



Part4：图算法

- 图搜索策略
- 最短路径问题
- 拓扑排序
- 最大流问题

图

图(Graph)是一种比线性表和树更为复杂的数据结构。

线性结构：是研究数据元素之间的一对一关系。在这种结构中，除第一个和最后一个元素外，任何一个元素都有唯一的一个直接前驱和直接后继。

树结构：是研究数据元素之间的一对多的关系。在这种结构中，每个元素对下(层)可以有0个或多个元素相联系，对上(层)只有唯一的一个元素相关，数据元素之间有明显的层次关系。

图

图结构：是研究数据元素之间的多对多的关系。在这种结构中，任意两个元素之间可能存在关系。即结点之间的关系可以是任意的，图中任意元素之间都可能相关。

图的应用极为广泛，已渗入到诸如语言学、逻辑学、物理、化学、电讯、计算机科学以及数学的其它分支。

图的定义和术语

一个图(G)定义为一个偶对(V, E)，记为 $G=(V, E)$ 。其中： V 是顶点(Vertex)的非空有限集合，记为 $V(G)$ ； E 是无序集 $V \& V$ 的一个子集，记为 $E(G)$ ，其元素是图的弧(Arc)。

将顶点集合为空的图称为空图。其形式化定义为：

$$G=(V, E)$$

$$V=\{v | v \in \text{data object}\}$$

$$E=\{<v, w> | v, w \in V \wedge p(v, w)\}$$

$p(v, w)$ 表示从顶点 v 到顶点 w 有一条直接通路。

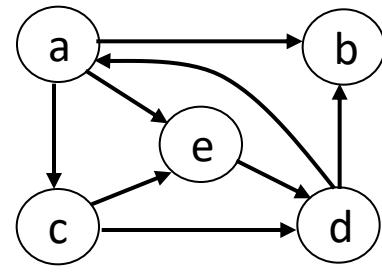


弧(Arc)：表示两个顶点 v 和 w 之间存在一个关系，用顶点偶对 $\langle v, w \rangle$ 表示。通常根据图的顶点偶对将图分为有向图和无向图。

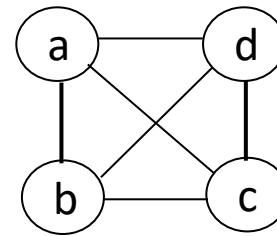
有向图(Digraph)：若图 G 的关系集合 $E(G)$ 中，顶点偶对 $\langle v, w \rangle$ 的 v 和 w 之间是有序的，称图 G 是有向图。

在有向图中，若 $\langle v, w \rangle \in E(G)$ ，表示从顶点 v 到顶点 w 有一条弧。其中： v 称为弧尾(tail)或始点(initial node)， w 称为弧头(head)或终点(terminal node)。

无向图(Undigraph)：若图 G 的关系集合 $E(G)$ 中，顶点偶对 $\langle v, w \rangle$ 的 v 和 w 之间是无序的，称图 G 是无向图。



(a) 有向图G1



(b) 无向图G2

图的示例

完全无向图：对于无向图，若图中顶点数为n，用e表示边的数目，则 $e \in [0, n(n-1)/2]$ 。具有 $n(n-1)/2$ 条边的无向图称为完全无向图。

完全无向图另外的定义是：

对于无向图 $G=(V, E)$ ，若 $\forall v_i, v_j \in V$ ，当 $v_i \neq v_j$ 时，有 $(v_i, v_j) \in E$ ，即图中任意两个不同的顶点间都有一条无向边，这样的无向图称为完全无向图。

完全有向图: 对于有向图，若图中顶点数为 n ，用 e 表示弧的数目，则 $e \in [0, n(n-1)]$ 。具有 $n(n-1)$ 条边的有向图称为完全有向图。

完全有向图另外的定义是：

对于有向图 $G=(V, E)$ ，若 $\forall v_i, v_j \in V$ ，当 $v_i \neq v_j$ 时，有 $\langle v_i, v_j \rangle \in E \wedge \langle v_j, v_i \rangle \notin E$ ，即图中任意两个不同的顶点间都有一条弧，这样的有向图称为**完全有向图**。

有很多边或弧的图 ($e < n \log n$) 的图称为**稀疏图**，反之称为**稠密图**。

权(Weight): 与图的边和弧相关的数。权可以表示从一个顶点到另一个顶点的距离或耗费。



顶点的邻接(Adjacent): 对于无向图 $G=(V, E)$, 若边 $(v, w) \in E$, 则称顶点 v 和 w 互为邻接点, 即 v 和 w 相邻接。边 (v, w) 依附(**incident**)与顶点 v 和 w 。

对于有向图 $G=(V, E)$, 若有向弧 $\langle v, w \rangle \in E$, 则称顶点 v “邻接到”顶点 w , 顶点 w “邻接自”顶点 v , 弧 $\langle v, w \rangle$ 与顶点 v 和 w “相关联”。

顶点的度、入度、出度: 对于无向图 $G=(V, E)$, $\forall v_i \in V$, 图 G 中依附于 v_i 的边的数目称为顶点 v_i 的度(**degree**), 记为 $TD(v_i)$ 。

显然，在无向图中，所有顶点度的和是图中边的2倍。即
 $\sum TD(v_i) = 2e \quad i=1, 2, \dots, n$ ， e 为图的边数。

对有向图 $G=(V, E)$ ，若 $\forall v_i \in V$ ，图 G 中以 v_i 作为起点的有向边(弧)的数目称为顶点 v_i 的 **出度(Outdegree)**，记为 $OD(v_i)$ ；以 v_i 作为终点的有向边(弧)的数目称为顶点 v_i 的 **入度(Indegree)**，记为 $ID(v_i)$ 。顶点 v_i 的出度与入度之和称为 v_i 的 **度**，记为 $TD(v_i)$ 。即

$$TD(v_i) = OD(v_i) + ID(v_i)$$

路径(Path)、路径长度、回路：对无向图 $G=(V, E)$ ，若从顶点 v_i 经过若干条边能到达 v_j ，称顶点 v_i 和 v_j 是 **连通的**，又称顶点 v_i 到 v_j 有 **路径**。

对有向图 $G=(V, E)$ ，从顶点 v_i 到 v_j 有 **有向路径**，指的是从顶点 v_i 经过若干条有向边(弧)能到达 v_j 。

连通图、图的连通分量: 对无向图 $G=(V, E)$, 若 $\forall v_i, v_j \in V$, v_i 和 v_j 都是连通的, 则称图 G 是**连通图**, 否则称为**非连通图**。若 G 是非连通图, 则**极大的连通子图**称为 G 的连通分量。

对有向图 $G=(V, E)$, 若 $\forall v_i, v_j \in V$, 都有**以 v_i 为起点, v_j 为终点**以及**以 v_j 为起点, v_i 为终点**的有向路径, 称图 G 是**强连通图**, 否则称为**非强连通图**。若 G 是非强连通图, 则**极大的强连通子图**称为 G 的强连通分量。

“极大”的含义: 指的是对子图再增加图 G 中的其它顶点, 子图就不再连通。



图的存储结构

图的存储结构比较复杂，其复杂性主要表现在：

- ◆ 任意顶点之间可能存在联系，无法以数据元素在存储区中的物理位置来表示元素之间的关系。
- ◆ 图中顶点的度不一样，有的可能相差很大，若按度数最大的顶点设计结构，则会浪费很多存储单元，反之按每个顶点自己的度设计不同的结构，又会影响操作。

图的常用的存储结构有：**邻接矩阵**、**邻接链表**、**十字链表**、**邻接多重表**和**边表**。



邻接矩阵(数组)表示法

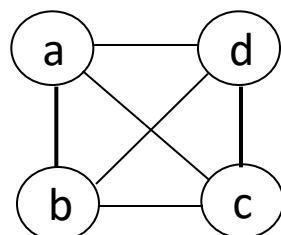
基本思想：对于有n个顶点的图，用一维数组vexs[n]存储顶点信息，用二维数组A[n][n]存储顶点之间关系的信息。该二维数组称为**邻接矩阵**。在邻接矩阵中，以顶点在vexs数组中的下标代表顶点，邻接矩阵中的元素A[i][j]存放的是顶点i到顶点j之间关系的信息。

1 无向图的数组表示

(1) 无权图的邻接矩阵

无向无权图 $G=(V, E)$ 有 $n(n \geq 1)$ 个顶点，其邻接矩阵是 n 阶对称方阵，如图7-5所示。其元素的定义如下：

$$A[i][j] = \begin{cases} 1 & \text{若 } (v_i, v_j) \in E, \text{ 即 } v_i, v_j \text{ 邻接} \\ 0 & \text{若 } (v_i, v_j) \notin E, \text{ 即 } v_i, v_j \text{ 不邻接} \end{cases}$$



(a) 无向图

vexs
a
b
c
d

(b) 顶点矩阵

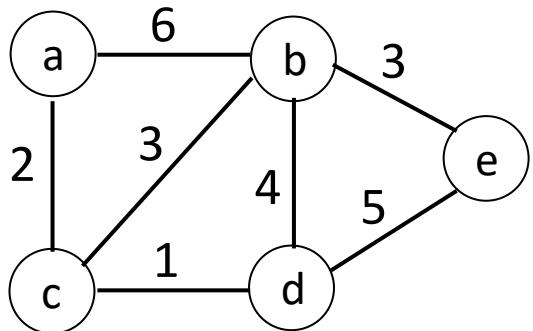
$$\left(\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array} \right)$$

(c) 邻接矩阵

(2) 带权图的邻接矩阵

无向带权图 $G = (V, E)$ 的邻接矩阵如图7-6所示。其元素的定义如下：

$$A[i][j] = \begin{cases} w_{ij} & \text{若 } (v_i, v_j) \in E, \text{ 即 } v_i, v_j \text{ 邻接, 权值为 } w_{ij} \\ \infty & \text{若 } (v_i, v_j) \notin E \} \text{ 即 } v_i, v_j \text{ 不邻接时} \end{cases}$$



(a) 带权无向图

vexs
a
b
c
d
e

(b) 顶点矩阵

$$\left(\begin{array}{ccccc} \infty & 6 & 2 & \infty & \infty \\ 6 & \infty & 3 & 4 & 3 \\ 2 & 3 & \infty & 1 & \infty \\ \infty & 4 & 3 & \infty & 5 \\ \infty & 3 & \infty & 5 & \infty \end{array} \right)$$

(c) 邻接矩阵

(3) 无向图邻接矩阵的特性

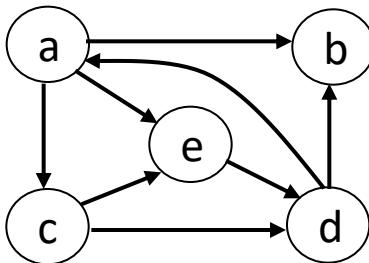
- ◆ 邻接矩阵是**对称方阵**；
- ◆ 对于顶点 v_i , 其**度数**是第*i*行的非0元素的个数；
- ◆ 无向图的**边数**是上(或下)三角形矩阵中非0元素个数。

2 有向图的数组表示

(1) 无权图的邻接矩阵

若有向无权图 $G=(V, E)$ 有 $n(n \geq 1)$ 个顶点，则其邻接矩阵是**n阶对称方阵**，如图7-7所示。元素定义如下：

$$A[i][j] = \begin{cases} 1 & \text{若 } \langle v_i, v_j \rangle \in E, \text{ 从 } v_i \text{ 到 } v_j \text{ 有弧} \\ 0 & \text{若 } \langle v_i, v_j \rangle \notin E \text{ 从 } v_i \text{ 到 } v_j \text{ 没有弧} \end{cases}$$



(a) 有向图

vexs
a
b
c
d
e

(b) 顶点矩阵

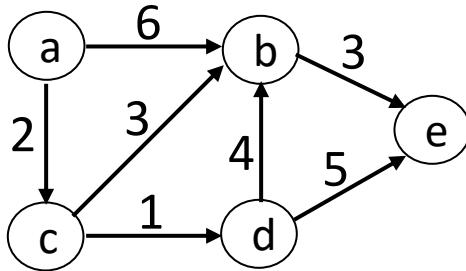
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(c) 邻接矩阵

(2) 带权图的邻接矩阵

有向带权图 $G=(V, E)$ 的邻接矩阵如图7-8所示。其元素的定义如下：

$$A[i][j] = \begin{cases} w_{ij} & \text{若 } \langle v_i, v_j \rangle \in E, \text{ 即 } v_i, v_j \text{ 邻接, 权值为 } w_{ij} \\ \infty & \text{若 } \langle v_i, v_j \rangle \notin E, \text{ 即 } v_i, v_j \text{ 不邻接时} \end{cases}$$



(a) 带权有向图

vexs	
a	
b	
c	
d	
e	

(b) 顶点矩阵

$$\left(\begin{array}{ccccc} \infty & 6 & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & 3 \\ \infty & 3 & \infty & 1 & \infty \\ \infty & 4 & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty \end{array} \right)$$

(c) 邻接矩阵

(3) 有向图邻接矩阵的特性

- ◆ 对于顶点 v_i , 第*i*行的非0元素的个数是其出度OD(v_i); 第*i*列的非0元素的个数是其入度ID(v_i)。
- ◆ 邻接矩阵中非0元素的个数就是图的弧的数目。

邻接链表法

基本思想：对图的每个顶点建立一个单链表，存储该顶点所有邻接顶点及其相关信息。每一个单链表设一个表头结点。

第*i*个单链表表示依附于顶点 v_i 的边(对有向图是以顶点 v_i 为头或尾的弧)。

1 结点结构与邻接链表示例

链表中的结点称为**表结点**，每个结点由三个域组成，。其中邻接点域(adjvex)指示与顶点 v_i 邻接的顶点在图中的位置(顶点编号)，链域(nextarc)指向下一个与顶点 v_i 邻接的表结点，数据域(info)存储和边或弧相关的信息，如权值等。对于无权图，如果没有与边相关的其他信息，可省略此域。

每个链表设一个表头结点(称为**顶点结点**)，由两个域组成。链域(firstarc)指向链表中的第一个结点，数据域(data)存储顶点名或其他信息。

表结点:

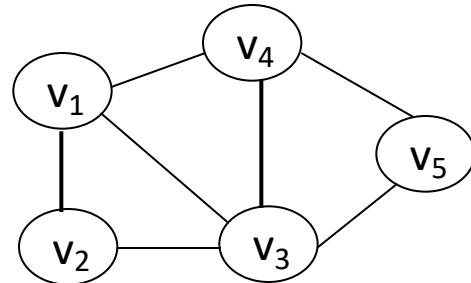
adjvex	info	nextarc
--------	------	---------

顶点结点:

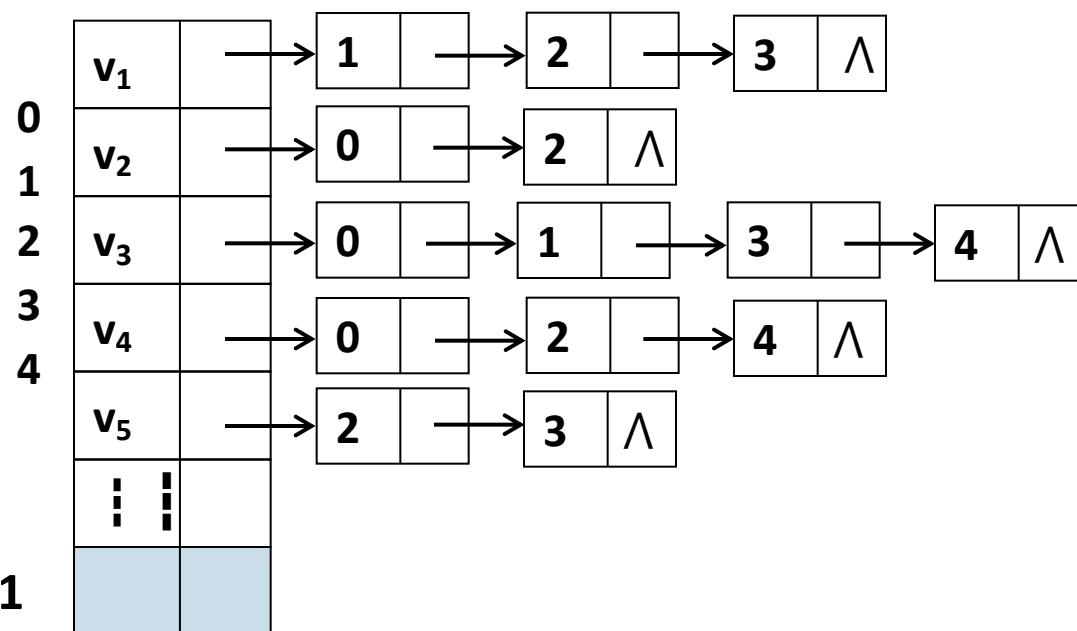
data	firstarc
------	----------

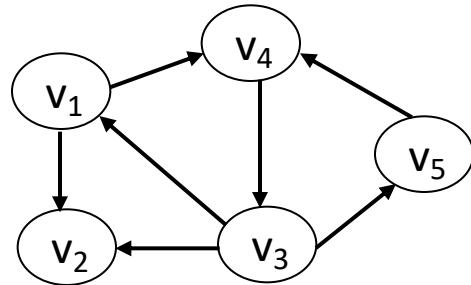
在图的邻接链表表示中，所有顶点结点用一个向量以顺序结构形式存储，可以随机访问任意顶点的链表，该向量称为表头向量，向量的下标指示顶点的序号。

用邻接链表存储图时，对无向图，其邻接链表是唯一的，对有向图，其邻接链表有两种形式

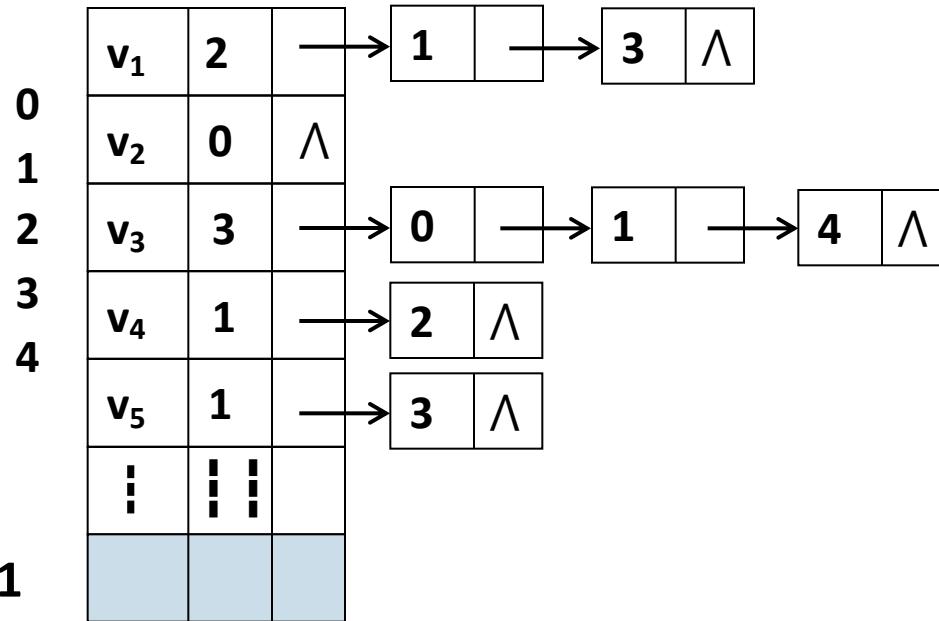


MAX_VEX-1

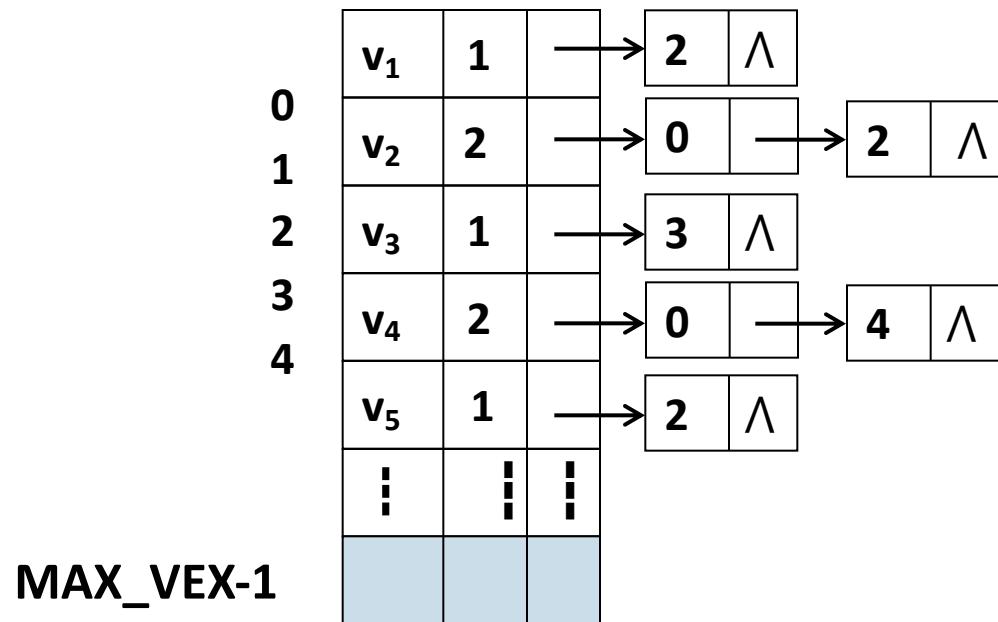




(a) 有向图



(b) 正邻接链表, 出度直观



(c) 逆邻接链表, 入度直观

2 邻接表法的特点

- ◆ 表头向量中每个分量就是一个单链表的头结点，分量个数就是图中的顶点数目；
- ◆ 在边或弧稀疏的条件下，用邻接表表示比用邻接矩阵表示节省存储空间；
- ◆ 在无向图，顶点 V_i 的度是第*i*个链表的结点数；
- ◆ 对有向图可以建立正邻接表或逆邻接表。正邻接表是以顶点 V_i 为出度(即为弧的起点)而建立的邻接表；逆邻接表是以顶点 V_i 为入度(即为弧的终点)而建立的邻接表；
- ◆ 在有向图中，第*i*个链表中的结点数是顶点 V_i 的出(或入)度；求入(或出)度，须遍历整个邻接表；



3.5 图搜索策略

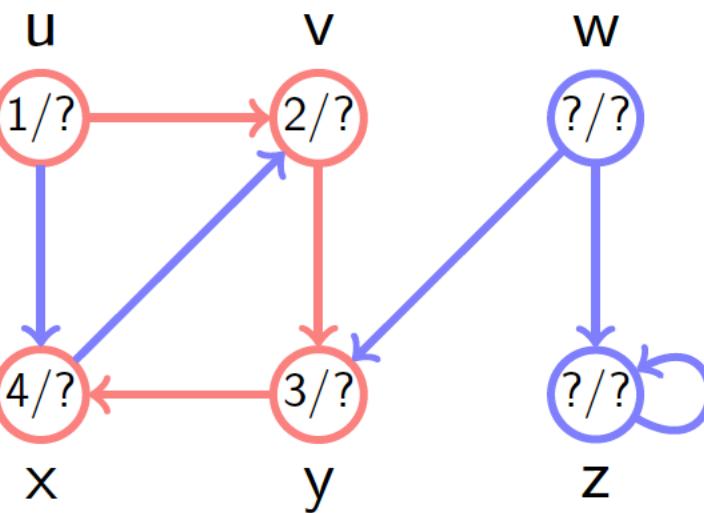
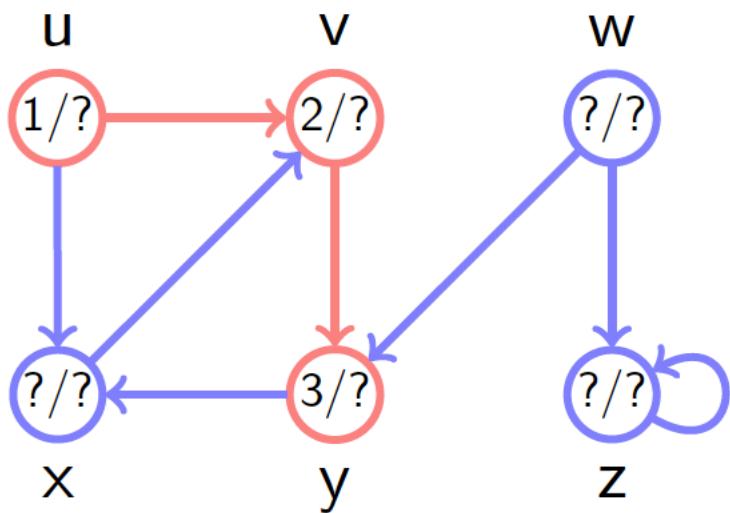
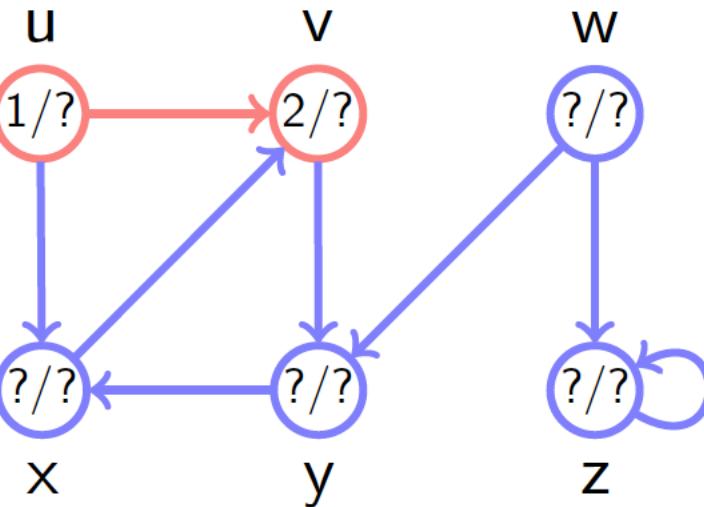
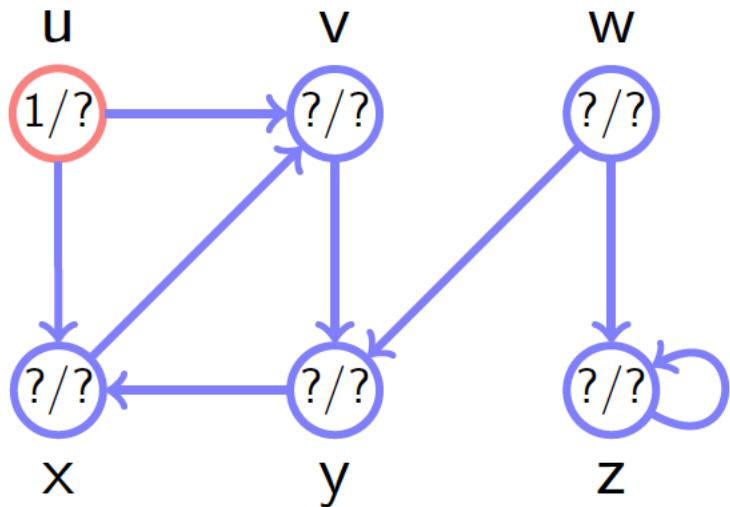
- 图的遍历
 - 从图的某一点出发访遍图中的其余顶点，且每个顶点仅被访问一次
- 深度优先查找
- 广度优先查找

3.5 图搜索策略——深度优先查找

算法流程

- ① 从任意顶点开始访问图的顶点，然后把该顶点标记为已访问；
- ② 每次迭代，算法接着处理与当前顶点邻接的未访问顶点（有若干顶点，任意选择一个）；
- ③ 重复步骤2，直到遇到一个终点（所有邻接顶点都已被访问）；
- ④ 在该终点上，算法沿着来路后退一条边，重复步骤2和3，并试着继续从那里访问未访问的顶点；
- ⑤ 重复步骤4，直到起始顶点变为一个终点。
- ⑥ 如果未访问的顶点仍然存在，继续从其中任意顶点开始，重复上述过程；

3.5 图搜索策略——深度优先查找实例

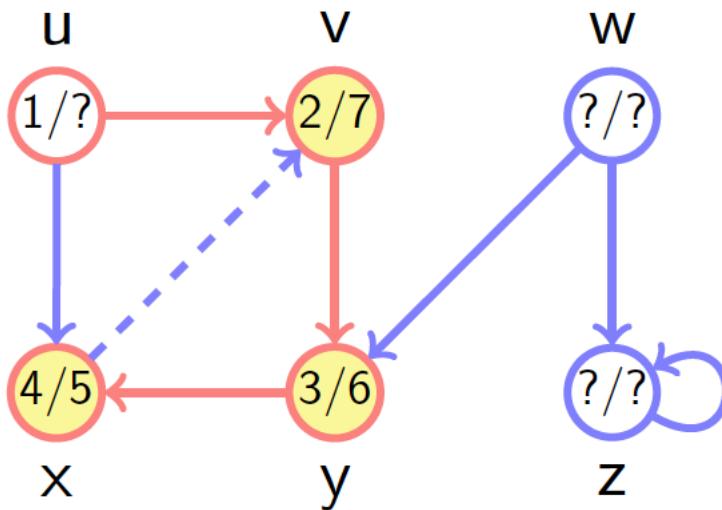
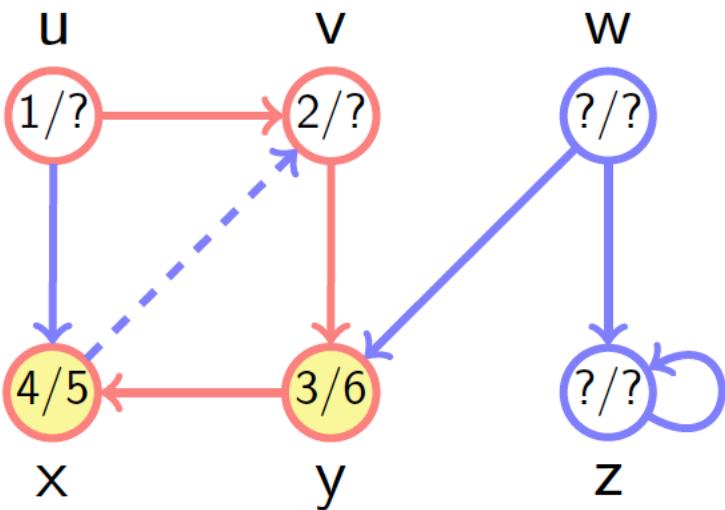
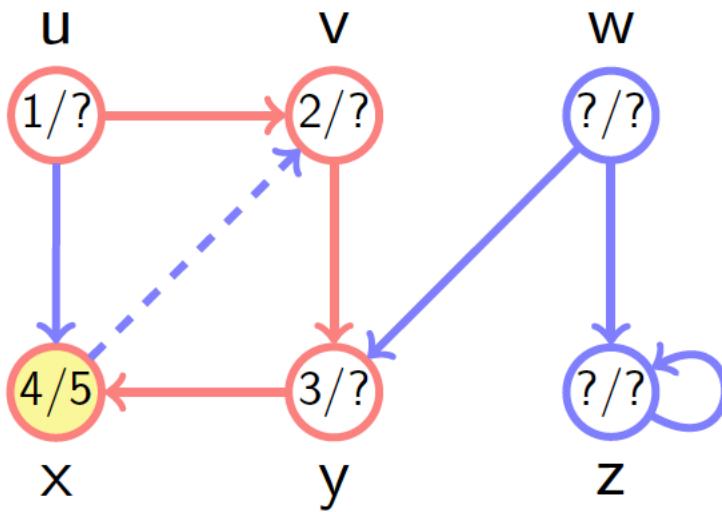
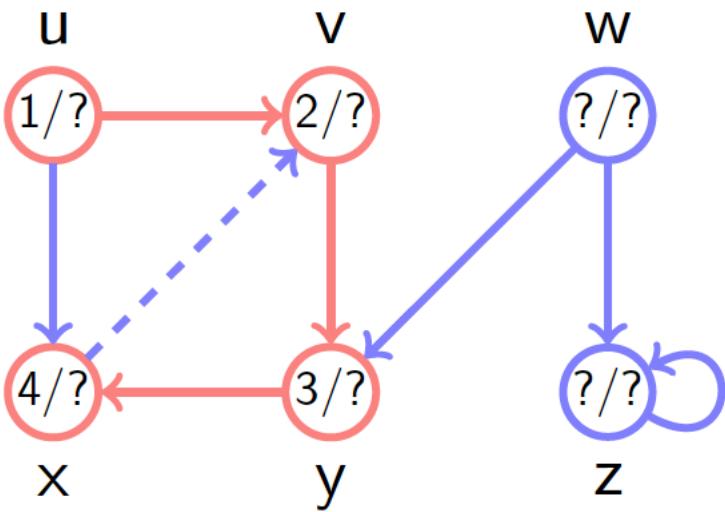


(s/t) 未发现节点

(s/t) 发现节点

(s/t) 完成节点 s:发现时间 t:完成时间

3.5 图搜索策略——深度优先查找实例

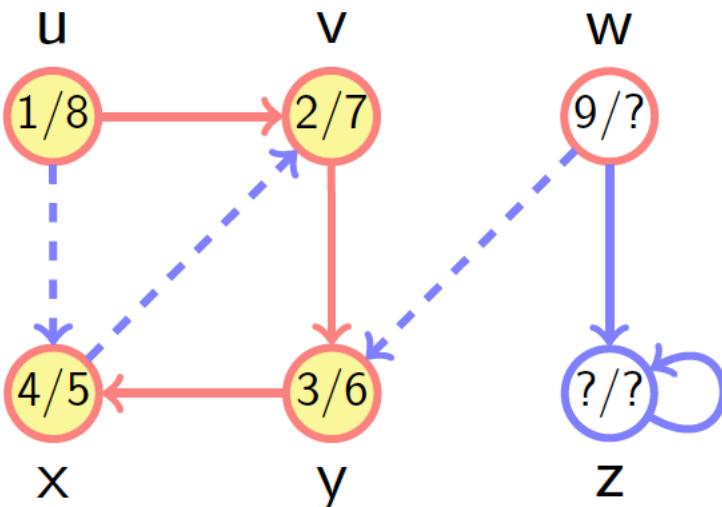
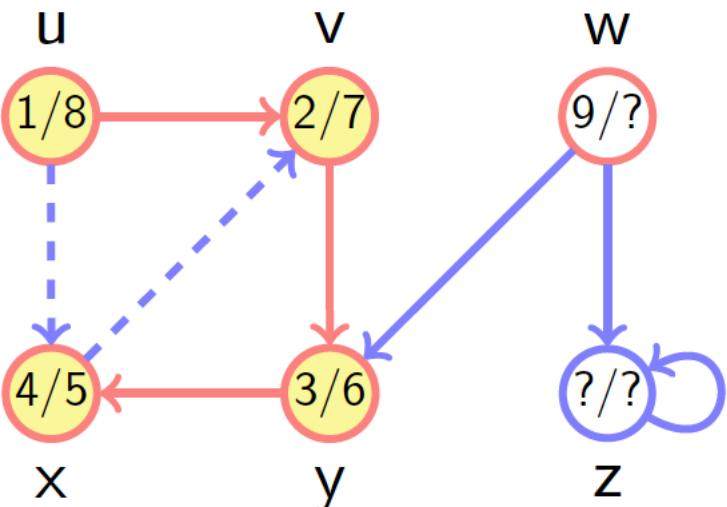
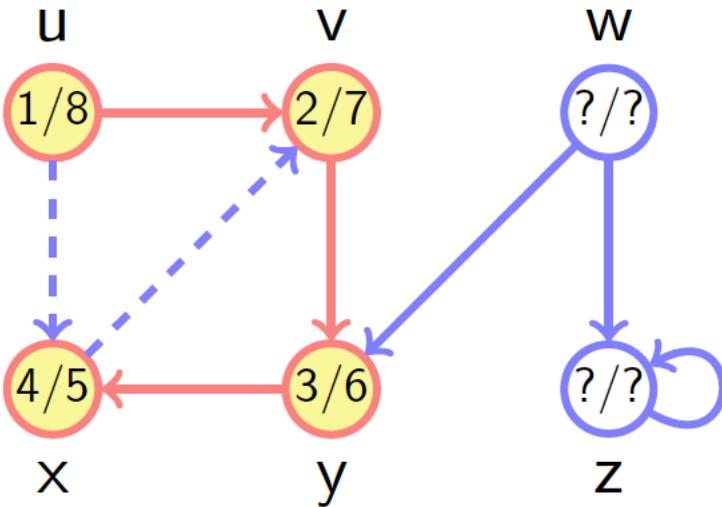
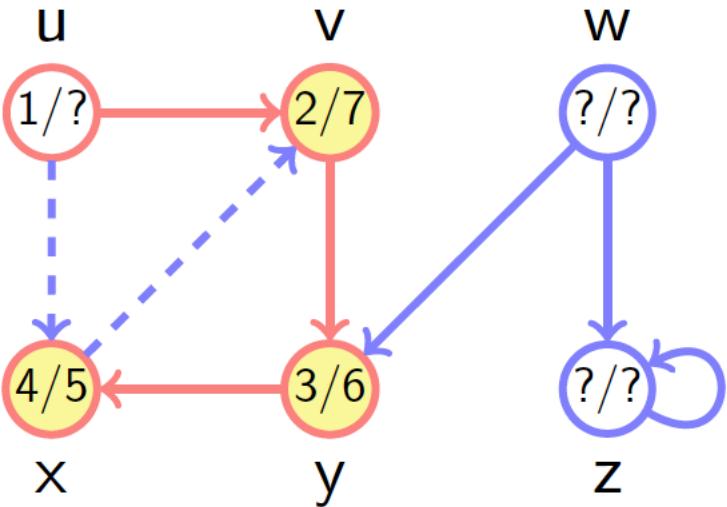


(s/t) 未发现节点

(s/t) 发现节点

(s/t) 完成节点 s:发现时间 t:完成时间

3.5 图搜索策略——深度优先查找实例

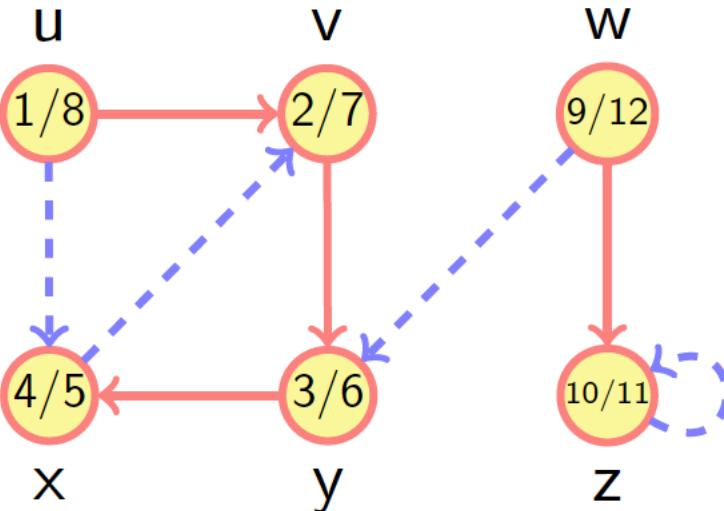
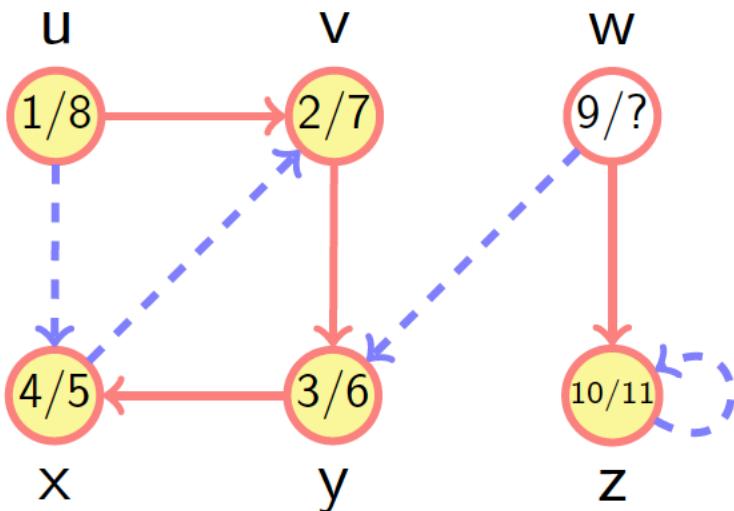
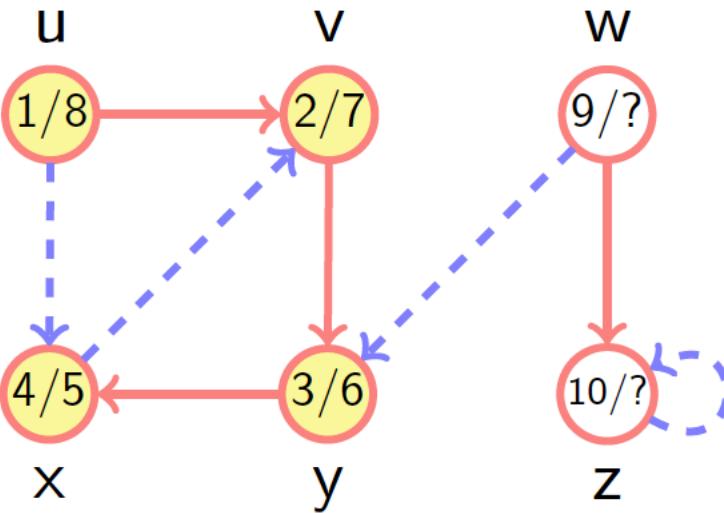
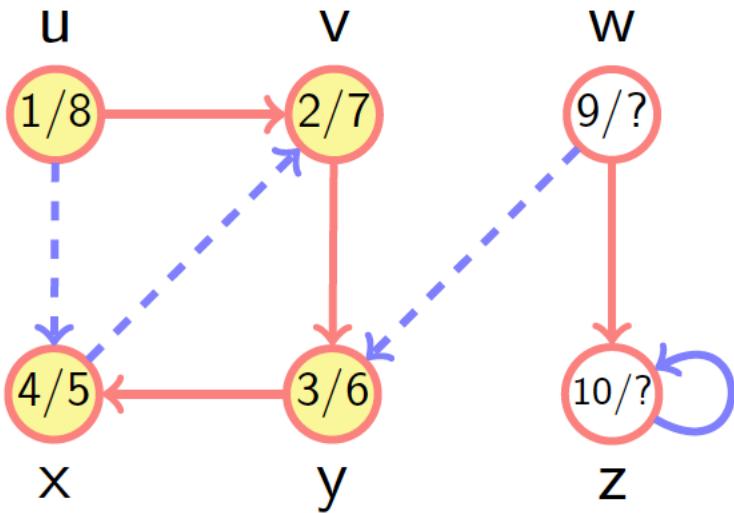


(s/t) 未发现节点

(s/t) 发现节点

(s/t) 完成节点 s:发现时间 t:完成时间

3.5 图搜索策略——深度优先查找实例

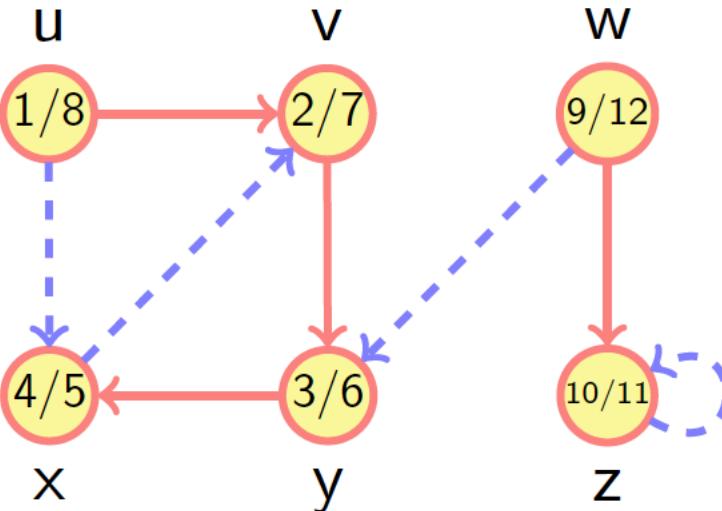
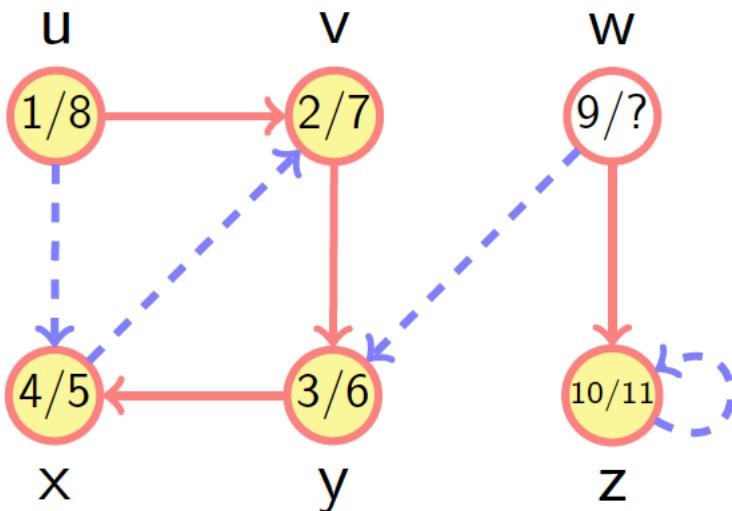
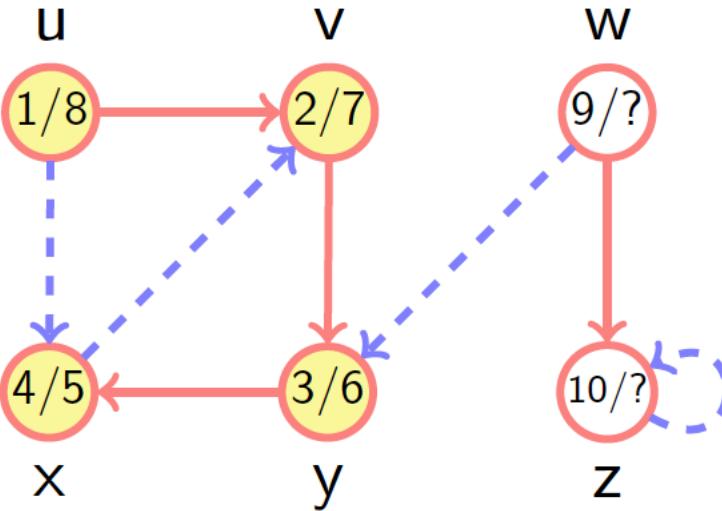
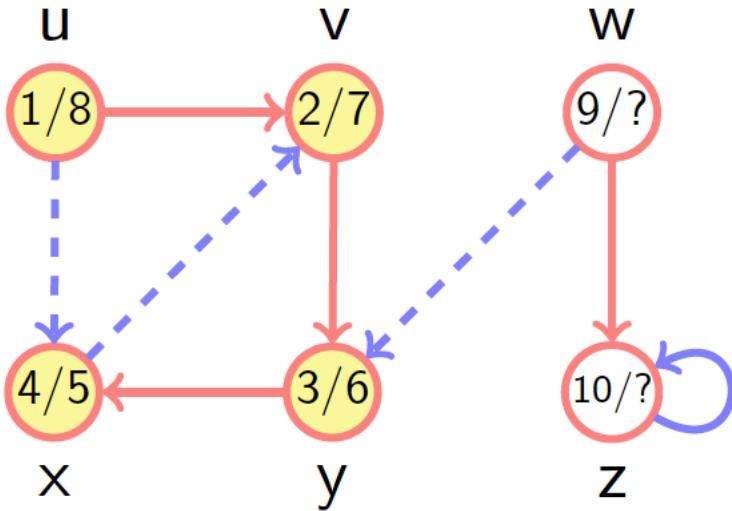


(s/t) 未发现节点

(s/t) 发现节点

(s/t) 完成节点 s:发现时间 t:完成时间

3.5 图搜索策略——深度优先查找时间



(s/t) 未发现节点

(s/t) 发现节点

(s/t) 完成节点 s:发现时间 t:完成时间

3.5 图搜索策略——深度优先查找时间效率

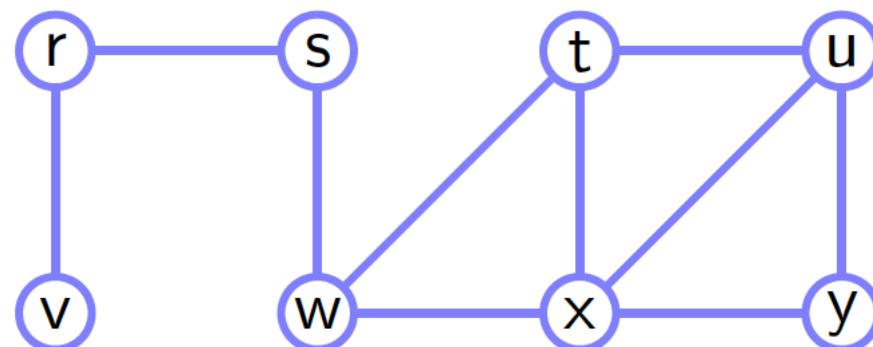
- 深度优先查找的效率与用来表示图的数据结构的规模成正比
 - 邻接矩阵表示法：遍历的时间效率属于 $\Theta(|V|^2)$
 - 邻接链表表示法：遍历的时间效率属于 $\Theta(|V| + |E|)$
- 使用栈
- 应用：检查图的连通性和无环性

3.5 图搜索策略——广度优先查找

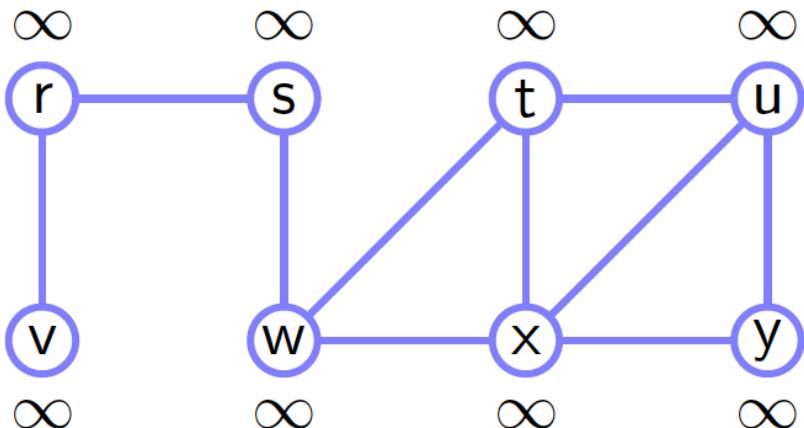
算法思想

首先访问和初始顶点邻接的顶点，然后是它两条边的所有未访问顶点。以此类推，直到所有与初始顶点连通的顶点都访问为止。如果未访问的顶点仍然存在，继续从其中任意顶点开始，重复此过程；

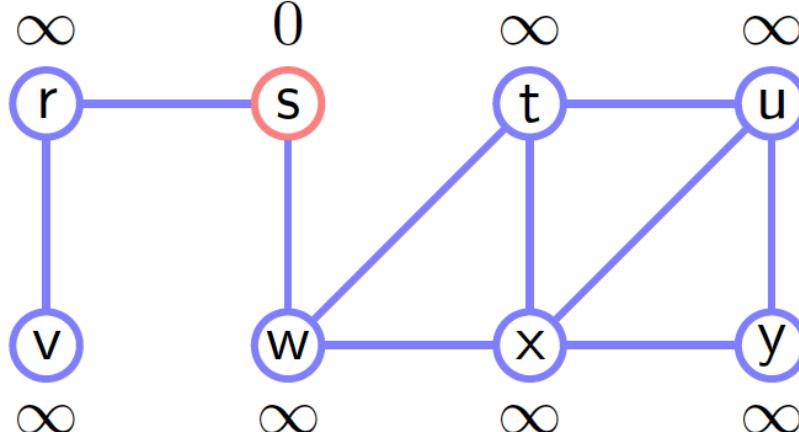
思考：如何利用广度优先搜寻如下的图？



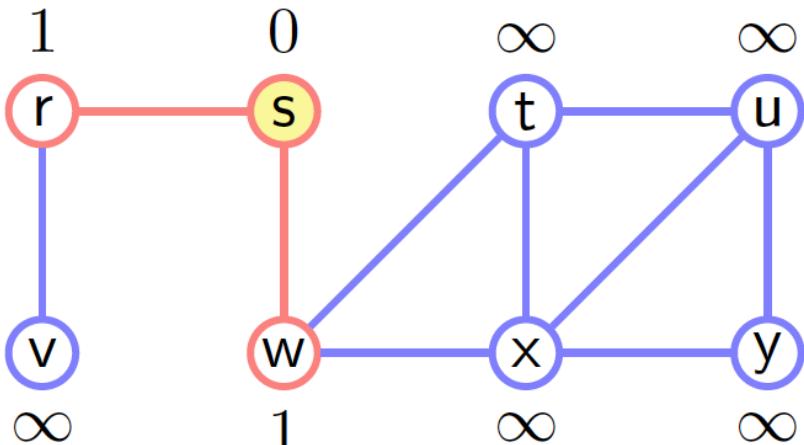
3.5 图搜索策略——广度优先查找实例



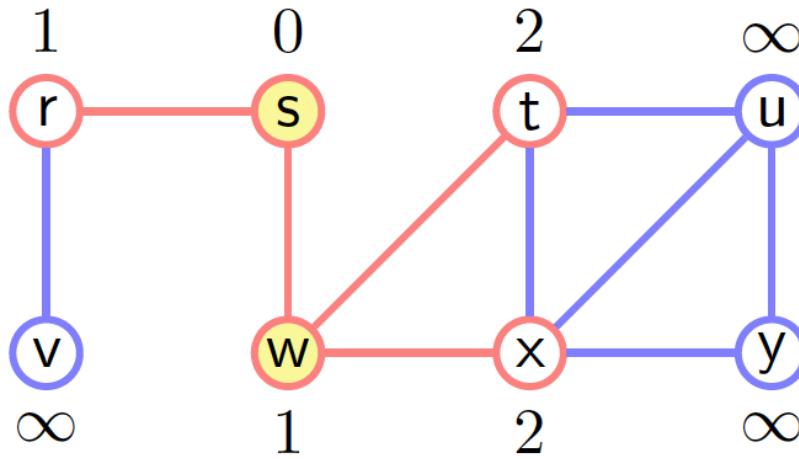
\emptyset



s

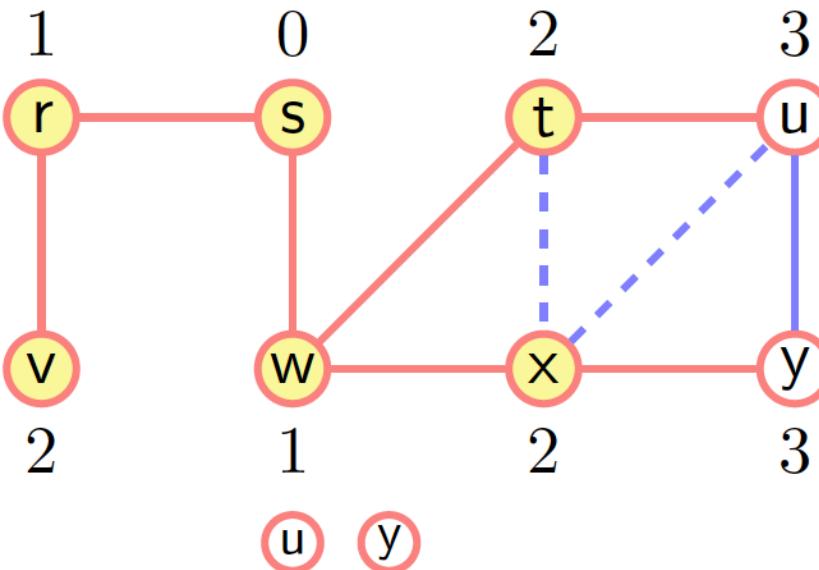
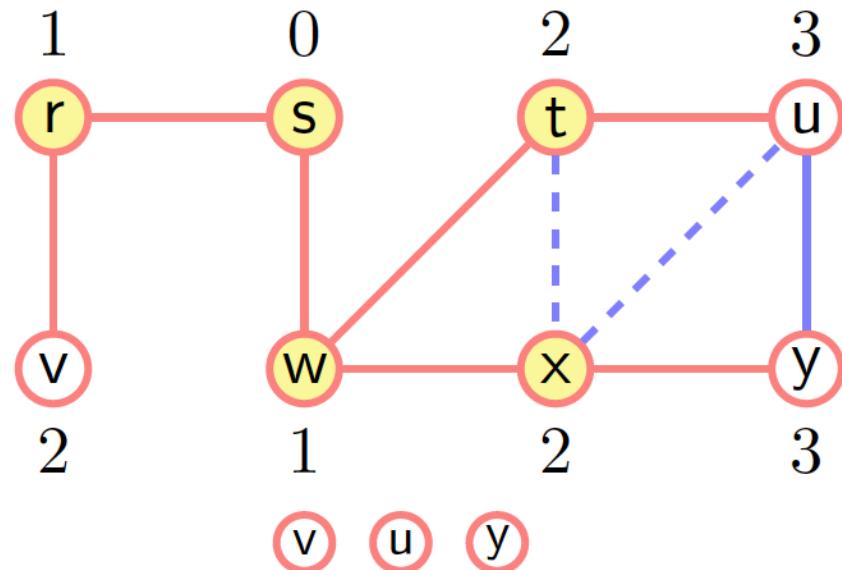
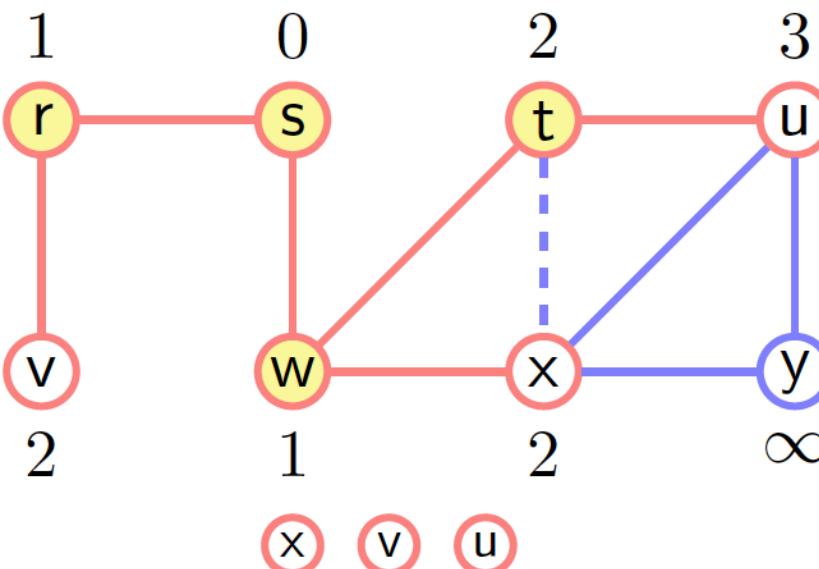
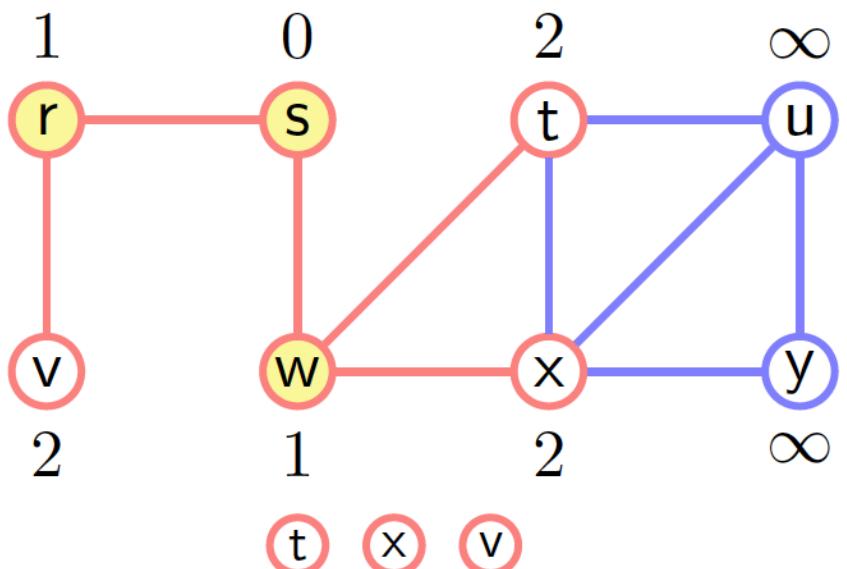


w r

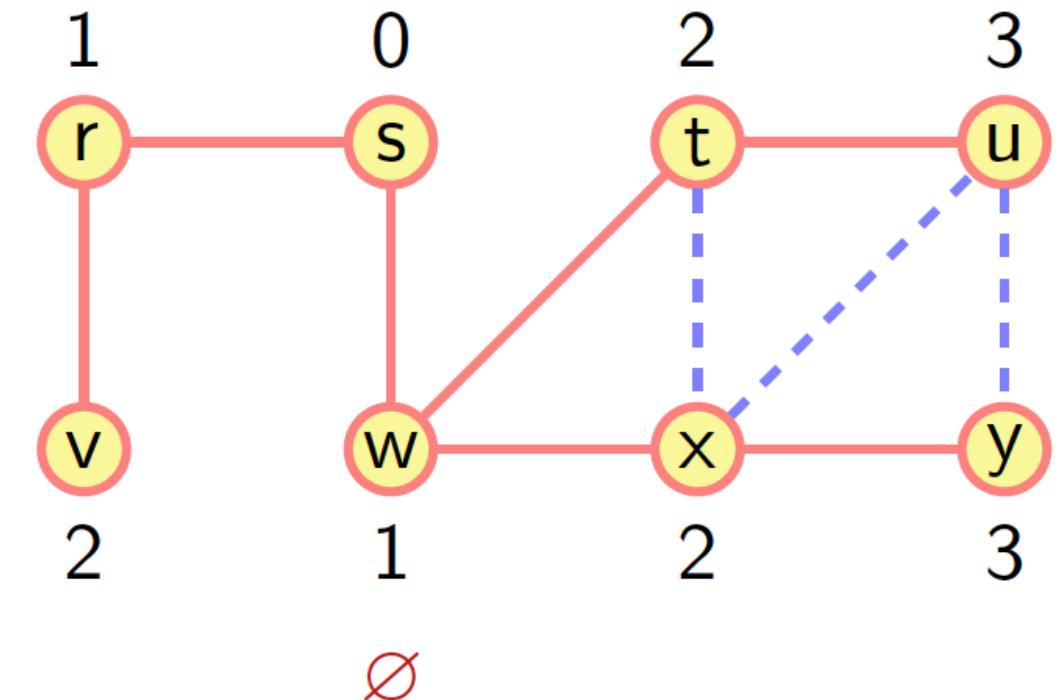
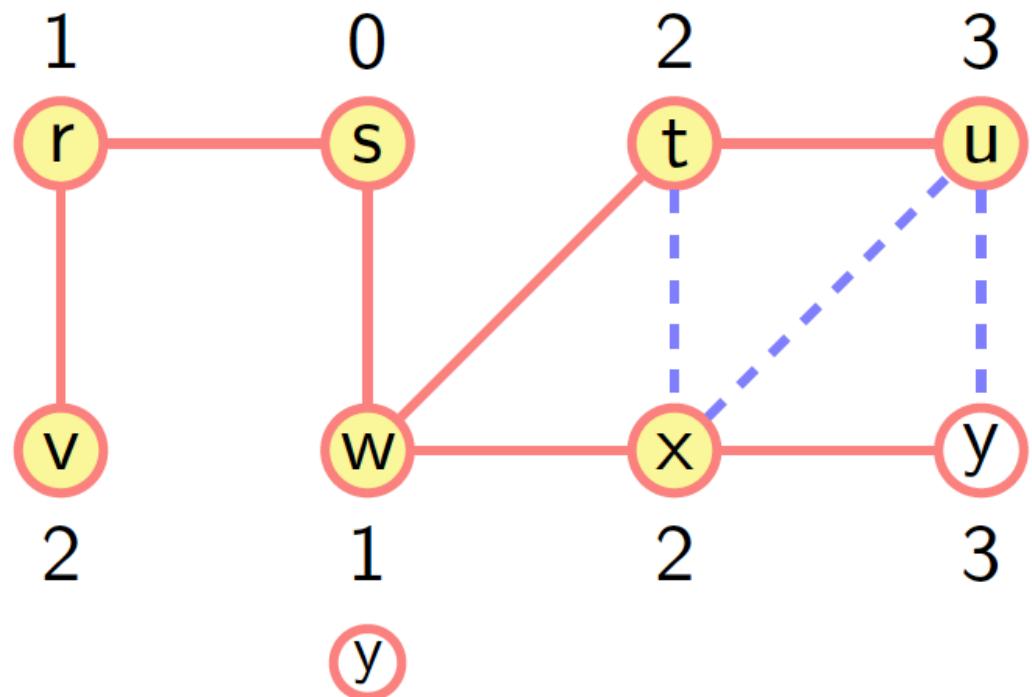


r t x

3.5 图搜索策略——广度优先查找实例



3.5 图搜索策略——广度优先查找实例



3.5 图搜索策略——广度优先查找时间效率

- 广度优先查找的效率与用来表示图的数据结构的规模成正比
 - 邻接矩阵表示法：遍历的时间效率属于 $\Theta(|V|^2)$
 - 邻接链表表示法：遍历的时间效率属于 $\Theta(|V| + |E|)$
- 使用队列
- 应用：检查图的连通性、无环性、最少边路径

9.3 最短路径问题

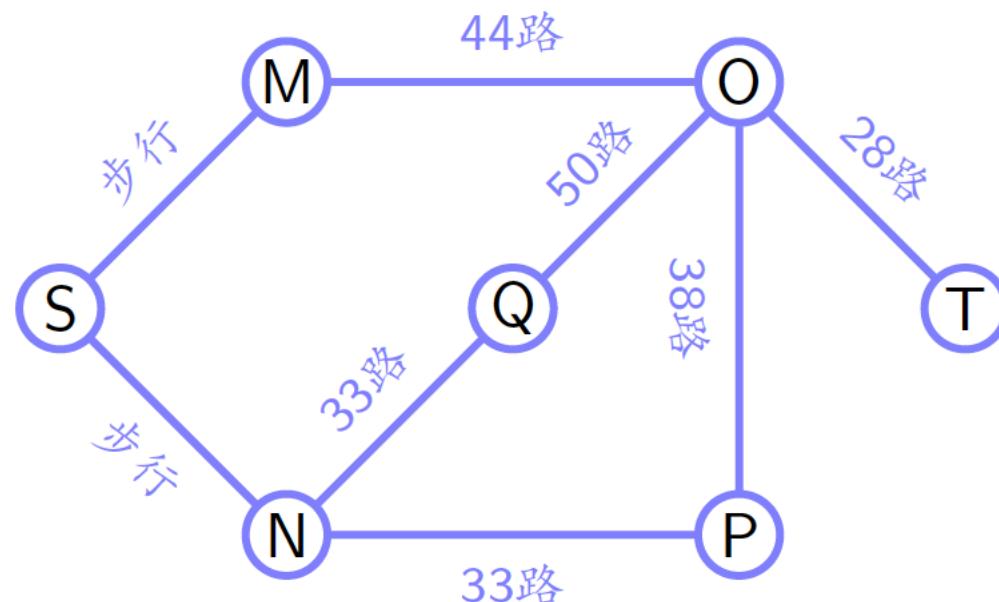
- 无权最短路径
- 单源最短路径

9.3 最短路径问题——无权最短路径

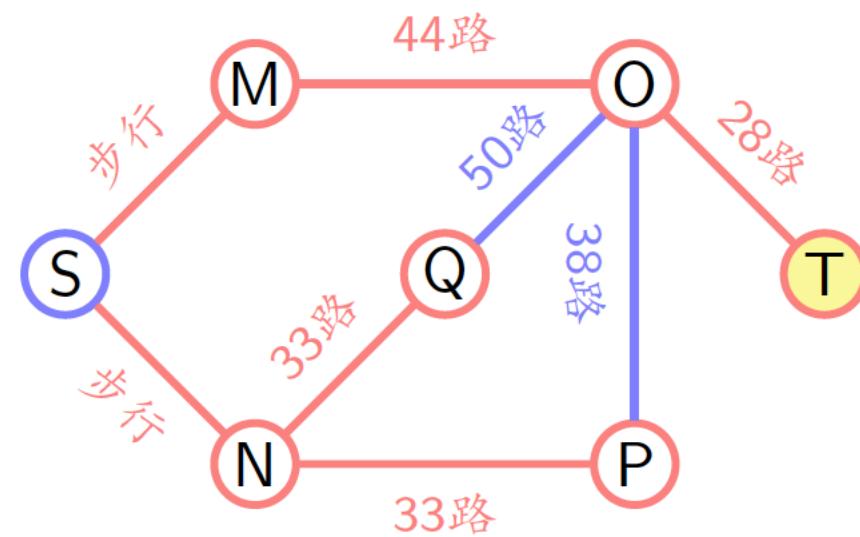
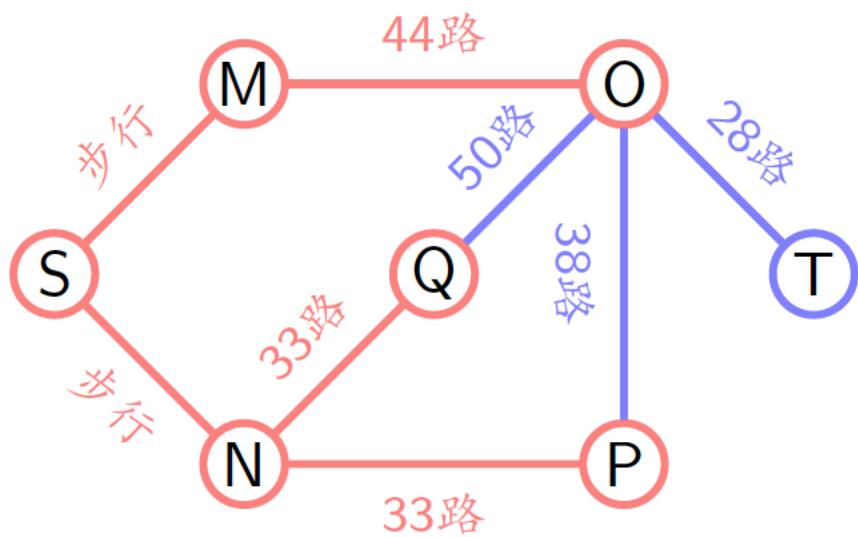
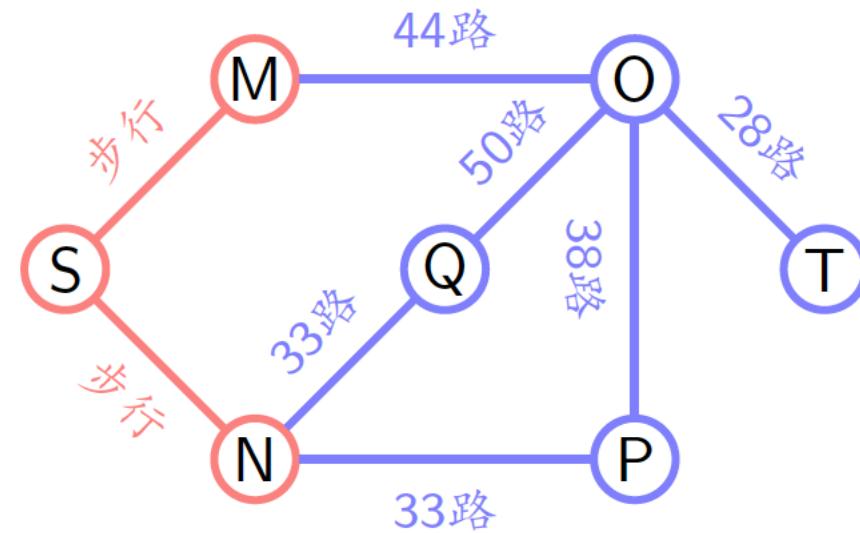
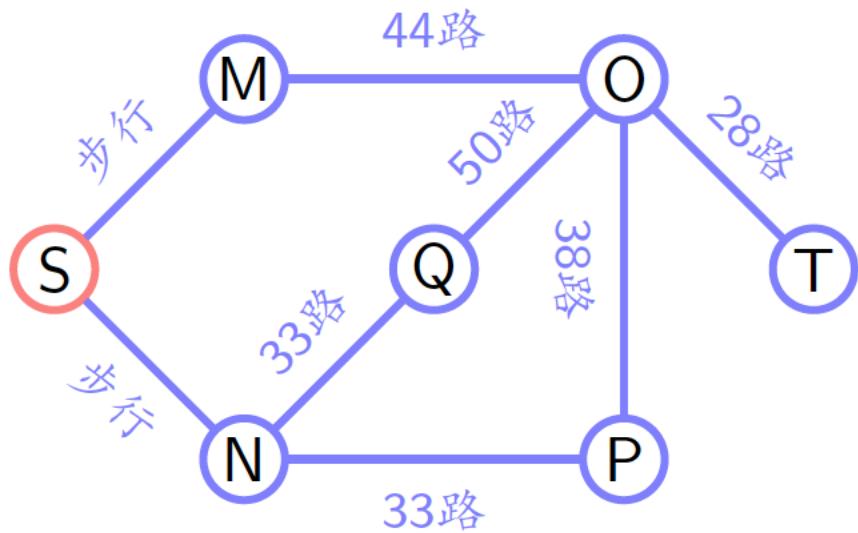
问题描述

对于一个无权图 G , 希望找到某个顶点到其他顶点的最短路径, 也就是说我们只对包含在路径中的边数感兴趣。

实例: 寻找从“双子峰”(S) 到“金门大桥”(T) 的最少换乘路径



9.3 无权最短路径——广度优先搜索



9.3 最短路径问题——单源最短路径问题

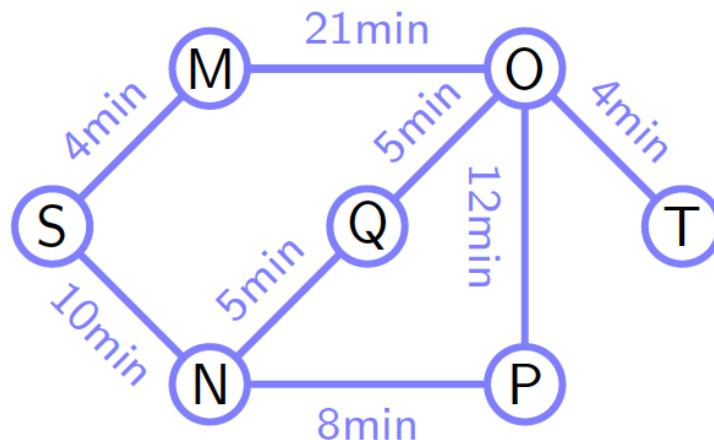
赋权路径长

对于加权图 $G = (V, E)$, 假设与每条边 (v_i, v_j) 相连的代价为 $c_{i,j}$, 定义一条路径 $v_1, v_2, v_3, \dots, v_N$ 的赋权路径长为 $\sum_{i=1}^{N-1} c_{i,i+1}$ 。

单源最短路径问题

找出从一个特定顶点 S 到 G 中每一个其他顶点的最短赋权路径。

思考：寻找从“双子峰”（S）到“金门大桥”（T）的耗时最短路径



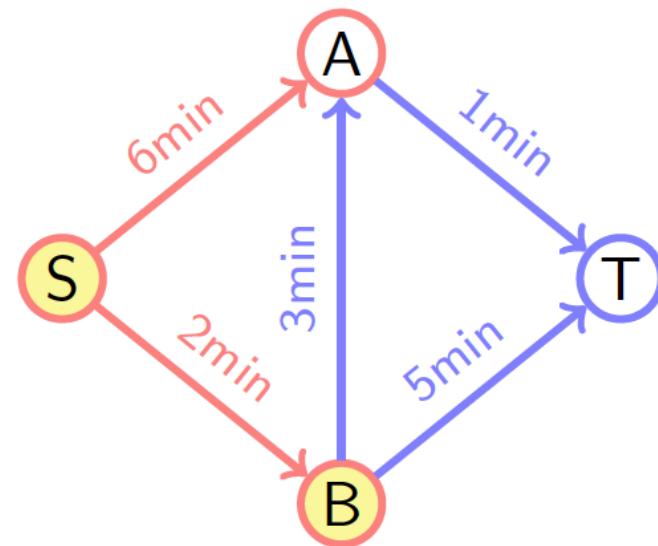
9.3 单源最短路径问题——Dijkstra算法

- 简单的例子

算法步骤

- ① 找出“最便宜”的节点
- ② 更新该节点的邻居的开销
- ③ 重复1和2直到访问所有节点
- ④ 计算最终路径

例子



节点	耗时
A	6
B	2
T	∞

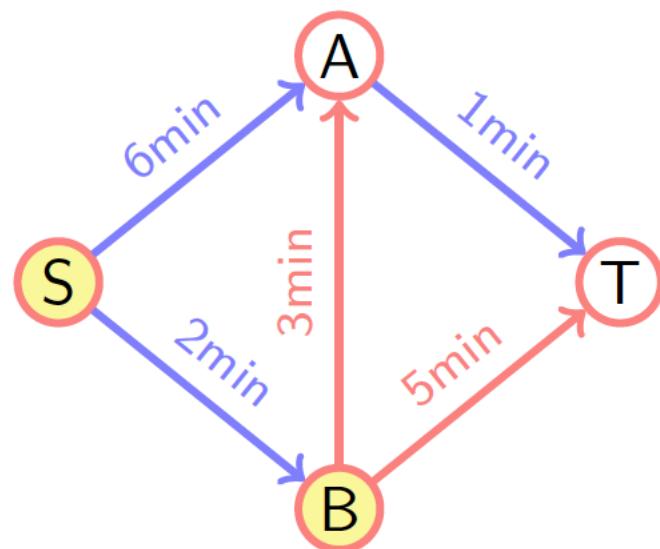
9.3 单源最短路径问题——Dijkstra算法

- 简单的例子

算法步骤

- ① 找出“最便宜”的节点
- ② 更新该节点的邻居的开销
- ③ 重复1和2直到访问所有节点
- ④ 计算最终路径

例子



节点	耗时
A	5
B	2
T	7

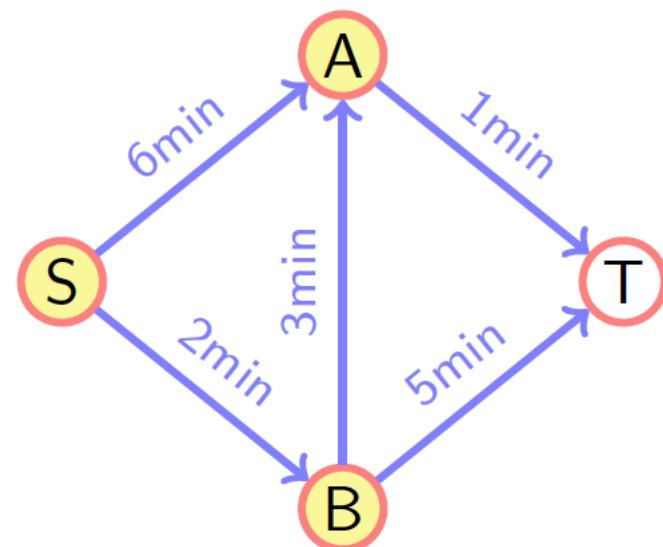
9.3 单源最短路径问题——Dijkstra算法

- 简单的例子

算法步骤

- ① 找出“最便宜”的节点
- ② 更新该节点的邻居的开销
- ③ 重复1和2直到访问所有节点
- ④ 计算最终路径

例子



节点	耗时
A	5
B	2
T	7

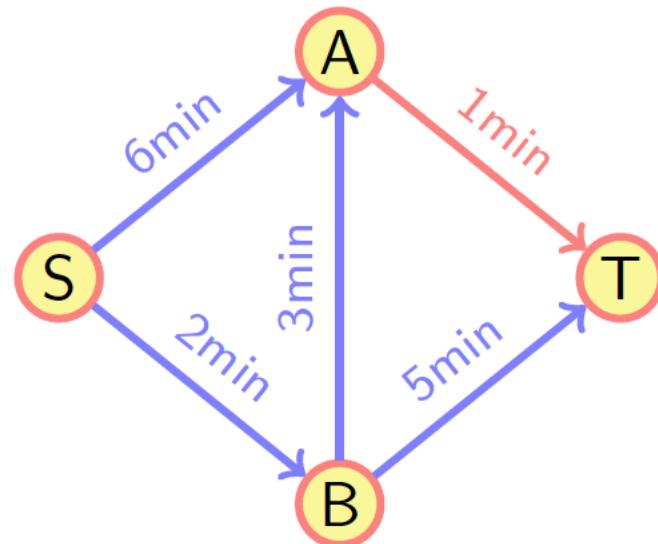
9.3 单源最短路径问题——Dijkstra算法

- 简单的例子

算法步骤

- ① 找出“最便宜”的节点
- ② 更新该节点的邻居的开销
- ③ 重复1和2直到访问所有节点
- ④ 计算最终路径

例子



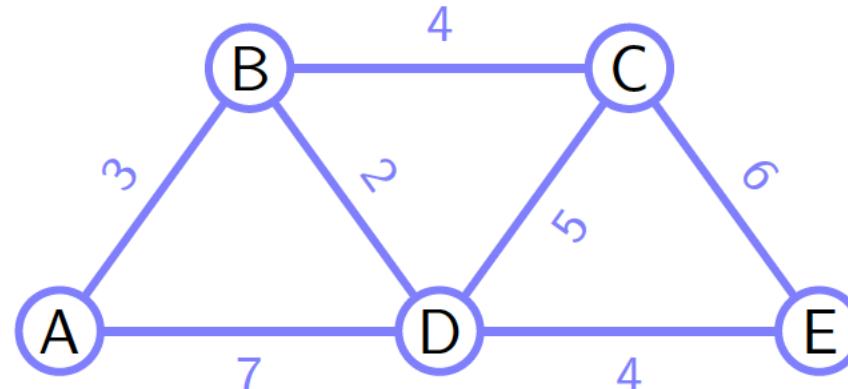
节点	耗时
A	5
B	2
T	6

9.3 单源最短路径问题——Dijkstra算法流程

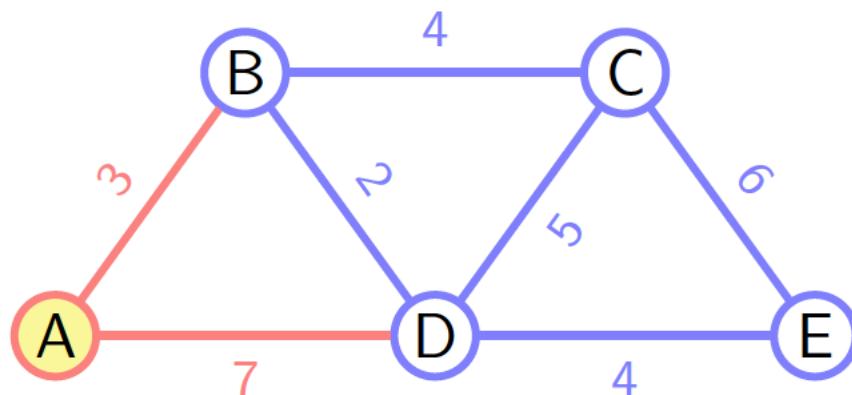
- ① 根据选择的起点 s , 将顶点 V 划分为两个部分, 树顶点 $O = \{s\}$ 和边缘顶点 $V - O$;
- ② 给图的所有顶点 V 添加标记 d , 记录目前为止算法得到从起点 s 到每个顶点的最短路径长度, 并记录该路径倒数第2个顶点 (父节点);
- ③ 从边缘顶点 $V - O$ 中选择具有最小 d 值的节点 u^* , 把 u^* 加入 O 中;
- ④ 对于 $V - O$ 中的每个顶点 u , 如果通过权重为 $w(u^*, u)$ 的边和 u^* 相连接, 当 $d_{u^*} + w(u^*, u) < d_u$ 时, u 的父节点标记更新为 u^* , 最短路径长度 d 更新为 $d_{u^*} + w(u^*, u)$;
- ⑤ 算法重复步骤3和4, 逐步从 $V - O$ 选择节点, 加入到 O 中, 直到 $V = O$;

9.3 单源最短路径问题——Dijkstra算法实例

如何利用Dijkstra算法寻找从点A开始的最短路径？

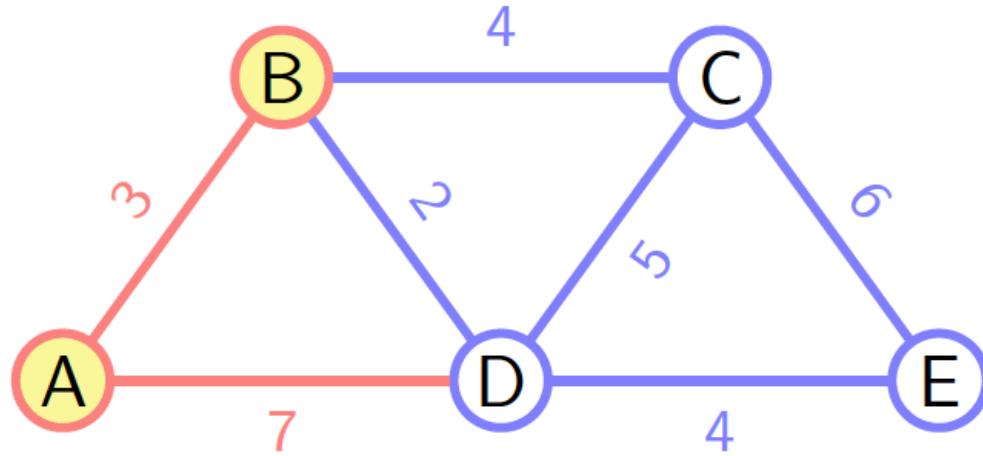


计算过程

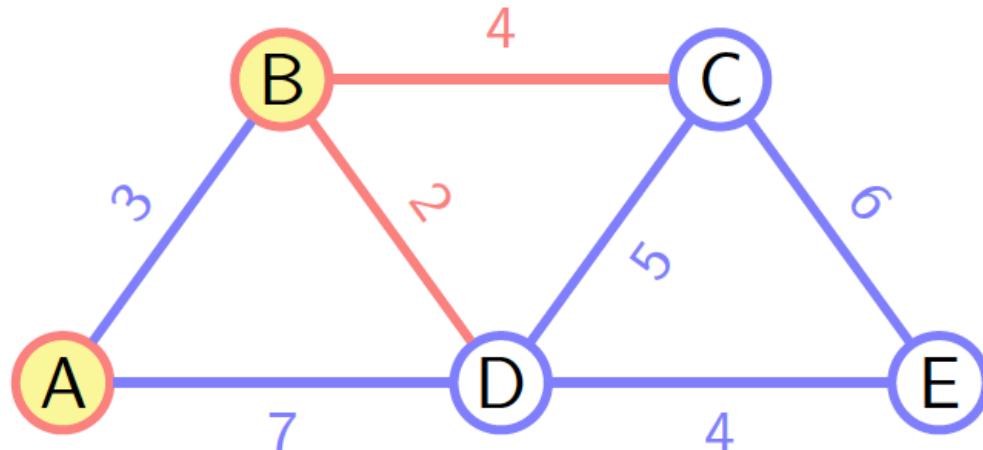


树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0	B	A	3
			C	-	∞
			D	A	7
			E	-	∞

9.3 单源最短路径问题——Dijkstra算法流程实例

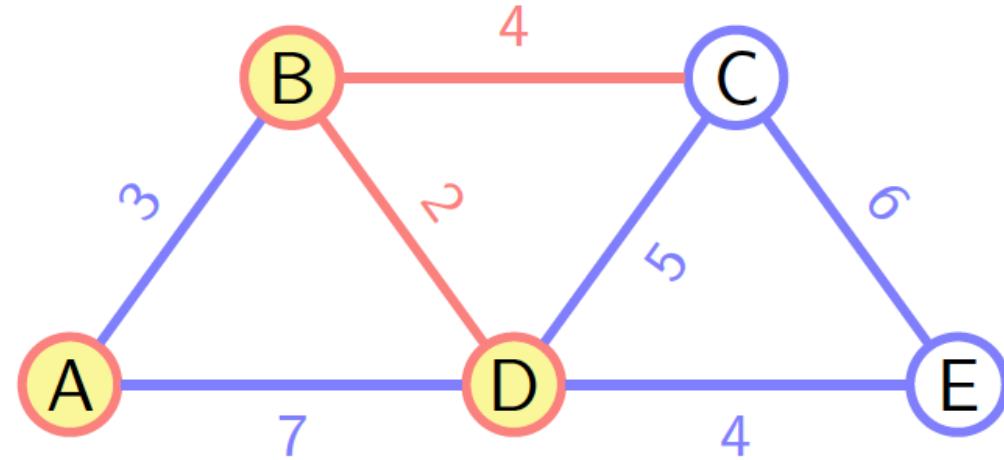


树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0			
B	A	3			
			C	-	∞
			D	A	7
			E	-	∞

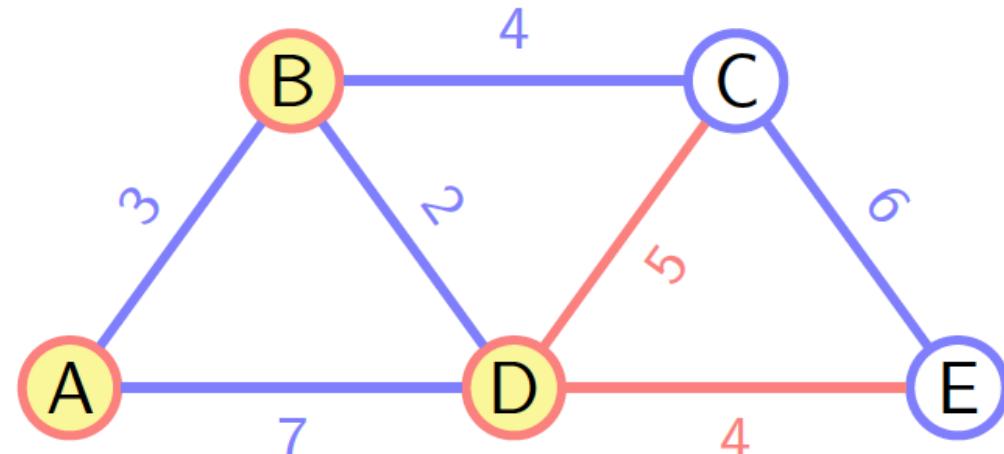


树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0			
B	A	3			
			C	B	$3+4$
			D	B	$3+2$
			E	-	∞

9.3 单源最短路径问题——Dijkstra算法流程实例

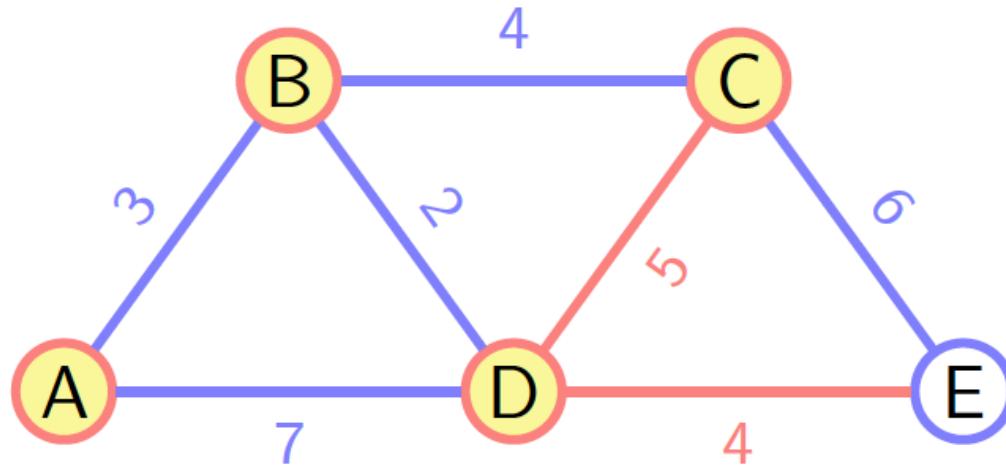


树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0			
B	A	3			
D	B	5			
			C	B	7
			E	-	∞

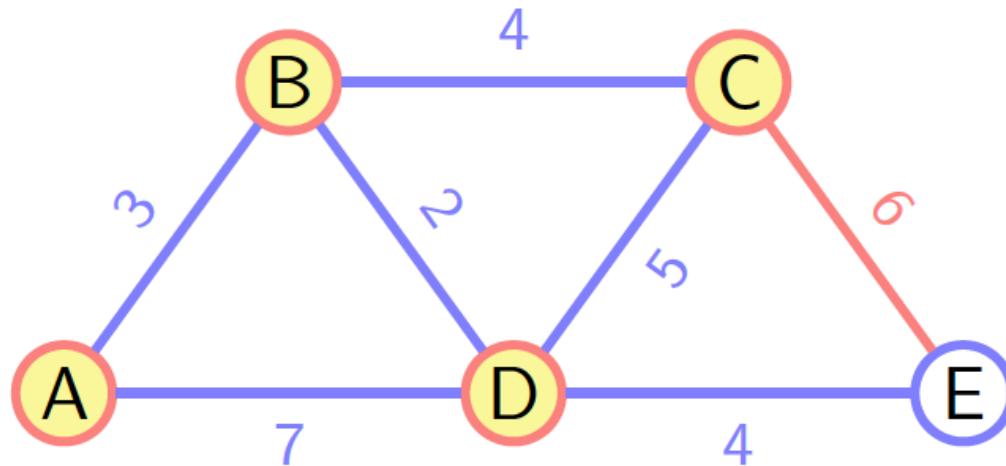


树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0			
B	A	3			
D	B	5			
			C	B	7
			E	D	9

9.3 单源最短路径问题——Dijkstra算法流程实例

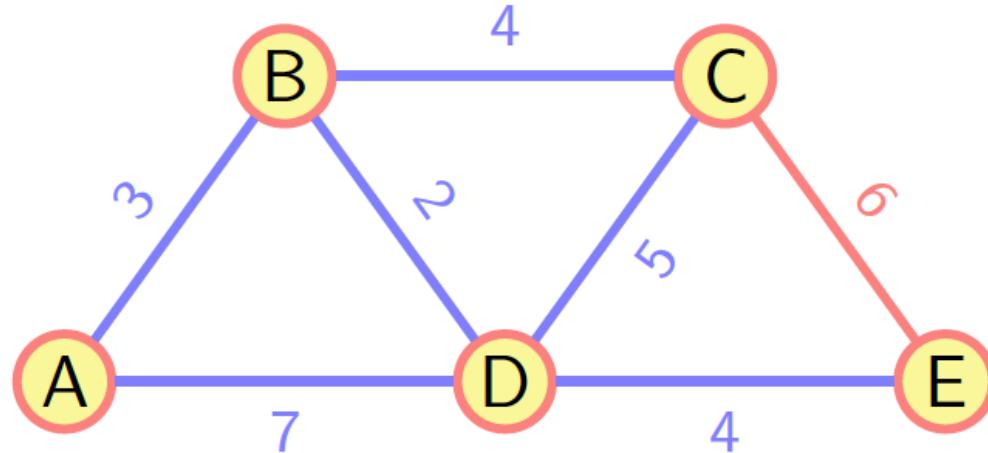


树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0			
B	A	3			
D	B	5			
C	B	7			
			E	D	9



树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0			
B	A	3			
D	B	5			
C	B	7			
			E	D	9

9.3 单源最短路径问题——Dijkstra算法流程实例



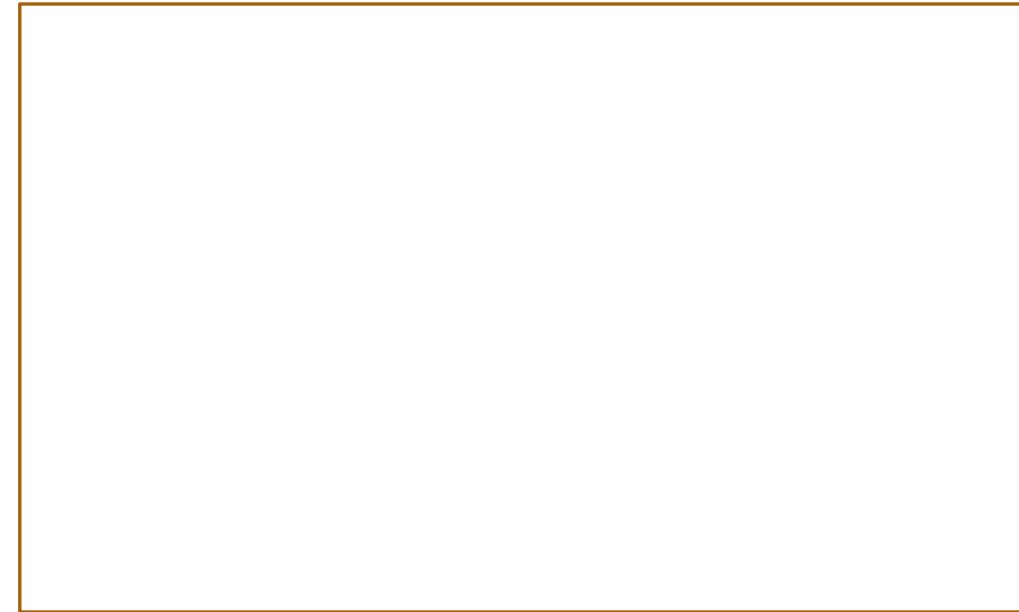
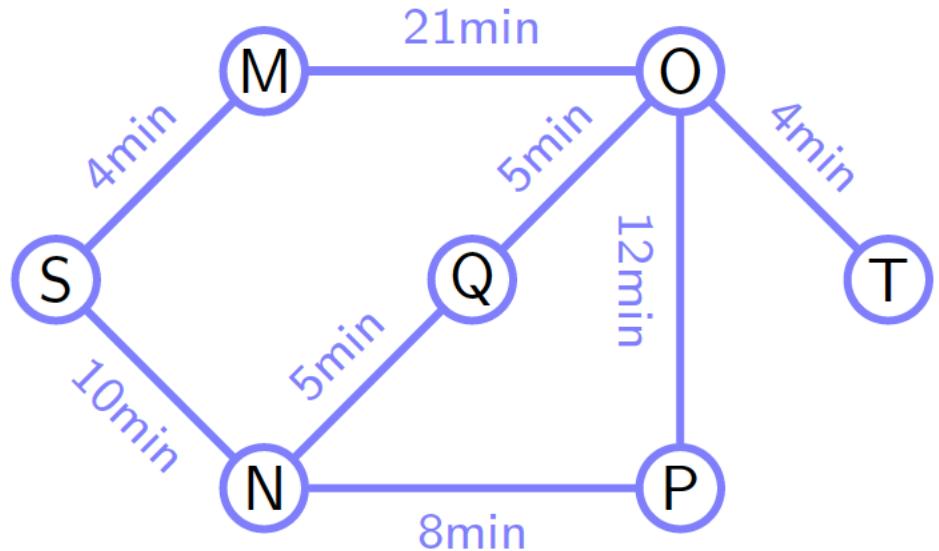
树顶点			边缘顶点		
节点	父节点	权重	节点	父节点	权重
A	-	0			
B	A	3			
D	B	5			
C	B	7			
E	D	9			

最短路径

- 从A到B: A-B, 长度为3;
- 从A到D: A-B-D, 长度为5;
- 从A到C: A-B-C, 长度为7;
- 从A到E: A-B-D-E, 长度为9;

9.3 单源最短路径问题——Dijkstra算法

寻找从“双子峰”(S)到“金门大桥”(T)的耗时最短路径



最短路径

从“双子峰”到“金门大桥”: S-N-Q-O-T, 长度为24分钟

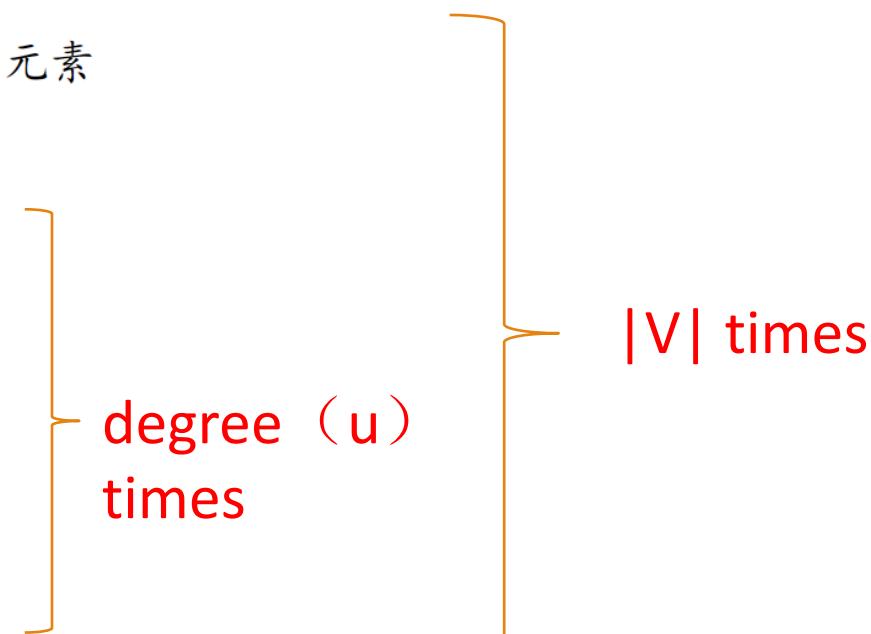
9.3 单源最短路径问题——Dijkstra算法伪代码

```
1: function Dijkstra( $G, s$ )
2:   Initialize( $Q$ ) //将顶点优先队列初始化为空
3:   for  $V$ 中每一个顶点 $v$  do
4:      $d_v \leftarrow \infty; p_v \leftarrow \text{null}$ 
5:     Insert( $Q, v, d_v$ ) //初始化优先队列中顶点的优先级
6:   end for
7:    $V_T \leftarrow \emptyset; d_s \leftarrow 0; \text{Decrease}(Q, s, d_s)$  //将 $s$ 的优先级更新为 $d_s$ 
8:   for  $i \leftarrow 0 \rightarrow |V| - 1$  do
9:      $u^* \leftarrow \text{DeleteMin}(Q)$  //删除优先级最小的元素
10:     $V_T \leftarrow V_T \cup \{u^*\}$ 
11:    for  $V - V_T$ 中每一个和 $u^*$ 相邻的顶点 $u$  do
12:      if  $d_{u^*} + w(u^*, u) < d_u$  then
13:         $d_u \leftarrow d_{u^*} + w(u^*, u); p_u \leftarrow u^*$ 
14:        Decrease( $Q, u, d_u$ )
15:      end if
16:    end for
17:  end for
18: end function
```

9.3 单源最短路径问题——Dijkstra算法时间效率

```
1: function Dijkstra( $G, s$ )
2:   Initialize( $Q$ ) //将顶点优先队列初始化为空
3:   for  $V$ 中每一个顶点 $v$  do
4:      $d_v \leftarrow \infty; p_v \leftarrow \text{null}$ 
5:     Insert( $Q, v, d_v$ ) //初始化优先队列中顶点的优先级
6:   end for
7:    $V_T \leftarrow \emptyset; d_s \leftarrow 0; \text{Decrease}(Q, s, d_s)$  //将 $s$ 的优先级更新为 $d_s$ 
8:   for  $i \leftarrow 0 \rightarrow |V| - 1$  do
9:      $u^* \leftarrow \text{DeleteMin}(Q)$  //删除优先级最小的元素
10:     $V_T \leftarrow V_T \cup \{u^*\}$ 
11:    for  $V - V_T$ 中每一个和 $u^*$ 相邻的顶点 $u$  do
12:      if  $d_{u^*} + w(u^*, u) < d_u$  then
13:         $d_u \leftarrow d_{u^*} + w(u^*, u); p_u \leftarrow u^*$ 
14:        Decrease( $Q, u, d_u$ )
15:      end if
16:    end for
17:  end for
18: end function
```

$$\text{Time} = \Theta(|V|T_{\text{Delete-MIN}} + |E|T_{\text{DECREASE-KEY}})$$



9.3 单源最短路径问题——Dijkstra算法时间效率

$$\text{Time} = \Theta(|V|T_{\text{Delete-MIN}} + |E|T_{\text{DECREASE-KEY}})$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
Binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ amortized



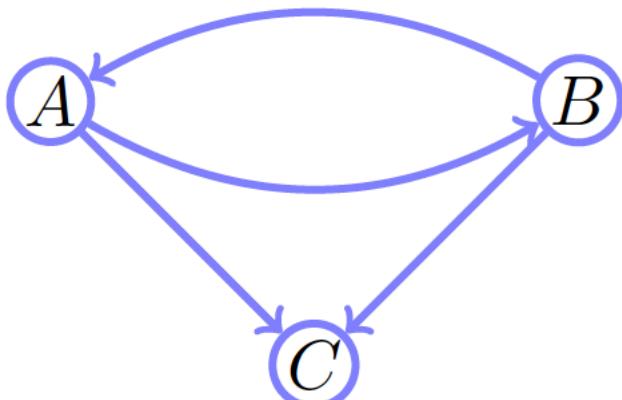
4.2 拓扑排序

- 深度优先法
- 源删除法

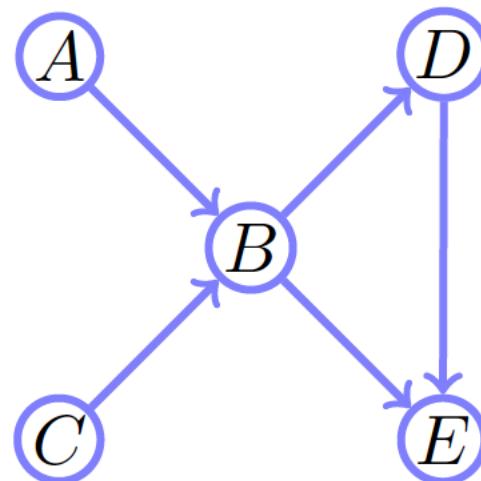
4.2 拓扑排序问题

问题描述

从有向无环图中寻找一个序列，使得图中每一条边来说，边的起始顶点总是排在边的结束顶点之前。



有向有环图



有向无环图

注意

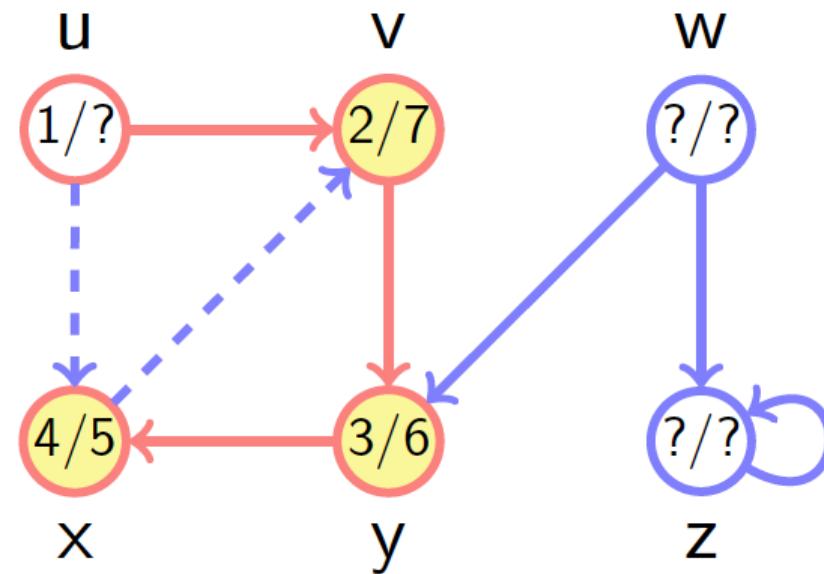
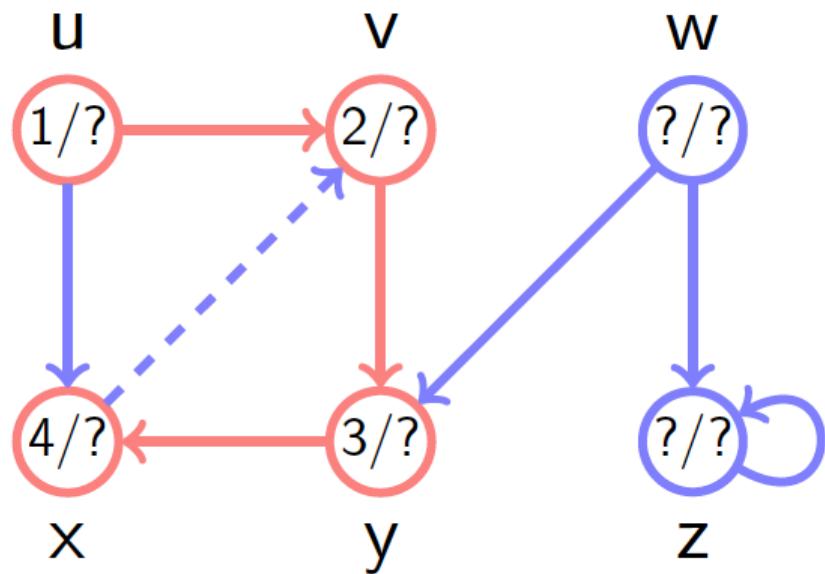
对于图中有一个回路（有向有环图），拓扑排序问题是无解的。

4.2 拓扑排序问题——无环图判断方法

要点

在DFS中，如果出现一条边指向发现且未完成的节点，则该图为有环图，该边称为回边。

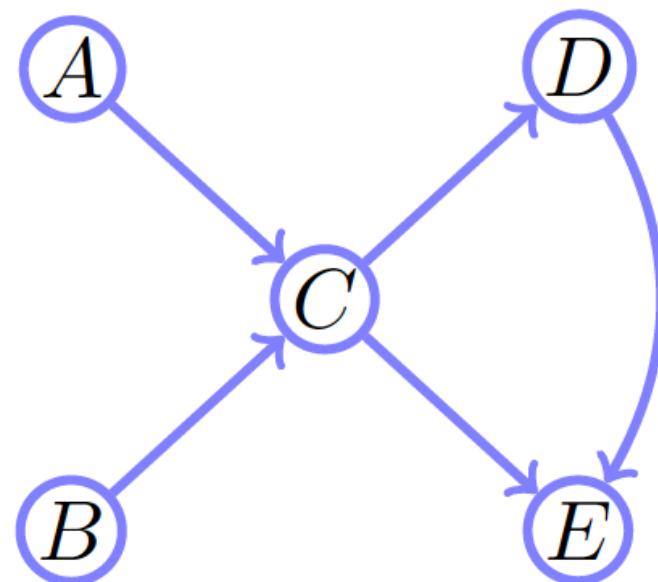
例子： (x, v) 是回边， (u, x) 不是回边



4.2 拓扑排序问题——实例

问题描述

考虑5门必修课的一个集合 $\{A, B, C, D, E\}$ ，一个在校学生必须在某个阶段修完这几门课程。学生可以按照任何次序修课，但是必须满足如下的条件，修完 A 和 B 才能修 C ，修完 C 才能修 D ，修完 C 和 D 才能修 E 。问学生该如何修课？

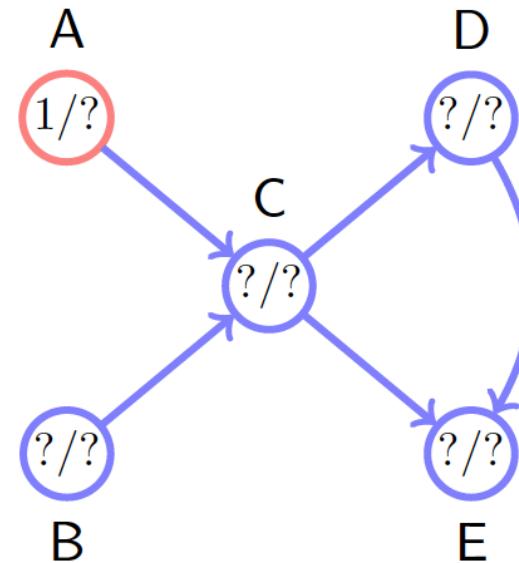
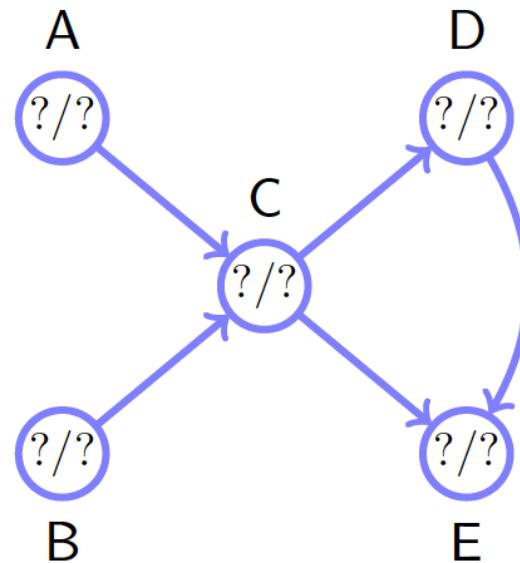


4.2 拓扑排序问题——深度优先法

算法思想

执行DFS遍历搜寻，并记住顶点变成终点（完成或出栈）的顺序，将该次序反过来就得到拓扑排序的一个解。

算法过程

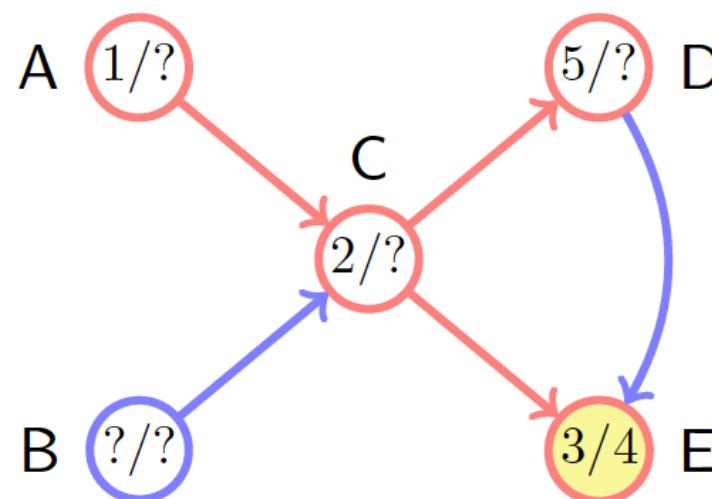
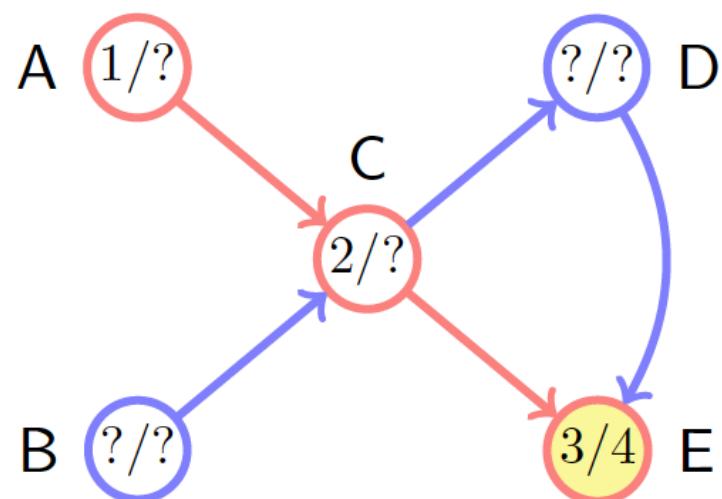
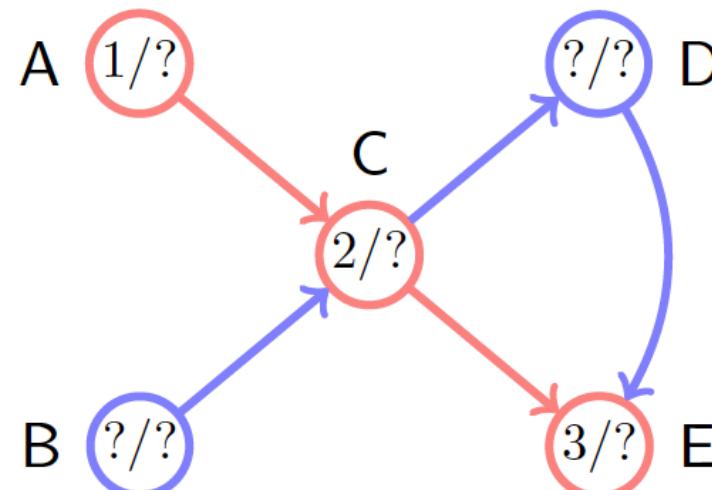
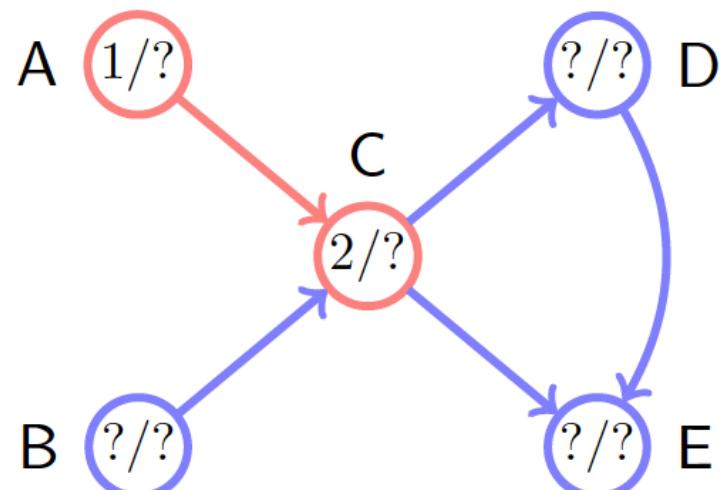


(s/t) 未发现节点

(s/t) 发现节点

(s/t) 完成节点 s:发现时间 t:完成时间

4.2 拓扑排序问题——深度优先法

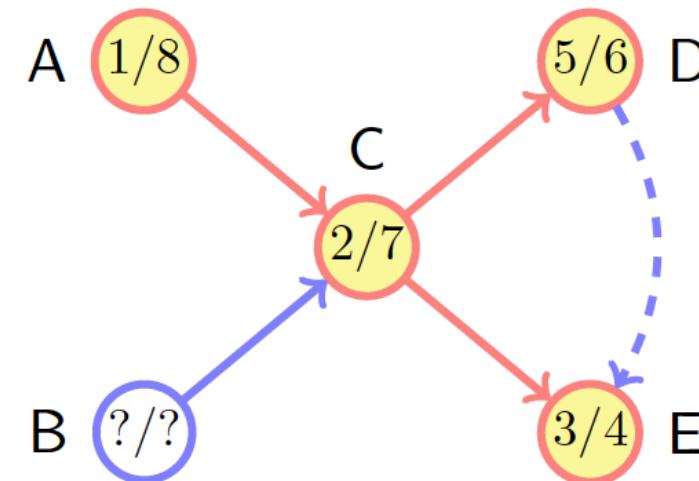
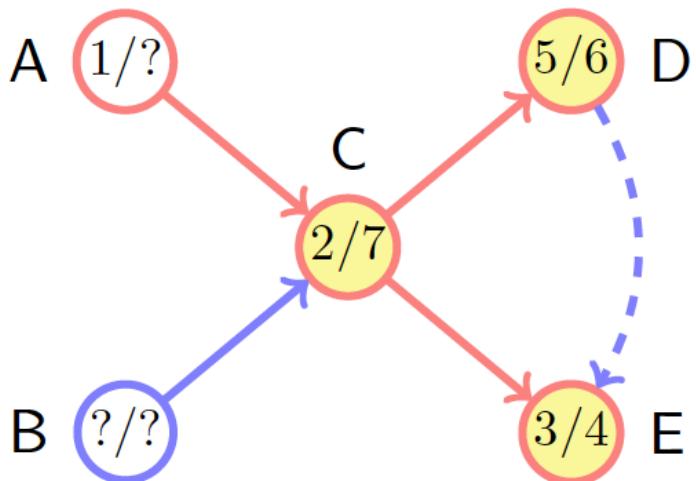
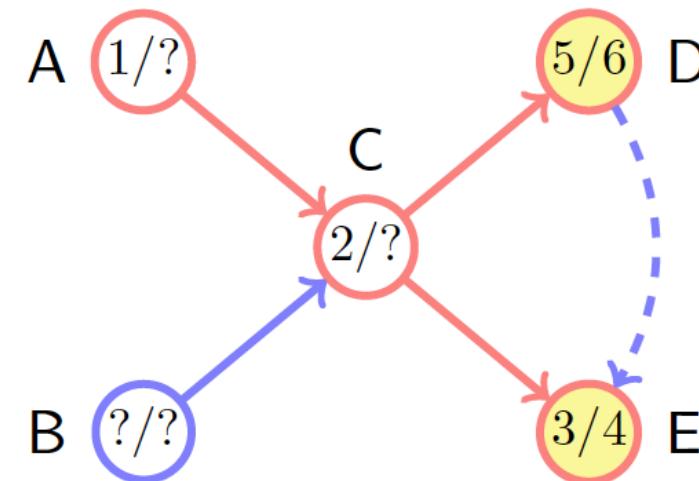
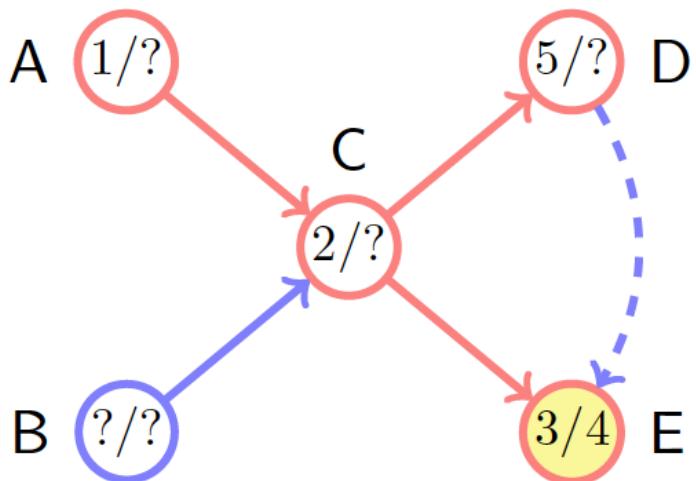


(s/t) 未发现节点

(s/t) 发现节点

(s/t) 完成节点 s:发现时间 t:完成时间

4.2 拓扑排序问题——深度优先法

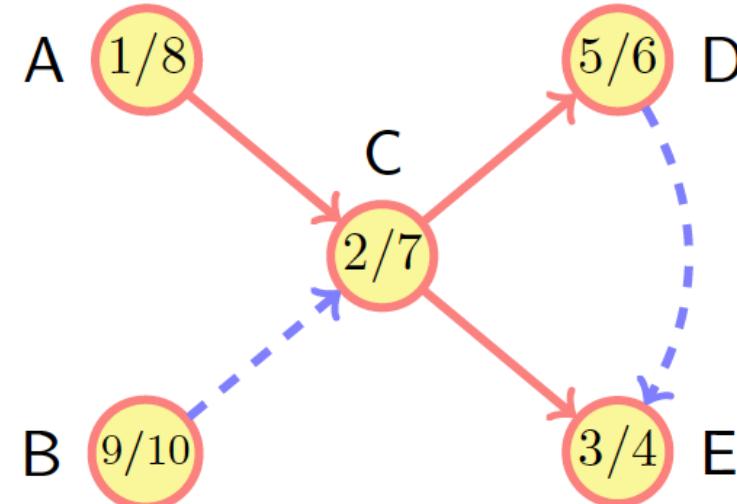
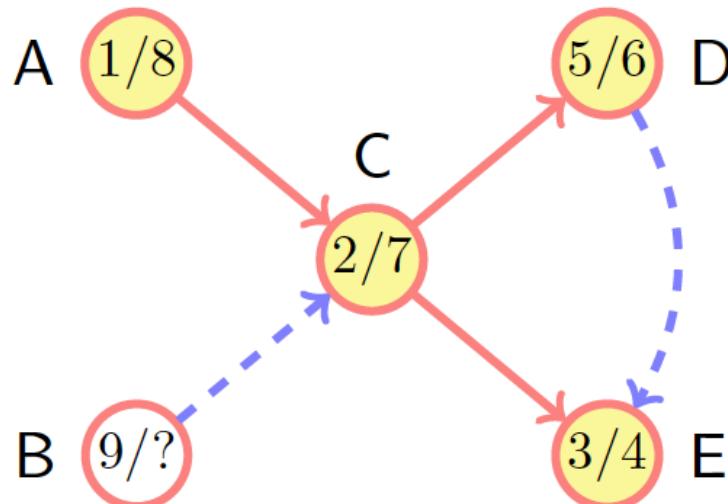


(s/t) 未发现节点

(s/t) 发现节点

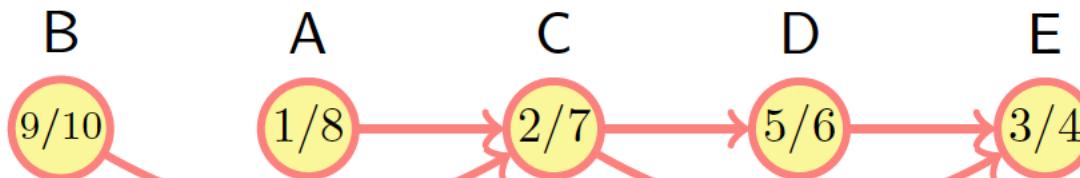
(s/t) 完成节点 s:发现时间 t:完成时间

4.2 拓扑排序问题——深度优先法



s/t 未发现节点 s/t 发现节点 s/t 完成节点 s :发现时间 t :完成时间

拓扑顺序

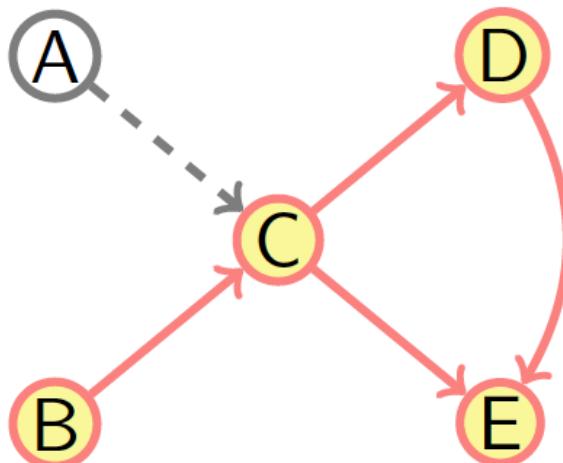
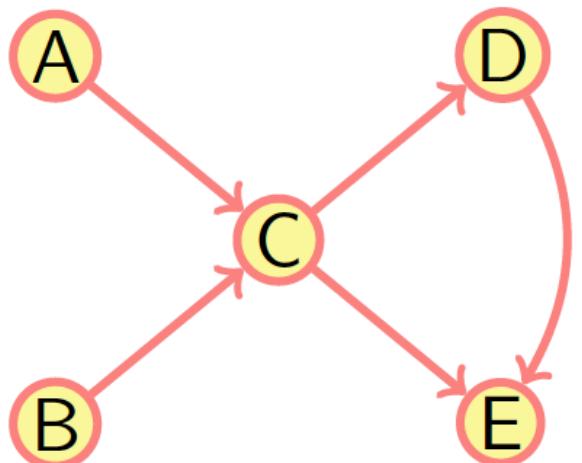


4.2 拓扑排序问题——源删除法

算法思想

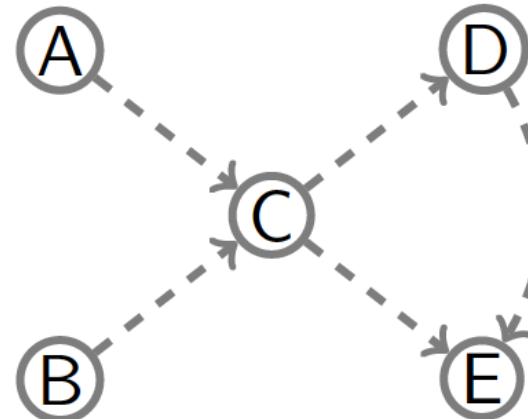
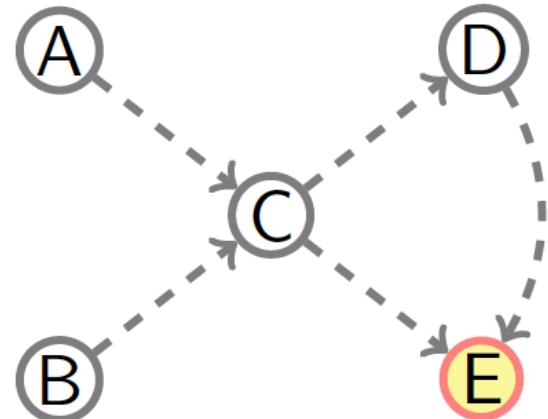
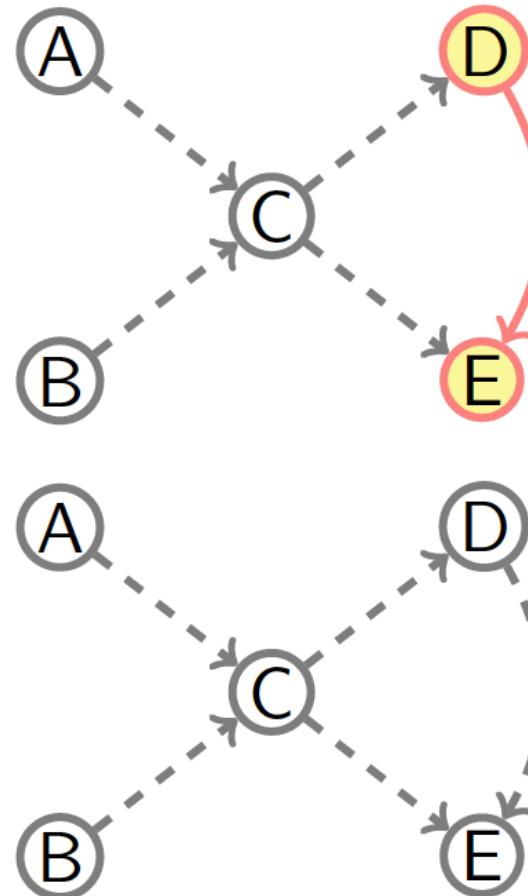
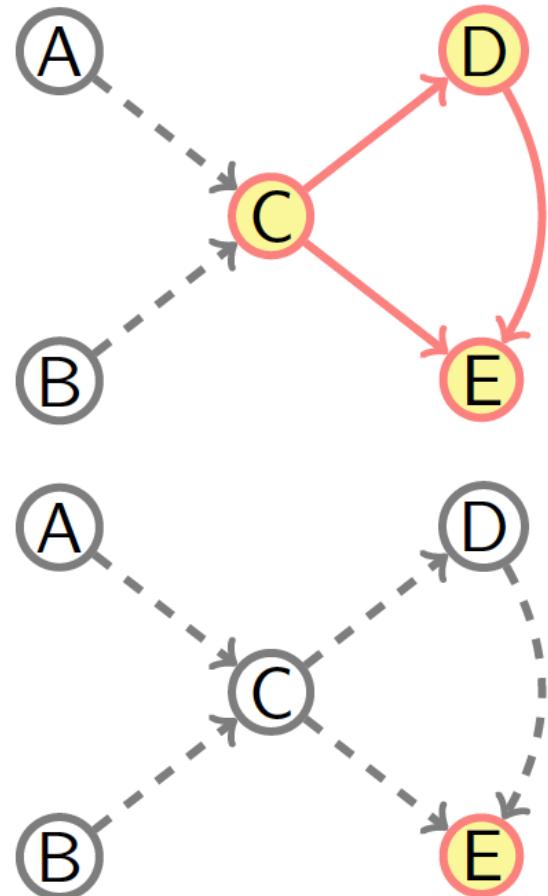
不断地在图中找出一个源点（没有输入边的顶点），然后把该点和所有从该点出发的边都删除，如此重复。顶点被删除的次序就是拓扑排序的次序。

算法过程



4.2 拓扑排序问题——源删除法

算法过程



拓扑顺序





10.2 最大流量问题

- 流量网络
- 最短路径增益法

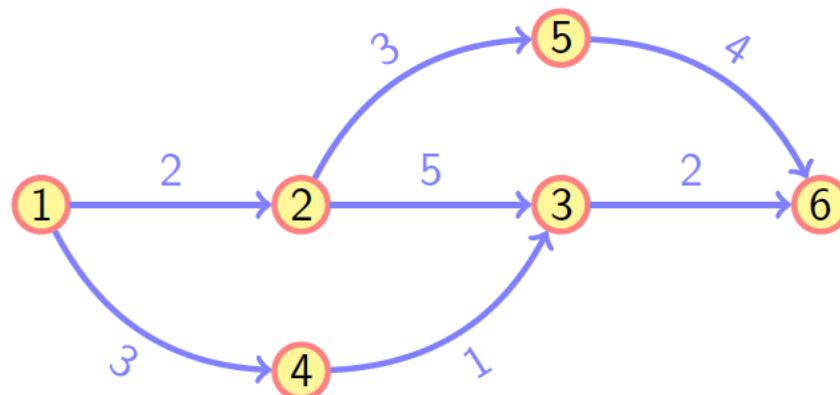
10.2 流量网络

- 流量网络定义

对于一个加权有向图 $G = (V, E)$, 具有 n 个顶点, 如具有如下的特性:

- 包含1个没有输入边的顶点 (称为源点, 标识为1);
- 包含1个没有输出边的顶点 (称为汇点, 标识为 n);
- 每条有向边 (i, j) 的权重 u_{ij} 是一个正整数 (该边的容量);

满足这些特性的有向图称为流量网络。



10.2 流量网络

流量守恒

进入任意中间顶点*i*的总流量等于离开的总流量：

$$\sum_{j:(j,i) \in E} x_{ji} = \sum_{j:(i,j) \in E} x_{ij}$$

最大流问题

对于每条边 $(i, j) \in E$ 的流量 x_{ij} ，满足容量约束：

$$0 \leq x_{ij} \leq u_{ij}$$

并且中间节点流量守恒要求，最大流问题的目标是最大化 v ：

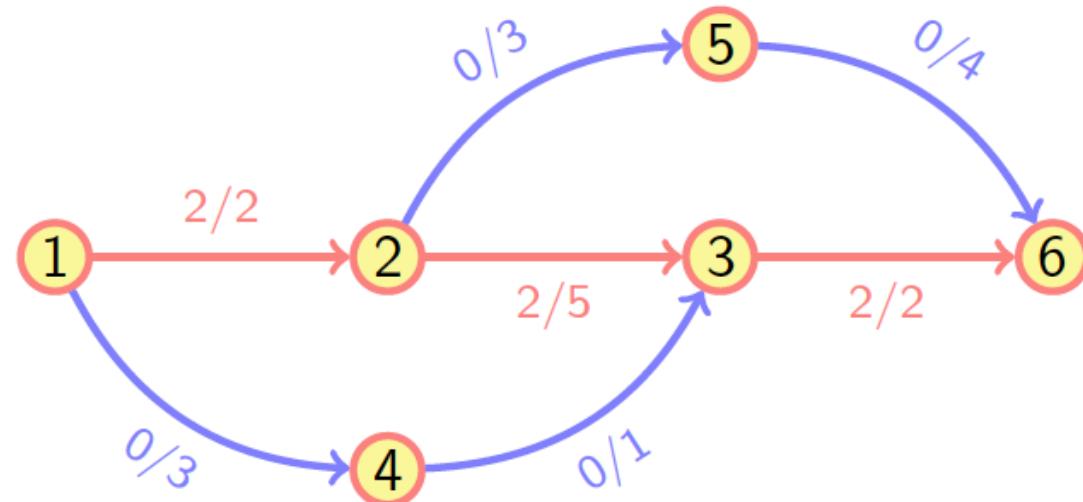
$$v = \sum_{j:(1,j) \in E} x_{1j} = \sum_{j:(j,n) \in E} x_{jn}$$

10.2 增益路径法

方法要点

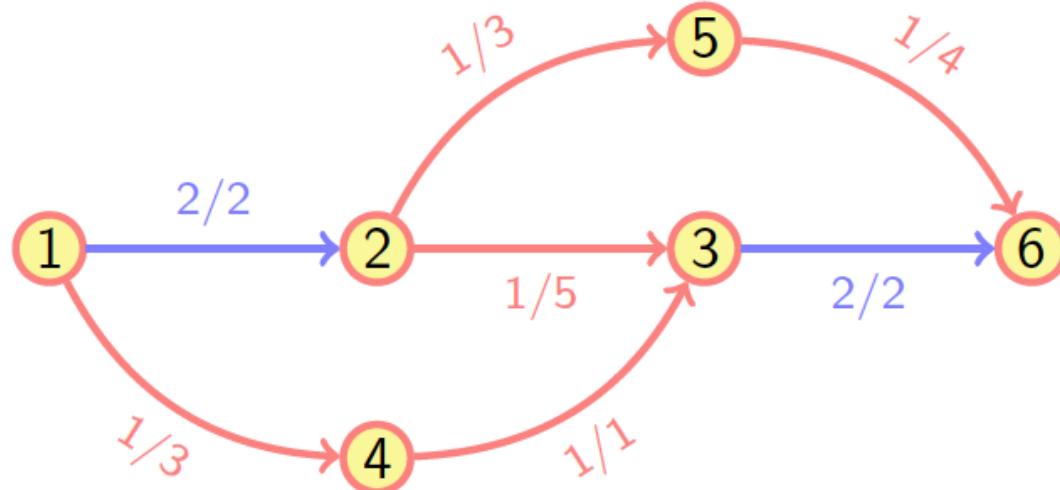
从流量为0开始，每次迭代，试着找到一条可以传输更多从源点到汇点流量的增益路径，并逐步调整各条边上的流量，直到找不到增益路径为止。

问题：流量是否是最大的？是否有其他增益路径？

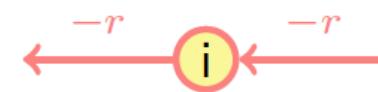
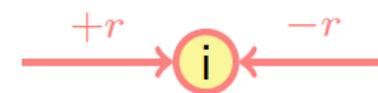
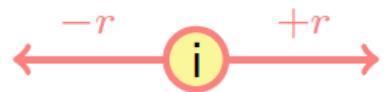


10.2 增益路径法

最优增益路径

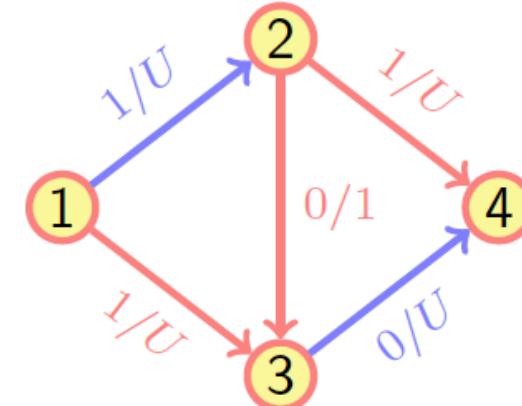
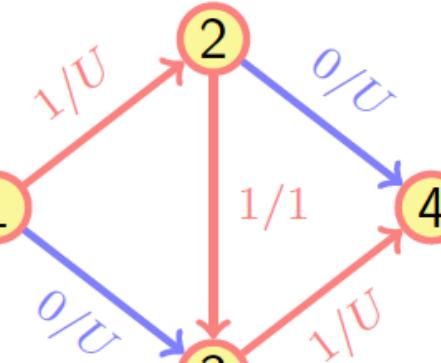
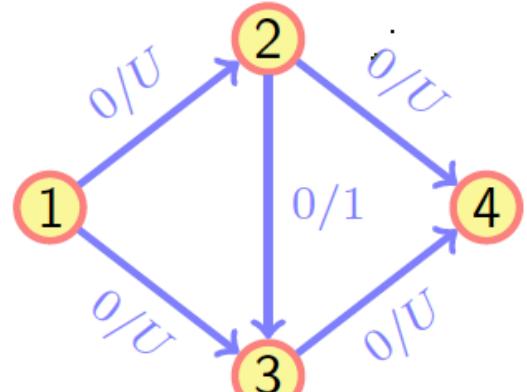


路径具有前向边和反向边时的增益方法

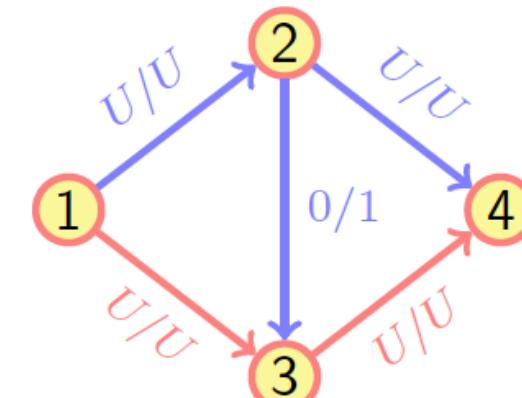
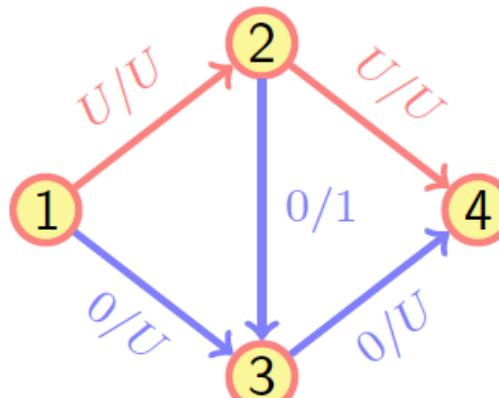
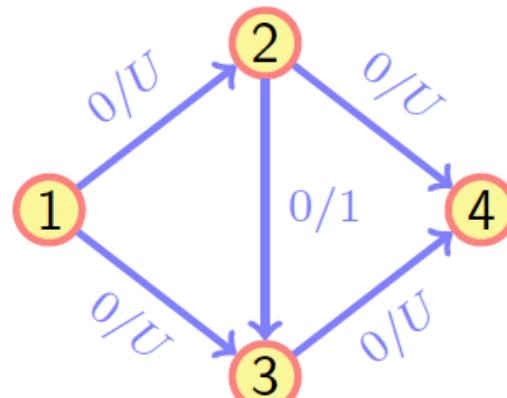


10.2 增益路径法——流量增益路径选择次序

次序1：总共需要迭代2U次



次序2：总共需要迭代2次

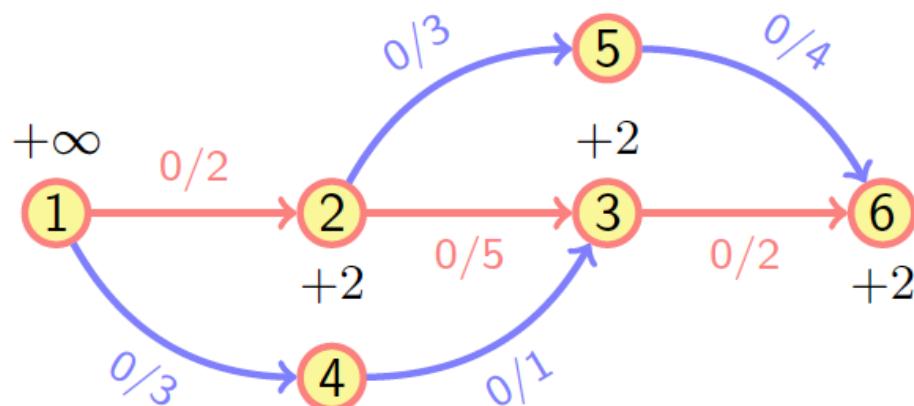


10.2 最短增益路径法

算法要点

- ① 用广度优先查找法生成增益路径（边数量最少）；
- ② 逐步标记当前增益路径每个顶点能够增加（正向边）或者减少（反向边）的流量大小；
- ③ 用汇点的流量标记修正当前增益路径每个顶点的流量；
- ④ 重复步骤1 – 3，直到优先队列为空（没有其他增益路径）；

示例

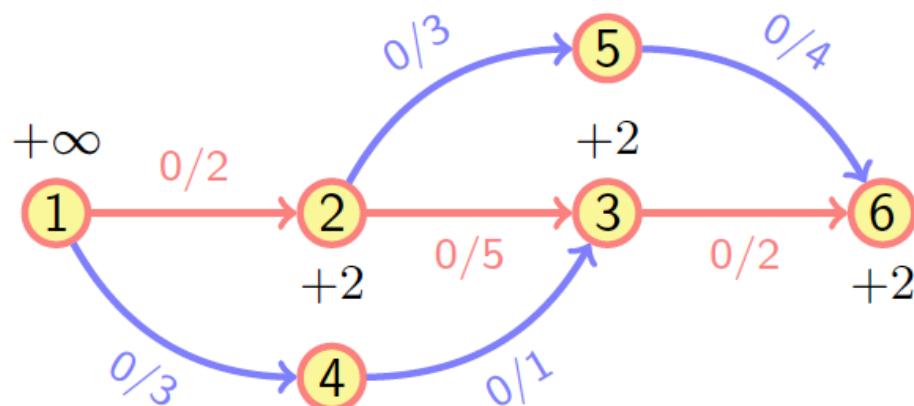


10.2 最短增益路径法——实例

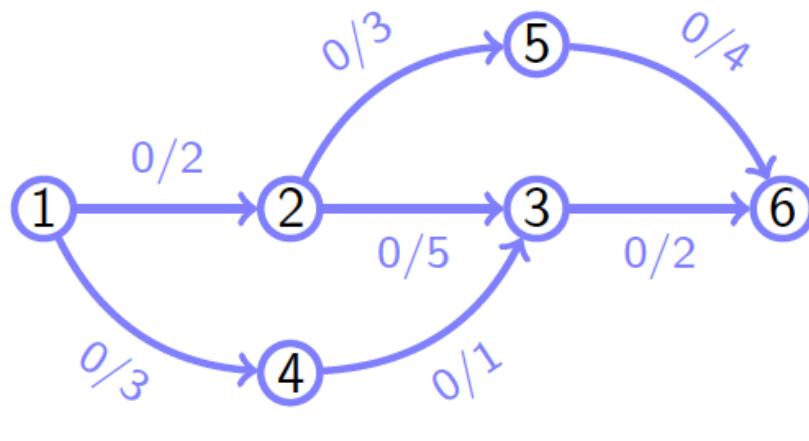
算法要点

- ① 用广度优先查找法生成增益路径（边数量最少）；
- ② 逐步标记当前增益路径每个顶点能够增加（正向边）或者减少（反向边）的流量大小；
- ③ 用汇点的流量标记修正当前增益路径每个顶点的流量；
- ④ 重复步骤1 – 3，直到优先队列为空（没有其他增益路径）；

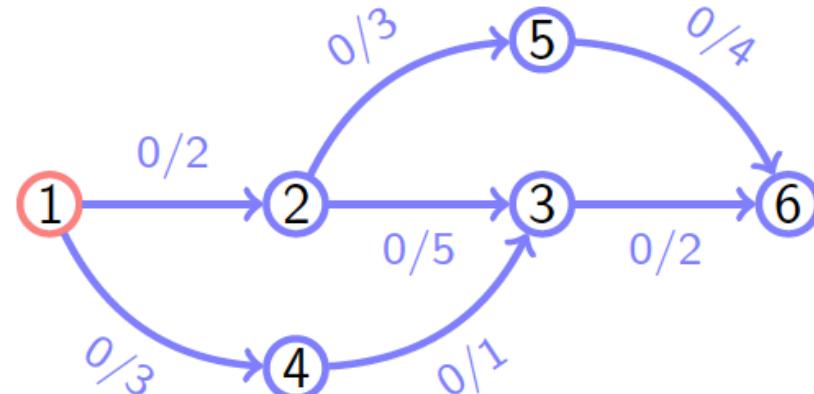
示例



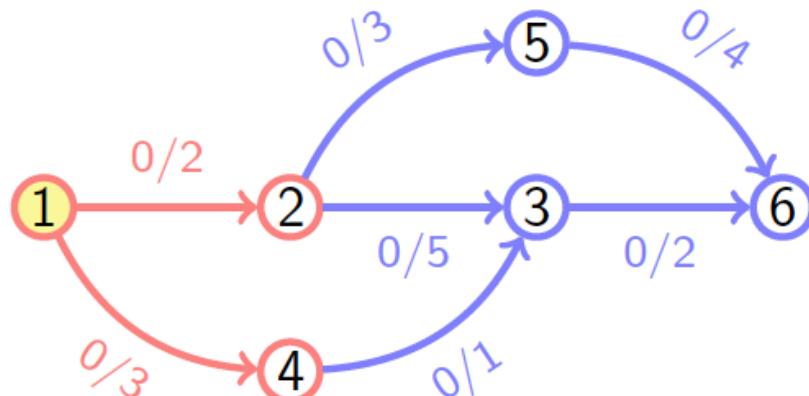
10.2 最短增益路径法——实例



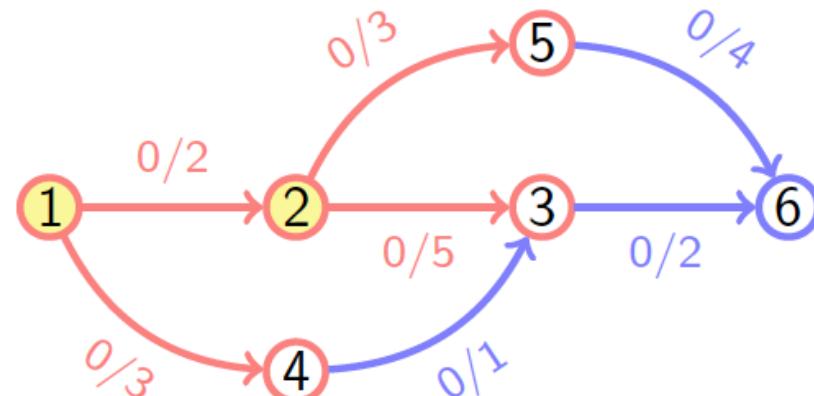
\emptyset



①

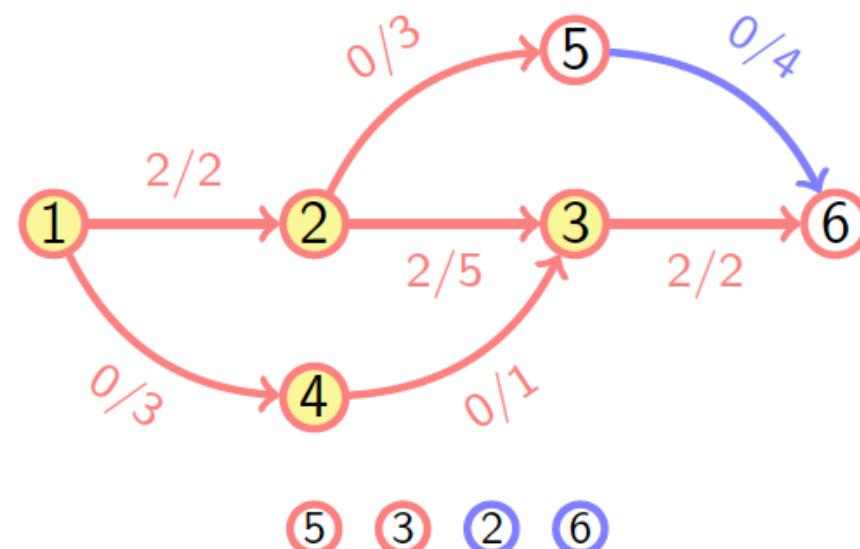
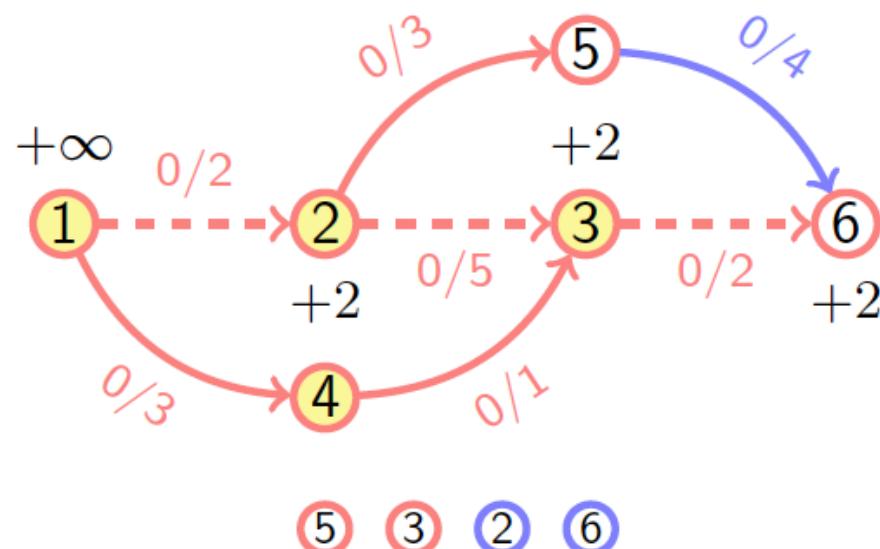
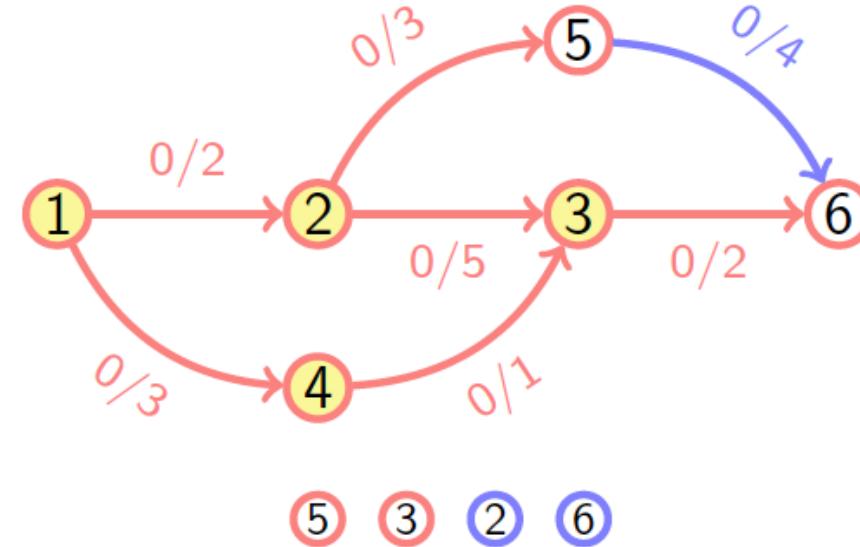
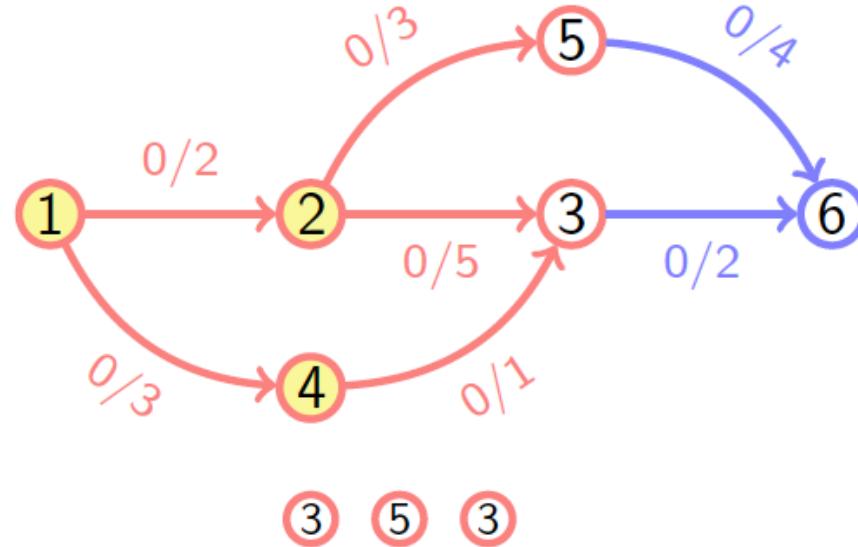


② ④

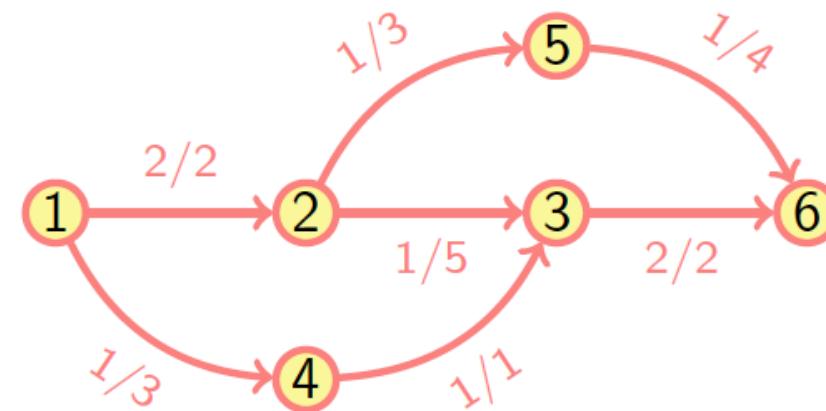
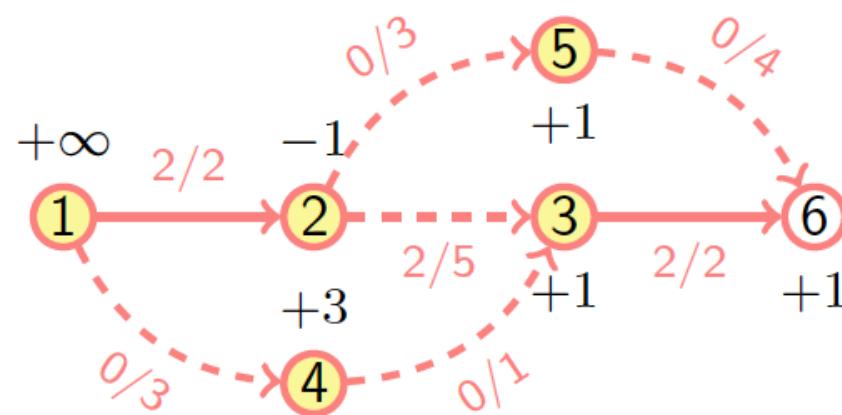
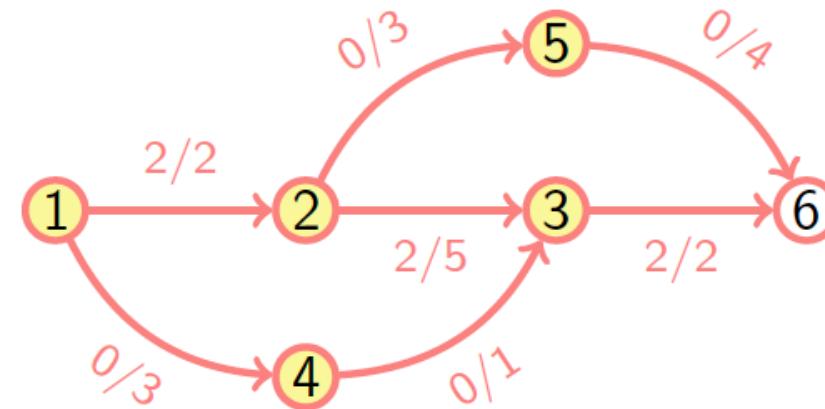
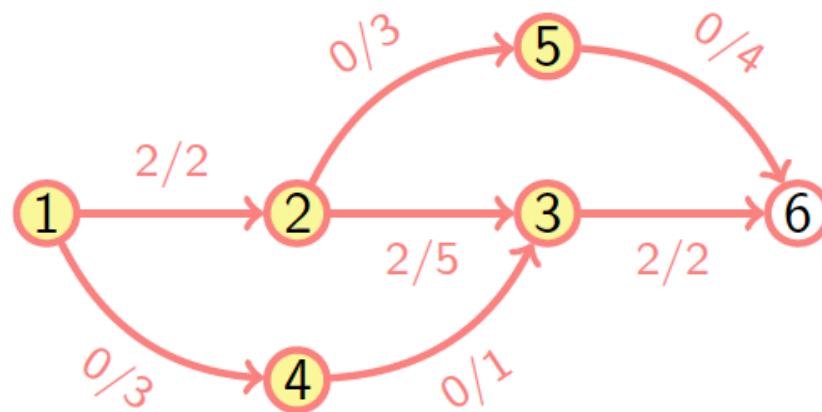


④ ③ ⑤

10.2 最短增益路径法——实例



10.2 最短增益路径法——实例



作业 4

- 阅读
 - 3.5, 9.3, 4.2, 10.2
- 习题
 - 9.3: 2, 4