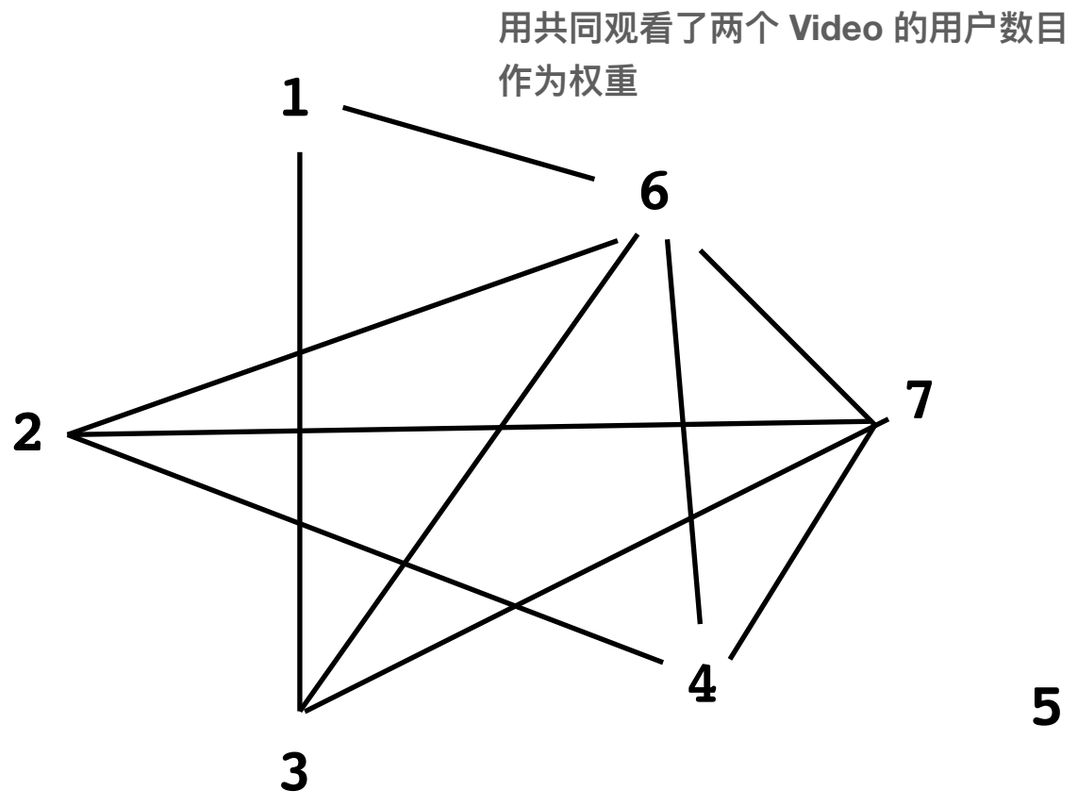
V - 顶点集合
E - 边集合
w - 边的权重
L - 标签集合
VL - 每个顶点在所有标签上的概率分布

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - “近朱者赤，近墨者黑”
      - 例如：
        - 假设我们目前只有4个标签：（性别男，性别女，战争片，爱情片），
        - 如果某个 Video 被用户打上标签, 比如某个已知男性用户观看了《我和我的祖国》， 而且它被认为是战争片。那么该 Video 的标签向量就是 (1,0,1,0) ,对于其他我们没有标签的 Video, 我们将其声明为 (0,0,0,0)

