Algorithm Design and Analysis

David N. JANSEN, Bohua ZHAN
名 姓

算法设计与分析

詹博华，杨大卫

Algorithm Design and Analysis

Elementary Graph Algorithms

David N. JANSEN

算法设计与分析

基本的图算法

杨大卫

Graph

- A graph consists of vertices[1] / nodes / points V and edges / lines / links $E \subseteq V \times V$.

- used for: relation
  map
  network
  process
  program state

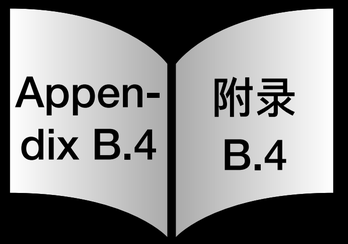
[1] The English word has an irregular plural: one vertex, many vertices.

图

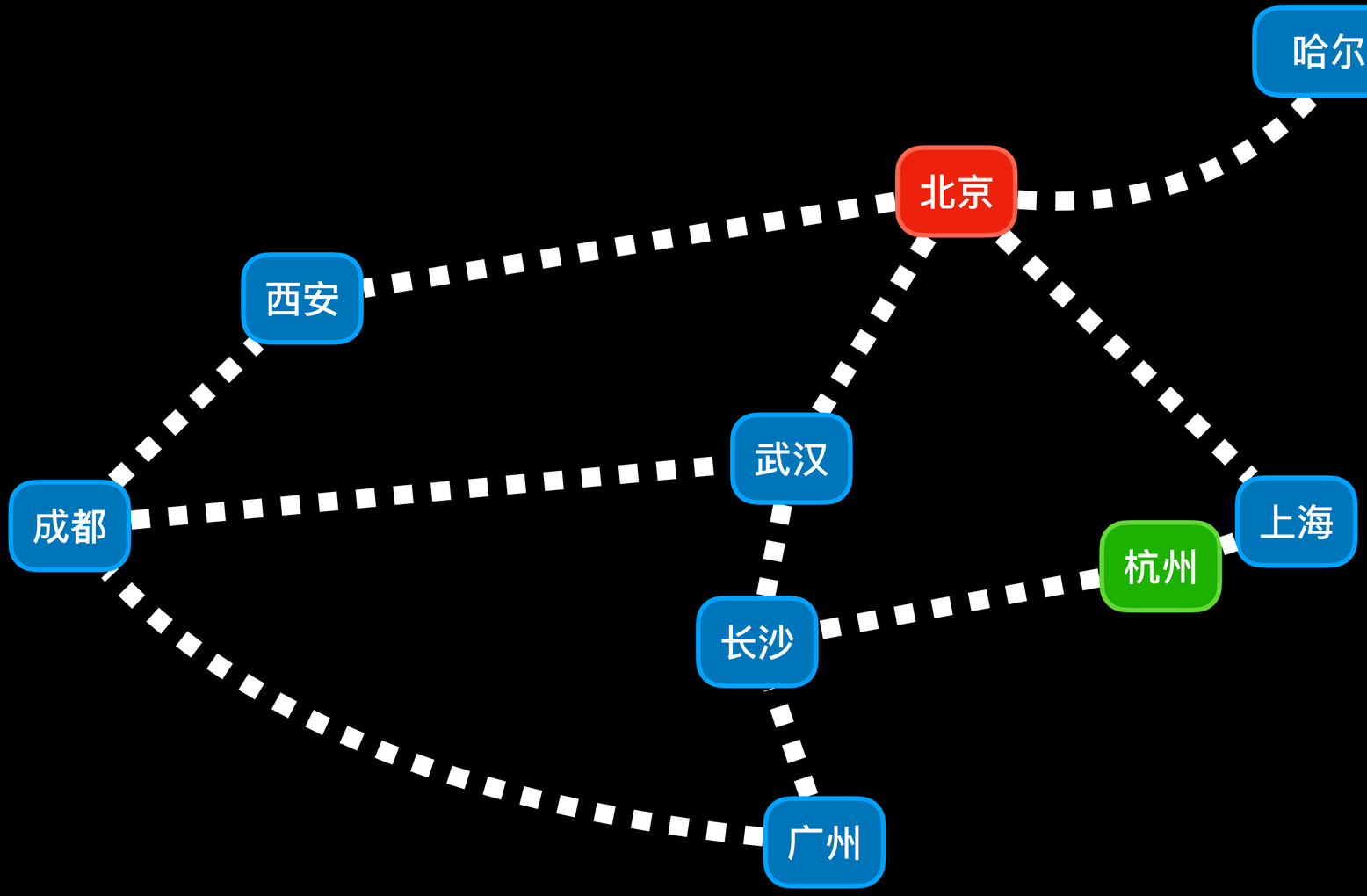
- 图存在于顶点/结点/节点/角顶 V 和边 $E \subseteq V \times V$。

- 关系
  地图
  网络
  过程
  程序状态

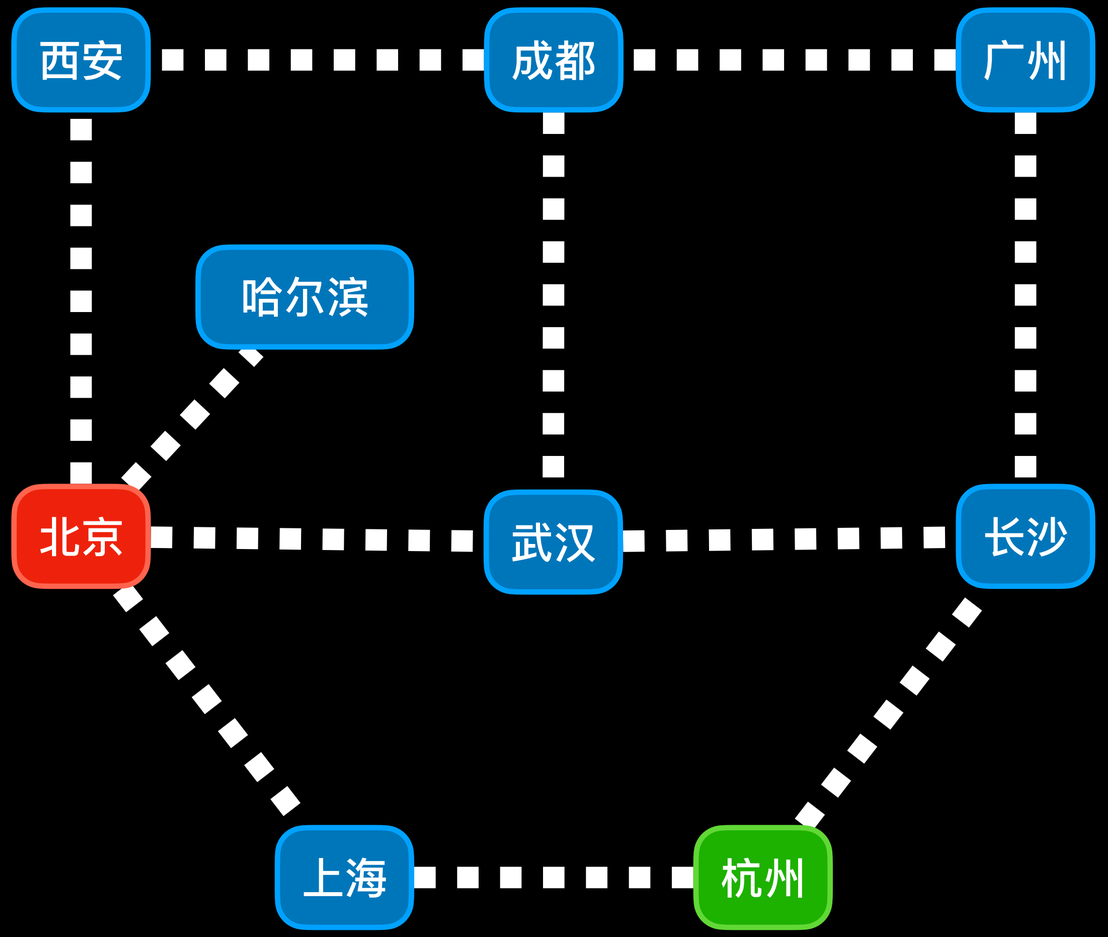
[1] 英文词的复数不规则的：单数vertex，复数vertices。



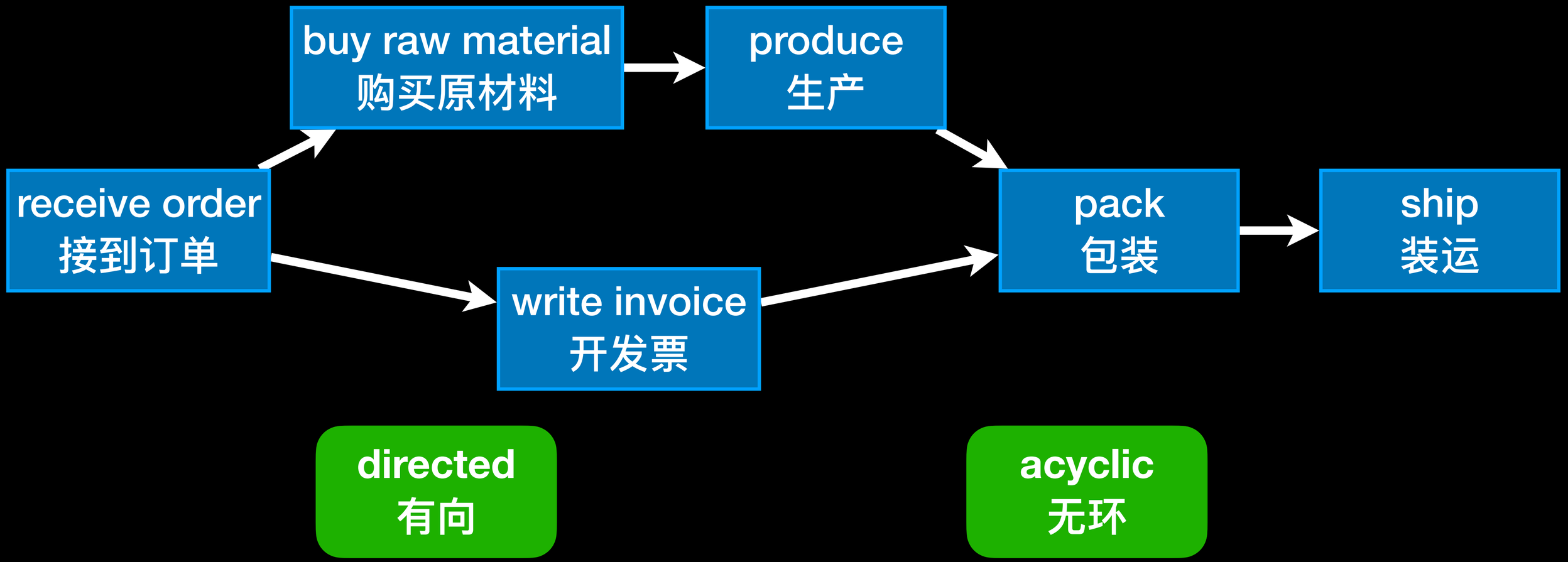
Graph Example 示例图



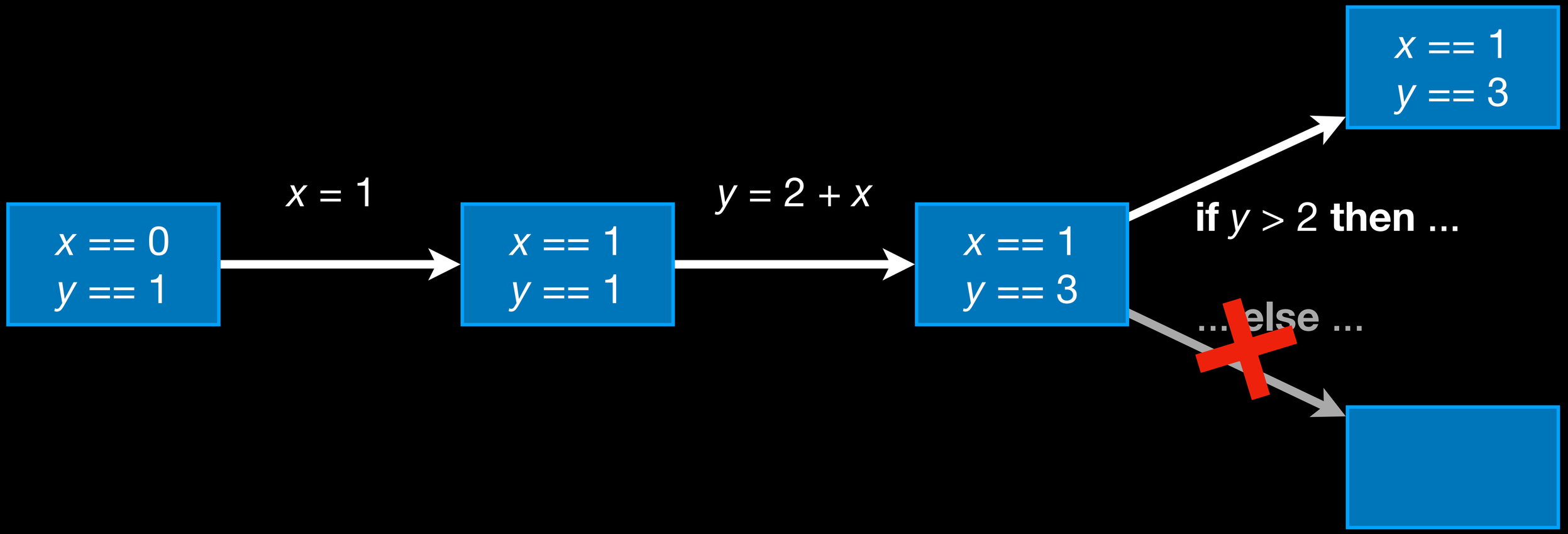
Graph Example 示例图



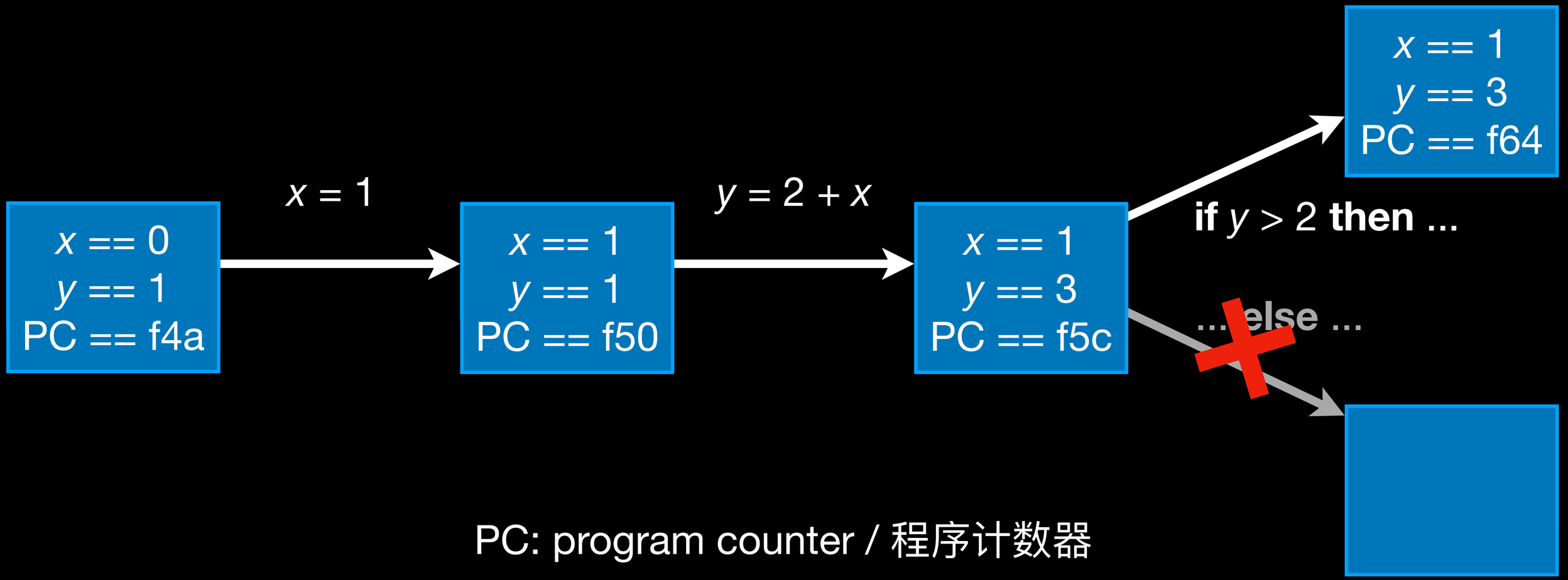
Graph Example 示例图



Graph Example 示例图



Graph Example 示例图



PC: program counter / 程序计数器

more graph terms

- directed / undirected
- path, length of a path ⇝
- connected, strongly connected
- cycle
- sparse graph $|E| \ll |V|^2$
  dense graph $|E|$ in $\Omega(|V|^2)$

更多图形术语

- 有向/无向
- 路径，路径的长度 ⇝
- 连通，强连通
- 环路
- 稀疏图 $|E| \ll |V|^2$
  稠密图 $|E|$ in $\Omega(|V|^2)$

Quiz

1. Which three methods of amortized analysis did I mention yesterday?

2. What is the difference between “amortize” in bookkeeping and in amortized analysis of algorithms?

3. Explain the word “credit” for amortized analysis.

测验

1. 我昨天提到了哪三种摊还分析方法?

2. 簿记中的“摊还”与
算法的摊还分析中的“摊销”
有什么区别?

3. 解释摊还分析中的“信用”一词。

Quiz

4. We looked at dynamic tables.
   Why was it impossible to keep the load factor ≥½
   when introducing TABLE-DELETE?

测验

4. 我们讨论了动态表。
   为什么不可能保持装载因子≥½
   引入TABLE-DELETE时?

Overview

- representation of graphs
  (data structures for graphs)
- breadth-first search
- depth-first search
  - topological sort
  - strongly connected components
- minimum spanning tree

概述

- 图的表示
  （图的数据结构）
- 广度优先搜索
- 深度优先搜索
  - 拓扑排序
  - 强连通分量
- 最小生成树

Representing graphs

How to store a graph in a computer

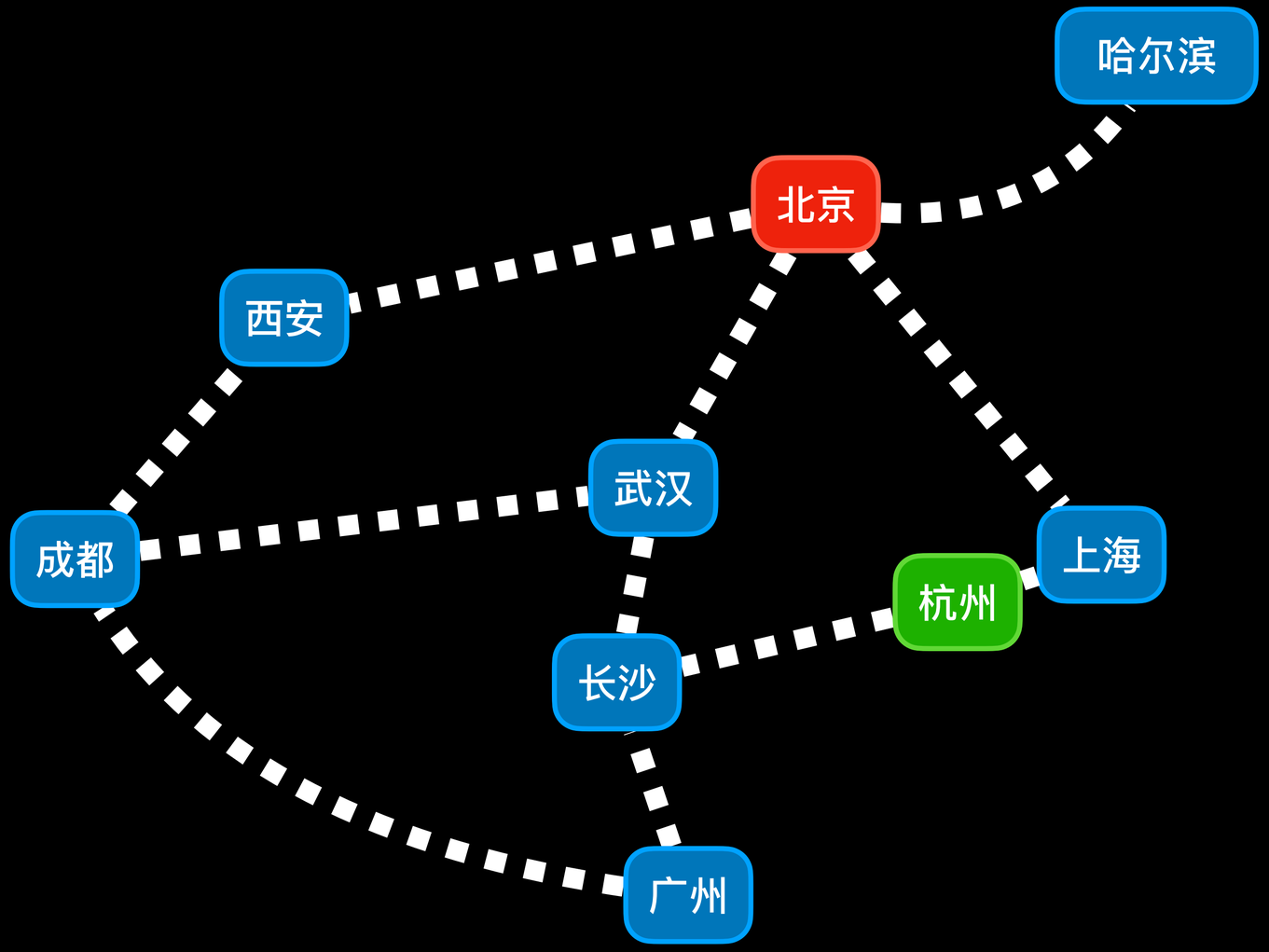
- **Adjacency list** = $|V|$ lists of edges, one per vertex
- **Adjacency matrix** = a matrix of bits, indicating for each pair of vertices whether they have an edge
- can be adapted to weighted graphs (Chapters 24/25)

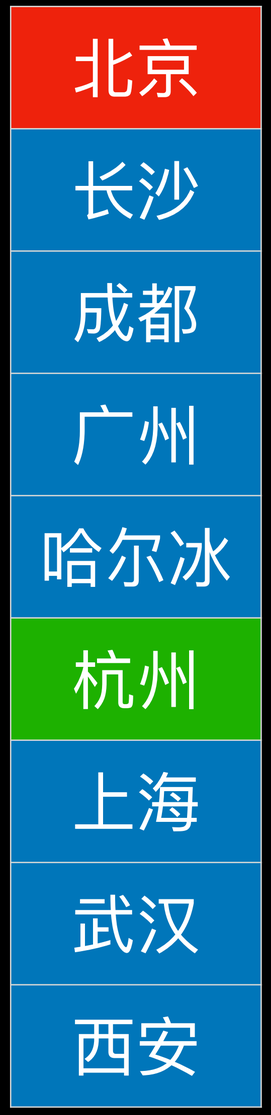
图的表示

- 如何在计算机中存储图形
- **邻接链表** = $|V|$条边的链表，每个结点有一条
- **邻接矩阵** = 位矩阵，指示每对顶点是否有边
- 可以适应于权重图（24/25章）

Example: Adjacency List

例子：邻接链表

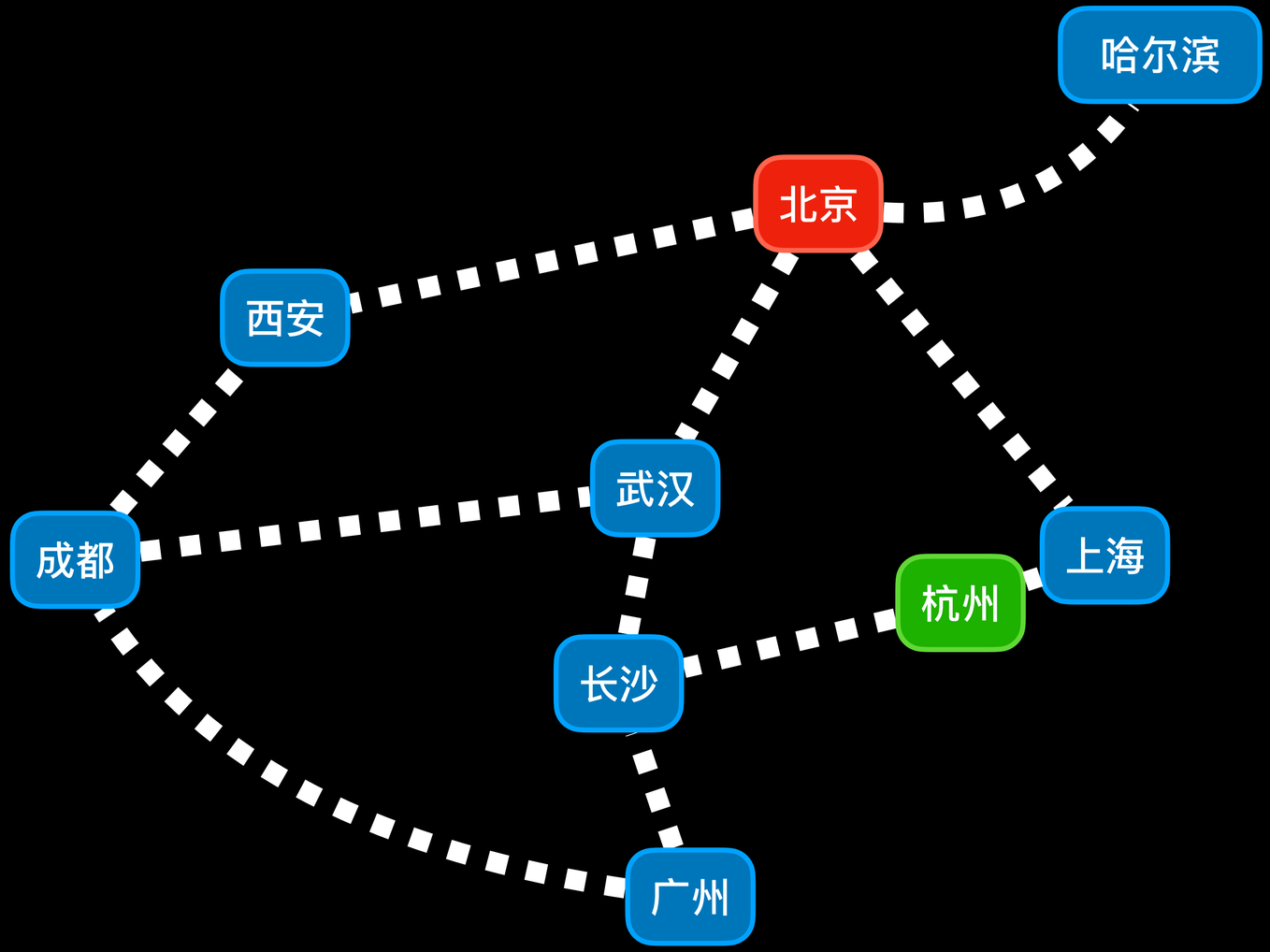


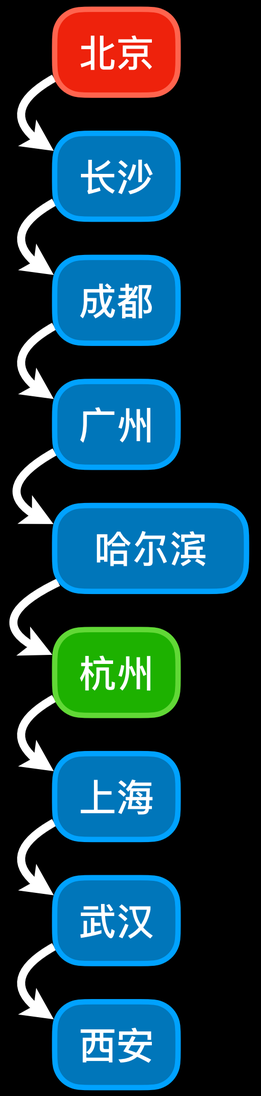


basic version:
array of vertices

Example: Adjacency List

例子：邻接链表

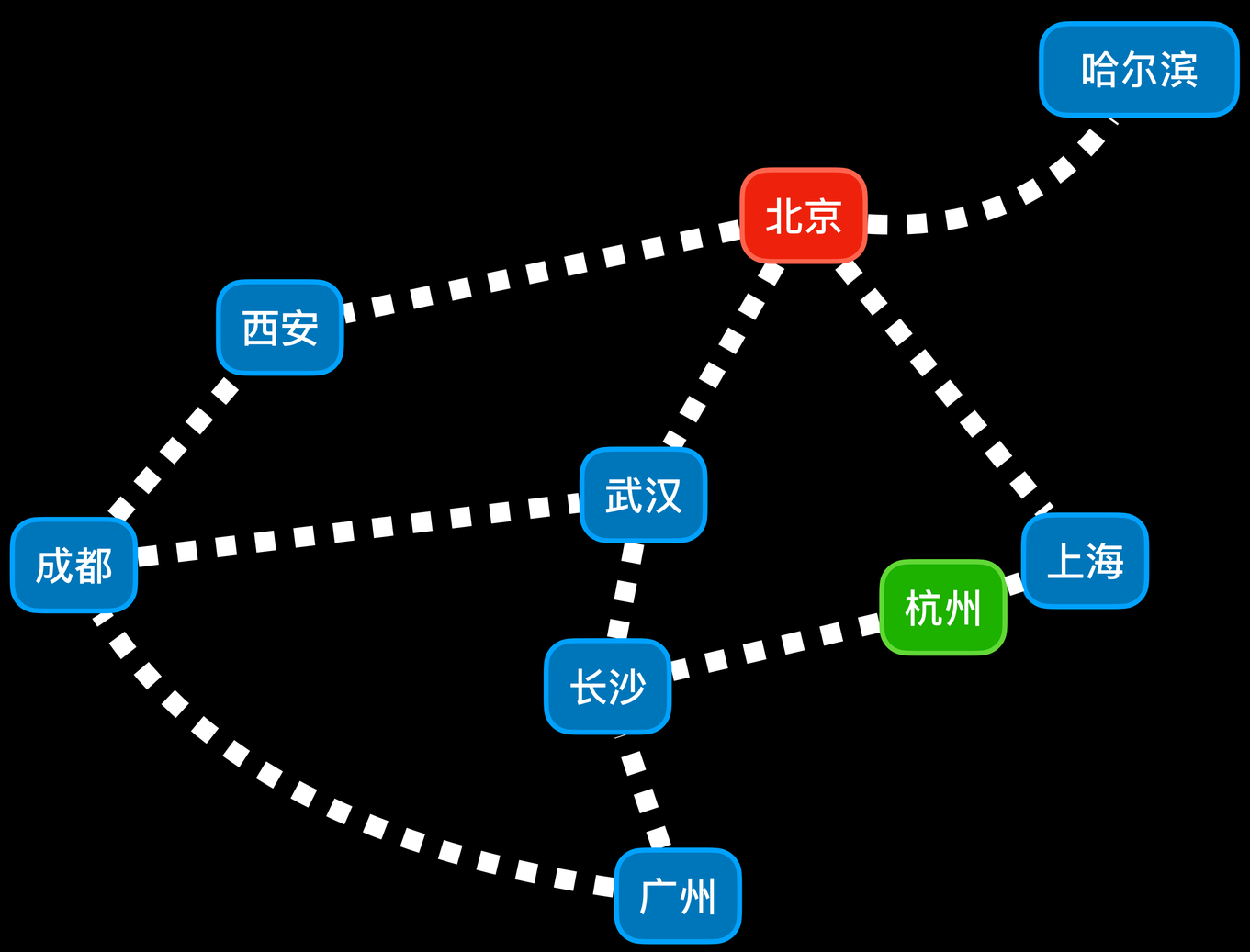


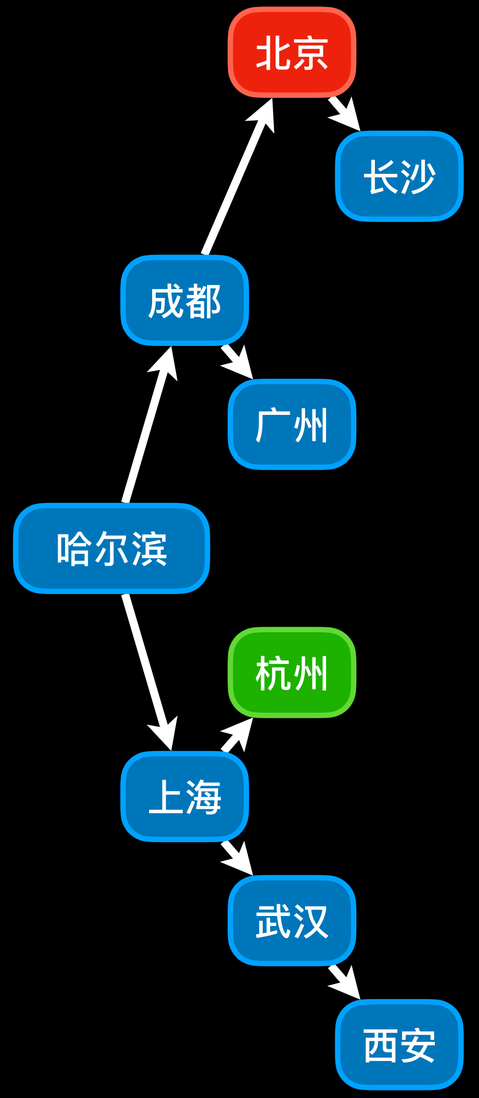


variant:
list of vertices

Example: Adjacency List

例子：邻接链表

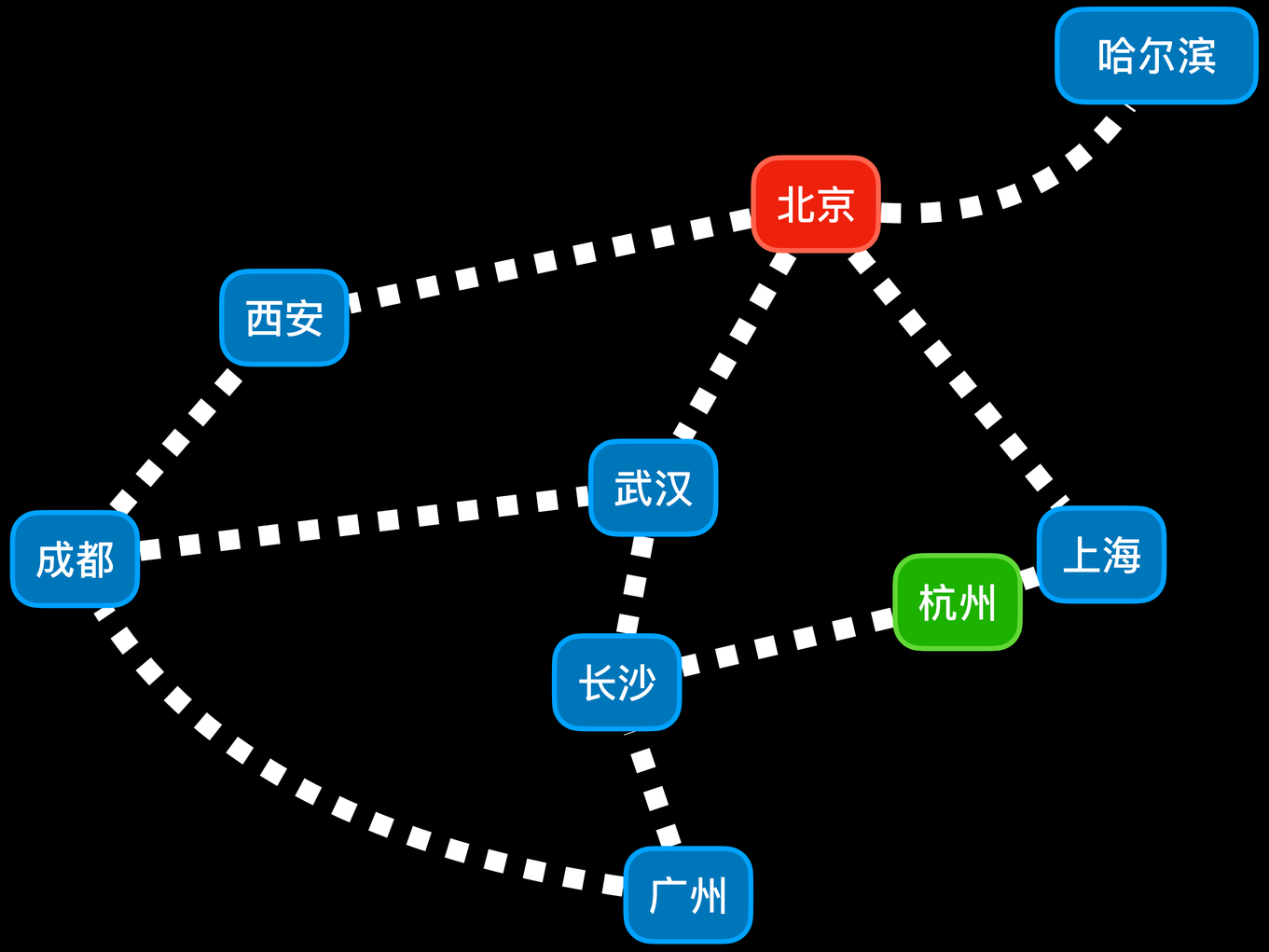


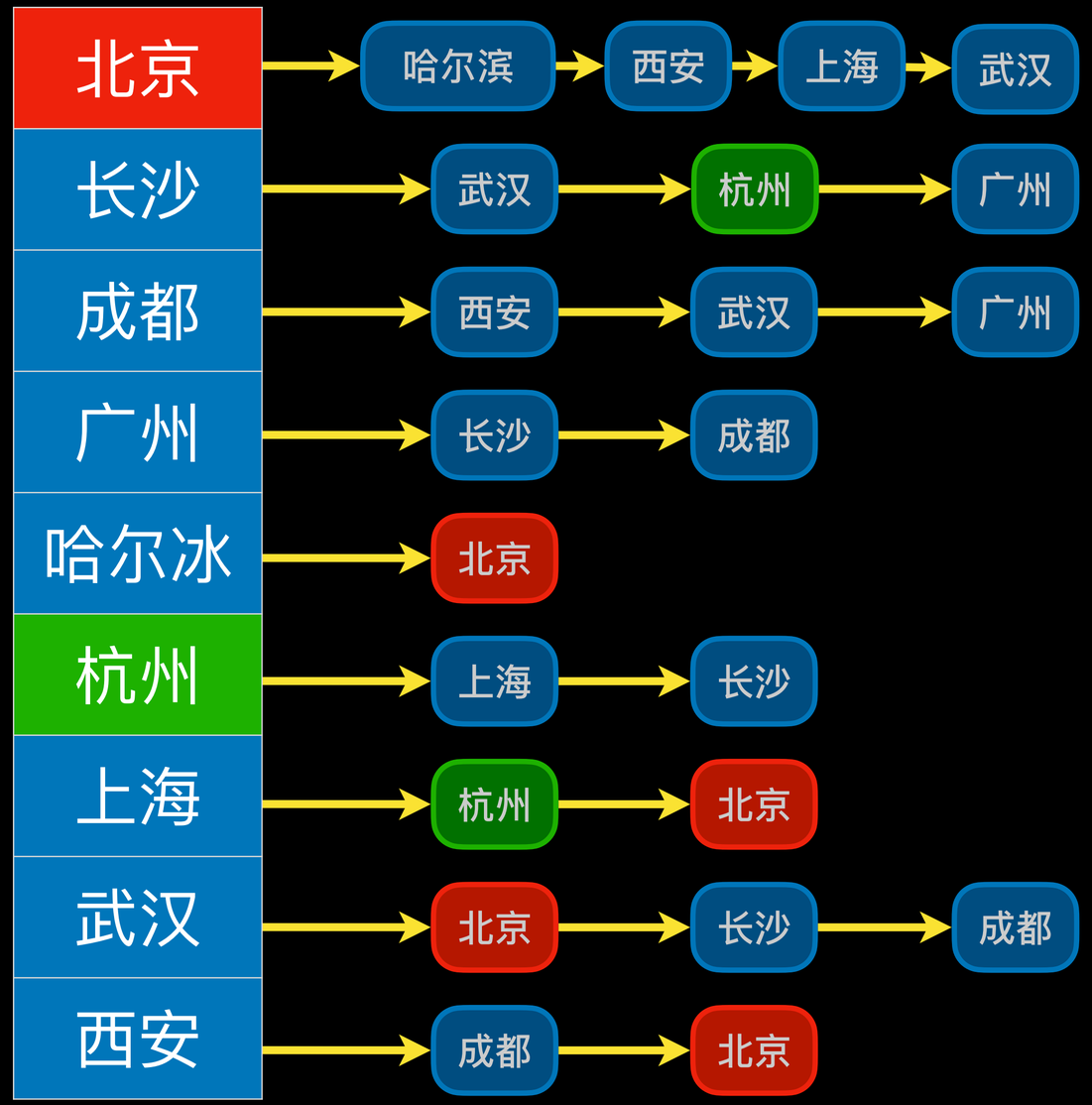


variant:
search tree
of vertices

Example: Adjacency List

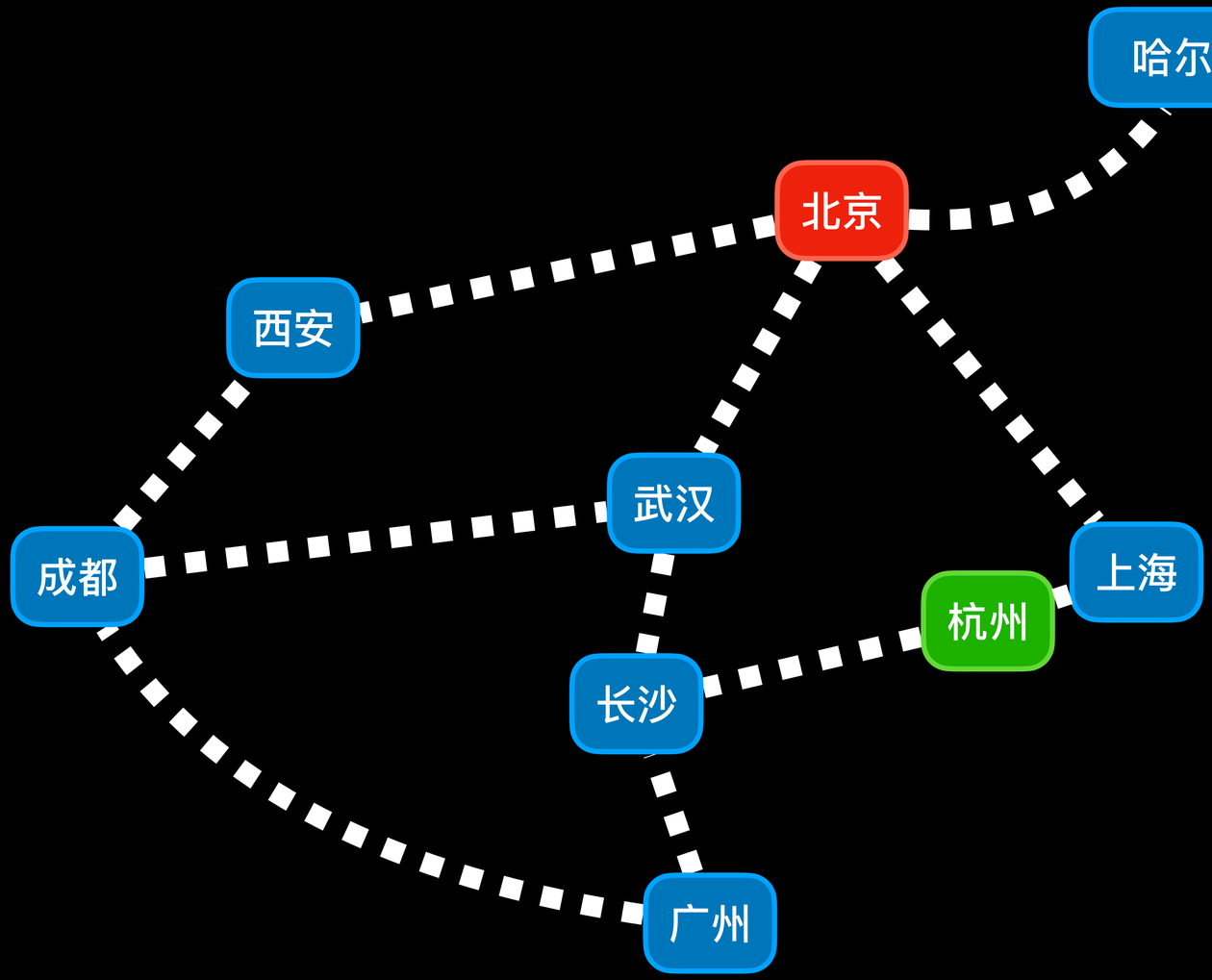
例子：邻接链表





Example: Adjacency Matrix

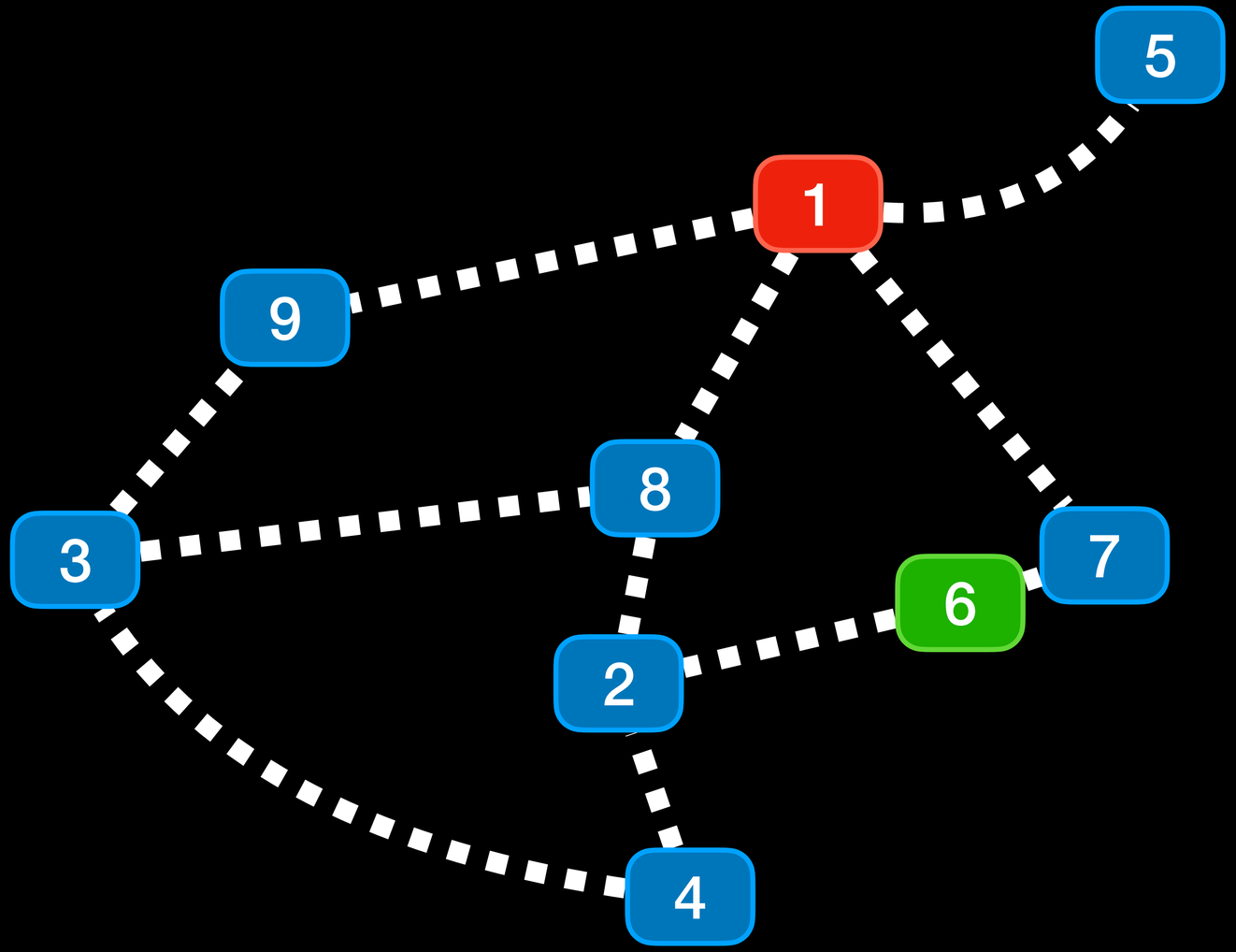
例子：邻接矩阵



	北京	长沙	成都	广州	哈尔滨	杭州	上海	武汉	西安
北京	0	0	0	0	1	0	1	1	1
长沙	0	0	0	1	0	1	0	1	0
成都	0	0	0	1	0	0	0	1	1
广州	0	1	1	0	0	0	0	0	0
哈尔滨	1	0	0	0	0	0	0	0	0
杭州	0	1	0	0	0	0	1	0	0
上海	1	0	0	0	0	1	0	0	0
武汉	1	1	1	0	0	0	0	0	0
西安	1	0	1	0	0	0	0	0	0

Example: Adjacency Matrix

例子：邻接矩阵



0	0	0	0	1	0	1	1	1
0	0	0	1	0	1	0	1	0
0	0	0	1	0	0	0	1	1
0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0

Representing graphs

- When to use which data structure
  [let students propose]
- The adjacency list is suitable for large sparse graphs.
- The adjacency matrix is suitable for small graphs and for dense graphs.

图的表示

- 什么情况使用什么表示?
- 邻接链表合适大的稀疏图
- 邻接矩阵合适小的图与稠密图

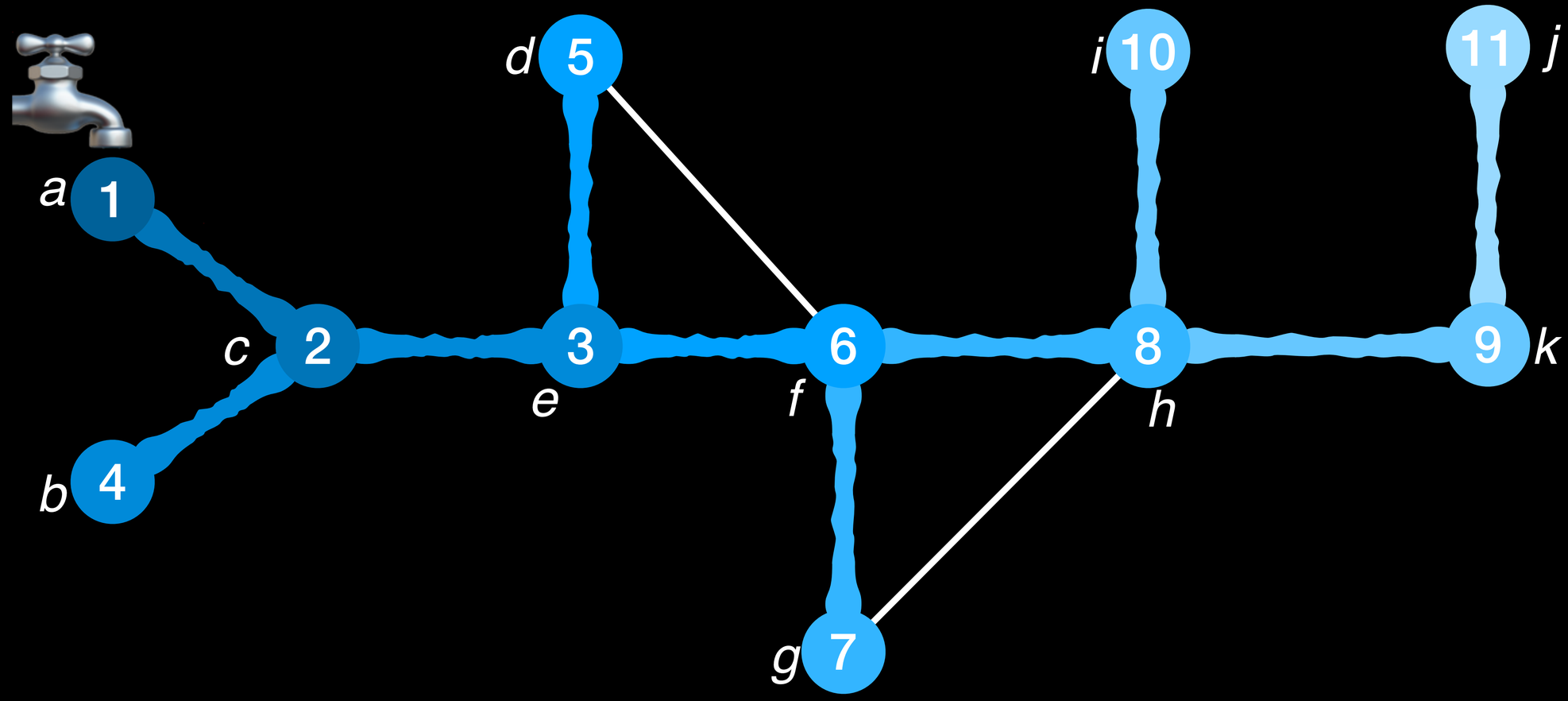
Breadth-first Search

- method to visit every vertex of a graph
- visit vertices close to source first
- image: a river overflowing

广度优先搜索

- 访问图的每个顶点的方法
- 首先访问靠近源的顶点
- 想象一条河泛滥

Breadth-First Search 广度优先搜索



Input and Output

Input: graph G, source vertex s

Output: for every vertex v:

- $v.d$ = distance from s
  (∞ if there is no path from s to v)
- a shortest path from s to v
  (stored efficiently as:
  predecessor $v.\pi$ on the shortest path
  $s \rightsquigarrow v.\pi \rightarrow v$)

输入和输出

输入： 图G， 源顶点s

输出： 于每个顶点v：

- $v.d$ = s的距离
  （∞ 如果不可到达）
- 一条最短从s到v的路径
  （有效存储为：
  前导$v.\pi$ 在最短路径
  $s \rightsquigarrow v.\pi \rightarrow v$上）

Order of Examination

- Examine a vertex = find new neighbors
- Q: In which order are vertices examined?
  A: oldest vertices first — queue (FIFO)
- Q: How to avoid examining vertex twice?
  A: mark vertices with a color:
  white = not examined
  gray = waiting to be examined or examination in progress
  black = completely examined

检查顺序

- 检查顶点=查找新邻居
- 问：顶点的检查顺序是什么?
  答：最早的顶点优先 — 队列（FIFO）
- 问：如何避免检查顶点两次?
  答：使用颜色标记顶点：
  白色 = 未检查
  灰色 = 等待检查或正在检查
  黑色 = 完全检查

Breadth-first search 广度优先搜索

initialize colors and queue

find vertex to examine

examine vertex *u*

```
BFS(G,s)
for each vertex u ∈ G.V \ {s}
        u.color = white, u.d = ∞, u.π = NIL
s.color = gray, s.d = 0, s.π = NIL
Q = empty queue
ENQUEUE(Q,s)
while Q is not empty
        u = DEQUEUE(Q)
        for each v adjacent to u
                if v.color == white
                        v.color = gray, v.d = u.d + 1, v.π = u
                        ENQUEUE(Q,v)
        u.color = black
```

预置颜色和队列

找到检查的顶点

检查*u*顶点

Loop Invariant

- Q consists of the gray vertices, ordered by distance from s.
- If $v.color \neq$ white, then $v.d$ is the length of a shortest path from s to v, and v.π = [ask students to fill in]
- There exists a distance $d_0 \in \mathbb{N}$ such that:
  if $v.d < d_0$, then $v.color$ = black
  if $v.d = d_0$, then $v.color \in$ {black,gray}
  if $v.d = d_0 + 1$, then $v.color \in$ {gray,white}
  if $v.d > d_0 + 1$, then $v.color$ = white

循环不变量

- Q由灰色顶点组成，按与s的距离排序。
- 如果$v.color \neq$ white，那么$v.d$是从s到v的最短路径的长度，和$v.\pi$ = [清学生填写]
- 存在一个距离$d_0 \in \mathbb{N}$ 以便：
  如果$v.d < d_0$ ，则$v.color$ = black
  如果$v.d = d_0$ ，则$v.color \in$ {black, gray}
  如果$v.d = d_0+1$，则$v.color \in$ {gray, white}
  如果$v.d > d_0+1$，则$v.color$ = white

Partial Correctness

- **Theorem 22.5:** BFS discovers every vertex v that is reachable from the source s, and $v.d$ = the shortest-path distance from s to v. Moreover, for every vertex $v \neq s$ reachable from s, one of its shortest paths uses $v.\pi$ as its butlast vertex.

部分正确性

- 定理22.5: BFS发现
从源s可到达的每个顶点v,
$v.d$ = 从s到v的最短路径距离。
此外，对于每个可从s到达的顶点$v \neq s$,
其最短路径之一使用$v.\pi$
作为其最后一个顶点。

Proof of Theorem 22.5

Lemma 22.2: The value $v.d$ computed by BFS satisfies $v.d \geq \delta(s,v)$.

Proof by induction over the number of ENQUEUE operations. Central step:
At the moment when $v.d$ is changed,
its new value is $u.d + 1 \geq \delta(s,u) + 1$,
which is $\geq \delta(s,v)$.

定理22.5的证明

引理 22.2: BFS所计算出的$v.d$满足$v.d \geq \delta(s,v)$.

证明：ENQUEUE操作的次数进行归纳。
主要步骤：
当$v.d$发生变化时，
它的新值是$u.d + 1 \geq \delta(s,u) + 1 \geq \delta(s,v)$.

Proof of Theorem 22.5

Proof by contradiction.
If there exists a vertex v with $v.d \neq \delta(s,v)$,
then there exists one with minimal $\delta(s,v)$.

By Lemma 22.5, we have $v.d > \delta(s,v)$.
Let $s \rightsquigarrow u \rightarrow v$ be a shortest path from s to v and u the last node visited before v.

Then we must have
$v.d \leq u.d + 1 = \delta(s,u) + 1 = \delta(s,v)$.

定理22.5的证明

用矛盾来证明。
如果存在顶点v满足$v.d \neq \delta(s,v)$,
则存在一个$\delta(s,v)$最小的。

通过引理22.5，我们得到了$v.d > \delta(s,v)$。
让我们$s \rightsquigarrow u \rightarrow v$是从$s$到$v$的最短路径，
u是v之前访问的最后一个顶点。

那我们一定有
$v.d \leq u.d + 1 = \delta(s,u) + 1 = \delta(s,v)$。

Running Time 运行时间

```
BFS(G,s)
for each vertex u ∈ G.V \ {s}
        u.color = white, u.d = ∞, u.π = NIL
s.color = gray, s.d = 0, s.π = NIL
Q = empty queue
ENQUEUE(Q,s)
while Q is not empty
        u = DEQUEUE(Q)
        for each v adjacent to u
                if v.color == white
                        v.color = gray, v.d = u.d + 1, v.π = u
                        ENQUEUE(Q,v)
        u.color = black
```

O(|V|)

O(|out(u)|)

every vertex is enqueued ≤1×

每个顶点入队≤1次

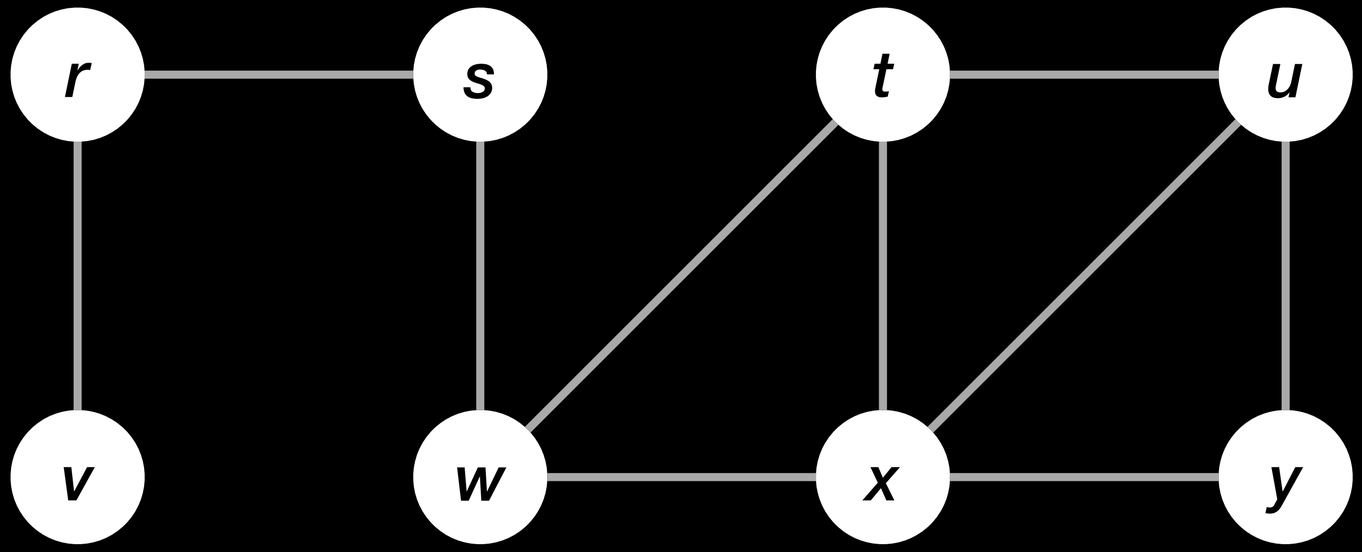
Running Time

- Initialization requires time $O(|V|)$.
- The main loop is executed $\leq |V|$ times.
- The inner loop is, in total, executed $O(|E|)$ times.
- The FIFO queue operations ENQUEUE and DEQUEUE require time $O(1)$ per call.
- So, the total running time is in $O(|V|+|E|)$, and the algorithm terminates always.

运行时间

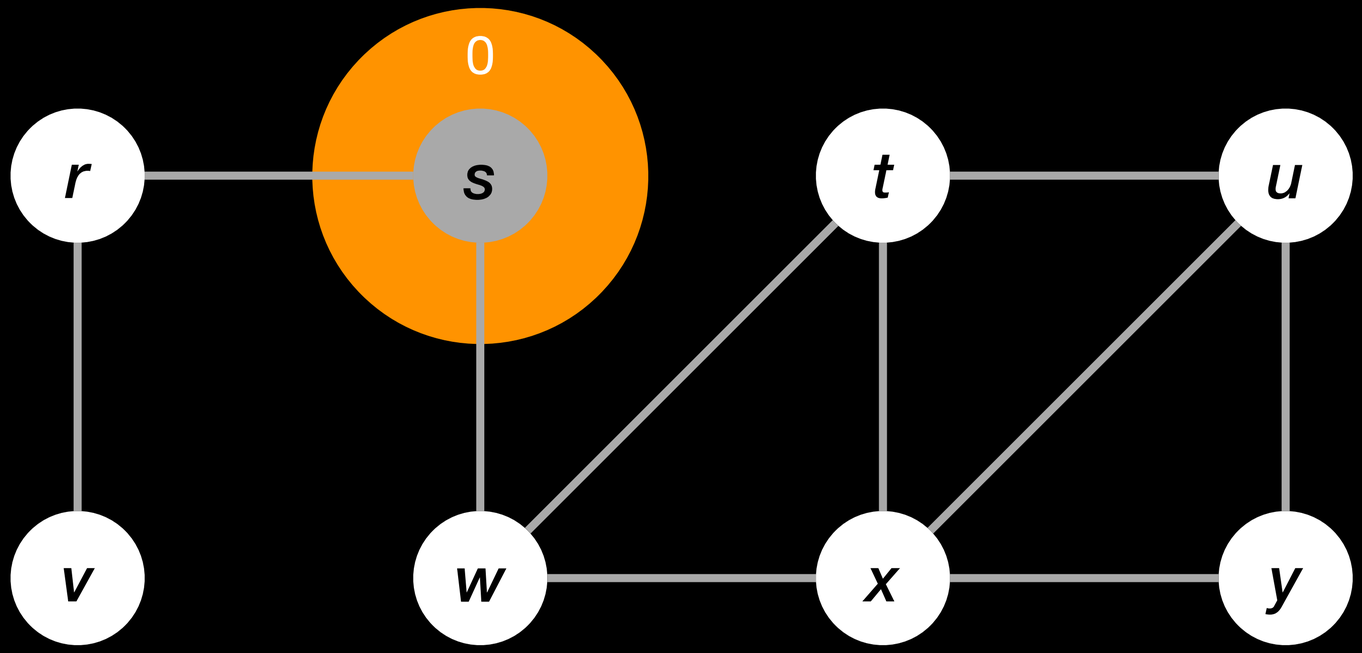
- 预置需要$O(|V|)$的时间。
- 主循环执行 $\leq |V|$次。
- 内部循环，总的来说，执行了 $O(|E|)$次。
- FIFO队列操作的ENQUEUE和DEQUEUE要求每次调用的时间为$O(1)$。
- 因此，总运行时间以$O(|V|+|E|)$内，并且算法每次都终止。

Example 例子

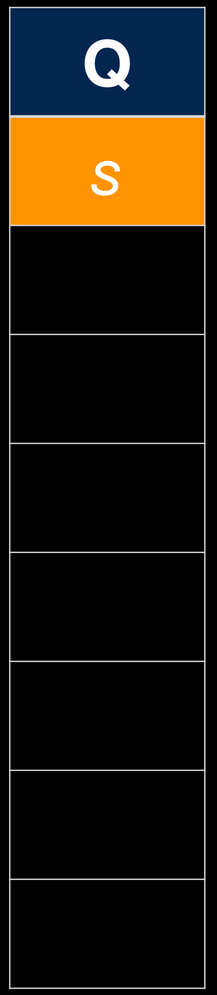


Example 例子

BFS(*G*,*s*)

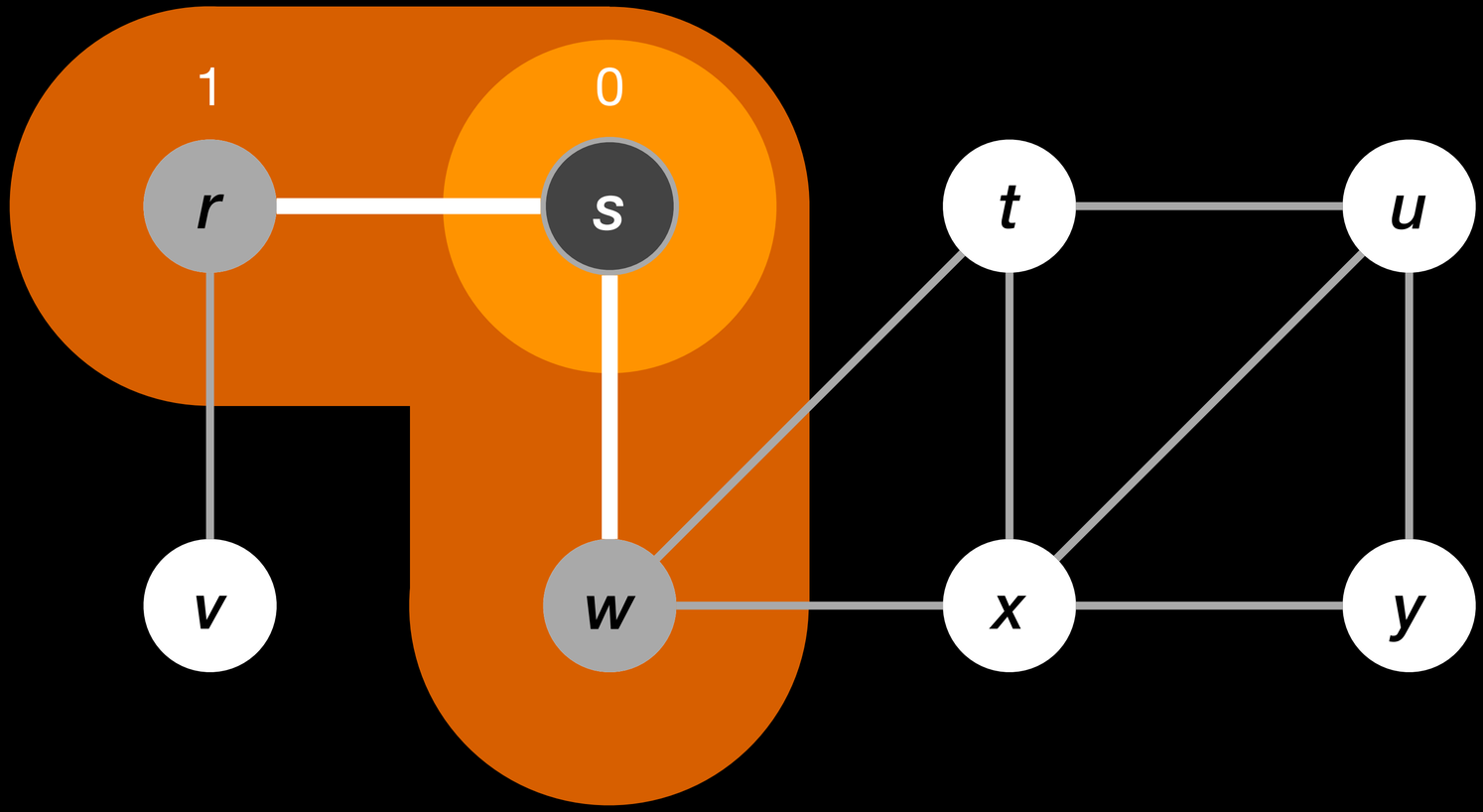


∞

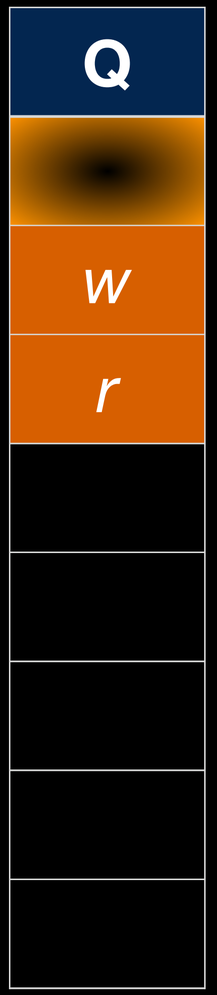


Example 例子

BFS(*G*,*s*)

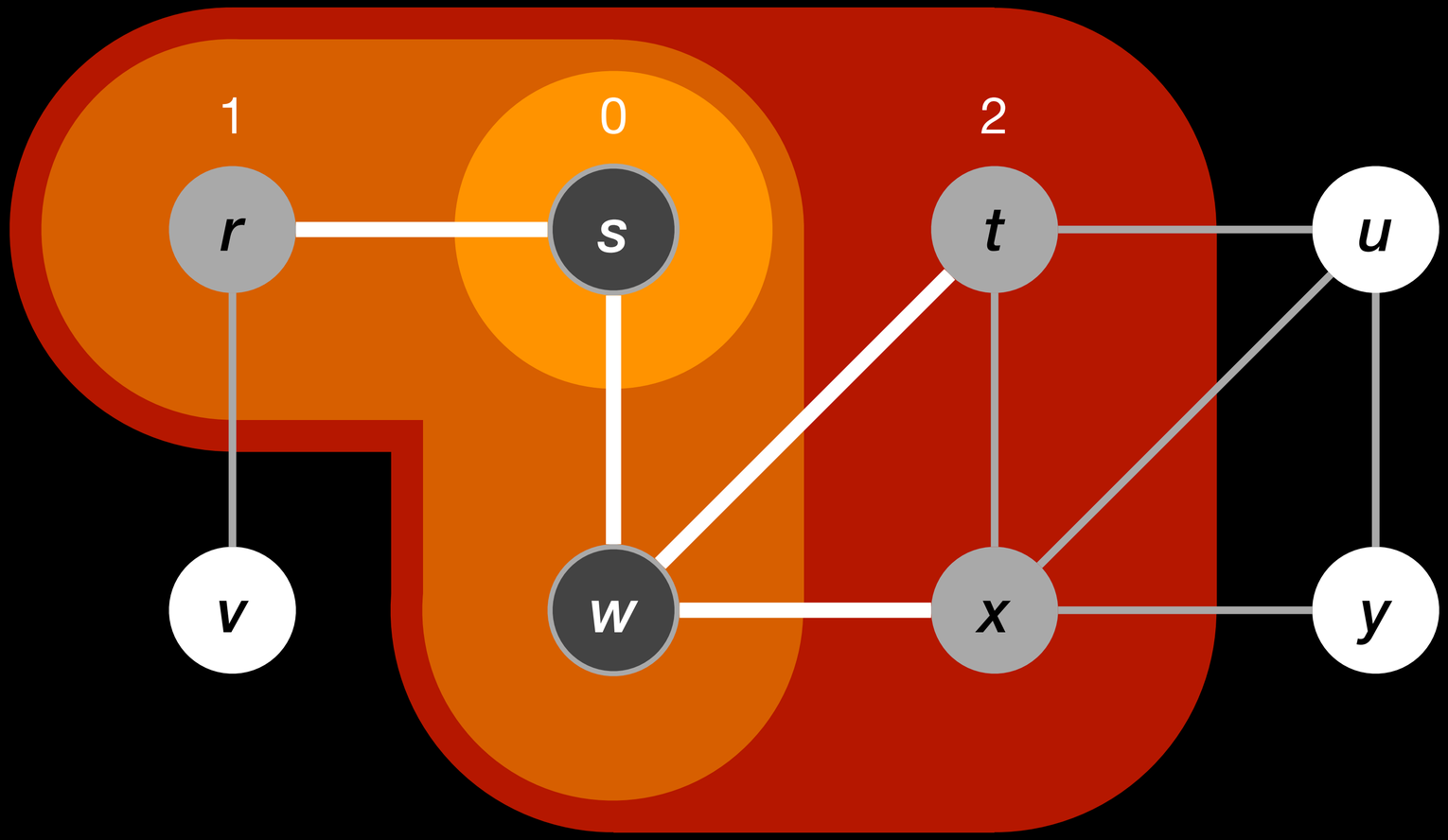


∞

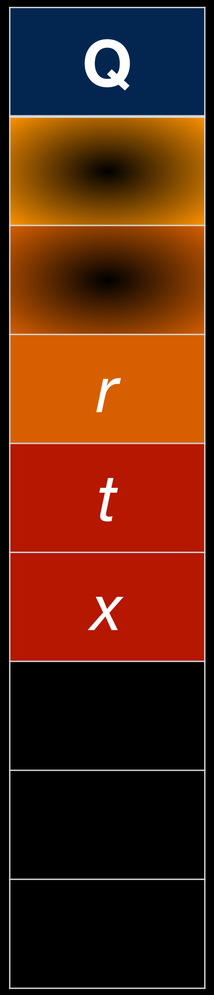


Example 例子

BFS(*G*,*s*)

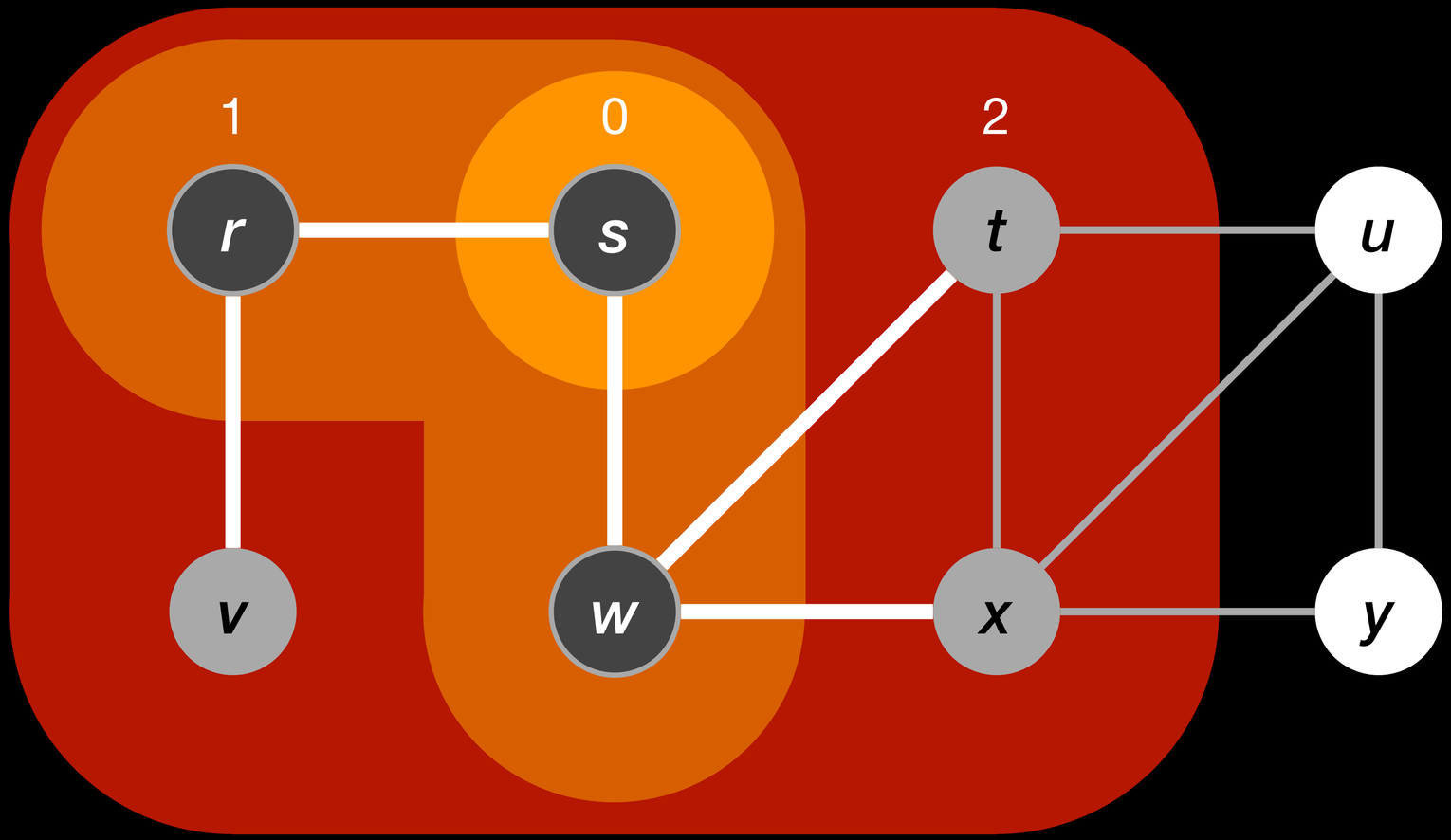


∞

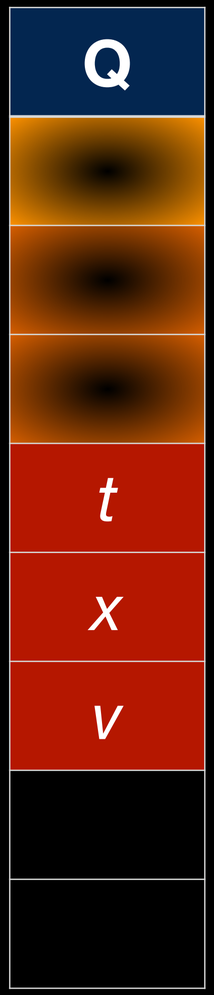


Example 例子

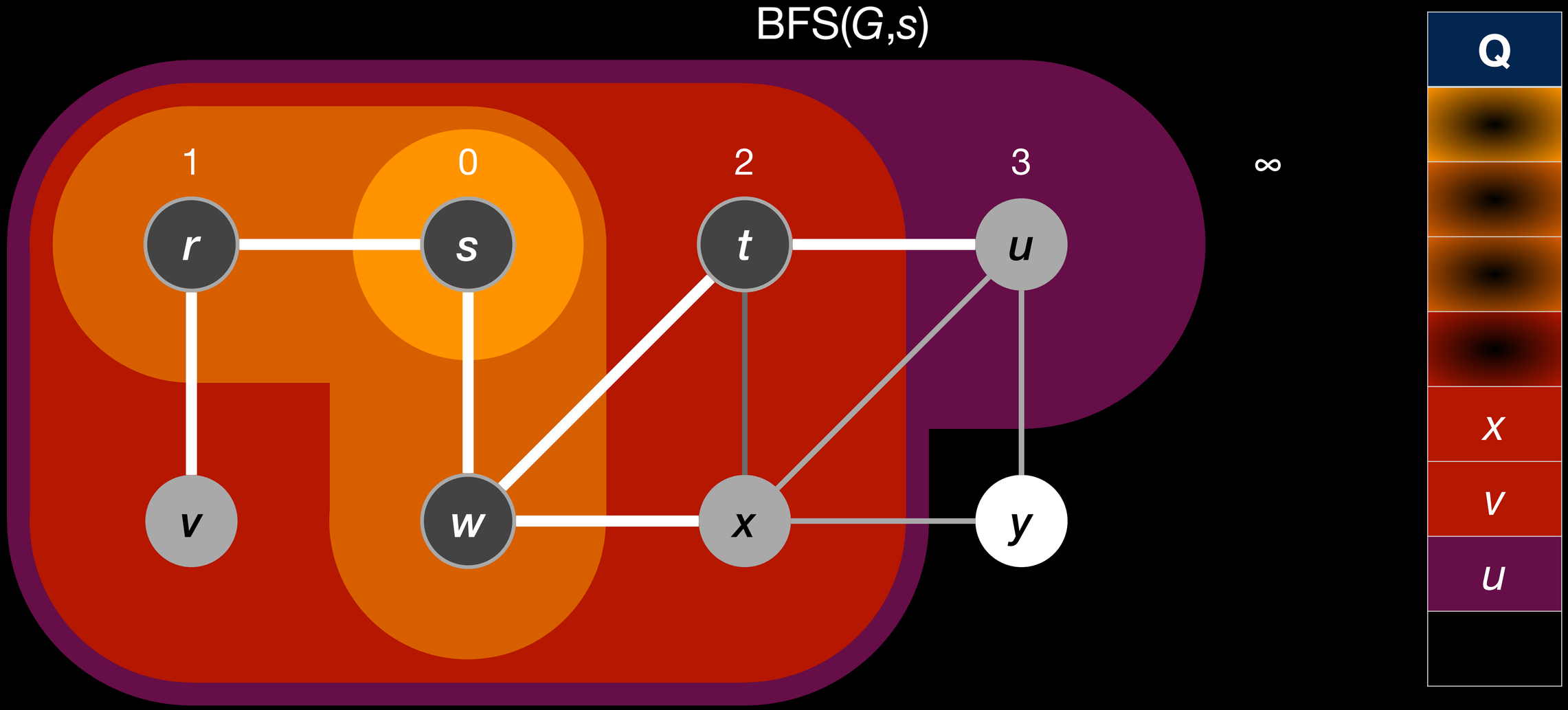
BFS(*G*,*s*)



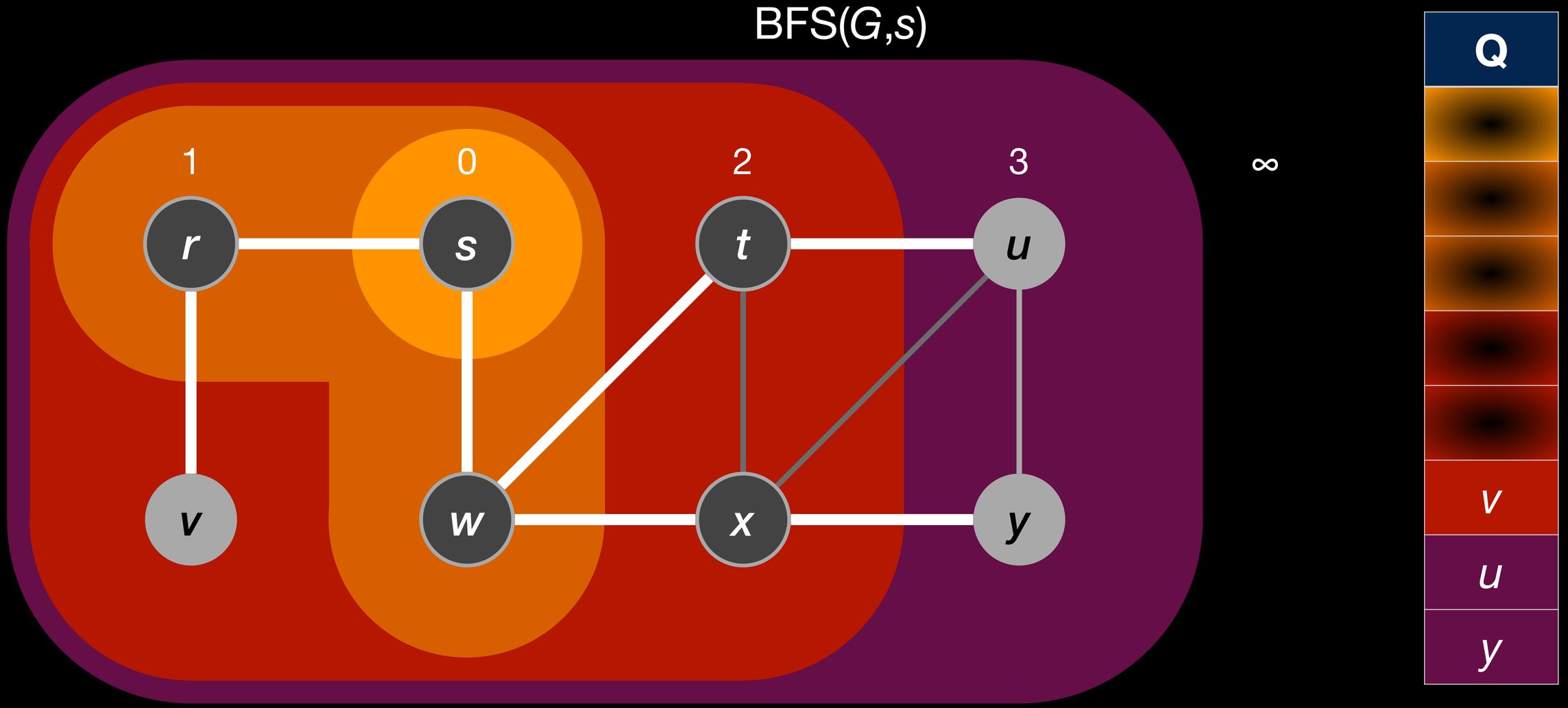
∞



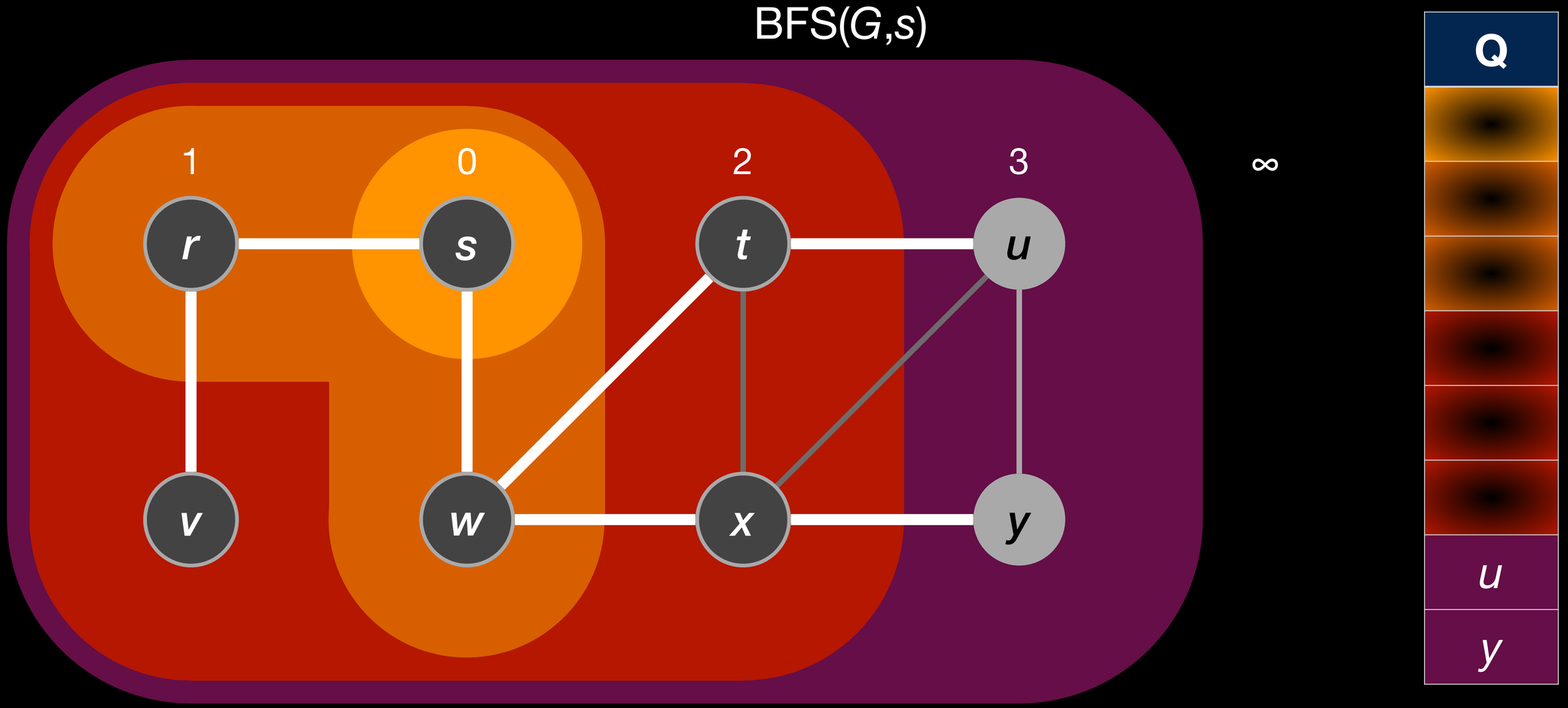
Example 例子



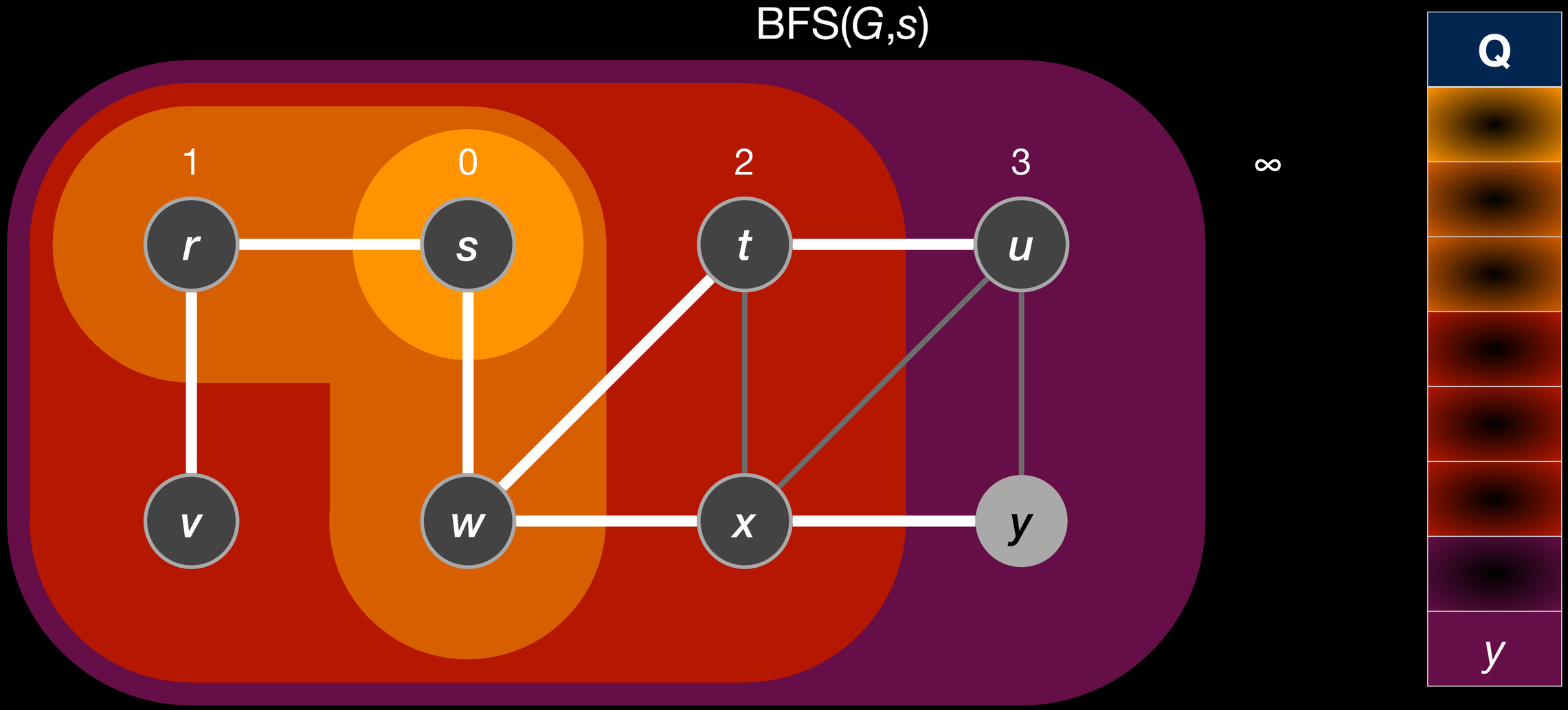
Example 例子



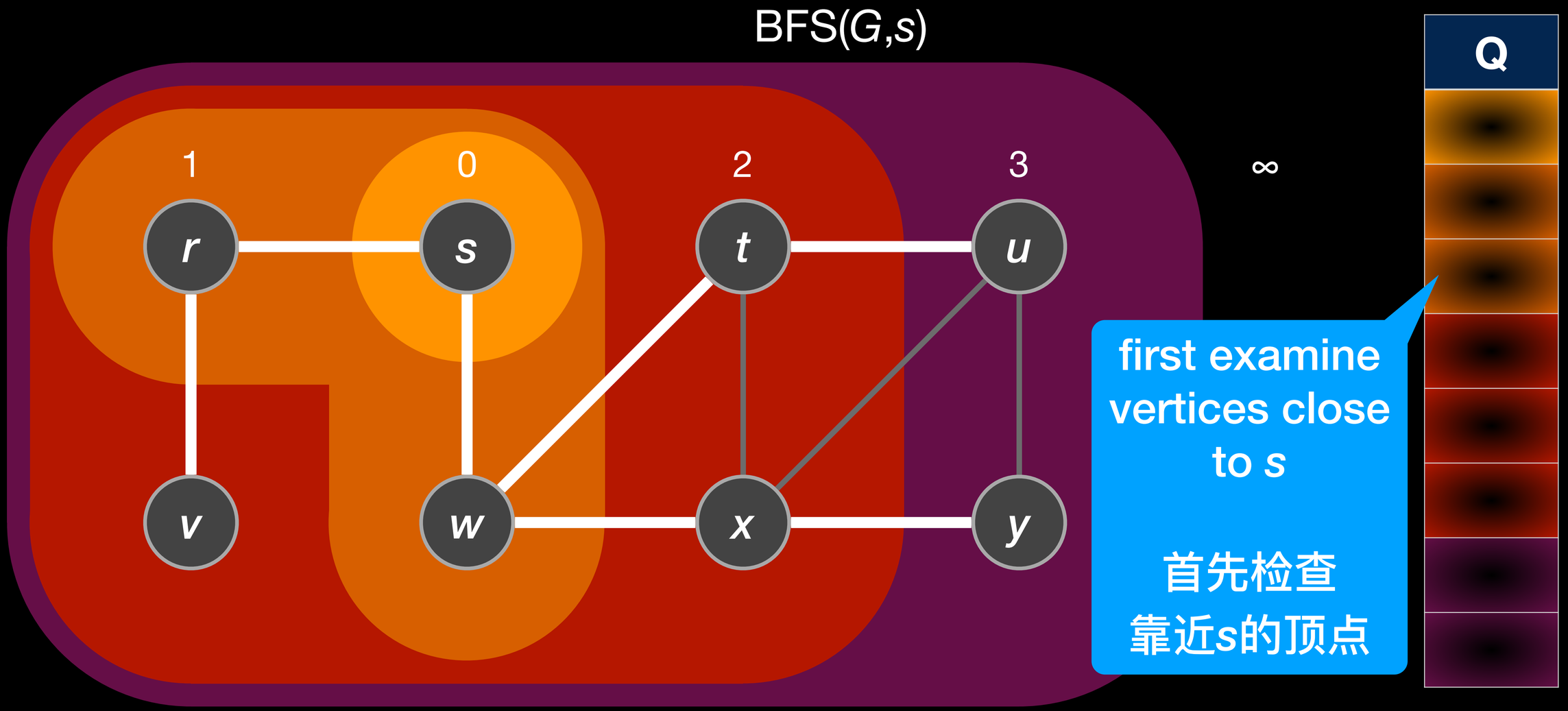
Example 例子



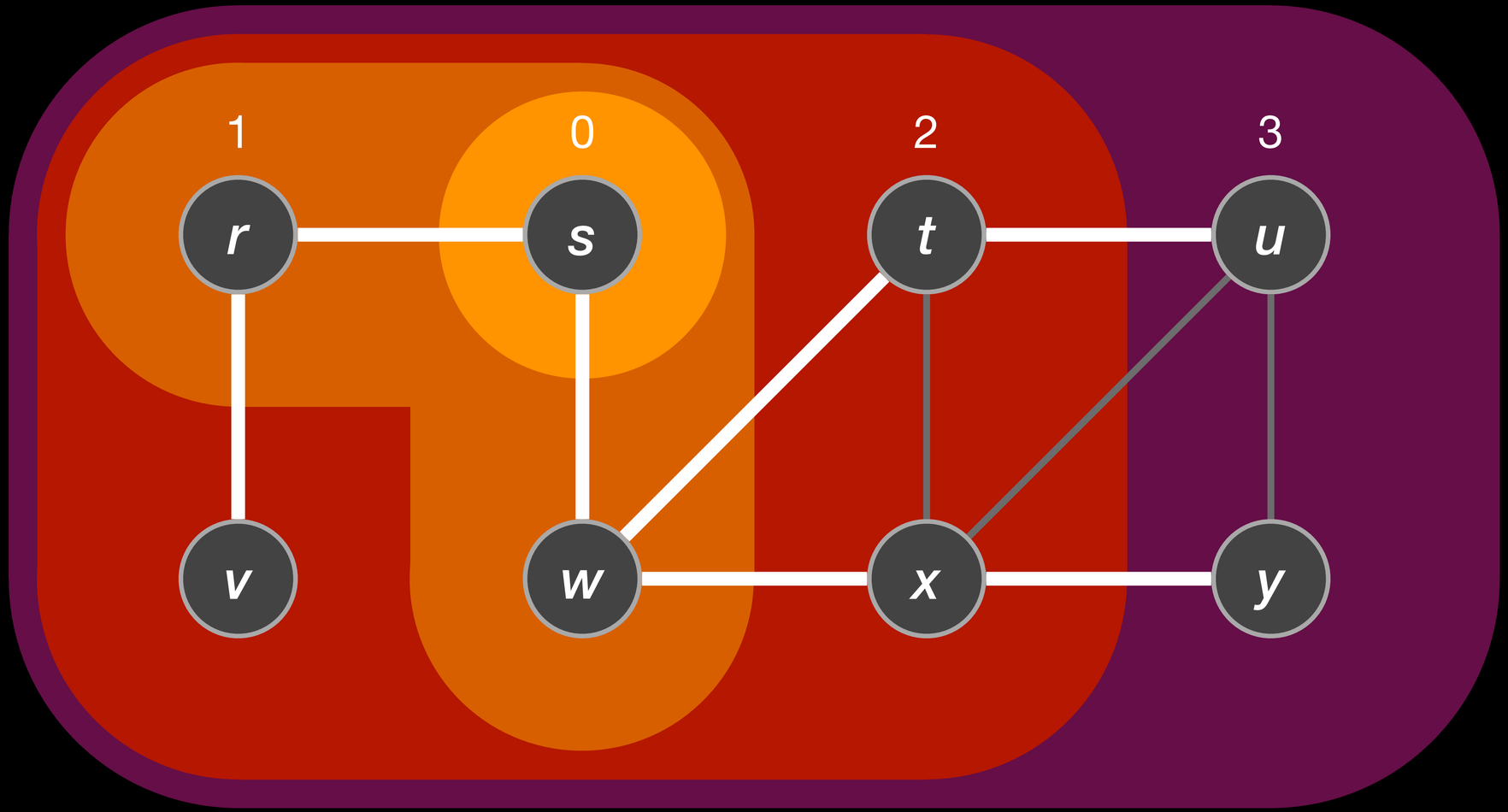
Example 例子



Example 例子


BFS(G,s)
1
0
2
3
∞
r
s
t
u
v
w
x
y
Q
first examine vertices close to s
首先检查靠近s的顶点

Example 例子



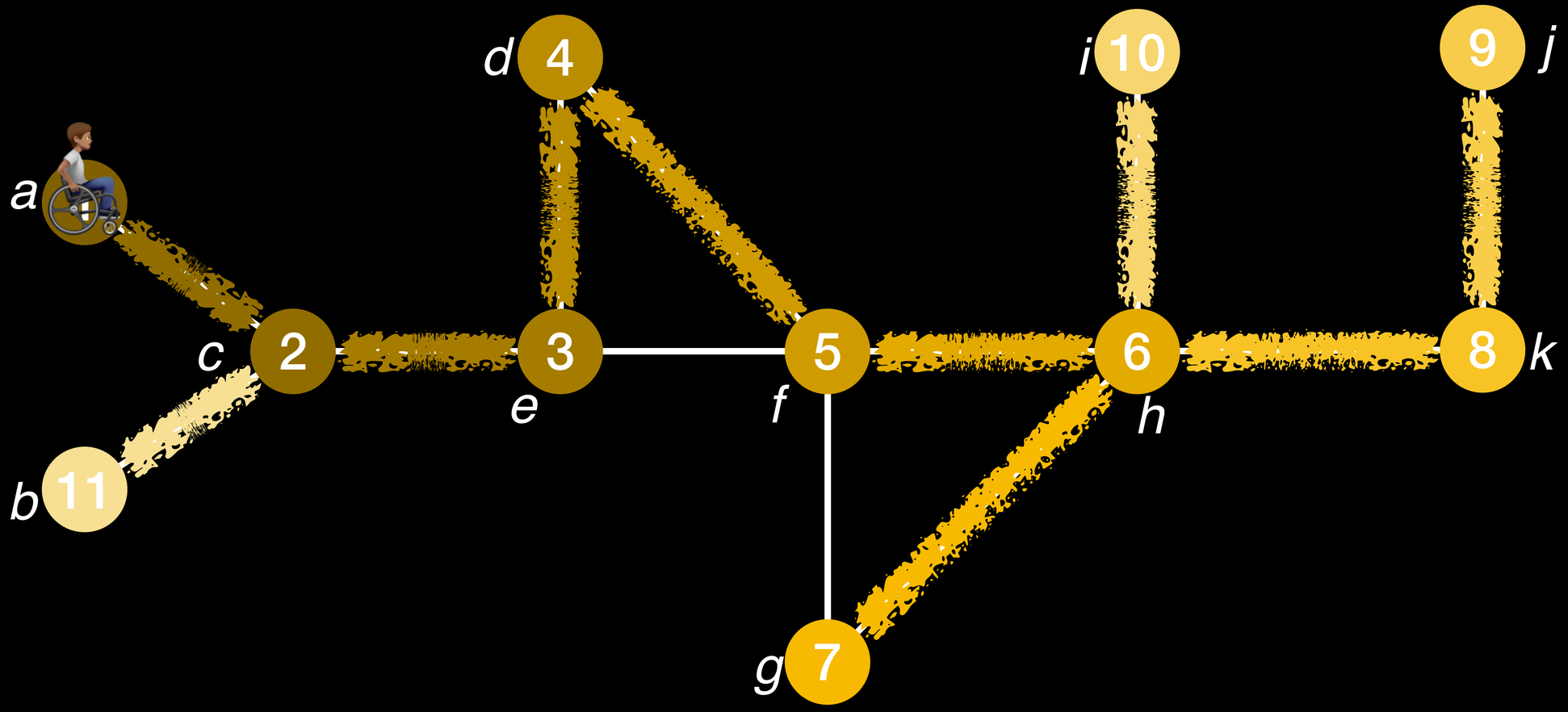
Depth-first Search

- another method to visit every vertex
- continue visiting from the newest vertex
- image: exploring a place

深度优先搜索

- 访问图的每个顶点的另一种方法
- 从最新顶点继续访问
- 想象探索一个地方

Depth-First Search 深度优先搜索



Input and Output

Input: graph G

Output: for every vertex v:

- $v.d$ = discovery time of v
  (when $v.color$ becomes gray)
- $v.f$ = finishing time of v
  (when $v.color$ becomes black)
- $v.\pi$ = predecessor of v

输入和输出

输入： 图G

输出： 对于每个顶点v：

- $v.d$ = v的发现时间
  （$v.color$ 变为gray时）
- $v.f$ = v的完成时间
  （$v.color$变为black时）
- $v.\pi$ = v的前身

Order of Examination

- Examine a vertex = find new neighbours
- Q: In which order are vertices examined?
  A: newest vertices first — stack/recursion
- Q: How to avoid examining vertex twice?
  A: mark vertices with a color:
  white = not examined
  gray = waiting to be examined or examination in progress
  black = completely examined

检查顺序

- 检查顶点 = 查找新邻居
- 问：顶点的检查顺序是什么?
  答：最新顶点优先 — 堆栈 / 递归
- 问：如何避免两次检查顶点?
  答：使用颜色标记顶点：
  白色 = 未检查
  灰色 = 等待检查或正在检查
  黑色 = 完全检查

Depth-first search 深度优先搜索

initialize colors

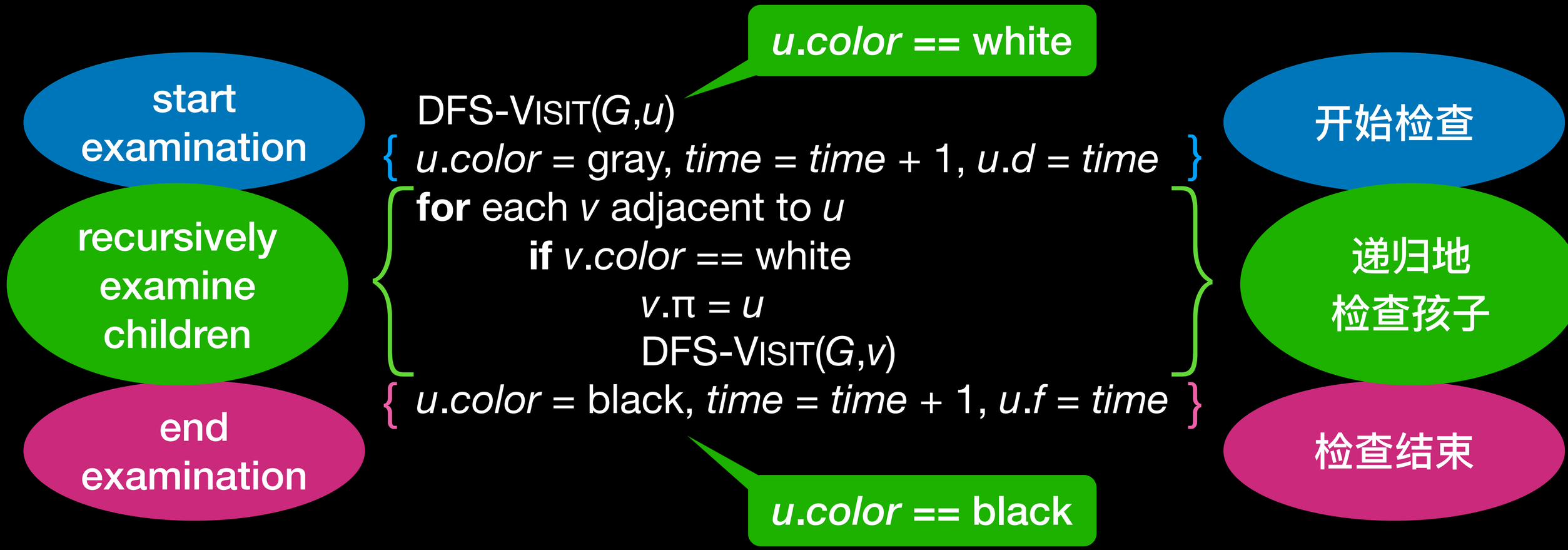
every vertex can be a root

```
DFS(G)
for each vertex u ∈ G.V
        u.color = white, u.π = NIL
time = 0
for each u ∈ G.V
        if u.color == white
                DFS-VISIT(G,u)
```

预置颜色

每个顶点可以当根

Depth-first search 深度优先搜索



White-Path Property

- Precondition:
  When DFS-VISIT(G,u) is called,
  u is a white vertex.

