Generic depth-first search
通用的深度优先搜索

Generic depth-first search
通用的深度优先搜索

The knight-tour problem is a special kind of depth-first search of graphs

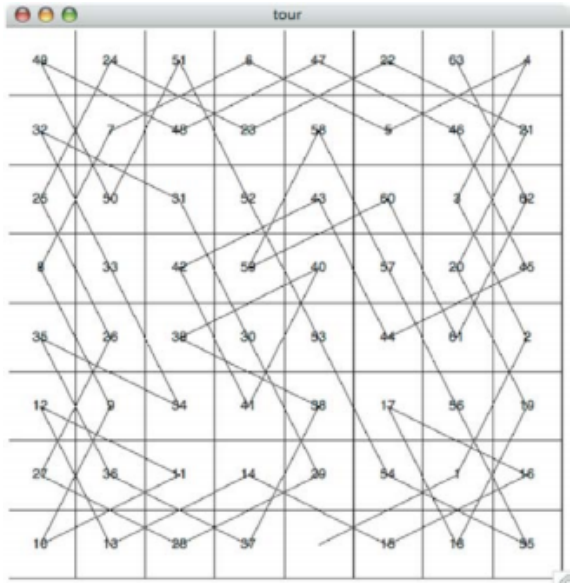
骑士周游问题是一种特殊的对图进行深度优先搜索

- Its purpose is to build a deepest depth-first tree without branches

  其目的是建立一个没有分支的最深的深度优先树

- Represents as a linear degenerate tree containing all nodes

  表现为一条线性的包含所有节点的退化树



Generic depth-first search
通用的深度优先搜索

The general depth-first search goal is to search as deep as possible on the graph, connect as many vertices as possible, and branch if necessary (a tree is created)

一般的深度优先搜索目标是在图上进行尽量深的搜索，连接尽量多的顶点，必要时可以进行分支(创建了树)

Sometimes a depth-first search creates multiple trees, called a "depth-first forest"

有时候深度优先搜索会创建多棵树，称为“深度优先森林”

Depth-first search also uses the "precursor" attribute of vertices to build trees or forests

深度优先搜索同样要用到顶点的“前驱”属性，来构建树或森林

Besides, set the "Discovery Time" and "End Time" properties

另外要设置“发现时间”和“结束时间”属性

- The former is to visit this vertex in the first step (set gray)

  前者是在第几步访问到这个顶点(设置灰色)

- The latter **completed** this vertex exploration (set black)

  后者是完成了此顶点探索(设置黑色)

These two new properties are important for later graph algorithms

这两个新属性对后面的图算法很重要

Generic depth-first search
通用的深度优先搜索

Graphs with DFS algorithms are implemented as subclasses of Graph

带有DFS算法的图实现为Graph的子类

- Vertex adds members *Discovery* and *Finish*

  顶点Vertex增加了成员Discovery及Finish

- Graph adds member *time* to record the number of steps performed by the algorithm

  图Graph增加了成员time用于记录算法执行的步骤数目

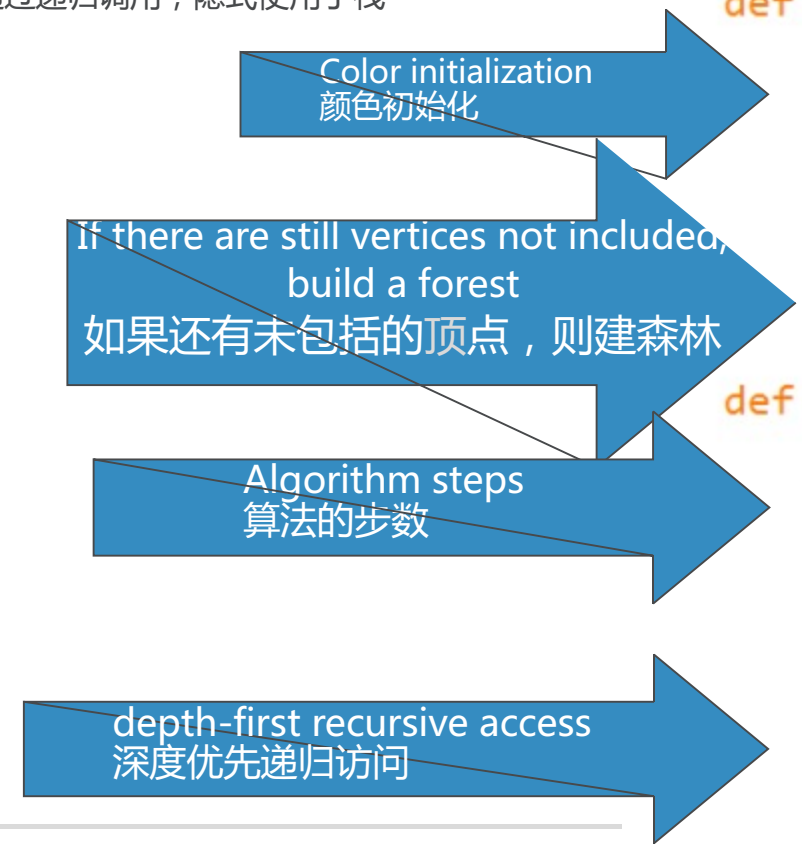
Generic depth-first search algorithm code
通用的深度优先搜索算法代码

BFS uses queues to store vertices to be accessed

BFS采用队列存储待访问顶点

DFS implicitly uses the stack through recursive calls

DFS则是通过递归调用，隐式使用了栈




```
from pythonds.graphs import Graph
class DFSGraph(Graph):
    def __init__(self):
        super().__init__()
        self.time = 0

    def dfs(self):
        for aVertex in self:
            aVertex.setColor('white')
            aVertex.setPred(-1)
        for aVertex in self:
            if aVertex.getColor() == 'white':
                self.dfsvisit(aVertex)

    def dfsvisit(self,startVertex):
        startVertex.setColor('gray')
        self.time += 1
        startVertex.setDiscovery(self.time)
        for nextVertex in startVertex.getConnections():
            if nextVertex.getColor() == 'white':
                nextVertex.setPred(startVertex)
                self.dfsvisit(nextVertex)
        startVertex.setColor('black')
        self.time += 1
        startVertex.setFinish(self.time)
```

Generic Depth-First Search Algorithm: Example

通用的深度优先搜索算法：示例

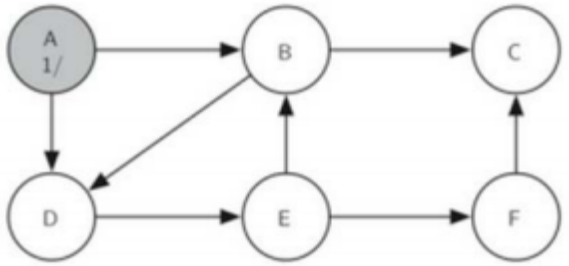


Figure 14: Constructing the Depth First Search Tree-10

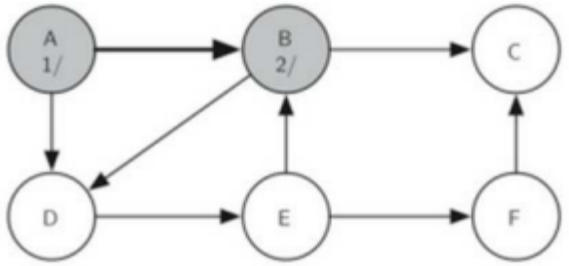


Figure 15: Constructing the Depth First Search Tree-11

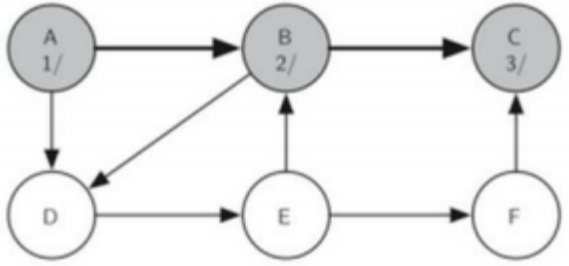


Figure 16: Constructing the Depth First Search Tree-12

Generic Depth-First Search Algorithm: Example

通用的深度优先搜索算法：示例

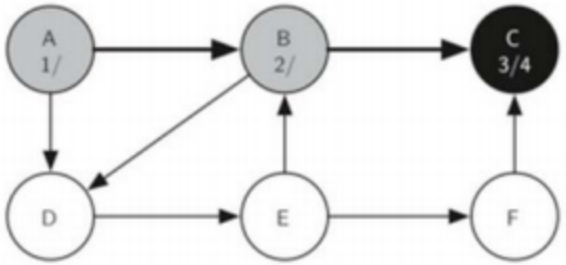


Figure 17: Constructing the Depth First Search Tree-13

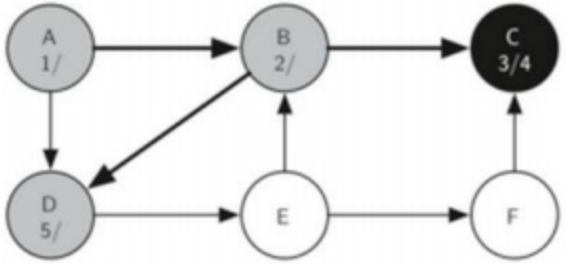


Figure 18: Constructing the Depth First Search Tree-14

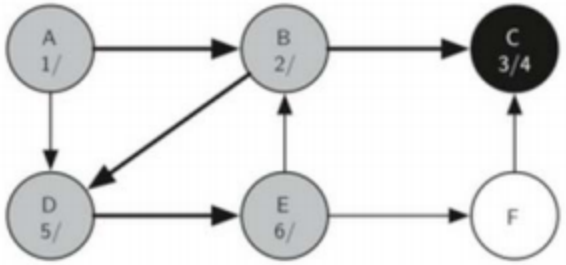


Figure 19: Constructing the Depth First Search Tree-15

Generic Depth-First Search Algorithm: Example
通用的深度优先搜索算法：示例

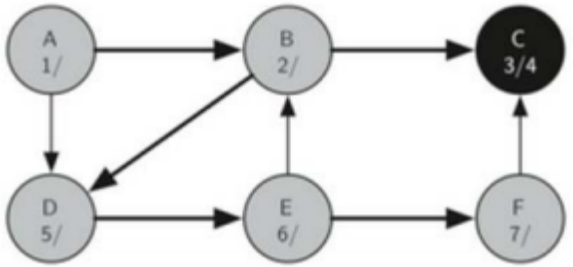


Figure 20: Constructing the Depth First Search Tree-16

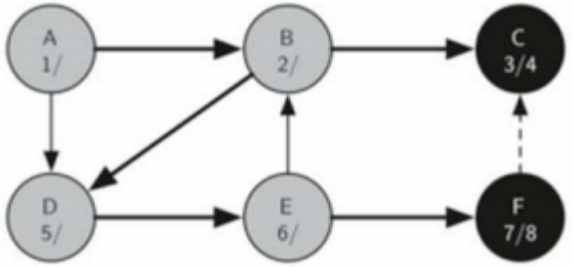


Figure 21: Constructing the Depth First Search Tree-17

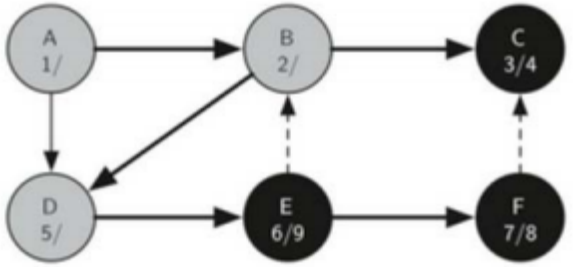


Figure 22: Constructing the Depth First Search Tree-18

Generic Depth-First Search Algorithm: Example
通用的深度优先搜索算法：示例

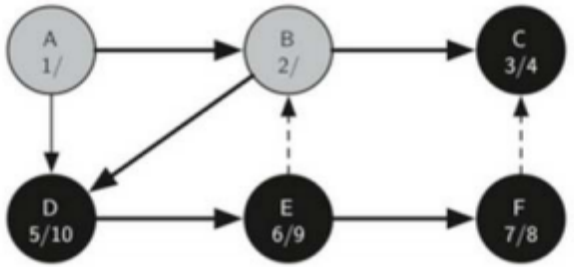


Figure 23: Constructing the Depth First Search Tree-19

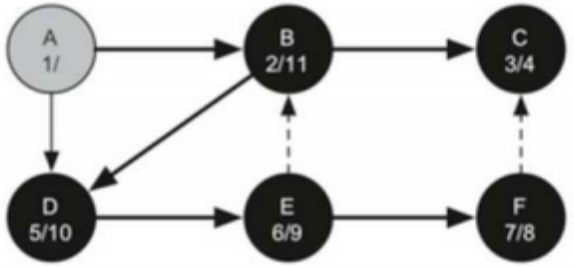


Figure 24: Constructing the Depth First Search Tree-20

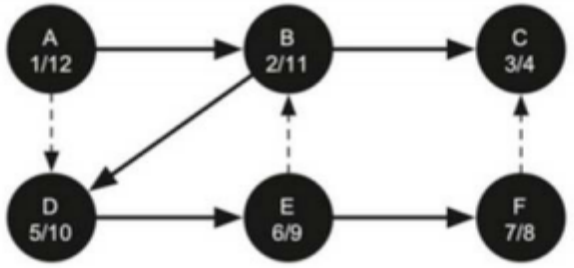


Figure 25: Constructing the Depth First Search Tree-21

General Depth-First Search Algorithms: Analysis

通用的深度优先搜索算法：分析

A tree constructed by DFS with the "discovery time" and "end time" attributes of its vertices, with **parentheses**-like properties

DFS构建的树，其顶点的“发现时间”和“结束时间”属性，具有类似括号的性质

That is, the "discovery time" of a vertex is always less than the "discovery time" of all child vertices

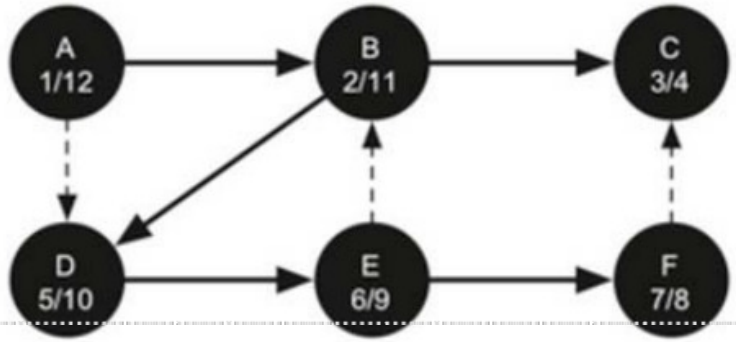
即一个顶点的“发现时间”总小于所有子顶点的“发现时间”

And the "end time" is greater than the "end time" of all child vertices

而“结束时间”则大于所有子顶点“结束时间”

Discovered earlier than child vertices and ended exploration later

比子顶点更早被发现，更晚被结束探索



General Depth-First Search Algorithms: Analysis
通用的深度优先搜索算法：分析

DFS runtime also includes two aspects:
DFS运行时间同样也包括了两方面：

There are two loops in the dfs function, each is |V| times, so it's O(|V|)

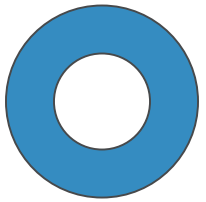
dfs函数中有两个循环，每个都是|V|次，所以是O(|V|)

The loop in the dfsvisit function is performed on the vertices connected to the current vertex, and the recursive call is only made when the vertex is white, so it will only run one step for each edge, so it is O(|E |)

而dfsvisit函数中的循环则是对当前顶点所连接的顶点进行，而且仅有在顶点为白色的情况下才进行递归调用，所以对每条边来说只会运行一步，所以是O(|E|)

It adds up to the same O(|V|+|E|) as BFS

加起来就是和BFS一样的O(|V|+|E|)



Applications of Graphs: The Shortest Path Problem

图的应用：最短路径问题

The Shortest Path Problem: An Introduction
最短路径问题：介绍

When we browse the web, send emails, and transmit QQ messages through the Internet, data flows between connected devices

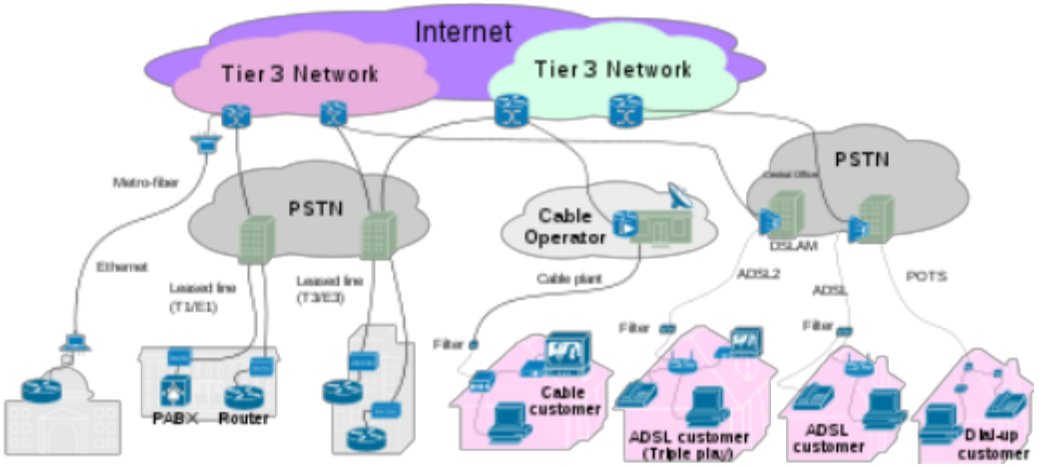
当我们通过网络浏览网页、发送电子邮件、QQ消息传输的时候，数据会在联网设备之间流动

The computer network specialization will study the technical details of the network at all levels in detail

计算机网络专业领域会详尽地研究网络各层面上的技术细节

We are primarily interested in the way the Internet works in the graph algorithms it contains

我们对Internet工作方式感兴趣的主要是其中包含的图算法



The Shortest Path Problem: An Introduction

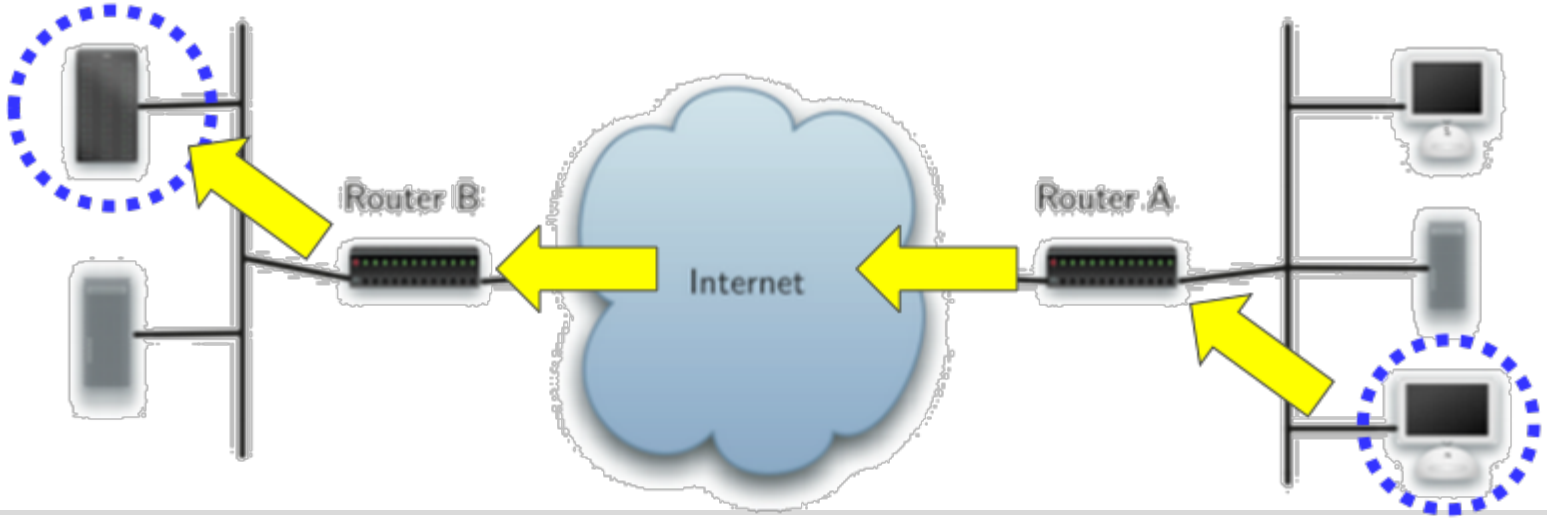
最短路径问题：介绍

As shown in the figure, when the browser on the PC requests a web page from the server, the requested information requires:

如图，当PC上的浏览器向服务器请求一个网页时，请求信息需要：

First, it is sent to the Internet by router A through the local local area network, and the request information is propagated along many routers in the Internet, and finally reaches router B, which belongs to the local area network of the server, and is transmitted to the server.

先通过本地局域网，由路由器A发送到Internet，请求信息沿着Internet中的众多路由器传播，最后到达服务器本地局域网所属的路由器B，从而传给服务器。



The Shortest Path Problem: An Introduction
最短路径问题：介绍

The cloud-like structure labeled "Internet" is actually a network of routers that work independently and together to transmit information from one end of the Internet to the other.

标注“Internet”的云状结构，实际上是一个由路由器连接成的网络，这些路由器各自独立而又协同工作，负责将信息从Internet的一端传送到另一端。

We can trace the path that the information went through using the "traceroute" command:

我们可以通过“traceroute”命来跟踪信息传送的路径：

```
traceroutewww.lib.pku.edu.cn
```

The Shortest Path Problem: An Introduction

最短路径问题：介绍

Let's take a look at a router path from the local computer to the Peking University library server, including 4 routers

我们来看看从本机到北大图书馆服务器之间的一条路由器路径，包含了4个路由器

```
traceroute to www.lib.pku.edu.cn (162.105.138.158), 64 hops max,
 1  rt-ac54u.lan (192.168.123.1)  1.299 ms  1.042 ms  1.026 ms
 2  10.128.224.1 (10.128.224.1)  4.148 ms  1.311 ms  1.266 ms
 3  162.105.253.237 (162.105.253.237)  1.288 ms  1.207 ms  1.23
 4  162.105.253.94 (162.105.253.94)  1.575 ms  78.726 ms  284.6
 5  www.lib.pku.edu.cn (162.105.138.158)  1.861 ms  1.982 ms  1.
```

Since the conditions of network can affect the path selection algorithm, the paths may be different at different times.

由于网络流量的状况会影响路径选择算法，在不同的时间，路径可能不同。

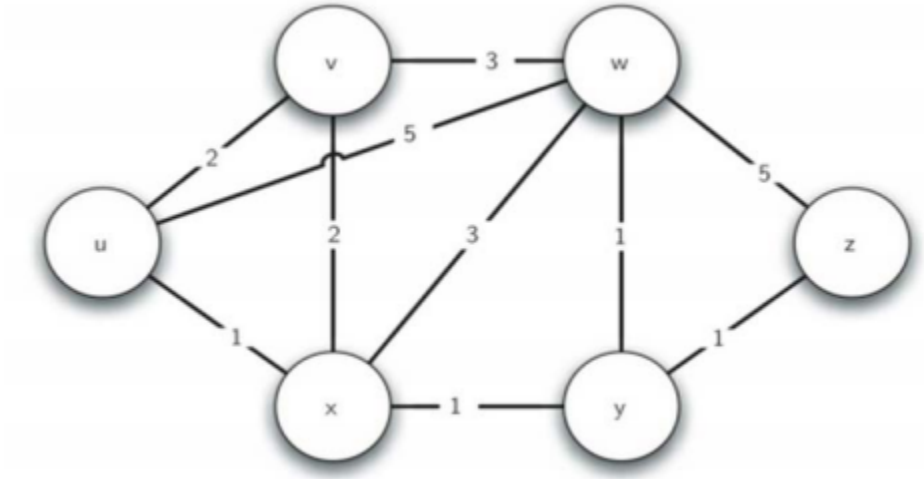
The Shortest Path Problem: An Introduction
最短路径问题：介绍

So we can represent the Internet router system as a graph with weighted edges
所以我们可以将互联网路由器体系表示为一个带权边的图

Routers are used as vertices, and network connections between routers are used as edges
路由器作为顶点，路由器之间网络连接作为边

The weight can include factors such as the speed of the network connection, the degree of network load, and the priority of the time period
权重可以包括网络连接的速度、网络负载程度、分时段优先级等影响因素

As an abstraction, we synthesize all influencing factors into a single weight
作为一个抽象，我们把所有影响因素合成为单一的权重



The Shortest Path Problem: An Introduction
最短路径问题：介绍

Solving the problem of choosing the fastest path for information to spread in the router network turns into the problem of the shortest path on a weighted graph.

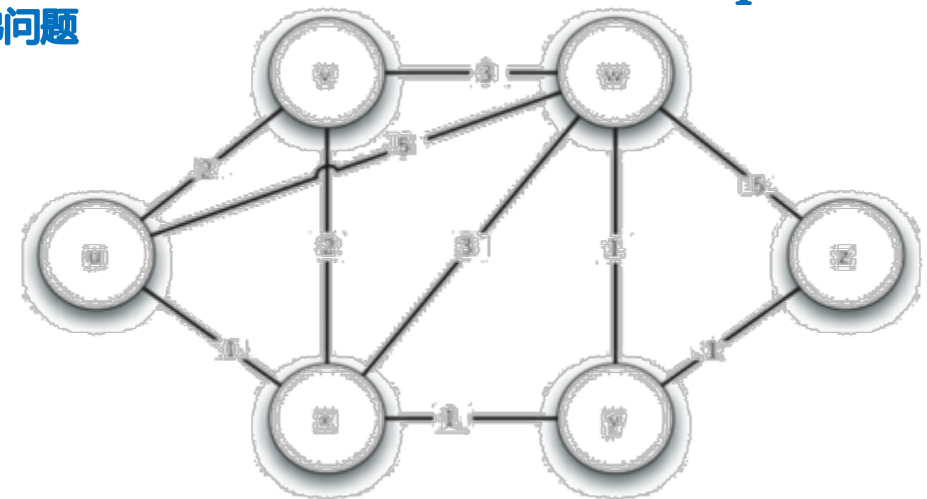
解决信息在路由器网络中选择传播速度最快路径的问题，就转变为在带权图上最短路径的问题。

This problem is similar to the word ladder problem solved by the breadth-first search BFS algorithm, except that the weights are added to the edges.

这个问题与广度优先搜索BFS算法解决的词梯问题相似，只是在边上增加了权重

If all weights are equal, it will return to the word ladder problem

如果所有权重相等，还是还原到词梯问题



The Shortest Path Problem: Dijkstra's Algorithm

最短路径问题：Dijkstra算法

The classic algorithm for solving the weighted shortest path problem is the "**Dijkstra algorithm**" named after the inventor /ˈdɛɪkstra/

解决带权最短路径问题的经典算法是以发明者命名的“Dijkstra算法” /ˈdɛɪkstra/

This is an iterative algorithm that finds the shortest path from one vertex to all other vertices, very close to the result of the breadth-first search algorithm BFS

这是一个迭代算法，得出从一个顶点到其余所有顶点的最短路径，很接近于广度优先搜索算法BFS的结果

In terms of specific implementation, the member ***dist*** in the Vertex class is used to record the length of the shortest weighted path (**sum of weights**) from the starting vertex to this vertex, and the algorithm iterates once for each vertex in the graph

具体实现上，在顶点Vertex类中的成员dist用于记录从开始顶点到本顶点的最短带权路径长度(权重之和)，算法对图中的每个点迭代一次

The Shortest Path Problem: Dijkstra Algorithm

最短路径问题：Dijkstra算法

The access order of vertices is controlled by a priority queue, and the priority in the queue is the dist attribute of the vertex.

顶点的访问次序由一个优先队列来控制，队列中作为优先级的是顶点的dist属性。

Initially, only the starting vertex dist is set to 0, and all other vertex dist is set to sys.maxsize (maximum integer), and all join the priority queue.

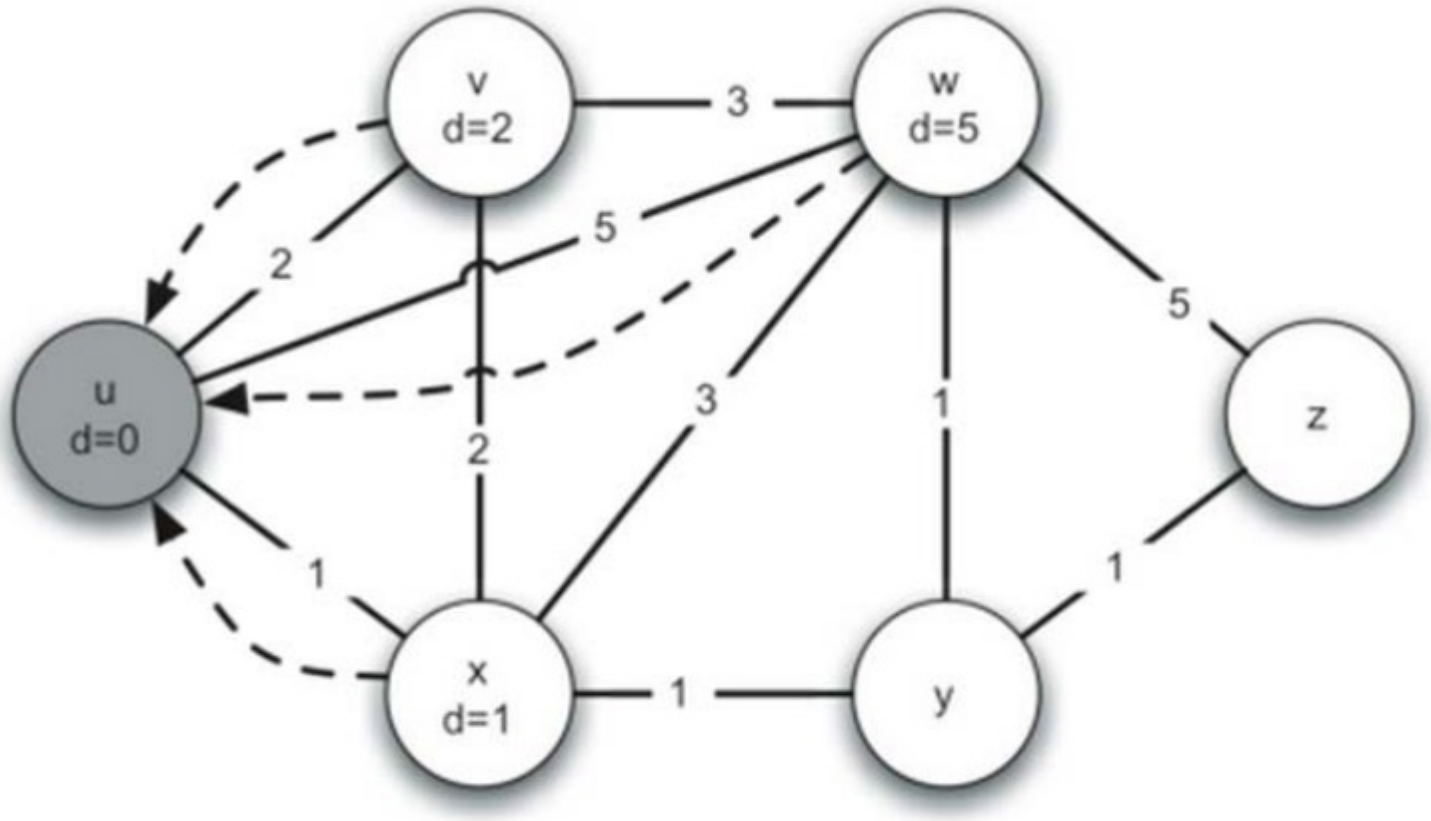
最初，只有开始顶点dist设为0，而其他所有顶点dist设为sys.maxsize(最大整数)，全部加入优先队列。

As each lowest dist vertex in the queue is dequeued first, and the weight of it and adjacent vertices is calculated, it will cause the reduction and modification of the dist of other vertices, causing the heap to rearrange, and then dequeue according to the updated dist priority.

随着队列中每个最低dist顶点率先出队，并计算它与邻接顶点的权重，会引起其它顶点dist的减小和修改，引起堆重排，并据更新后的dist优先级再依次出队

The Shortest Path Problem: Dijkstra Algorithm

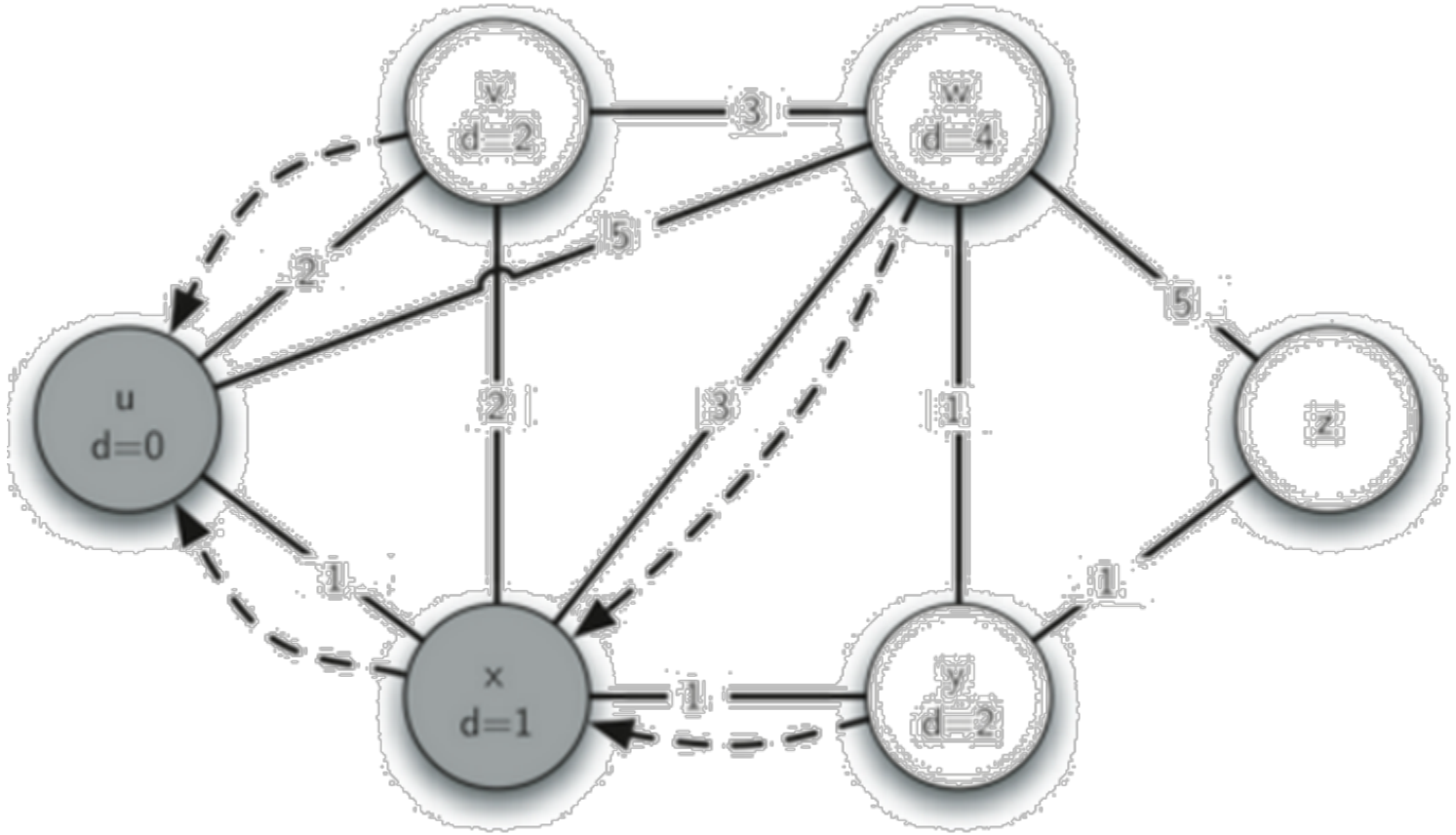
最短路径问题：Dijkstra算法



PQ = x,v,w

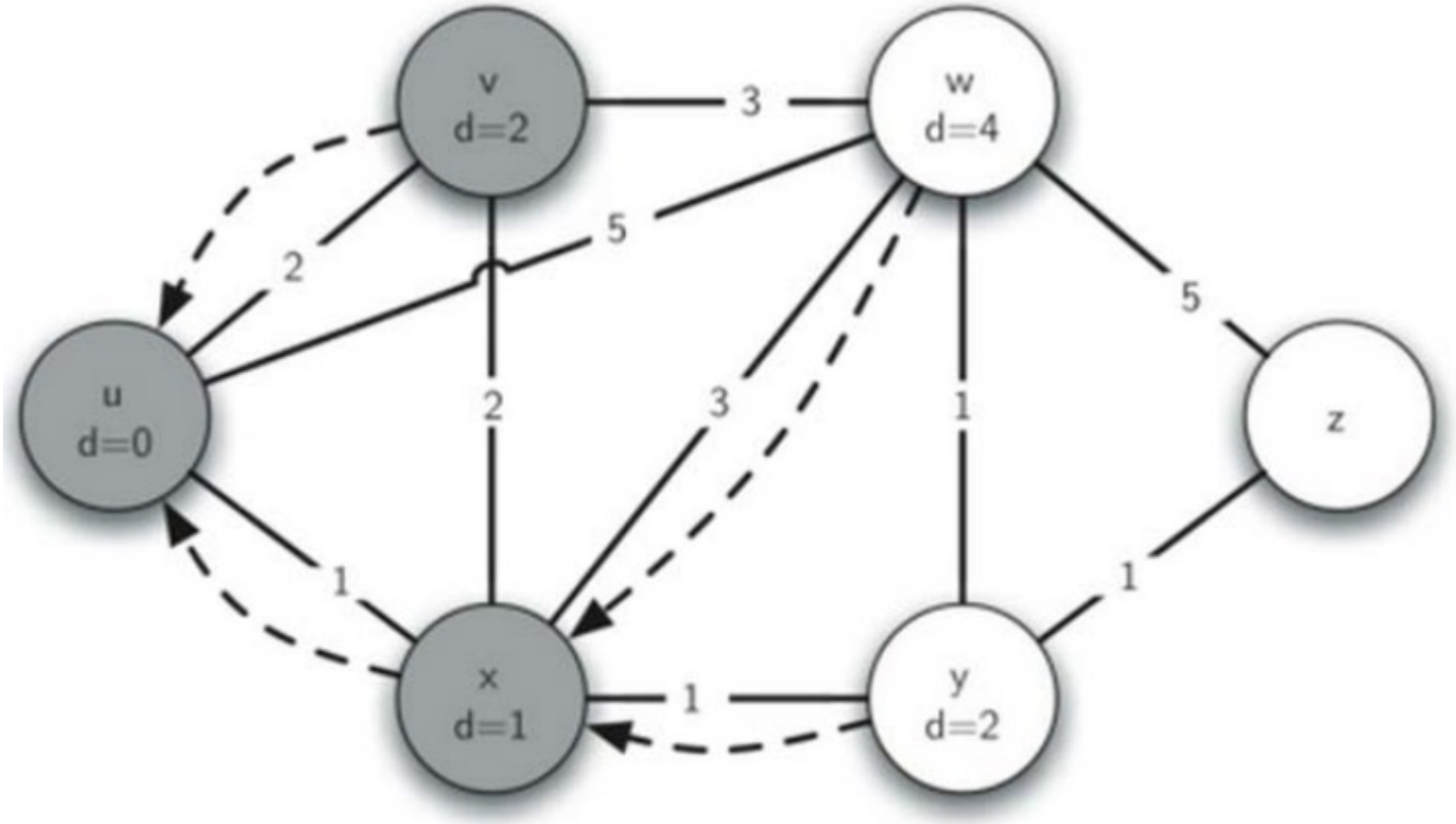
The Shortest Path Problem: Dijkstra Algorithm

最短路径问题：Dijkstra算法



The Shortest Path Problem: Dijkstra Algorithm

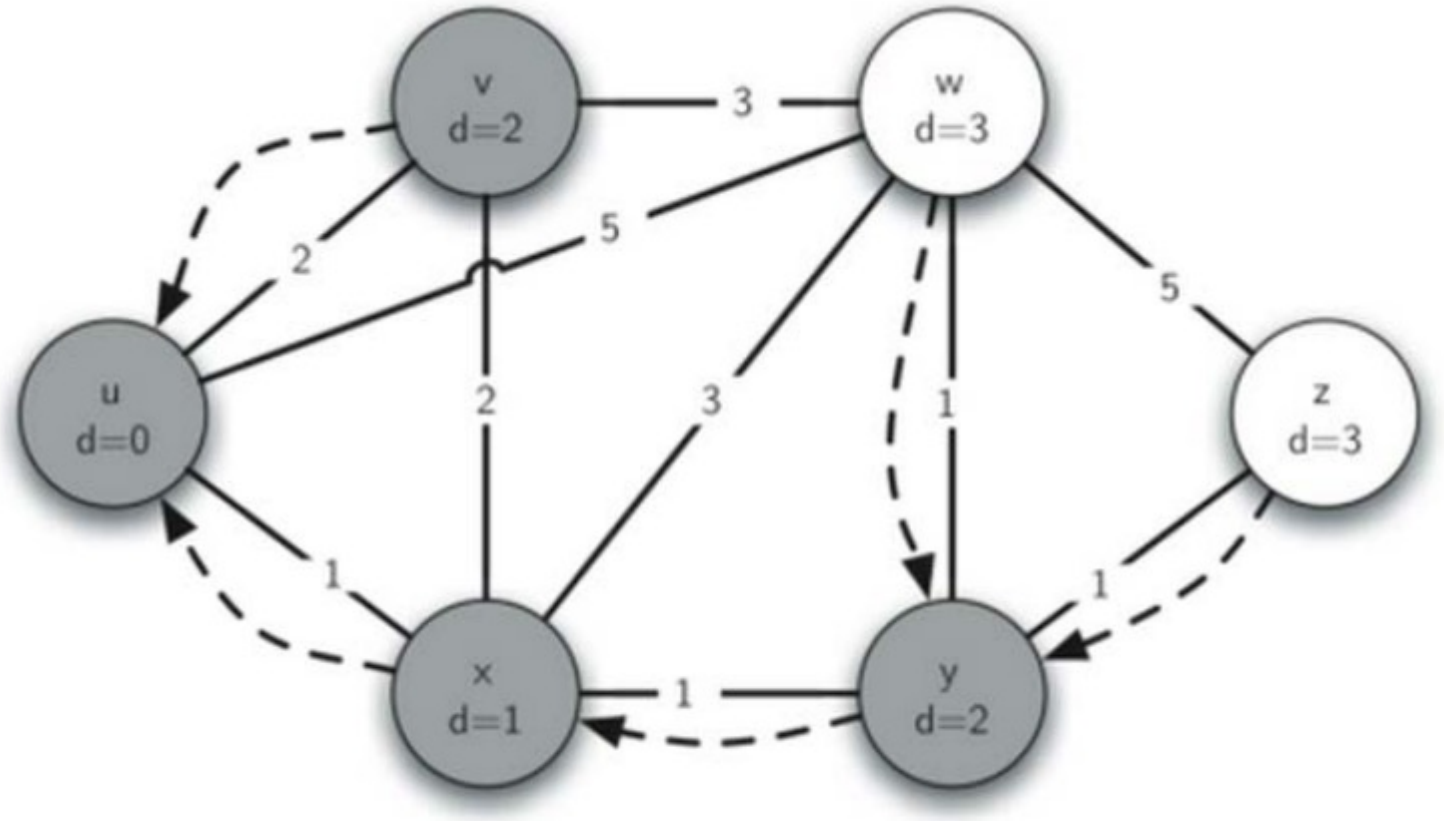
最短路径问题：Dijkstra算法



PQ = yw

The Shortest Path Problem: Dijkstra Algorithm

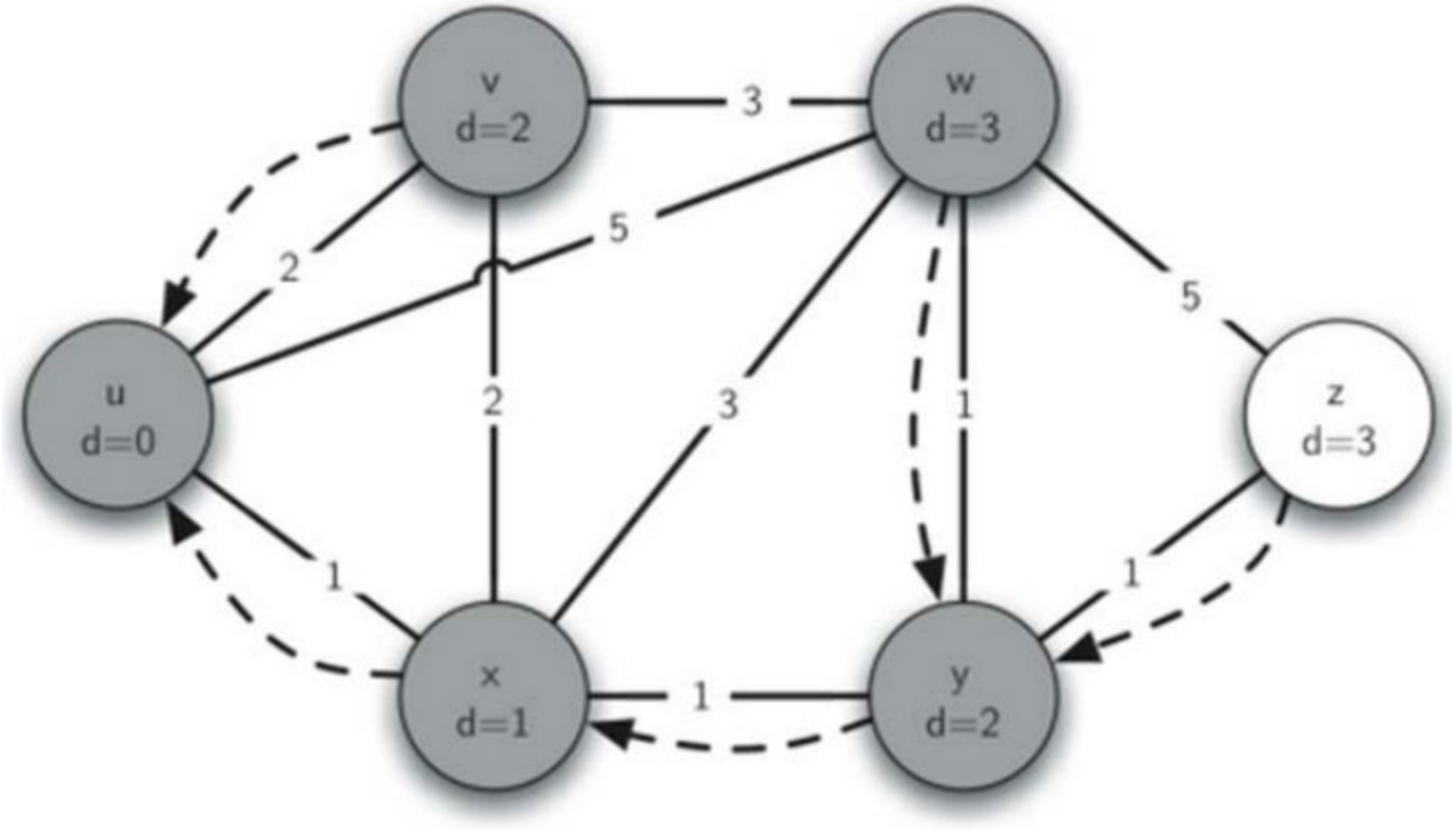
最短路径问题：Dijkstra算法



PQ = wz

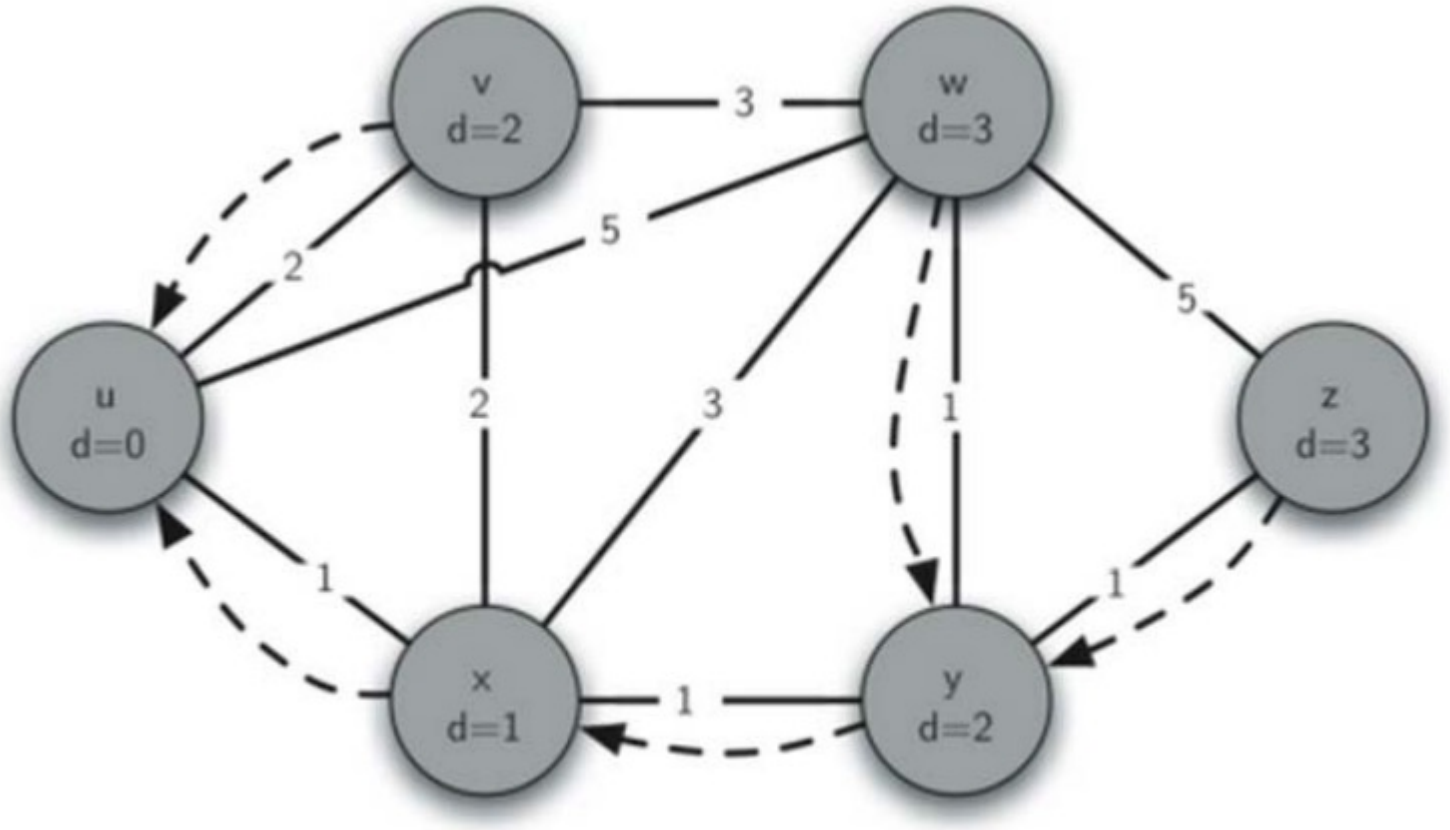
The Shortest Path Problem: Dijkstra Algorithm

最短路径问题：Dijkstra算法



The Shortest Path Problem: Dijkstra Algorithm

最短路径问题：Dijkstra算法



PQ = None

Shortest Path Problem: Dijkstra's Algorithm Code

最短路径问题：Dijkstra算法代码

Build a heap for all vertices to form a priority queue
对所有顶点建堆，形成优先队列

Priority queue dequeue
优先队列出队

Modify the dist of the vertices adjacent to the dequeuing vertex, and rearrange the queue one by one
修改出队顶点所邻接顶点的dist，并逐个重排队列

```
from pythonds.graphs import PriorityQueue, Graph, Vertex
def dijkstra(aGraph,start):
    pq = PriorityQueue()
    start.setDistance(0)
    pq.buildHeap([(v.getDistance(),v) for v in aGraph])
    while not pq.isEmpty():
        currentVert = pq.delMin()
        for nextVert in currentVert.getConnections():
            newDist = currentVert.getDistance() \
                    + currentVert.getWeight(nextVert)
            if newDist < nextVert.getDistance():
                nextVert.setDistance( newDist )
                nextVert.setPred(currentVert)
                pq.decreaseKey(nextVert,newDist)
```

The Shortest Path Problem: Dijkstra Algorithm

最短路径问题：Dijkstra算法

It should be noted that Dijkstra's algorithm can only handle weights greater than 0

需要注意的是，Dijkstra算法只能处理大于0的权重

If negative weights appear in the graph, the algorithm gets stuck in an infinite loop

如果图中出现负数权重，则算法会陷入无限循环

Although Dijkstra algorithm perfectly solves the shortest path problem with weighted graphs, other algorithms are actually used in Internet routers ( )

虽然Dijkstra算法完美解决了带权图的最短路径问题，但实际上Internet的路由器中采用的是其它算法()

The Shortest Path Problem: Dijkstra Algorithm

最短路径问题：Dijkstra算法

The most important reason is that the Dijkstra algorithm needs to have the data of the entire graph, but for the routers of the Internet, it is obviously impossible to save all the routers and their connection information of the entire Internet locally.

其中最重要的原因是，Dijkstra算法需要具备整个图的数据，但对于Internet的路由器来说，显然无法将整个Internet所有路由器及其连接信息保存在本地

This is not only a problem of the amount of data, but also the dynamic characteristics of the Internet make it unrealistic to save the whole image.

这不仅是数据量的问题，Internet动态变化的特性也使得保存全图缺乏现实性。

The router's routing algorithm (or "routing algorithm") is extremely important for the Internet. If you are interested, you can refer to "distance vector routing algorithm".

路由器的选径算法(或“路由算法”)对于互联网极其重要，有兴趣可以进一步参考“距离向量路由算法“

https://baike.baidu.com/item/distance vector routing algorithm

https://baike.baidu.com/item/距离向量路由算法

The Shortest Path Problem: An Analysis of Dijkstra Algorithm

最短路径问题：Dijkstra算法分析

First, add all vertices to the priority queue and build a heap, the time complexity is O(|V|)

首先，将所有顶点加入优先队列并建堆，时间复杂度为O(|V|)

Secondly, each vertex is only dequeued once, and each delMin costs O(log|V|), a total of O(|V|log|V|)

其次，每个顶点仅出队1次，每次delMin花费O(log|V|)，一共就是O(|V|log|V|)

In addition, the vertex associated with each edge will perform a decreaseKey operation (O(log|V|)), a total of O(|E|log|V|)

另外，每个边关联到的顶点会做一次decreaseKey操作(O(log|V|))，一共是O(|E|log|V|)

Adding the above three together, the order of magnitude is O((|V|+|E|)log|V|)

上面三个加在一起，数量级就是O((|V|+|E|)log|V|)

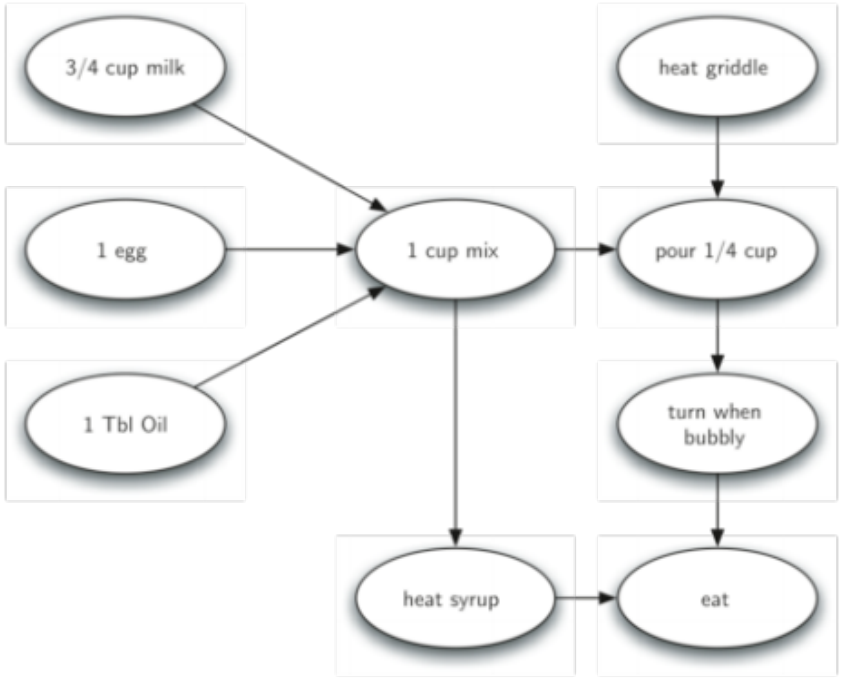
Applications of Graphs: Topological Sort
图的应用：拓扑排序

Topological Sort
拓扑排序

Many problems can be transformed into graphs and solved by graph algorithms
很多问题都可转化为图，利用图算法解决

For example, the process of eating pancakes for breakfast
例如早餐吃薄煎饼的过程

Take actions as vertices, and order as directed edges
以动作为顶点，以先后次序为有向边

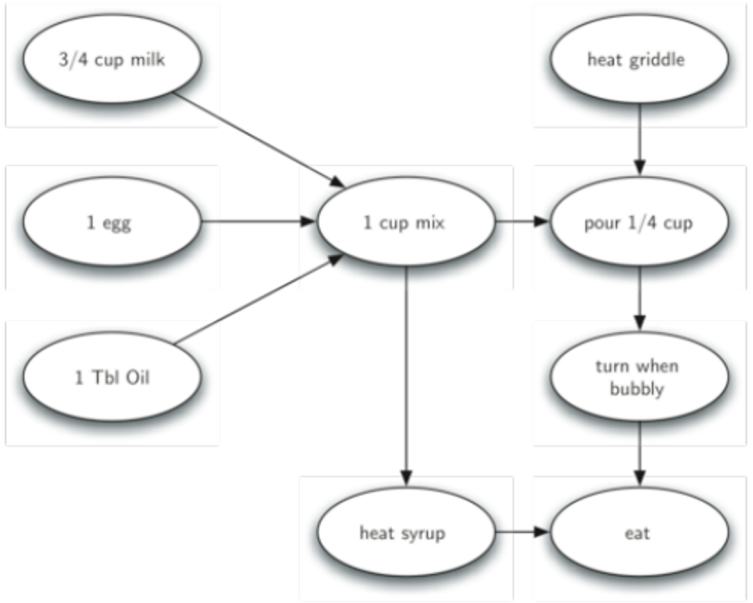


Topological Sort
拓扑排序

The problem is for the whole process
问题是对整个过程而言

If a person does it alone, what is the order of all actions?
如果一个人独自做，所有动作的先后次序？

Start with toppings? Or start with a heated griddle?
从加料开始？还是从加热烤盘开始？



Topological Sort
拓扑排序

The algorithm that obtains the work order arrangement from the work flow chart is called "topological sort"

从工作流程图得到工作次序排列的算法，称为“拓扑排序”

Topological sort processes a DAG, outputting a linear sequence of vertices

拓扑排序处理一个DAG，输出顶点的线性序列

> Make two vertices v,w, if G has a (v,w) edge, v appears before w in the linear sequence.
>
> 使得两个顶点v,w，如果G中有(v,w)边，在线性序列中v就出现在w之前。

Topological sort is widely used in event-dependent scheduling, and can also be used in project management, database query optimization, and order optimization of matrix multiplication

拓扑排序广泛应用在依赖事件的排期上，还可以用在项目管理、数据库查询优化和矩阵乘法的次序优化上

Topological Sort
拓扑排序

Topological sort can be well implemented using DFS:

拓扑排序可以采用DFS很好地实现：

- Build a workflow as a graph, where work items are vertices and dependencies are directed edges

  将工作流程建立为图，工作项是节点，依赖关系是有向边

- The work flow chart must be a DAG graph, otherwise there is a cyclic dependency to call the DFS algorithm on the DAG graph to get the "end time" of each vertex

  工作流程图一定是个DAG图，否则有循环依赖对DAG图调用DFS算法，以得到每个顶点的“结束时间”

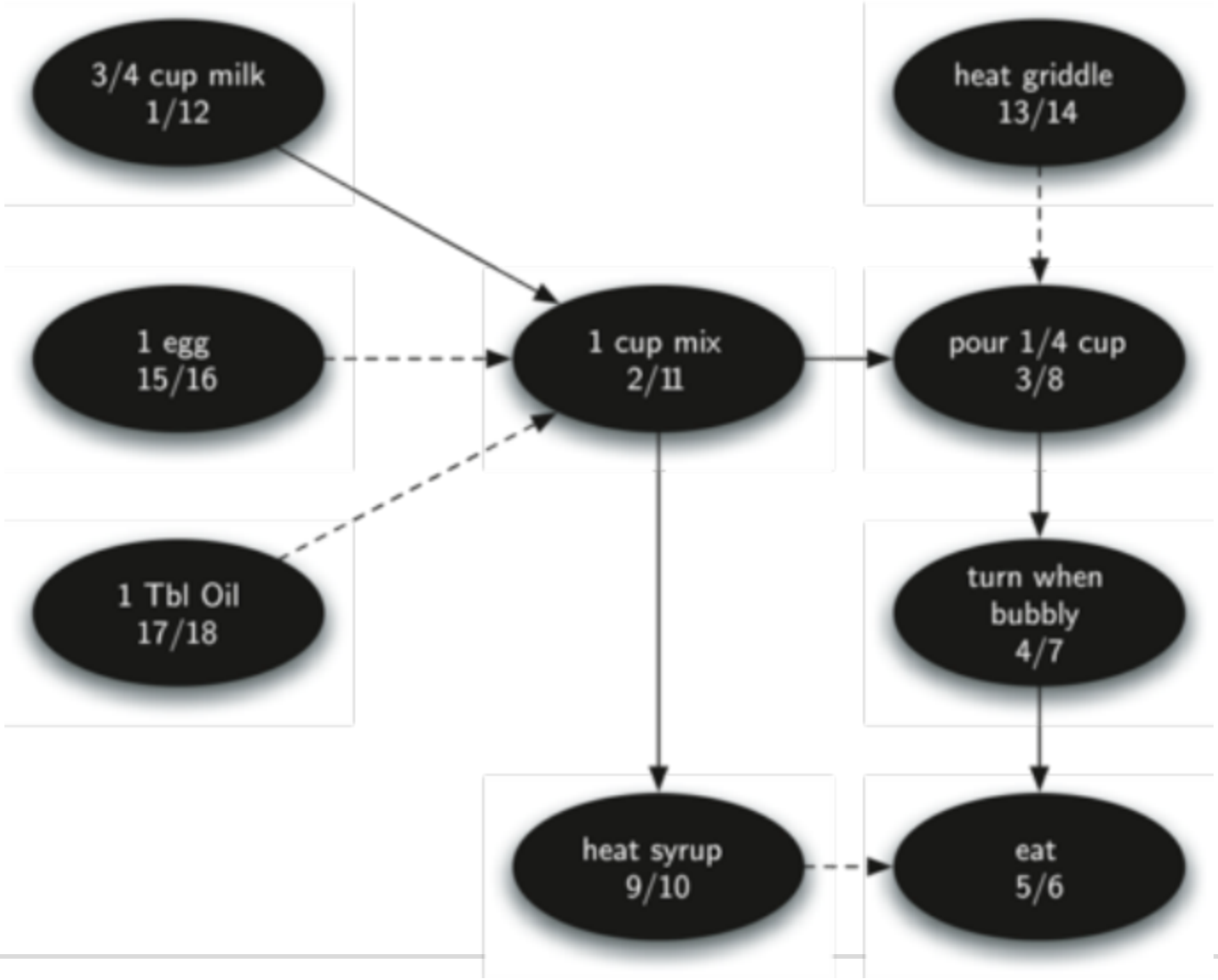
- Sort by the "end time" of each vertex from largest to smallest

  按照每个顶点的“结束时间”从大到小排序

- output the list of vertices in this order

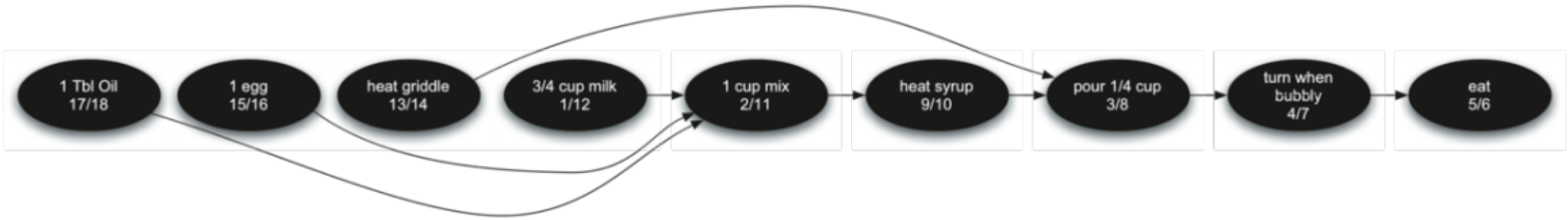
  输出这个次序下的顶点列表

Topological Sort: Example
拓扑排序：示例



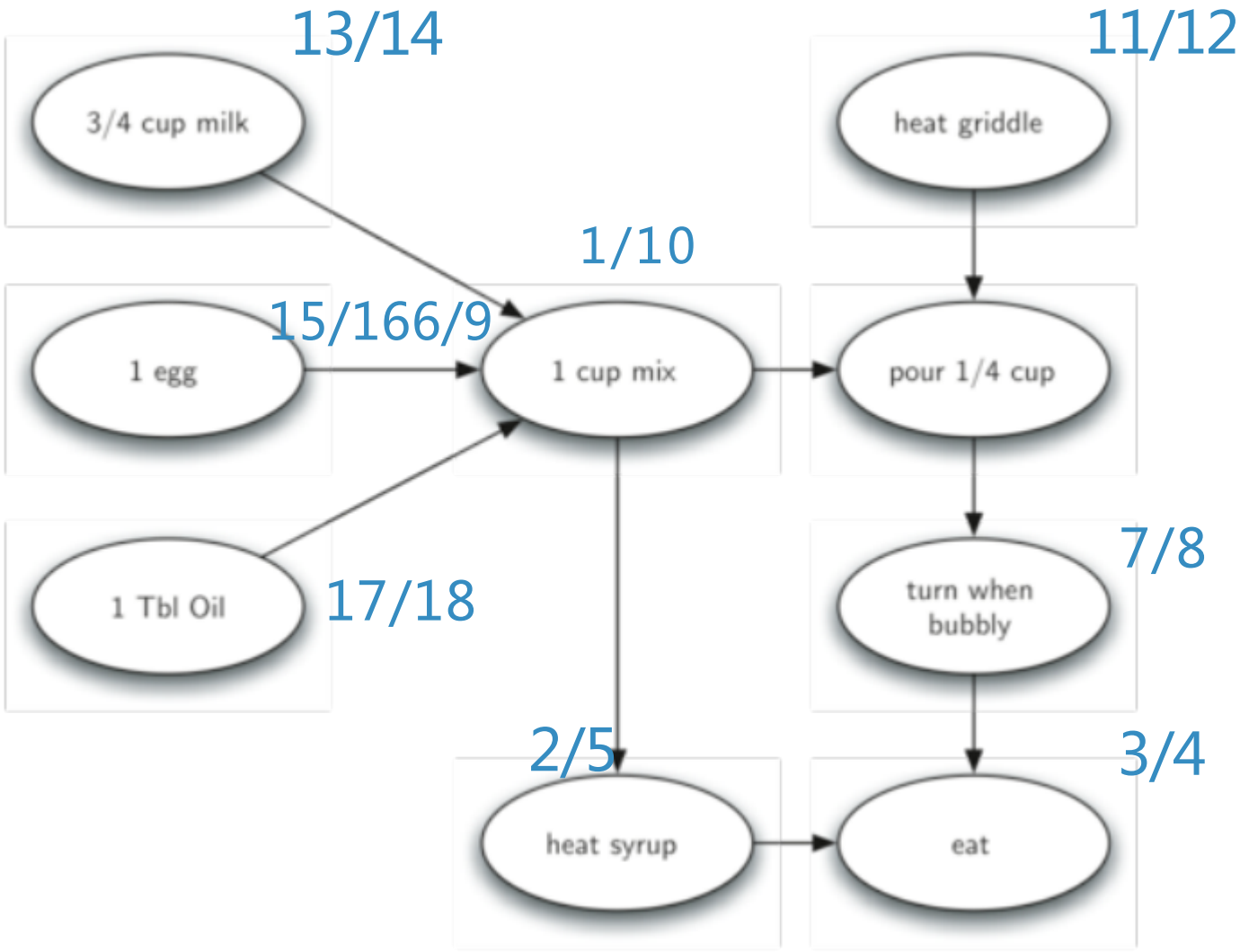
Topological Sort: Example
拓扑排序：示例

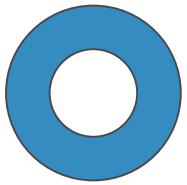
Topological Sort: Example 1
拓扑排序：示例1



Topological Sort: Example 2

拓扑排序：示例2





Applications of Graphs: Strongly Connected Branches

图的应用：强连通分支

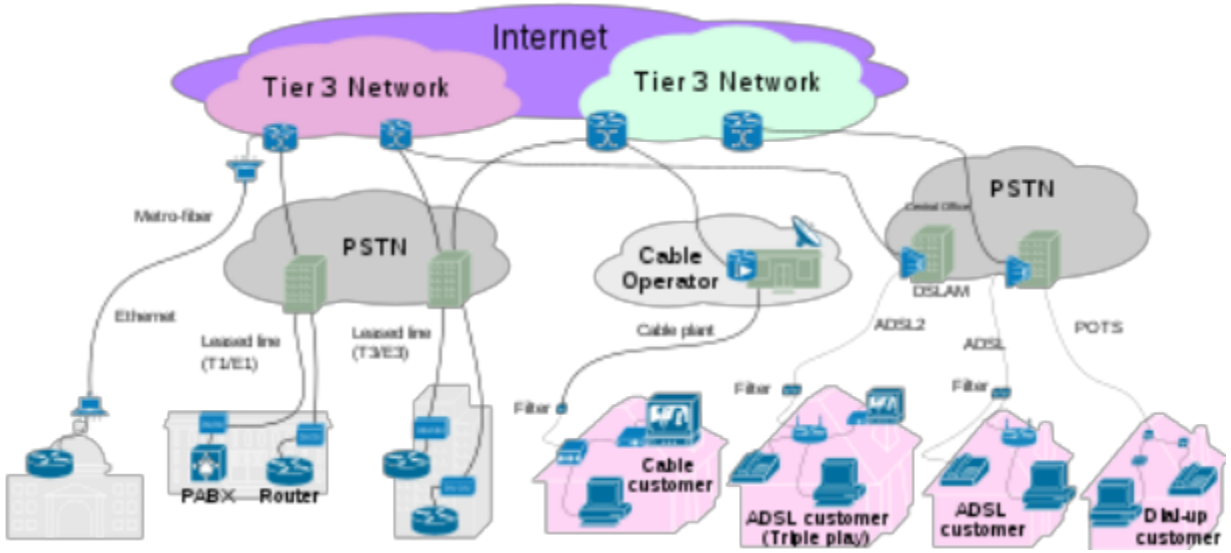
Strongly connected branch
强连通分支

Let's take a look at a very gigantic picture related to the Internet:

我们关注一下互联网相关的非常巨大图：

A graph formed by a host computer connected by a network cable (or wireless); and a graph formed by a web page connected by a hyperlink.

由主机通过网线(或无线)连接而形成的图；以及由网页通过超链接连接而形成的图。

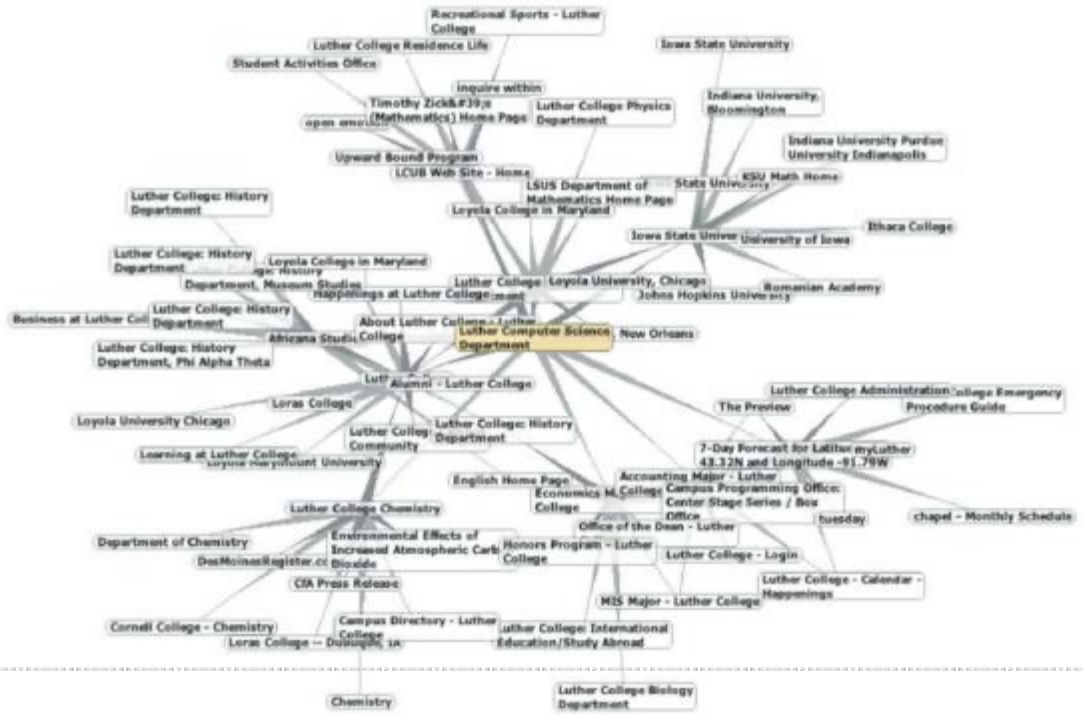


Strongly connected branch
强连通分支

First look at the picture formed by the webpage
先看网页形成的图

Taking the web page (URI as the id) as the vertex and the hyperlink contained in the web page as the edge, it can be converted into a directed graph.

以网页(URI作为id)为顶点，网页内包含的超链接作为边，可以转换为一个有向图。



Strongly connected branch
强连通分支

There are three interesting phenomena in the website link situation of the Department of Computer Science of Lutheran College

路德学院计算机系网站链接情况，有三个有趣的现象

①The image contains the websites of many other departments at Lutheran College

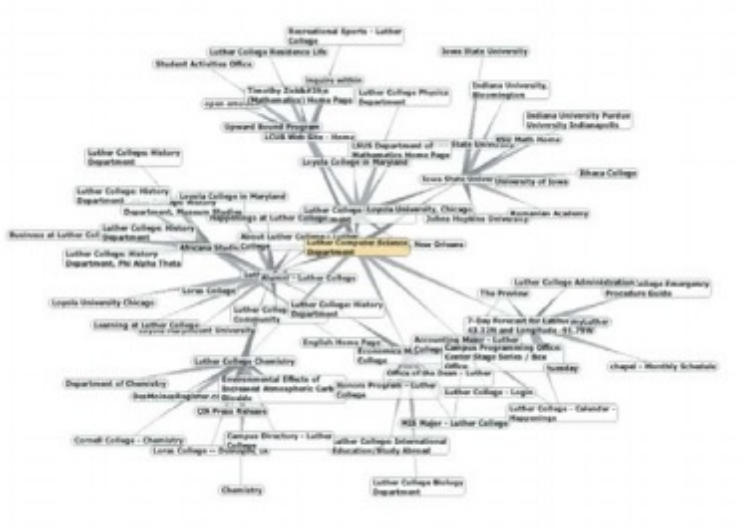
图中包含了许多路德学院其它系的网站

②Contains the websites of some of the other university colleges in Iowa

包含了一些爱荷华其它大学学院的网站

③And also contains the websites of some of the Colleges of Humanities

还包含了一些人文学院的网站



Strongly connected branch
强连通分支

We can guess that the underlying structure of the Web may have aggregations of some similar websites
我们可以猜想，Web的底层结构可能存在某些同类网站的聚集

An algorithm for finding highly aggregated node groups in the graph, that is, finding the "StronglyConnectedComponents" algorithm
在图中发现高度聚集节点群的算法，即寻找“强连通分支“算 法

Strongly connected branch, defined as a subset C of the graph G
强连通分支，定义为图G的一个子集C

There is a path back and forth between any two vertices v,w in C, that is, (v,w)(w,v) is the path of C, and C is the largest subset with such properties
C中的任意两个顶点v,w之间都有路径来回，即(v,w)(w,v)都是C的路径，而且C是具有这样性质的最大子集

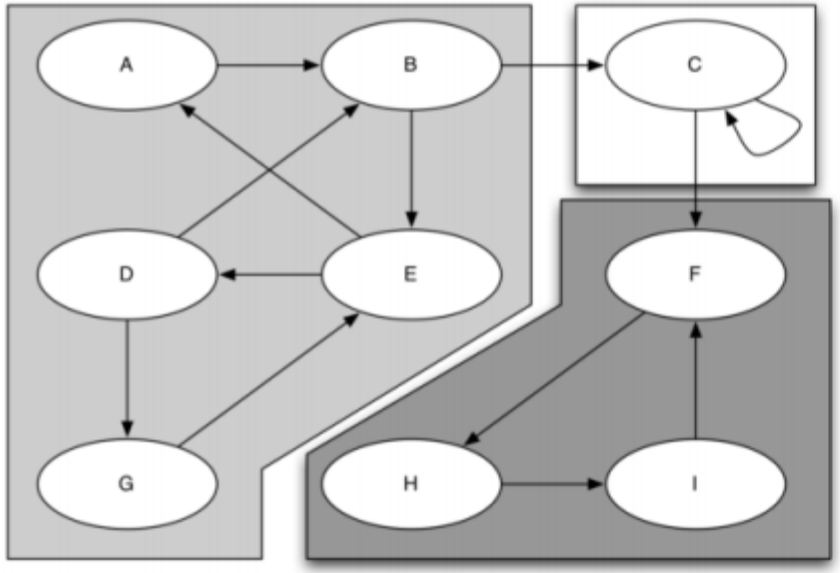
Strongly connected branch example
强连通分支例子

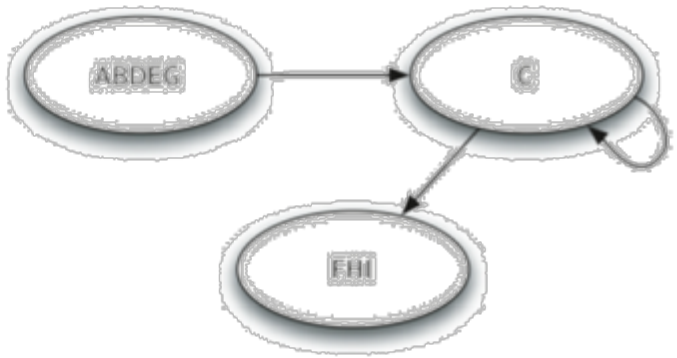
The following figure is a 9-vertex directed graph with 3 strongly connected branches

下图是具有3个强连通分支的9顶点有向图

Once the strongly connected branch is found, the vertices of the graph can be classified accordingly and the graph can be simplified.

一旦找到强连通分支，可以据此对图的顶点进行分类，并对图进行化简。





Strongly Connected Branch Algorithm: Transpose Concept

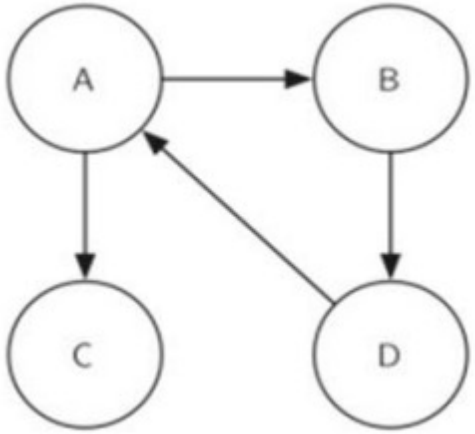
强连通分支算法：转置概念

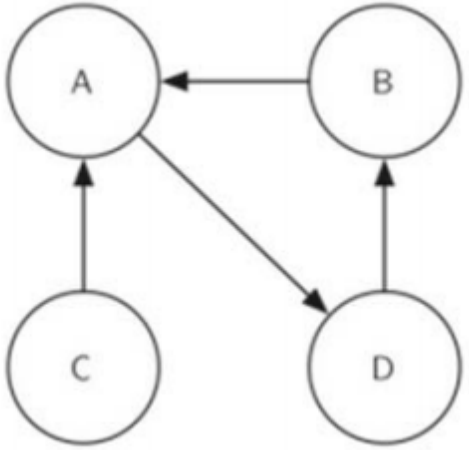
Before using depth-first search to find strongly connected branches, familiarize yourself with a concept: Transposition

在用深度优先搜索来发现强连通分支之前，先熟悉一个概念：转置

A transpose G^T of a directed graph G is defined as the exchange of the order of vertices of all edges of the graph G, such as converting (v, w) to (w, v), it can be observed that the graph and the transposed graph are in the strongly connected branch. The number and division are the same.

一个有向图G的转置G^T，定义为将图G的所有边的顶点交换次序，如将(v,w)转换为(w,v)，可以观察到图和转置图在强连通分支的数量和划分上，是相同的。





Strongly connected branch algorithm: Kosaraju algorithm idea

强连通分支算法：Kosaraju算法思路

First, call the DFS algorithm on the graph G to calculate the "end time" for each vertex;

首先，对图G调用DFS算法，为每个顶点计算“结束时间”；

Then, transpose the graph G to get G^T;

然后，将图G进行转置，得到G^T；

Then call the DFS algorithm on G^T, but in the *dfs* function, in the search loop for each vertex, search in the reverse order of the "end time" of the vertices

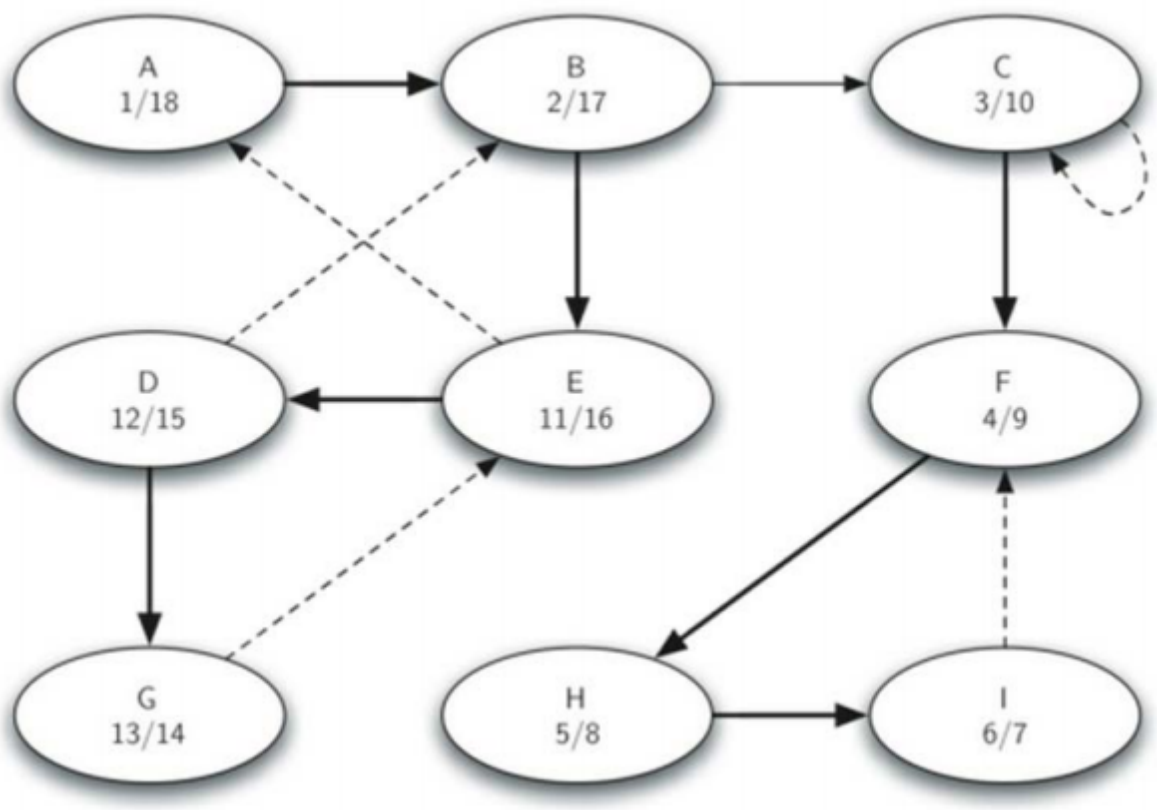
再对G^T调用DFS算法，但在dfs函数中，对每个顶点的搜索循环里，要以顶点的“结束时间”倒序的顺序来搜索

Finally, every tree in a depth-first forest is a strongly connected branch

最后，深度优先森林中的每一棵树就是一个强连通分支

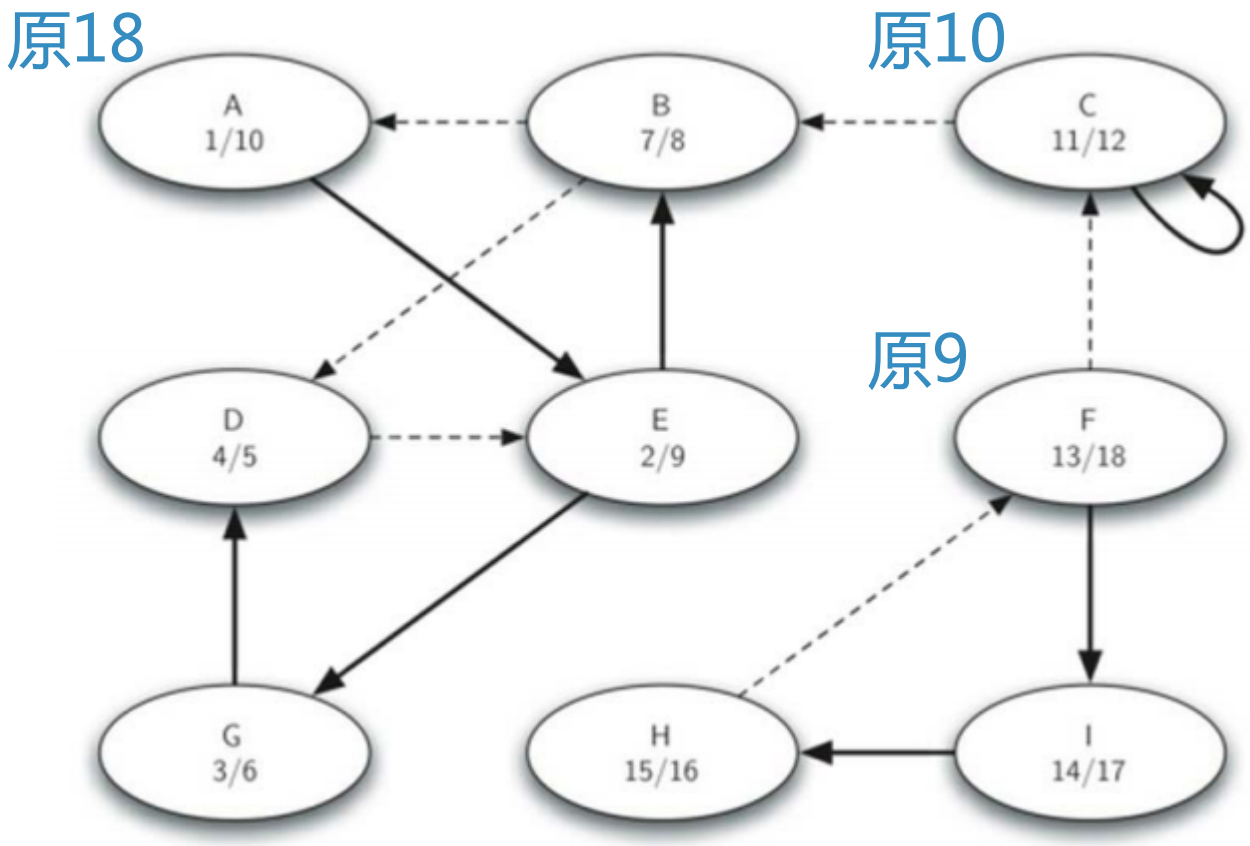
Kosaraju Algorithm Example: The First DFS

Kosaraju算法实例：第一趟DFS



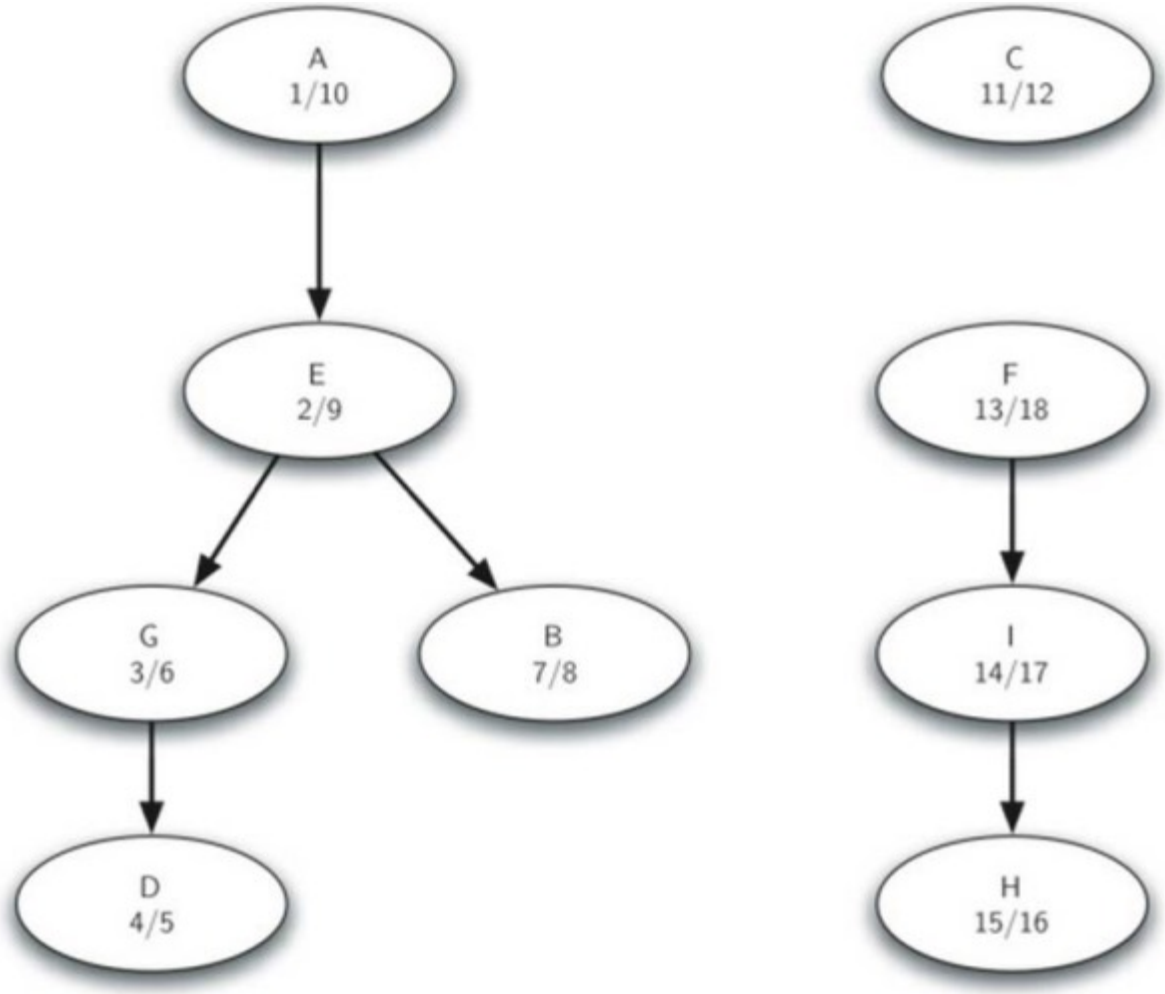
Kosaraju algorithm example: second DFS after transposition

Kosaraju算法实例：转置后第二趟DFS



Kosaraju Algorithm Example: Results

Kosaraju算法实例：结果



Strongly Connected Branch Algorithm
强连通分支算法

Another commonly used strongly connected branch algorithm
另外的常用强连通分支算法

①Tarjan's algorithm

Tarjan算法

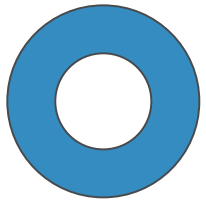
②Gabow's algorithm, an improvement on Tarjan

Gabow算法，对Tarjan的改进

reference reading
参考阅读

https://baike.baidu.com/item/tarjan algorithm

https://baike.baidu.com/item/tarjan算法



Applications of Graphs: Minimum Spanning Trees

图的应用：最小生成树

minimum spanning tree
最小生成树

This algorithm deals with the problems faced by online game designers and Internet radios in the Internet: the problem of information broadcasting

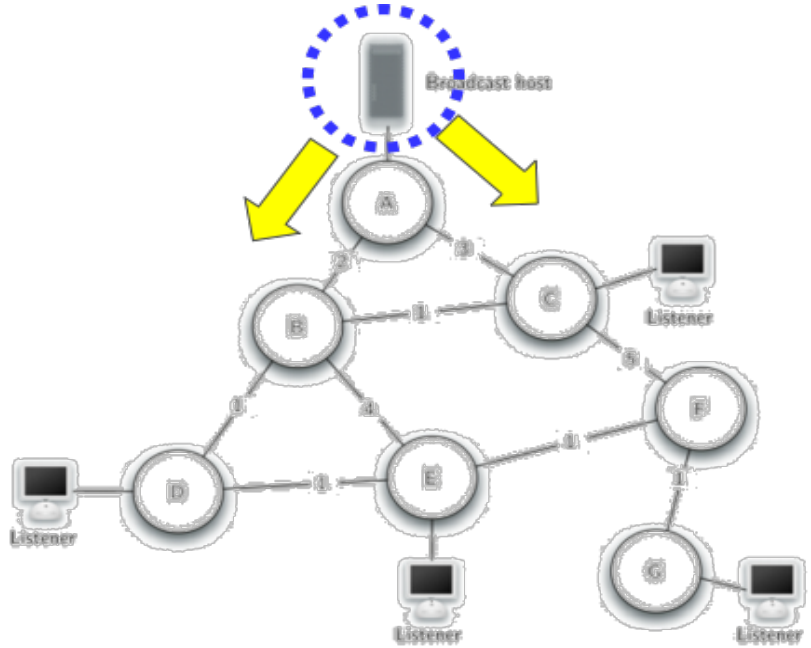
本算法涉及到在互联网中网游设计者和网络收音机所面临的问题：信息广播问题

Online games need to let all players know where other players are

网游需要让所有玩家获知其他玩家所在的位置

The radio needs to make the live audio data available to all listeners

收音机则需要让所有听众获取直播的音频数据



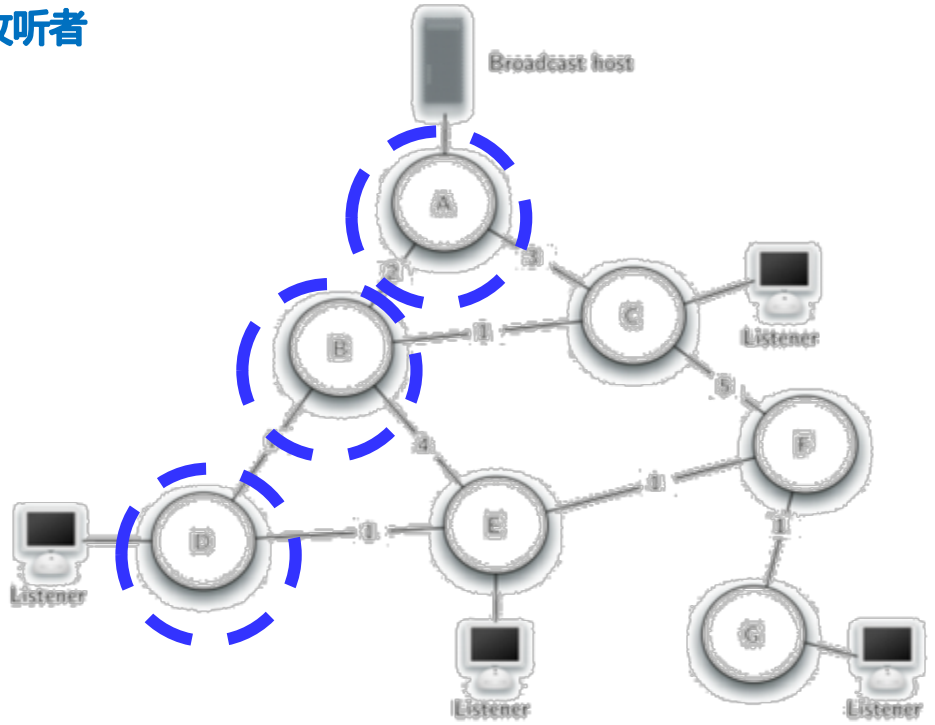
Information broadcasting problem: unicast solution
信息广播问题：单播解法

The simplest solution to the information broadcasting problem is that the broadcast source maintains a list of listeners and sends each message to each listener once

信息广播问题最简单的解法是由广播源维护一个收听者的列表，将每条消息向每个收听者发送一次

As shown in the figure, each message will be sent 4 times, and each message will use the shortest path algorithm to reach the listener

如图，每条消息会被发送4次，每个消息都采用最短路径算法到达收听者

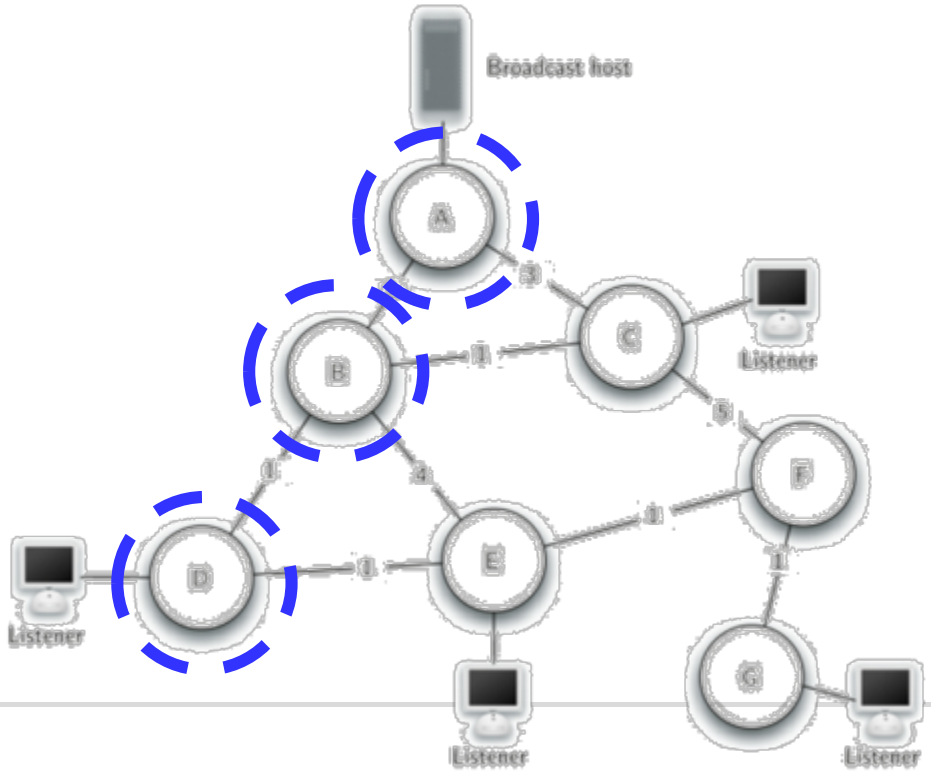


Information broadcasting problem: unicast solution

信息广播问题：单播解法

Router A will process the same message 4 times, C will only process it once; while B/D is on the shortest path of the other 3 listeners, each will process and forward the same message 3 times, which will generate a lot of extra stream

路由器A会处理4次相同消息，C仅会处理1次；而B/D位于其它3个收听者的最短路径上，则各会处理转发3次相同消息会产生许多额外流量



Information Broadcasting Problem: Flood Solution

信息广播问题：洪水解法

The brutal solution to the information broadcast problem is to spread each message between routers. All routers forward the received message to their adjacent routers and listeners. Obviously, if there are no restrictions, this method will cause network flooding disaster, as many routers and listeners keep getting the same message over and over again, never stop!

信息广播问题的暴力解法，是将每条消息在路由器间散布出去所有的路由器都将收到的消息转发到自己相邻的路由器和收听者，显然，如果没有任何限制，这个方法将造成网络洪水灾难，很多路由器和收听者会不断重复收到相同的消息，永不停止！

Information Broadcasting Problem: Flood Solution

信息广播问题：洪水解法

Therefore, the flood solution generally adds a life value (TTL: TimeToLive) to each message, and the initial setting is the distance from the message source to the farthest listener;

所以，洪水解法一般会给每条消息附加一个生命值(TTL:TimeToLive)，初始设置为从消息源到最远的收听者的距离；

Each router receives a message, if its TTL value is greater than 0, it reduces the TTL by 1, and then forwards it out

每个路由器收到一条消息，如果其TTL值大于0，则将TTL减少1，再转发出去

If the TTL is equal to 0, the message is discarded directly.

如果TTL等于0了，则就直接抛弃这个消息。

The TTL setting prevents disasters, but this flood solution obviously cause more stream than the aforementioned unicast method does

TTL的设置防止了灾难发生，但这种洪水解法显然比前述的单播方法所产生的流量还要大。

Information Broadcasting Problem: Minimum Spanning Tree
信息广播问题：最小生成树

The optimal solution to the information broadcast problem relies on selecting the minimum weight spanning tree on the router graph.

信息广播问题的最优解法，依赖于路由器关系图上选取具有最小权重的生成树

Spanning Tree: A subgraph that has all the vertices in the graph and the minimum number of edges to keep it connected.

生成树：拥有图中所有的顶点和最少数量的边，以保持连通的子图。

The minimum spanning tree T of a graph G(V,E) is defined as

图G(V,E)的最小生成树T，定义为

A circle-free subset that contains all vertices V and E, with the smallest sum of edge weights

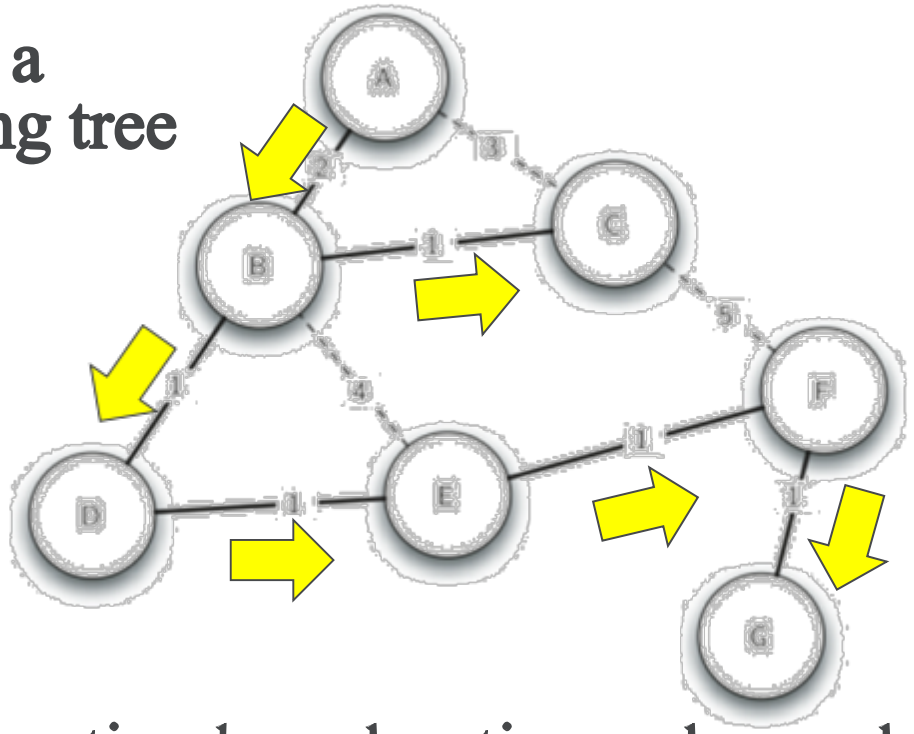
包含所有顶点V，以及E的无圈子集，并且边权重之和最小

Information Broadcasting Problem: Minimum Spanning Tree

信息广播问题：最小生成树

The figure shows a minimum spanning tree

图为一个最小生成树



In this way, information broadcasting only needs to start from A and propagate down the path level of the tree.

这样信息广播就只需要从A开始，沿着树的路径层次向下传播

It can be achieved that each router only needs to process a message once, and the total cost is minimal.

就可以达到每个路由器只需要处理1次消息，同时总费用最小。

Minimum Spanning Tree: Prim Algorithm
最小生成树：Prim算法

The Prim algorithm which solves the minimum spanning tree problem belongs to the "**greedy algorithm**" with each step searching forward along the edge of the minimum weight.

解决最小生成树问题的Prim算法，属于“**贪心算法**”，即每步都沿着最小权重的边向前搜索。

The idea of constructing a minimum spanning tree is very simple. If T is not a spanning tree, then do it repeatedly:

构造最小生成树的思路很简单，如果T还不是生成树，则反复做：

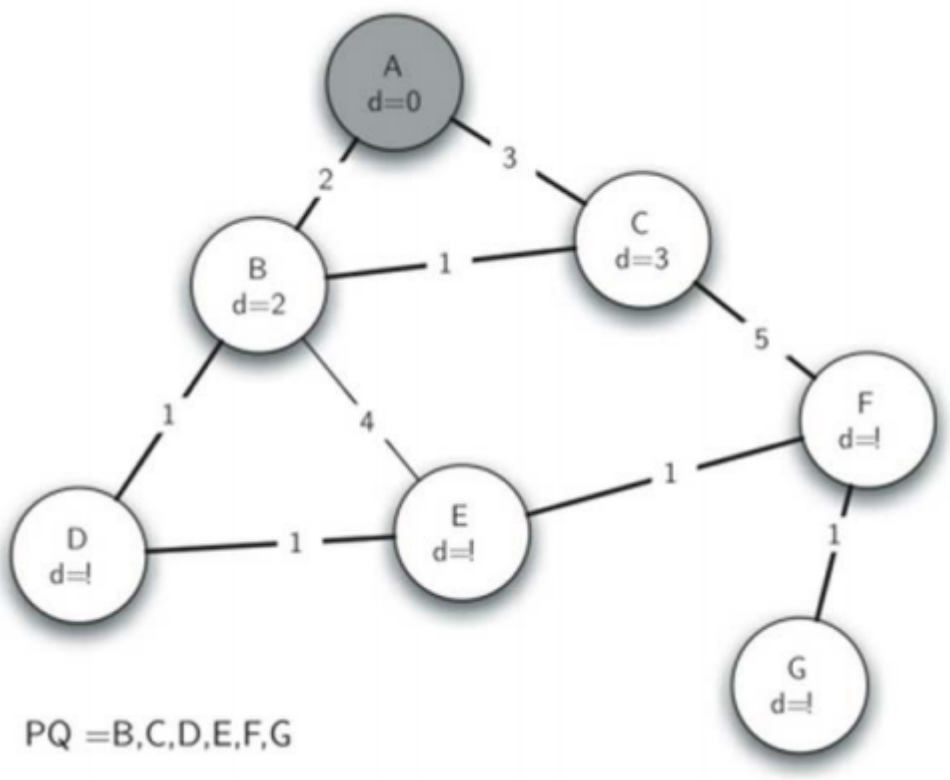
> Find an edge with minimum weight that can be safely added, add the edge to the tree T
>
> 找到一条最小权重的可以安全添加的边，将边添加到树T

An edge that is "**safe to add**", defined as one whose vertex is in the tree at one end and not in the tree at the other, in order to maintain the acyclic nature of the tree

“**可以安全添加**”的边，定义为一端顶点在树中，另一端不在树中的边，以便保持树的无圈特性

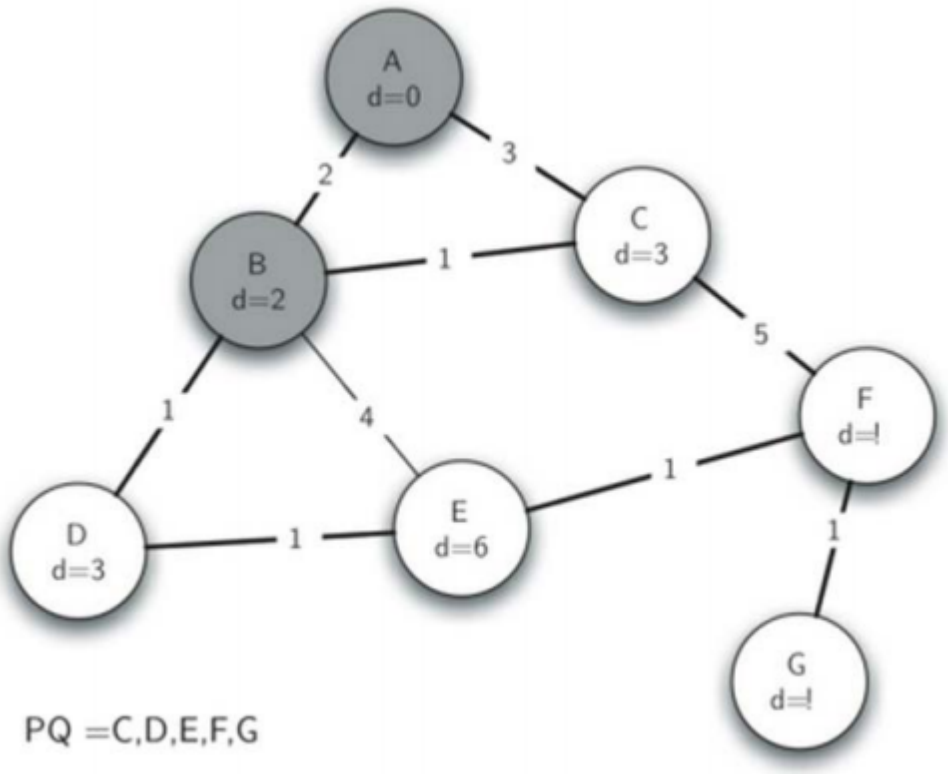
Minimum Spanning Tree: Example of Prim Algorithm