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - 先从用户行为的描述方式讲起

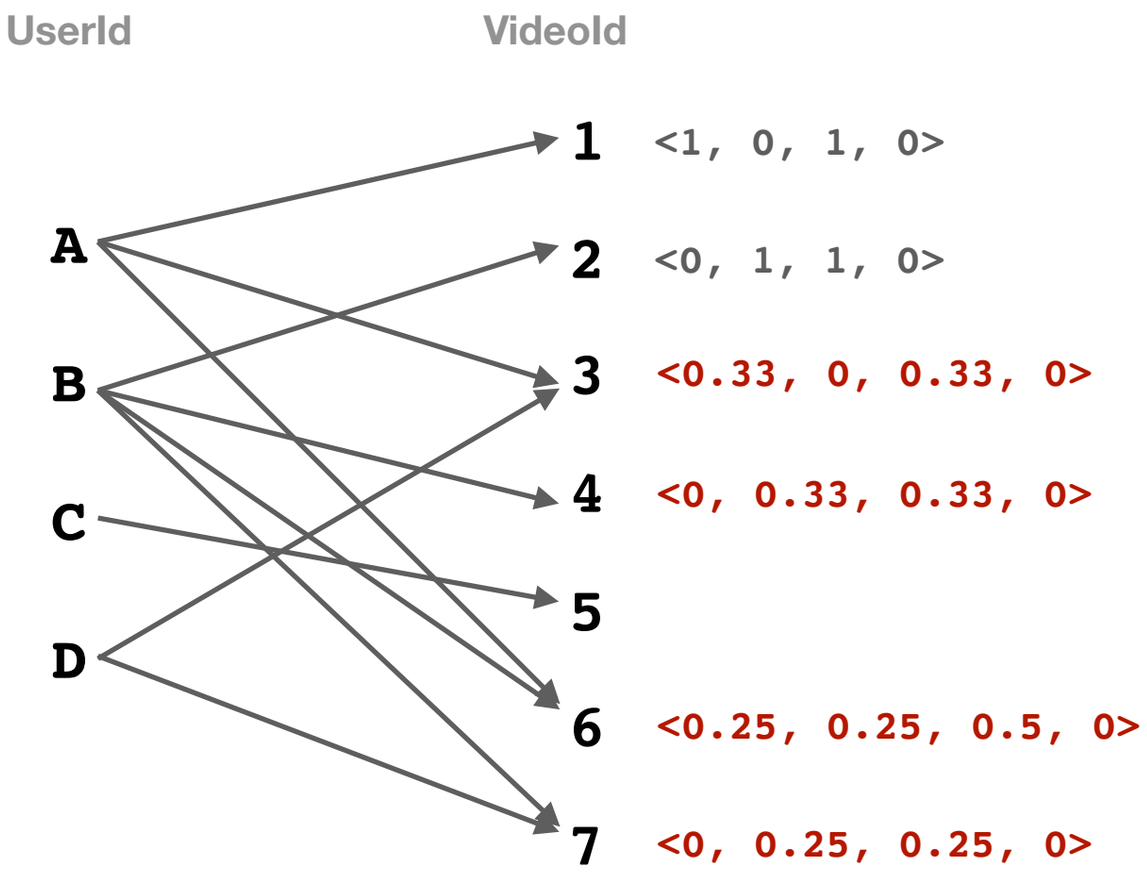


User-Video Graph



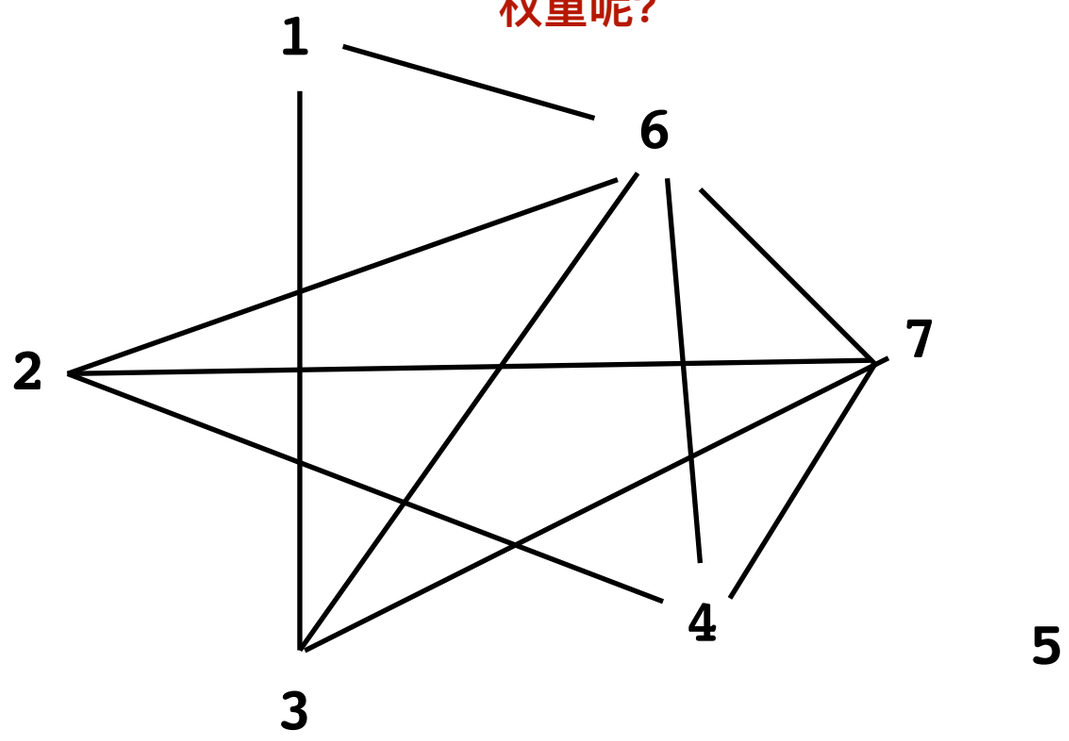
Co-Video Graph

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - 先从用户行为的描述方式讲起



User-Video Graph

如果不用 Averaging Walk，而考虑 Video-Video 之间的权重呢?



Co-Video Graph

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - Averaging Walk TO Random Walk
      - 我们在游走的过程中，将标签信息按权重作为概率传递
      - 研究：Video-Video 中 Edge 和 Weight 生成方式：
        - Edge：
          - 被一定量用户共同观看的视频；
          - 在同一个 Session 中经常被同时观看的视频；
          - 考虑顺序信息的，在同一个 Session 中经常先后被观看的视频。
        - Weight：
          - 被共同观看的用户数量；
          - 同时出现的 Session 数目；

意思的有效性逐渐更好，但数据则逐渐稀疏

- 社群发现问题
  - <Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph>
    - 还有什么问题：
      - 热门 vs 长尾
      - 丰富度

- 社群发现问题
  - 扩展阅读：
    - Youtube 推荐算法系列论文
      - 第一版 基于 Adsorption 算法
        - <Video Suggestion And Discovery For Youtube: Taking Random Walks Through The View Graph>
      - 第二版 基于 Item-Based 协同过滤
        - <The YouTube Video Recommendation System>
      - 第三版 基于 Topic Representation
        - <Retrieval Methods for Large Scale Related Video Suggestion>
      - 第四版 基于深度学习 (Deep&Wide），利用用户在浏览、Like 等行为序列进行 Graph-Embedding 建模和推荐
        - <Deep Neural Networks for YouTube Recommendations>
    - 阿里巴巴推荐算法
      - EGNS，基于深度学习，利用用户在电商的行为序列进行 Graph-Embedding 建模和推荐
        - <Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba >