网络

The network changes our lives

引言

- 网络在我们的生活中随处可见
  - 电话网络
  - 公共供给网络
  - 计算机网络
  - 交通网络
  - 社会网络
- 它们以多种方式将人们的日常生活高效地连接起来

引言

- 网络对于我们有着如此重大的意义，因此我们总是希望能够设计出更加完美的网络建设方案，或是在已有的网络中找出最符合我们要求的调度与安排
- 人们可以很容易地处理一些小范围的网络问题，但是对于那些超大规模的网络，我们需要计算机来帮助我们应对在网络应用中遇到的各种挑战

概要

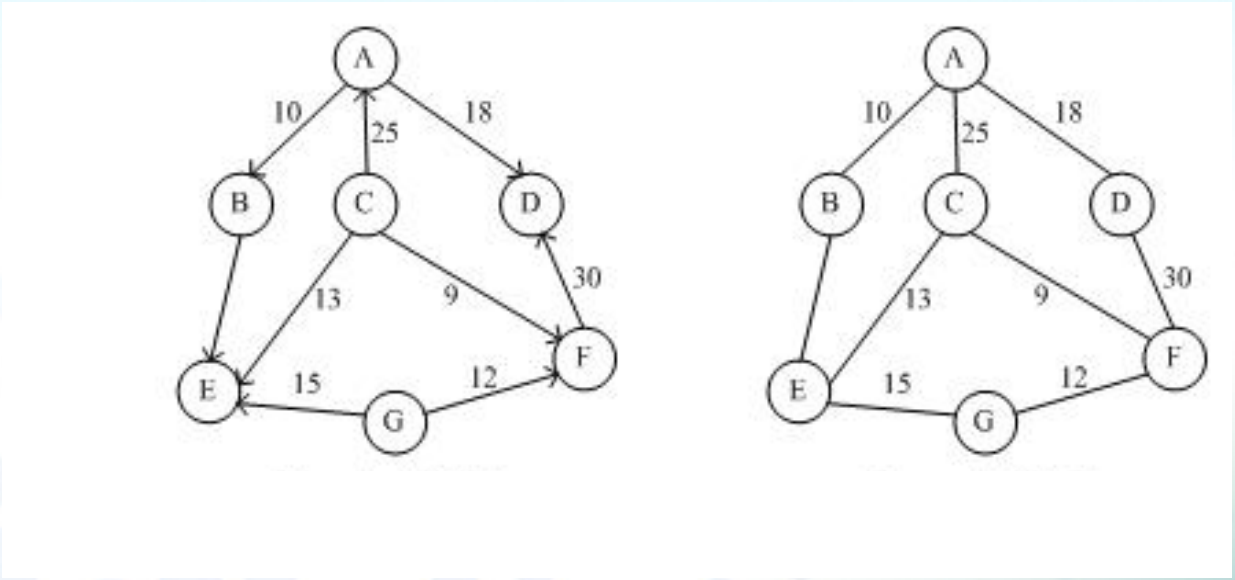
- 这一章我们将讨论在网络中会遇到的一些难题以及其解决方案
  - 最小生成树
  - 关键路径
  - 最短路径
  - 旅行商问题

网络的定义

- 从图论的角度来说，网络（network）实际上就是我们上一节讲过的图的一种特例
- 它与一般图的不同点在于，网络的边具有与之相关的数值，称为权重（weight）
- 权重可以表示顶点之间的距离，花费代价，所需时间、次数等等信息
- 从而网络即是一个带权图，可以表示为 $G=(V, E, w)$，其中w为权重函数：$E \rightarrow \mathbf{R}$

网络的种类

- 与一般图相同，网络也可以分为有向网络和无向网络两种，下面分别是有向网络和无向网络的例子



泥泞的城市

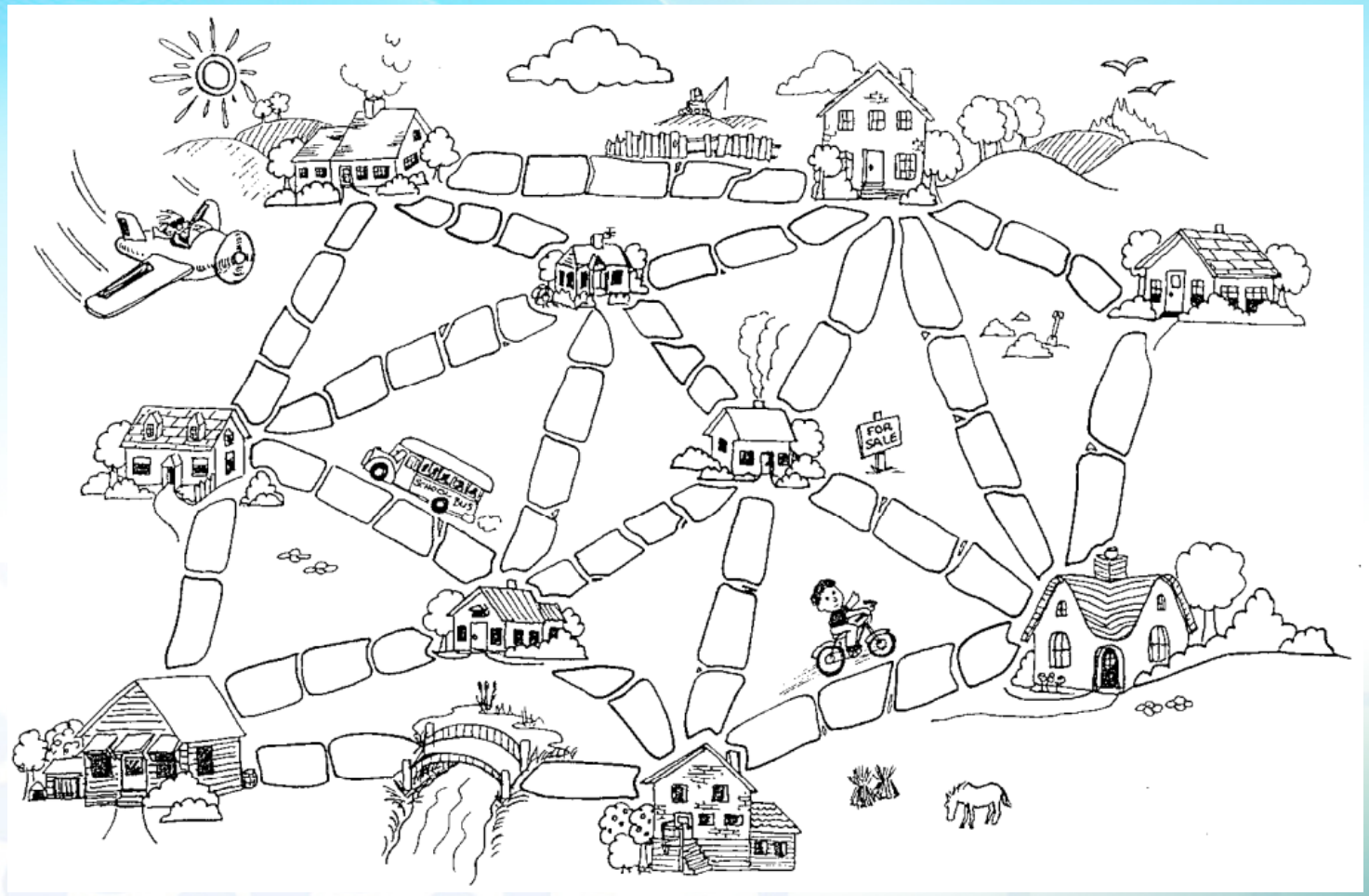
- 假设有一座城市还未铺上道路，下雨之后在这座城市行走是一件非常困难的事情，地面被雨水浸湿，满是泥泞，车辆纷纷陷入泥沼，人们的鞋子上沾满了污泥
- 于是市长痛下决心，一定要为一些道路砌上石砖，但是钱必须花在刀刃上，因为市民会监督政府的财政预算与执行

泥泞的城市

- 市长为了不辜负市民的期望，对于道路的建设提出了以下两个要求
  - 必须铺设足够的道路，让每个人都能从他的家里沿着铺好的道路到达别人的房子
  - 所花费的经费越少越好，而在两栋房子之间的道路上铺上的石砖数就代表了铺路所需要的费用

请帮助市长完成这一任务

城市布局图

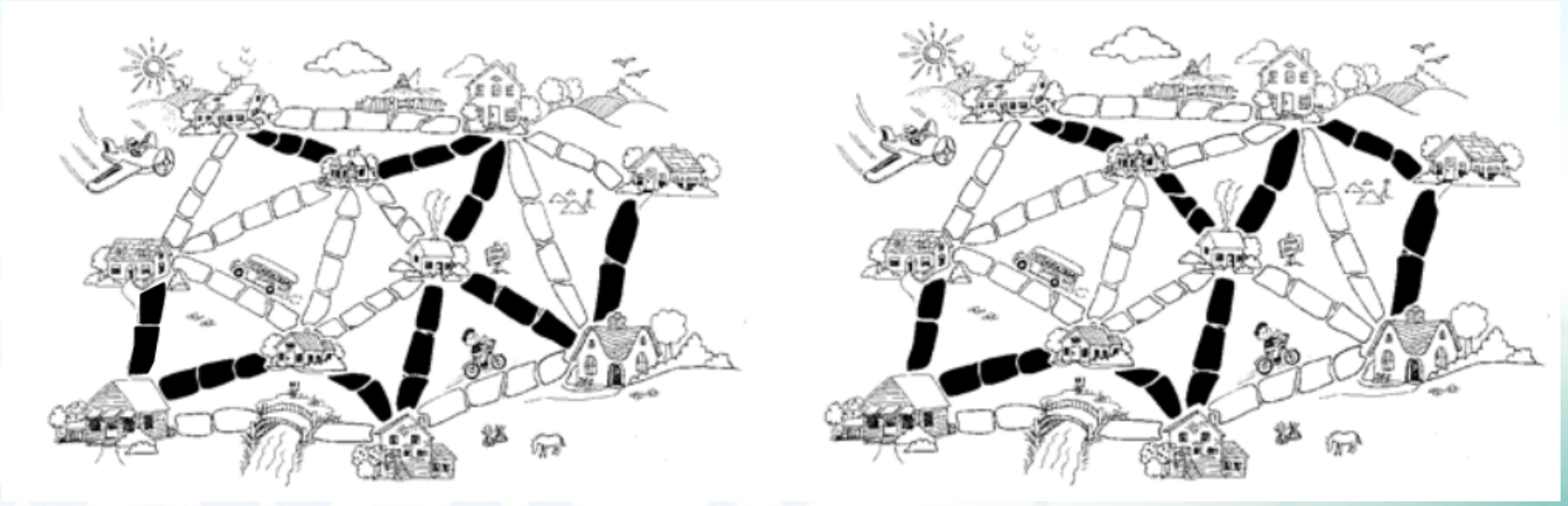


城市布局图

- 每栋房子之间的石砖数量反映了修这条道路的费用，桥算作一块石砖
- 你能在使用最少的铺路石砖的情况下，找出连接全部房子的最佳路径么？
- 提示：如果你的设计方案中有循环路线，即从一条路离开移动房子之后还能从另一条路再回到这栋房子，那么说明你用的石砖数量太多了

解决方案

- 以下是两种可能的最佳道路铺设方案，看看你设计的路线所需要的石砖是否和它们一样少



最小生成树

- 将我们前面的问题形式化，就是所谓的最小生成树问题（the minimal spanning tree problem）
- “最小”是因为它要求使用最少的石砖总数，即连接网络的总代价最小
- “生成树”我们前面已经提到过，即是用全部顶点和部分边组成的树，因为我们要求城市是连通的，代价最小则意味着无环

最小生成树的形式定义

- 对于一个无向带权图$G=(V, E, w)$，其中V是顶点集，E是边集，对于图中每一条边$(u, v)\in E$，都有一个权值$w(u, v)$表示连接u和v的代价，我们希望找出一个子集$T \subseteq E$，它连接了所有的顶点，且其权值之和

$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

为最小

最小生成树

- 我们接下来将介绍解决最小生成树问题的两种算法：Kruskal算法和Prim算法
- 他们都应用贪心（greedy）策略，即在迭代的每一步都选择当时最佳的可能性
- 一般来说，这种策略不一定能获得全局最优解，然而对于最小生成树问题，却可以证明某些贪心策略的确可以获得具有最小权值的生成树

Kruskal算法

- 为了使我们的生成树最小，我们会很自然地想到，应到考虑那些权值较小的边，可以证明，权值最小的边必然至少被一棵最小生成树采用（像我们前面城市的例子中，最小生成树可能有多棵）
- 那是否只要将所有边按权值大小排序，则权值最小的(|*V*|- 1)条边就能组成我们需要的最小生成树呢？

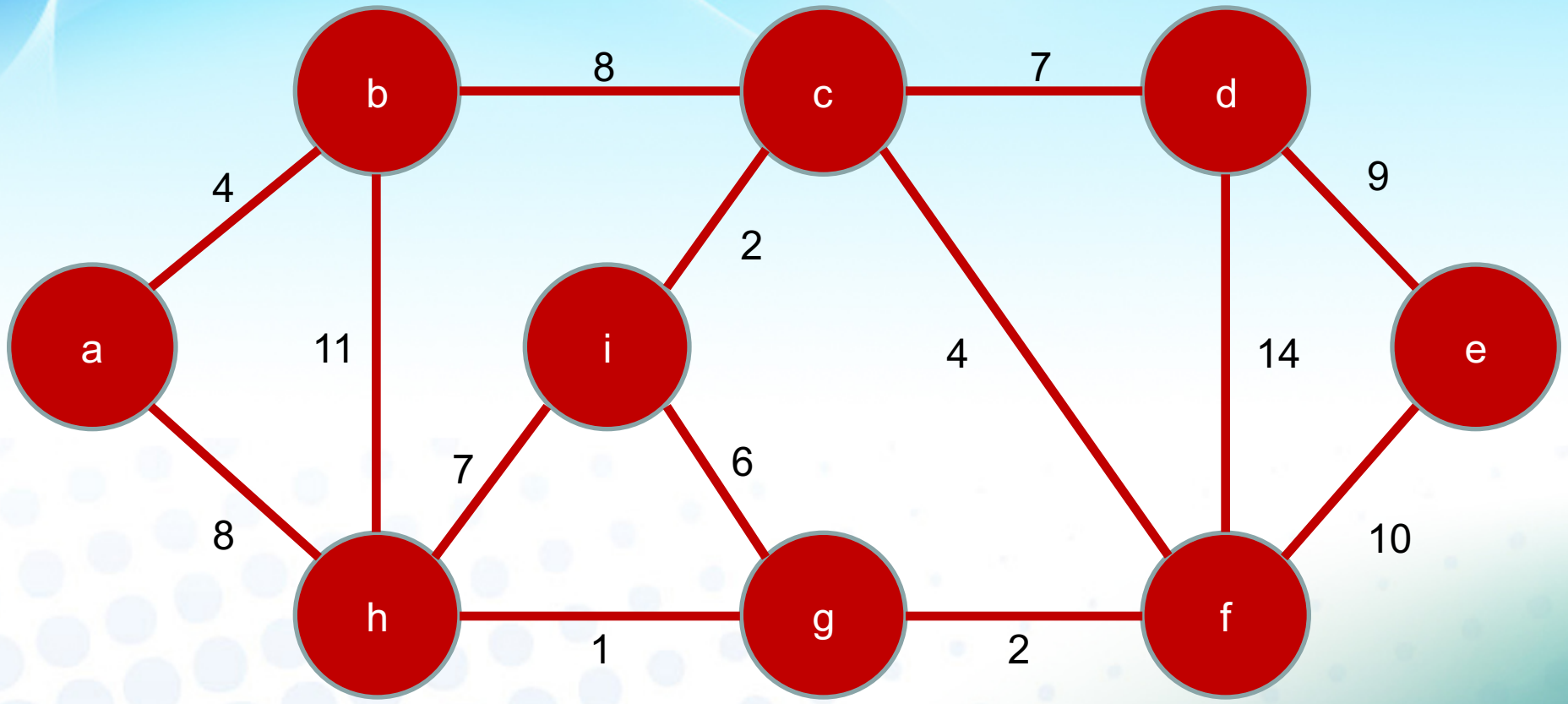
Kruskal算法

- 答案是否定的！
- 很遗憾，从第三条最小边开始，就不一定能够被最小生成树采用
- 原因在于，这些边的引入有可能会导致回路（环）的出现，这就违背了树结构的基本要求
- 尽管如此，只要对上述贪心策略稍作修改，就可以得到一个可行的算法，即Kruskal算法

Kruskal算法

- 该算法的基本过程如下：
  - 首先构造一个由所有|V|个顶点组成，不含任何边的图$T = \{V, \emptyset\}$
  - 不断从E中取出权值最小的一条边（如果有多条，则任选其中的一条），若该边的两个端点不都是已连接到生成树内的顶点，则将此边加入到T中，否则将其忽略
  - 如此重复，直到所有的顶点都已连接到生成树当中为止

Kruskal算法的例子



Kruskal算法

- 如果想要高效地实现Kruskal算法，有两个步骤是非常关键的：对边集按权重值排序和判断当前边加入后是否会形成回路
- 对于前者，我们已经学习过了许多排序的方法，比如快速排序和归并排序都能很好地解决这一问题
- 对于后者，需要用到叫做并查集（union-find set）的数据结构

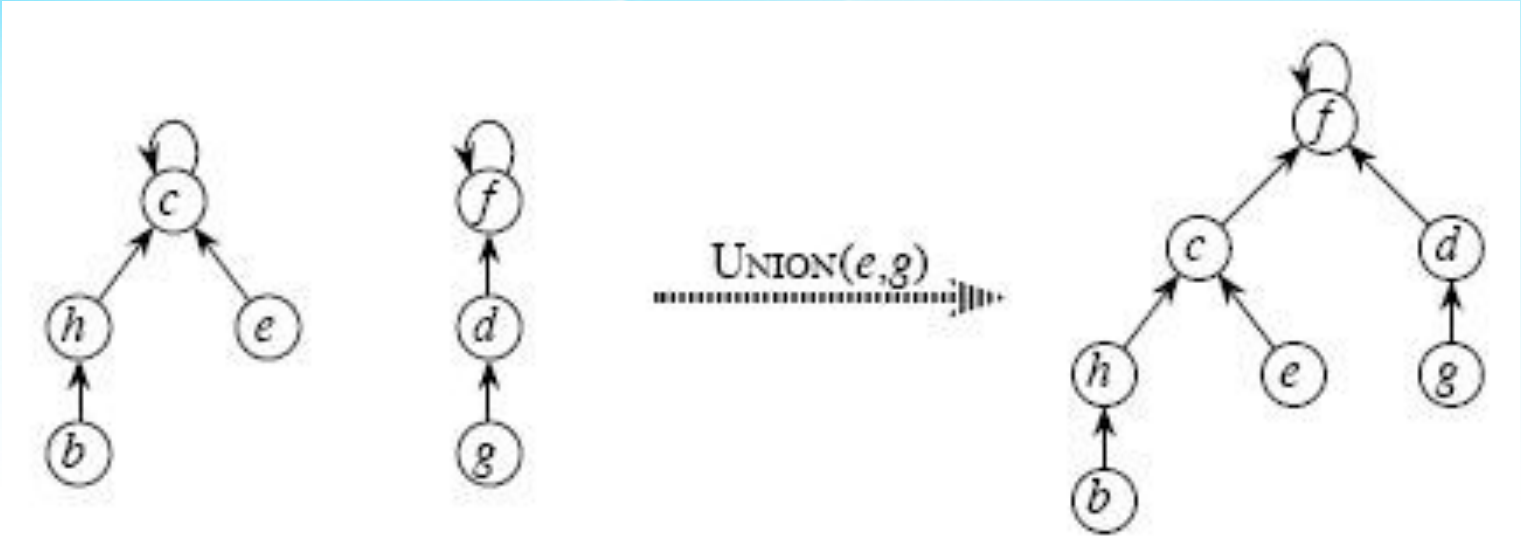
Kruskal算法

- 并查集，又称为不相交集合（disjoint set）数据结构，用于保持一组不相交的集合，并可能会按一定的规律动态地将某些集合合并，而在这变化过程中，支持查询某个元素属于哪个集合
- 这与我们将边加入到生成树的过程十分类似，加入一条边就是将两个顶点所在的集合合并，合并前需检查顶点是否已经在同一集合中

Kruskal算法的伪代码

MST-Kruskal(G, w)
$A \leftarrow \varnothing$
for each vertex $v \in V[G]$ **do**
    Make-Set(v)
sort the edges of E into nondecreasing order by w
for each edge$(u, v) \in E$ **do**
    if Find-Set(u)≠Find-Set(v)
        then $A \leftarrow A \cup \{(u, v)\}$
            Union(u, v)
return A

并查集的Union



Kruskal算法

- 在使用了并查集结构以及按秩结合、路径压缩等启发式方法之后，判断回路的步骤可以实现得非常高效，操作次数大约为$|E|$的级别
- 因此主要的操作时间在于排序，前面我们已经学习过，快速排序所需要的操作次数是$|E|\log_2|E|$的级别，因此总体的操作次数也大约是这个级别，还是非常快速的

Prim算法

- Prim算法也是按照贪心的策略不断迭代进行的，即每次添加到树中的边都是使树的权尽可能小的边
- Kruskal算法在执行过程的中间结果可能有多棵树（称为森林），最终才合并成我们所需的最小生成树
- 而Prim算法在生成树集合扩展时，总是形成单棵树

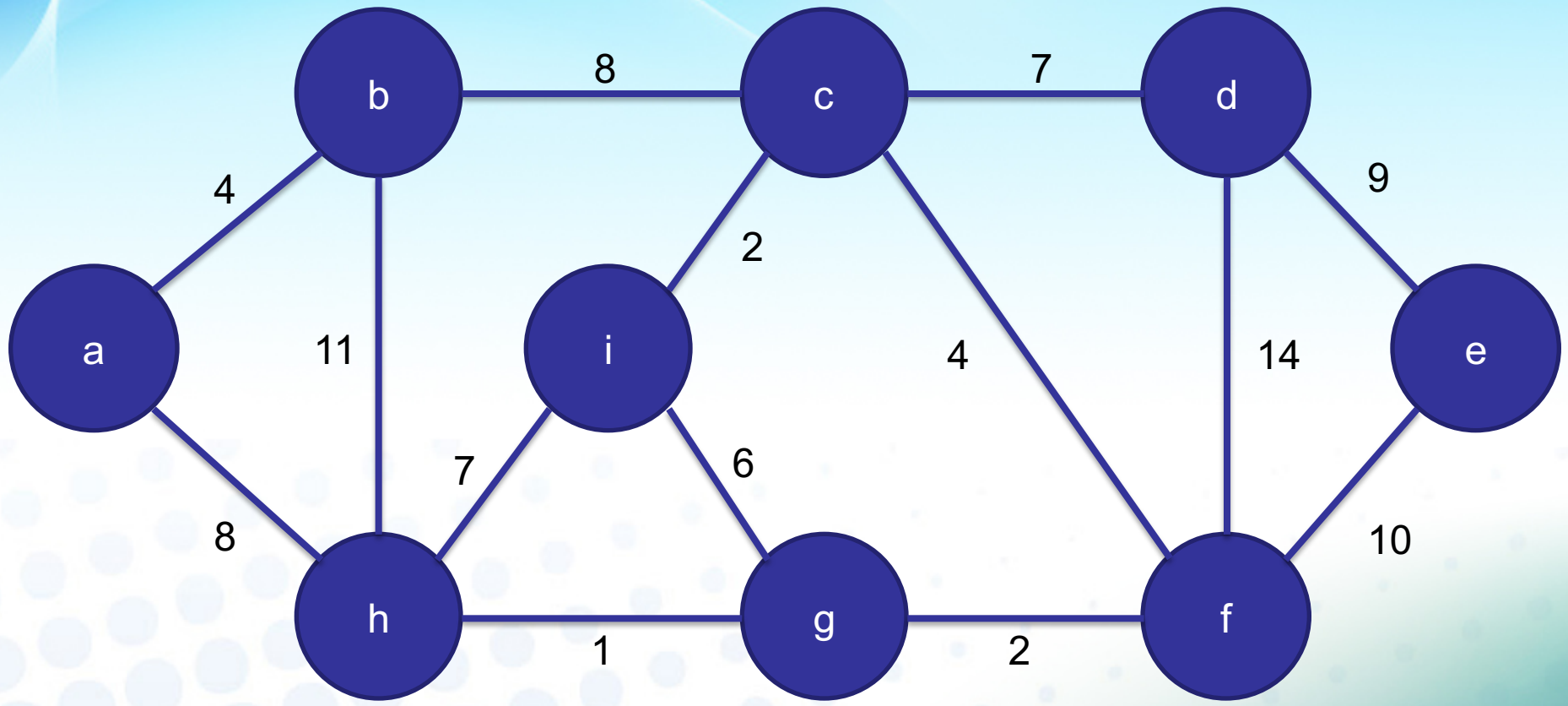
Prim算法

- Prim算法始终将顶点集合分成为不重叠的两部分：$V = V_{mst} \cup (V - V_{mst})$，这称为图的一个割（cut）
- 其中V_{mst}是当前在生成树当中的顶点集合，$V - V_{mst}$是图中不在生成树内顶点的集合
- 网络连通的情况下，只要V_{mst}与$V - V_{mst}$均非空，他们之间就至少有一条边相连，称这条边为该割的一座桥（bridge）

Prim算法

- 每一轮迭代中，都要在当前割的所有桥中，挑选出权值最小者(u, v)，其中$u \in V_{mst}$，$v \in V - V_{mst}$
- 将顶点v加入到生成树的顶点集合中，即令$V_{mst} \leftarrow V_{mst} \cup \{v\}$；同时将边$(u, v)$加入到生成树的边集合$E_{mst}$中，即令$E_{mst} \leftarrow E_{mst} \cup \{(u, v)\}$
- 不断重复这一过程，直到生成树包含所有顶点，算法终止

Prim算法的例子



Prim算法

- 有效实现Prim算法的关键是设法较为高效地选择出已经在生成树内和尚不在生成树内的顶点之间的最小权值边
- 由于生成树顶点集合是不断变化的，我们需要能够支持动态查询最小值的数据结构
- 我们前面学习过的二叉搜索树就能满足这一要求，其它可用的结构有二叉堆（heap）和斐波那契堆（Fibonacci heap）

Prim算法

- Prim算法的性能取决于查询最小边的数据结构Q是如何实现的
- 如果使用二叉最小堆或者平衡二叉搜索树来实现Q的话，更新的操作是$|E|\log_2|E|$级别的，取最小值的操作是$|V|\log_2|V|$级别的，总的操作次数是$|E|\log_2|E|$级别的
- 如果使用斐波那契堆，更新的操作次数可以减少到$|E|$的级别，总的操作次数下降到$(E + |V|\log_2|V|)$级别

活动网络

- 通常我们把计划、施工过程、生产流程、程序流程等都当成一个工程
- 除了很小的工程之外，一般都把工程分为若干个叫做“活动”的子工程，完成了这些活动，整个工程就可以完成了
- 有时，活动与活动之间存在着一定的依赖关系，我们需要在满足这些依赖的前提下设计高效合理的工程计划

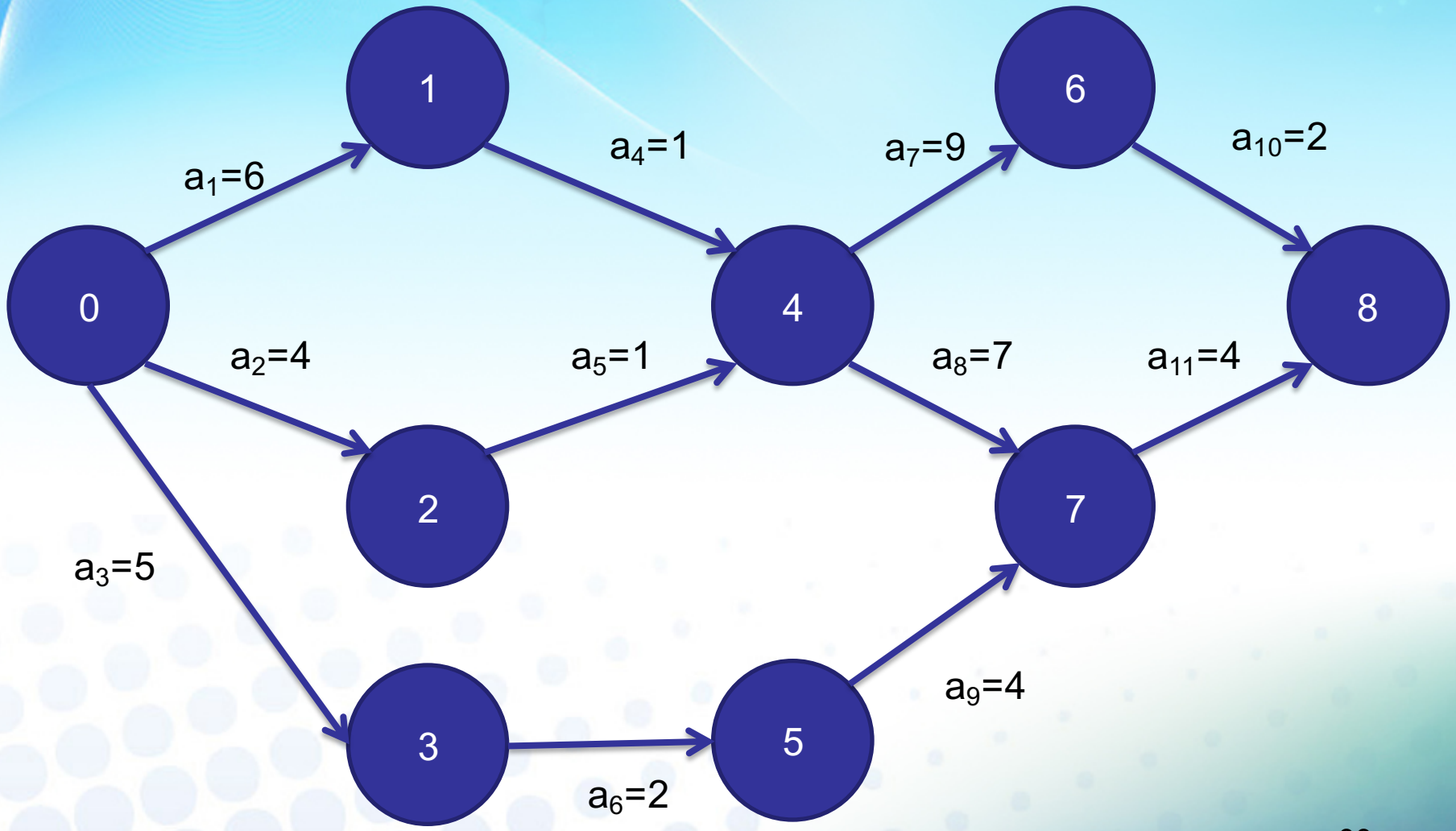
活动网络

- 比较简单的情况我们在上一节中已经做过讨论：活动的进行是有先后顺序要求的，而表示这种约束的有向图，我们也称为顶点表示活动的网络（activity on vertices），记作AOV网络
- 经过上一节的学习，我们知道使用拓扑排序可以很好地处理AOV网络的调度与安排问题，这里不再赘述

活动网络

- 另一种更为复杂的情况是：活动除了有先后顺序的限制之外，还需要考虑每项活动所需要花费的时间
- 这时，我们通常用有向边表示工程中的各项活动，用边上的权值表示活动的持续时间，用顶点表示事件
- 这样的有向图叫做用边表示活动的网络（activity on edges, AOE网络）

活动网络



活动网络

- 上图中共有9个事件，事件0表示整个工程的开始，事件8表示整个工程的结束
- 其它每个事件发生则表示在它之前的活动都已完成，在它之后的活动可以开始
- 比如事件4发生表示活动a_4和a_5已经完成，活动a_7和a_8可以开始。工程开始后，活动a_1、a_2、a_3可以并行进行，事件4发生后，活动a_7和a_8也可以并行进行

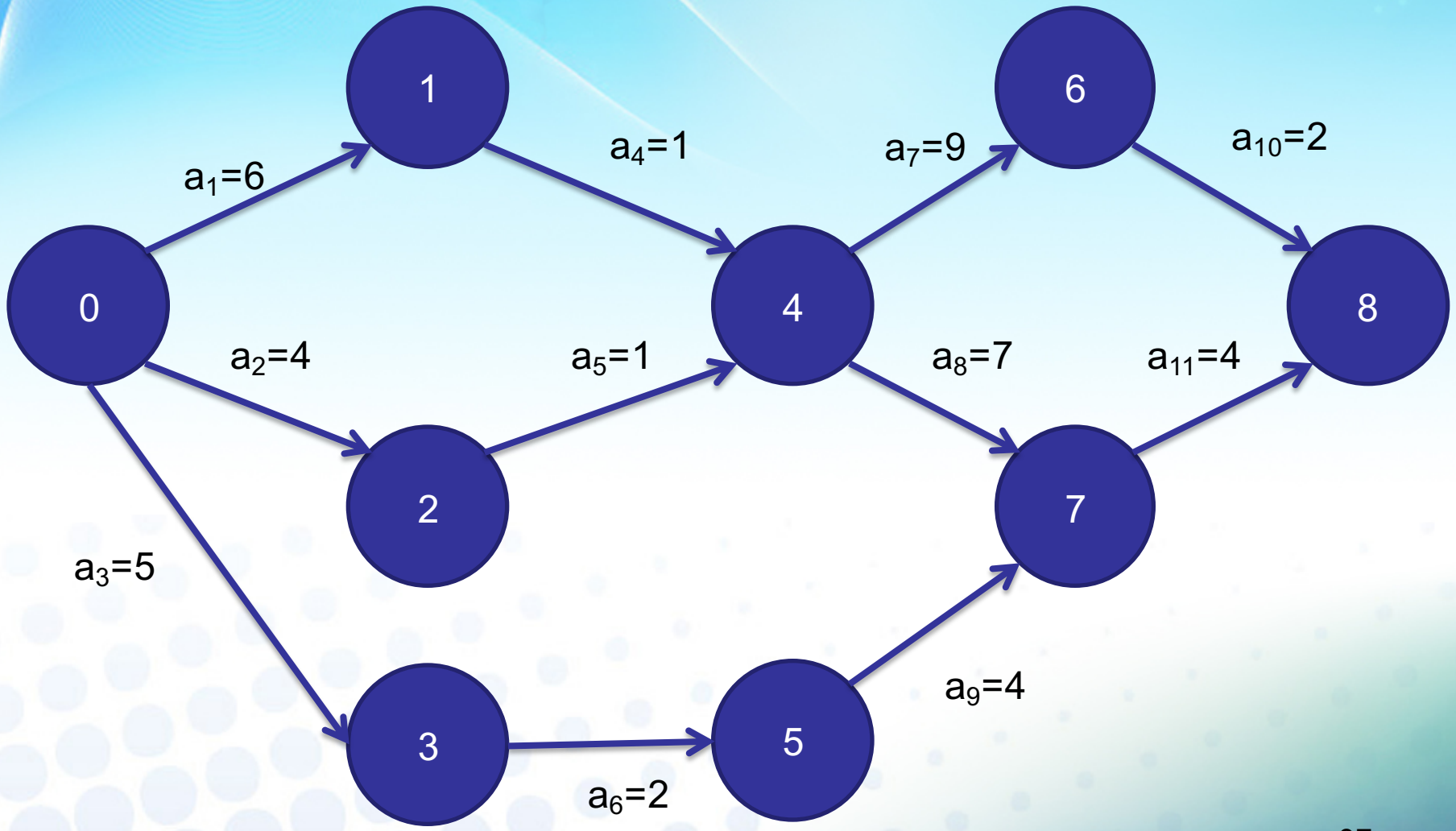
活动网络

- 通常整个工程只有一个开始点和一个完成点，我们称为源点（source）和汇点（sink）
- AOE网络在某些工程的估算方面非常有用，它可以使人们了解以下问题
  - 完成整个工程至少需要多少时间？
  - 为缩短完成工程所需的时间，应当加快哪些活动
- 其中第二个问题尤其重要

活动网络

- 在AOE网络中，有些活动可以并行进行，从源点到汇点的路径可能不止一条，不同路径的活动所需的时间虽然不同，但只有各条路径上所有活动都完成了，整个工程才算完成（木桶原理）
- 因此，完成整个工程所需的时间取决于从源点到汇点的最长路径长度，这条最长路径通常称为关键路径（critical path）

关键路径



关键路径

- 要找出关键路径，必须找出关键活动，即不按期完成就会影响整个工程完成的活动
- 关键路径上的所有活动都是关键活动，因此，只要找到了关键活动，就可以找到关键路径
- 比如上图中的a_1、a_4、a_7 、 a_8、a_{10}、a_{11}都是关键活动
- 接下来我们来考虑如何计算关键活动

相关术语

- 事件i的最早可能开始时间Ve(i)是从源点到顶点V_i的最长路径长度。例如，只有a_4、 a_5都完成，事件4才能开始，所以虽然a_2— a_5这条路径只需5天，但Ve(4)=7
- 事件i的最迟允许开始时间Vl(i)是保证汇点在最早可能时刻完成的前提下，事件i最迟的开始时间。它等于汇点的Ve减去从V_i到汇点的最长路径长度

相关术语

- 活动a_k的最早可能开始时间Ae(k)，由于每个活动必须在相关联的事件发生后才能开始，设活动在有向边<V_i, V_j>上，则Ae(k) = Ve(i)
- 活动a_k的最迟允许开始时间Al(k)，这需要保证其相关联的事件不会延误，则有Al(k) = Vl(j)-w(V_i, V_j)
- 如果Ae(k)=Al(k)，则说明活动a_k是没有时间余量的，即活动a_k是关键活动

关键路径算法

- 根据前面的定义可知，只要计算出Ve和Vl，就能够找出关键活动
- Ve需要在前面所有活动都完成的前提下才能发生，因此有

$$Ve(i)=max\{Ve(j)+w(Vj, Vi)$$

- Vl需要保证后面的活动都不会发生延误，因此有

$$Vl(i)=min\{Vl(j) - w(Vi, Vj)\}$$

关键路径算法

- 其中，为了能满足所有的约束，第一个递推式需要按拓扑顺序从前向后递推，源点的初始Ve为0
- 在计算出Ve之后，为了保证整个工程的进度，我们认为汇点的Vl等于其Ve，然后按拓扑顺序的逆序反向递推
- 最后再计算出Ae和Al，从而找出满足条件的关键活动

计算结果

事件	Ve	Vl
0	0	0
1	6	6
2	4	6
3	5	8
4	7	7
5	7	10
6	16	16
7	14	14
8	18	18

计算结果

活动	Ae	Al	关键活动
a_1	0	0	是
a_2	0	2	
a_3	0	3	
a_4	6	6	是
a_5	4	6	
a_6	5	8	
a_7	7	7	是
a_8	7	7	是
a_9	7	10	
a_{10}	16	16	是
a_{11}	14	14	是

关键路径

- 计算出网络中的关键路径，就可以辨明哪些是影响整个工程进度的关键活动，以便科学合理地安排工作
- 只有缩短关键活动的时间才有可能加快进度，但如果一个关键活动不在所有关键路径上，缩短其时间并不能加快进度
- 若是关键活动发生了延误，则一定会导致工程进度减缓

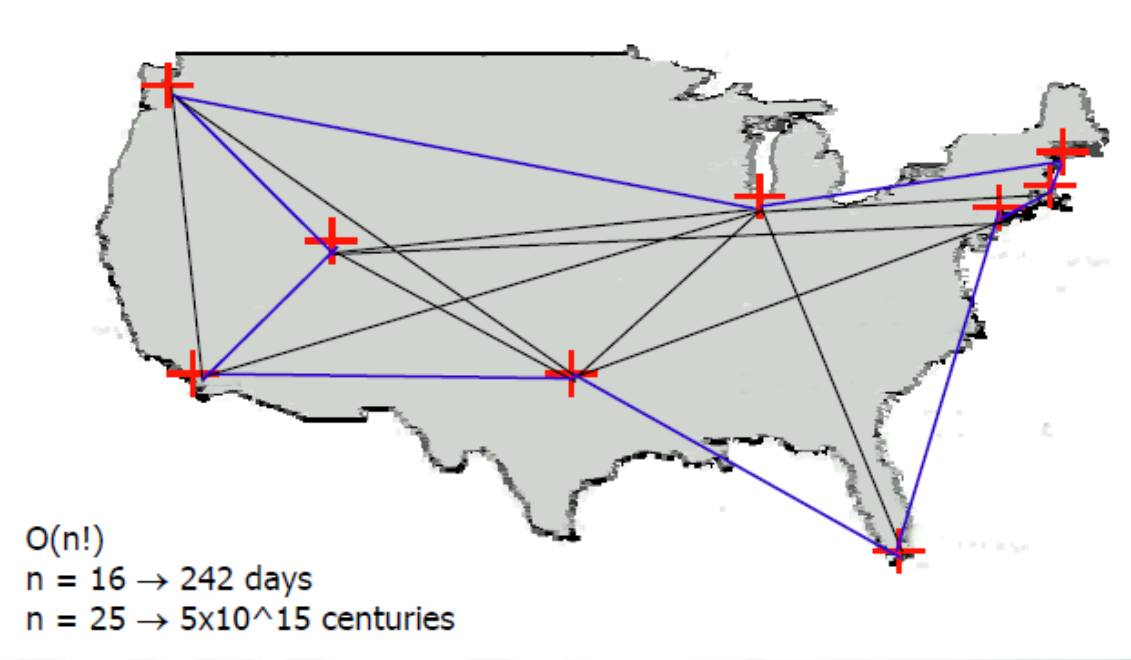
其它重要的网络问题

最短路径

- 最短路径问题是指在带权图的源点出发，找出一条通往汇点的路径，其组成边的权值之和最小
- 我们上一节介绍过的广度优先搜索，就是一种在无权图上执行的最短路径算法
- 有权图的最短路径最经典的算法是Dijkstra算法，其主要思想与广度优先搜索算法和Prim算法类似

旅行商问题

- 一个售货员希望访问n个城市，而且他希望这是一次巡回路行，即恰好访问每个城市一次，并最终回到出发的城市。任意两个城市之间都有一定的旅行费用，售货员希望能找到一条总费用最低的路线



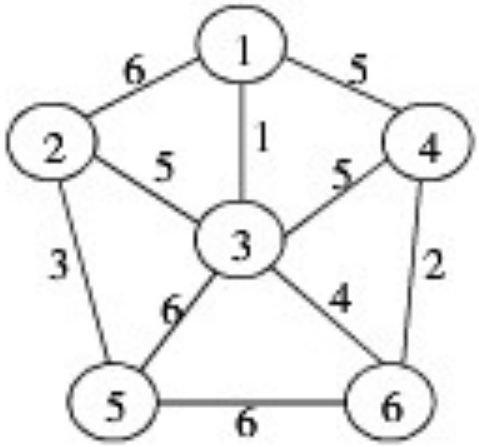
旅行商问题

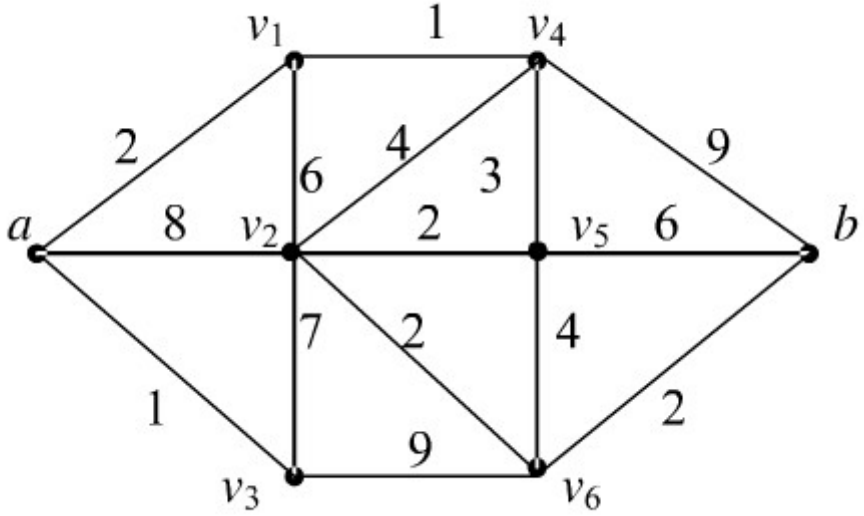
- 这个问题目前还没有很高效的算法，事实上它是NP完全的: 搜索空间是n个点的所有排列的集合，大小为(n-1)!

- 在花丛中飞来飞去的小蜜蜂显示出了轻易破解“旅行商问题”的能力。 英国伦敦大学皇家霍洛韦学院等机构研究人员利用人工控制的假花进行了实验，结果显示，不管怎样改变花的位置，蜜蜂在稍加探索后，很快就可以找到在不同花朵间飞行的最短路径。这是首次发现能解决这个问题的动物，研究报告发表在《美国博物学家》杂志上。

习题

- 给定以下两图
  - 用深度优先和广度优先两种搜索方法遍历
  - 用Kruskal和Prim两种算法求最小生成树





Thank you

Q&A