最小生成树：Prim算法示例



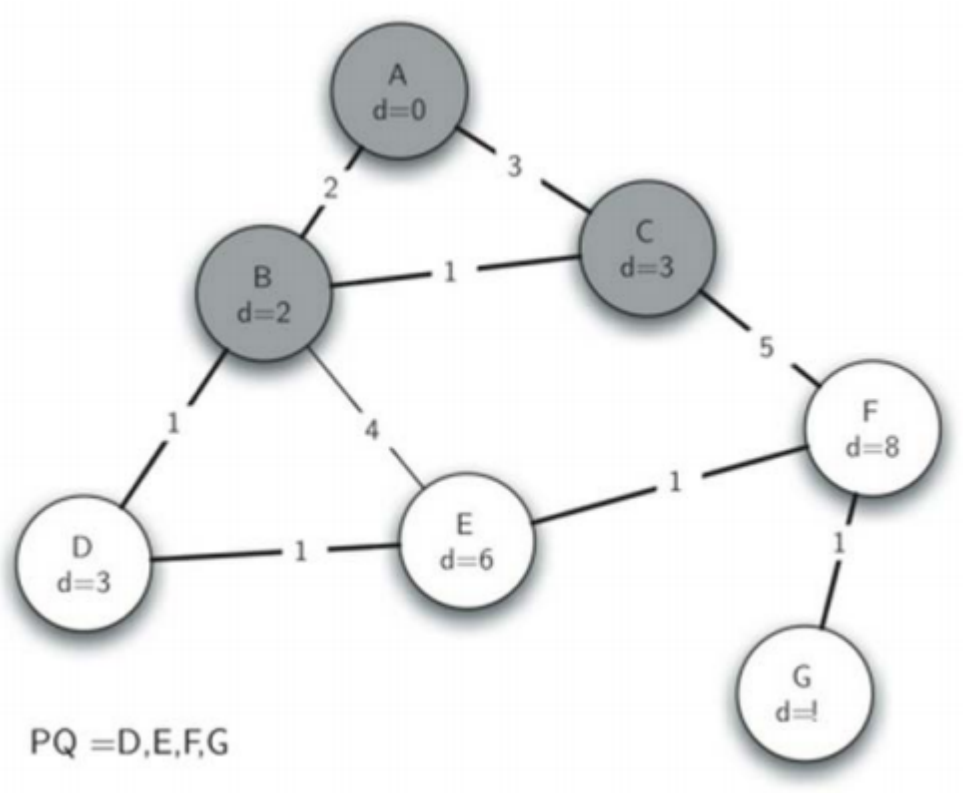
Minimum Spanning Tree: Example of Prim Algorithm

最小生成树：Prim算法示例



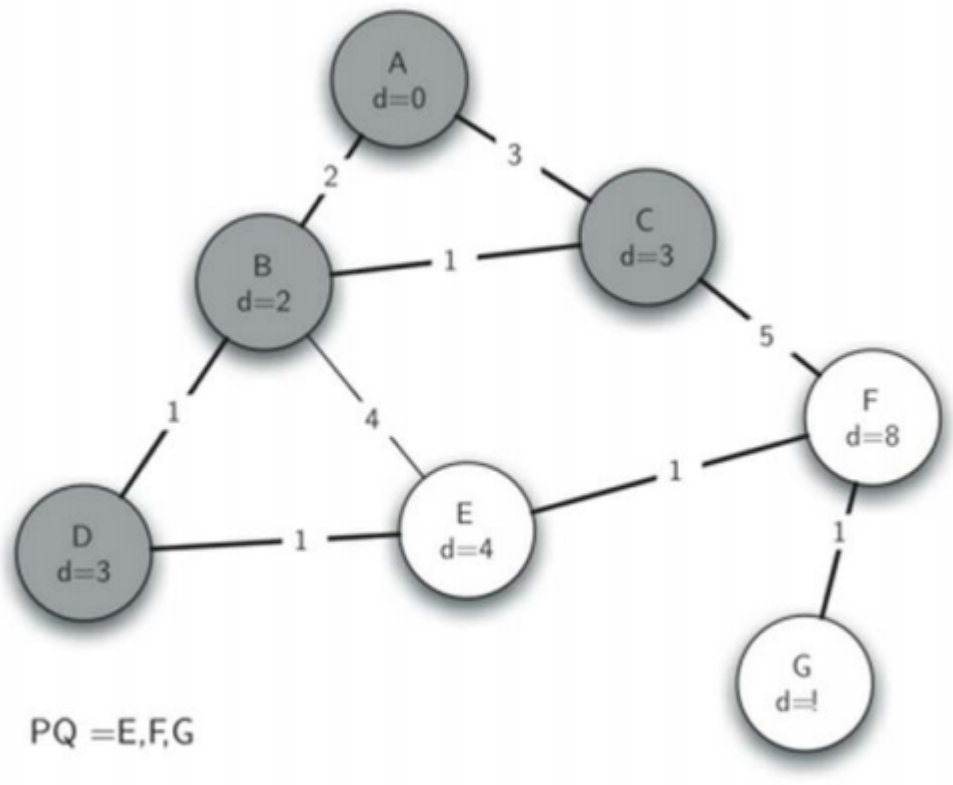
Minimum Spanning Tree: Example of Prim Algorithm

最小生成树：Prim算法示例



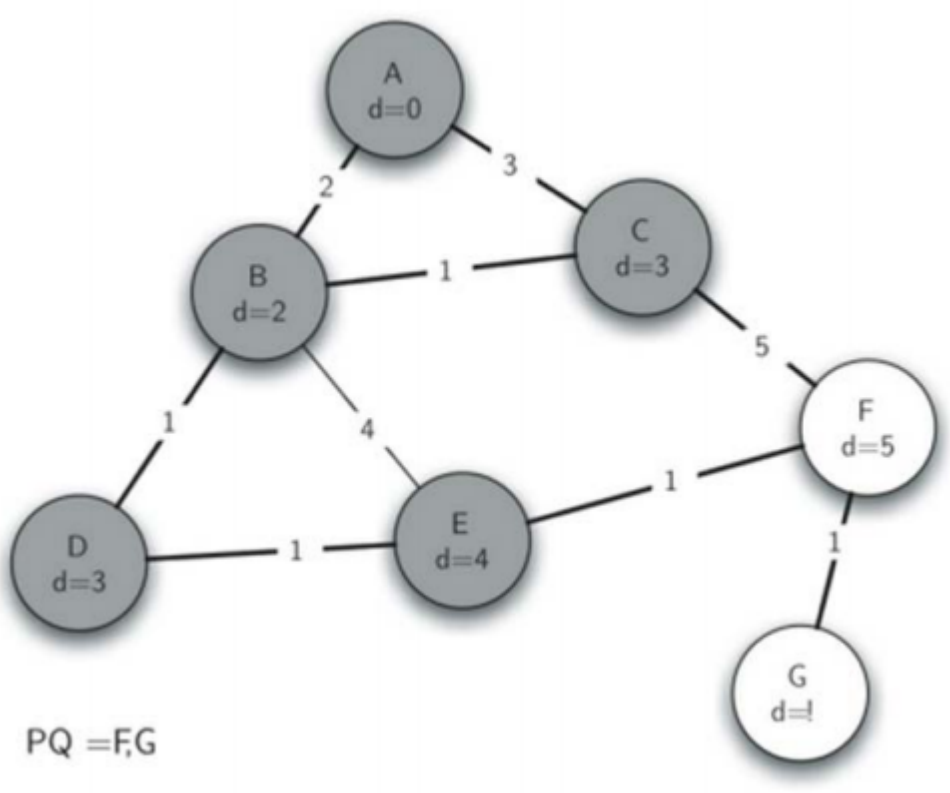
Minimum Spanning Tree: Example of Prim Algorithm

最小生成树：Prim算法示例



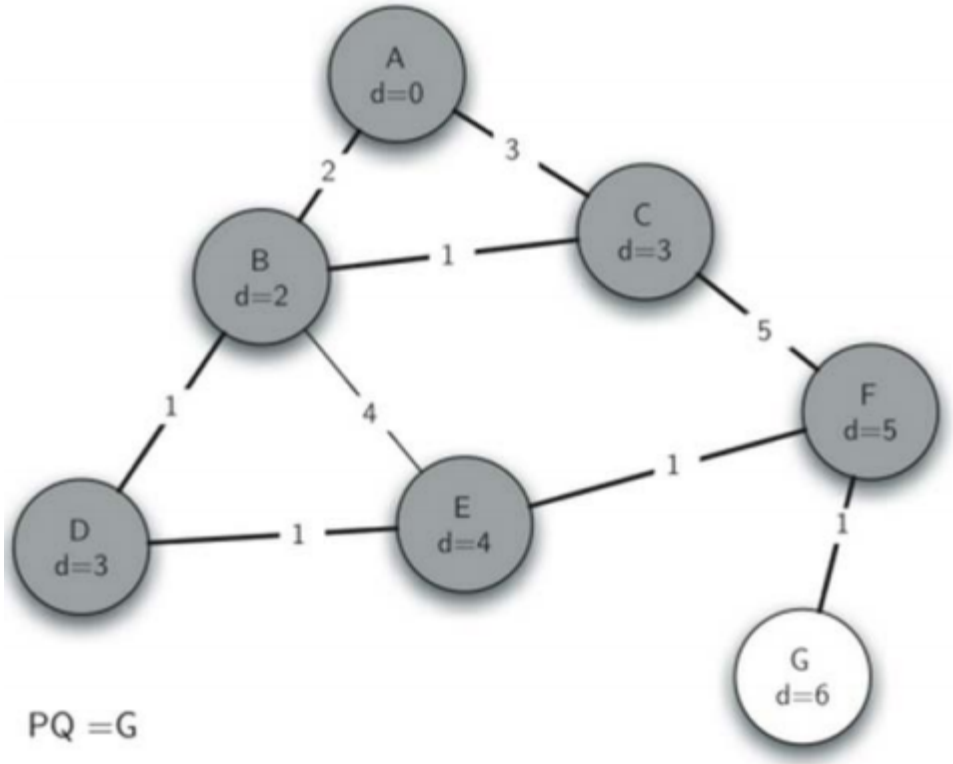
Minimum Spanning Tree: Example of Prim Algorithm

最小生成树：Prim算法示例



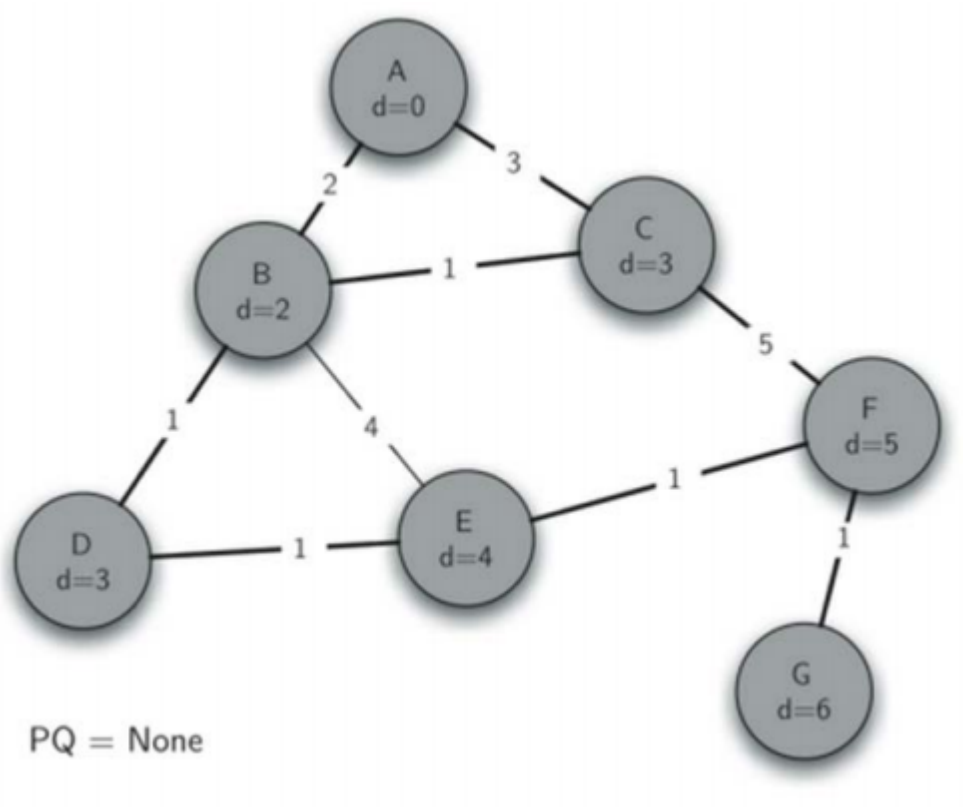
Minimum Spanning Tree: Example of Prim Algorithm

最小生成树：Prim算法示例



Minimum Spanning Tree: Example of Prim Algorithm

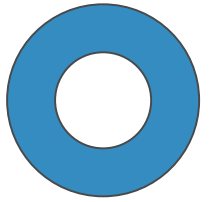
最小生成树：Prim算法示例



prim algorithm: minimum spanning tree
prim算法：最小生成树

```
from pythonds.graphs import PriorityQueue, Graph, Vertex
import sys


def prim(G, start):
    pq = PriorityQueue()
    for v in G:
        v.setDistance(sys.maxsize)
        v.setPred(None)
    start.setDistance(0)
    pq.buildHeap([(v.getDistance(), v) for v in G])
    while not pq.isEmpty():
        currentVert = pq.delMin()
        for nextVert in currentVert.getConnections():
            newCost = currentVert.getWeight(nextVert)
            if nextVert in pq and newCost < nextVert.getDistance():
                nextVert.setPred(currentVert)
                nextVert.setDistance(newCost)
                pq.decreaseKey(nextVert, newCost)
```



Chapter Summary
本章总结

Learned the graph abstract data type and several implementation methods, and discussed some graph algorithms and applications

学习了图抽象数据类型以及若干实现方法，讨论了一些图的算法和应用

①Breadth-first search algorithm BFS to solve the shortest path problem of unweighted graphs (word ladder problem);

广度优先搜索算法BFS，解决无权图的最短路径问题(词梯问题)；

②Dijkstra's algorithm for shortest paths in weighted graphs;

带权图最短路径的Dijkstra算法；

③Depth-first search algorithm DFS for graphs (knight tour problem);

图的深度优先搜索算法DFS(骑士周游问题)；

④A strongly connected branch algorithm for simplified graphs;

用于简化图的强连通分支算法；

⑤Topological sorting algorithm for sorting related tasks;

用于关联任务排序的拓扑排序算法；

⑥Minimum spanning tree algorithm for broadcast messages.

用于广播消息的最小生成树算法。