- Postcondition:
  When DFS-VISIT(G,u) terminates,
  u is the root of a depth-first tree
  containing all vertices
  that can be reached from u
  (only visiting previously white vertices
  —Theorem 22.9).

白色路径定理

- 前提条件：
  调用DFS-VISIT(G,u)时，
  u是一个白色顶点。

- 后置条件：
  当DFS-VISIT(G,u)终止时，
  u是包含所有顶点的深度优先树的根
  这可以从美国获得
  （仅访问以前的白色顶点
  —定理22.9）。

Parenthesis Property

Postcondition on times $v.d$ and $v.f$:

Theorem 22.7

For any two vertices $u \neq v$, exactly one of the following three conditions holds:

- $[u.d, u.f] \cap [v.d, v.f] = \varnothing$ and neither u nor v is a descendant of the other.
- $[u.d, u.f] \subseteq [v.d, v.f]$ and u is a descendant of v.
- $[v.d, v.f] \subseteq [u.d, u.f]$ and v is a descendant of u.

括号化定理

时间v.d和v.f的后置条件：

定理22.7

对于任意两个顶点$u \neq v$、以下三个条件中正好有一个适用：

- $[u.d, u.f] \cap [v.d, v.f] = \varnothing$，与$u$和$v$都不是对方的后代。
- $[u.d, u.f] \subseteq [v.d, v.f]$，与$u$是$v$的后代。
- $[v.d, v.f] \subseteq [u.d, u.f]$，与$v$是$u$的后代。

Edge Types

Tree edge = edge in the depth-first forest

Back edge = edge from a vertex to its ancestor

Forward edge = edge from a vertex to its grand(-grand-...)child

Cross edge = other edge

边缘类型

树边 = 深度优先林中的边

后边 = 从顶点到其祖先的边

前向边 = 从顶点到孙子（等）顶点的边

横向边 = 其他边

Running Time

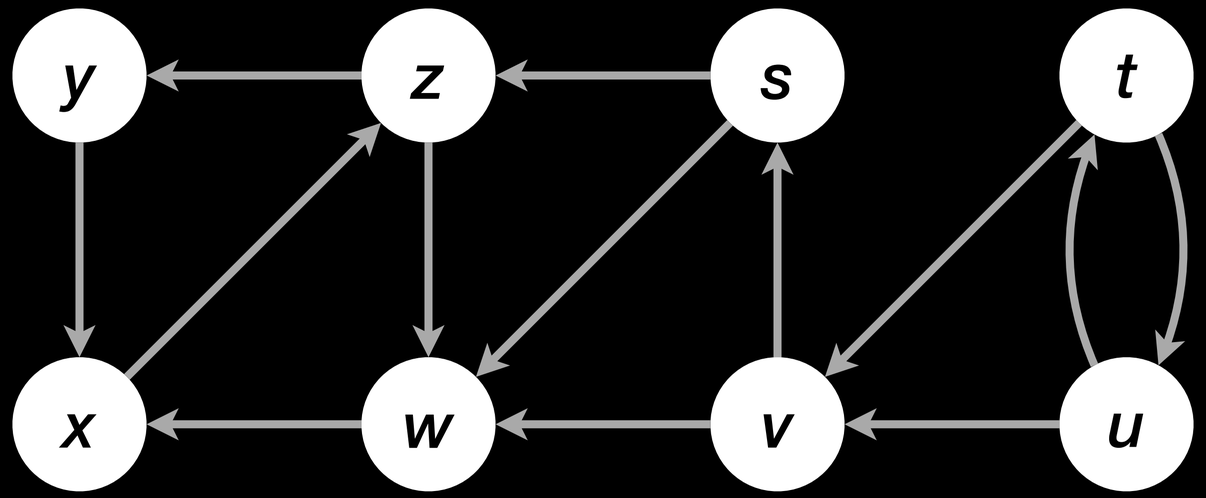
- Initialization requires time $O(|V|)$.
- DFS-Visit(G,u) is called exactly once for every vertex and requires time $O(|out(u)|)$ (not counting other calls to DFS-Visit).
- So, the total running time is in $O(|V|+|E|)$, and the algorithm terminates always.

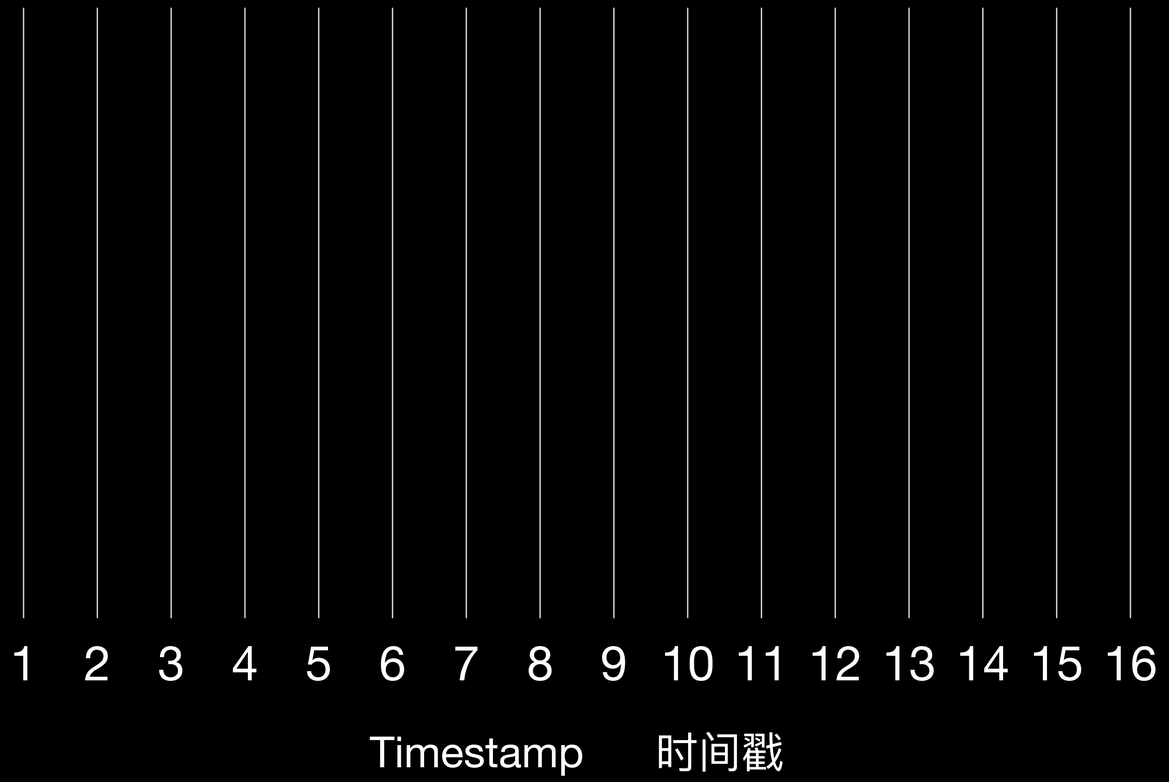
运行时间

- 初始化需要时间$O(|V|)$。
- DFS-Visit(G,u)称为每个顶点正好一次并且需要时间$O(|out(u)|)$（不包括其他DFS-Visit调用）。
- 因此，总运行时间以$O(|V|+|E|)$为单位，并且算法始终终止。
-

Example

例子

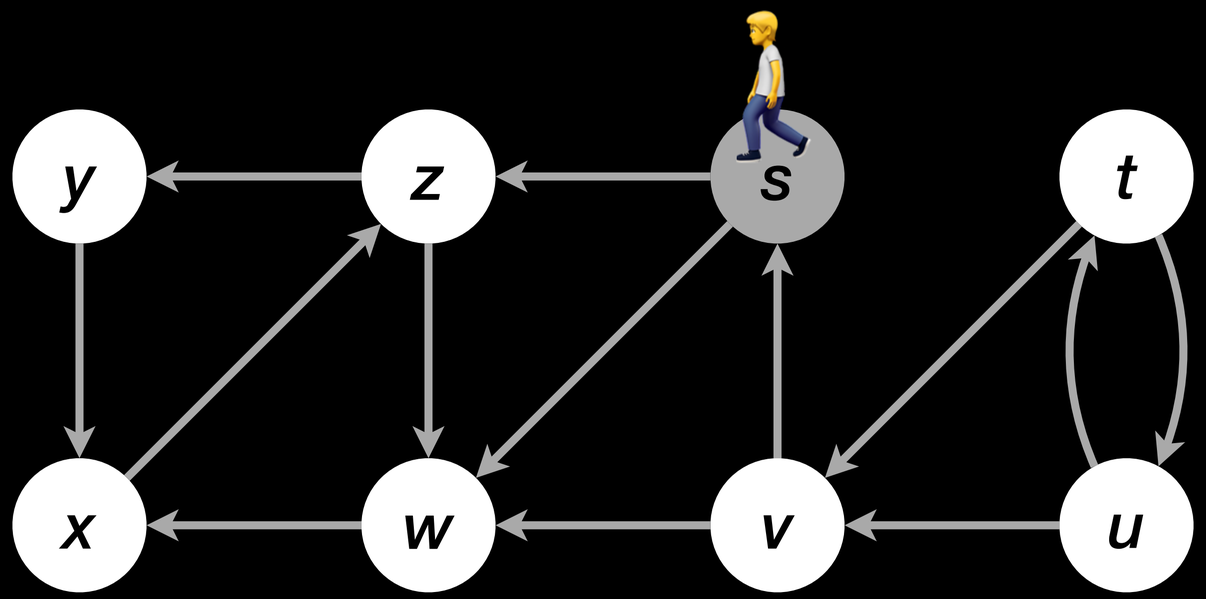


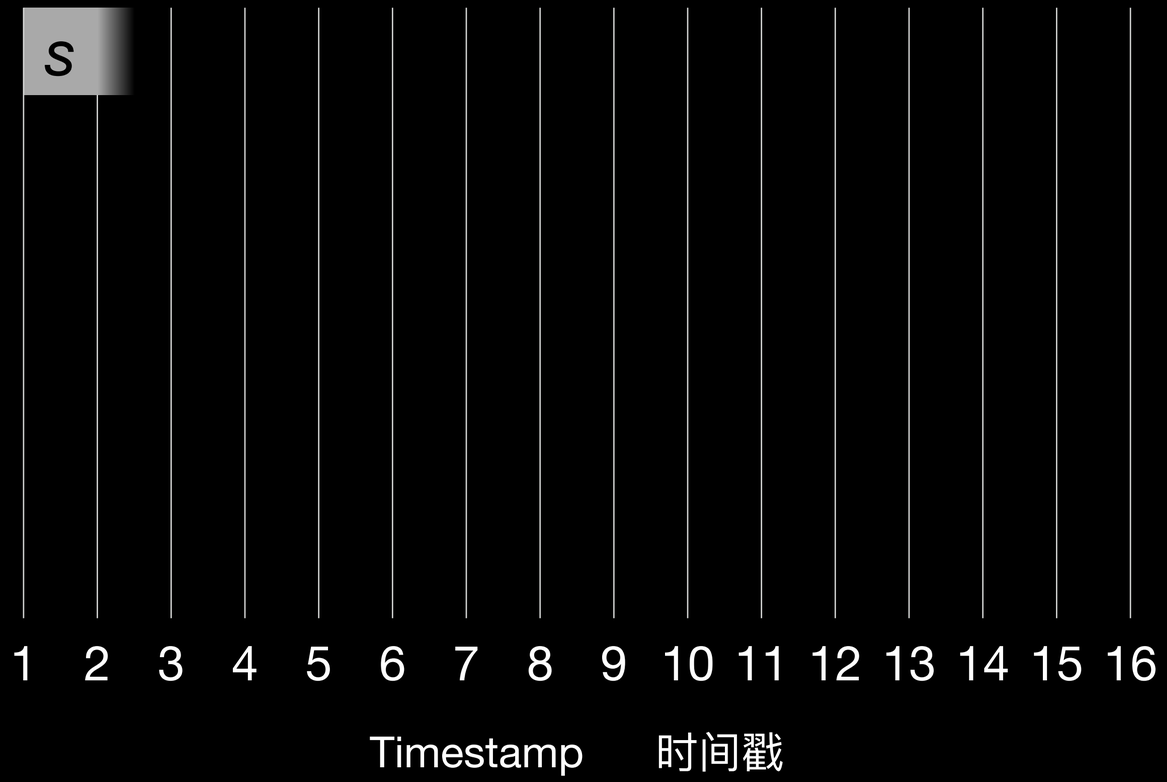


DFS(*G*)

Example

例子

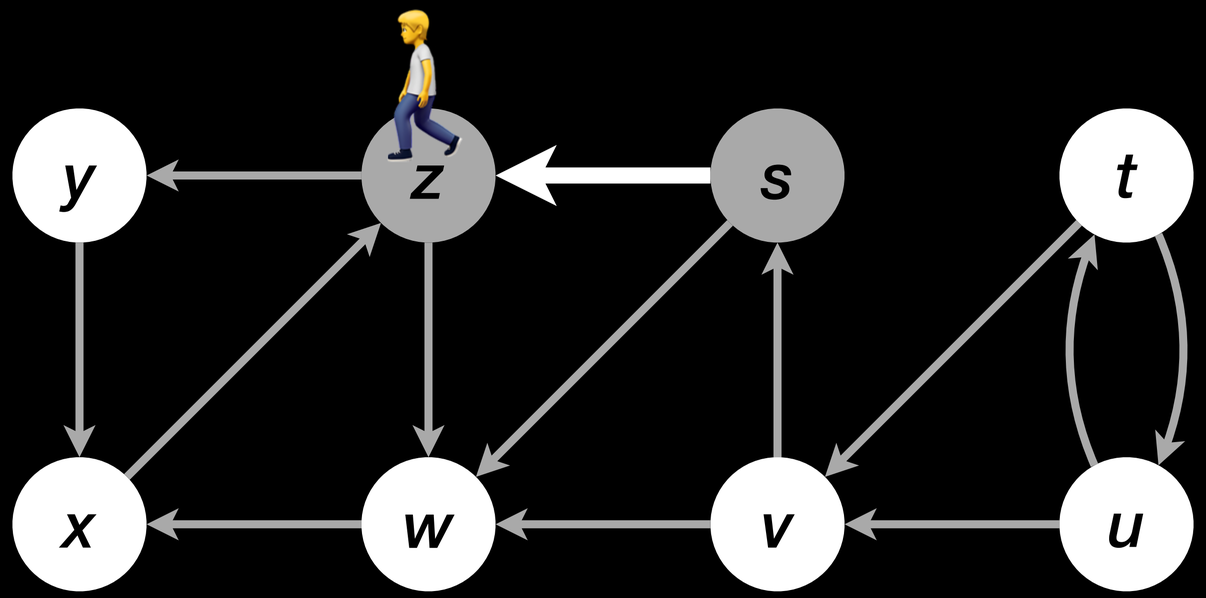


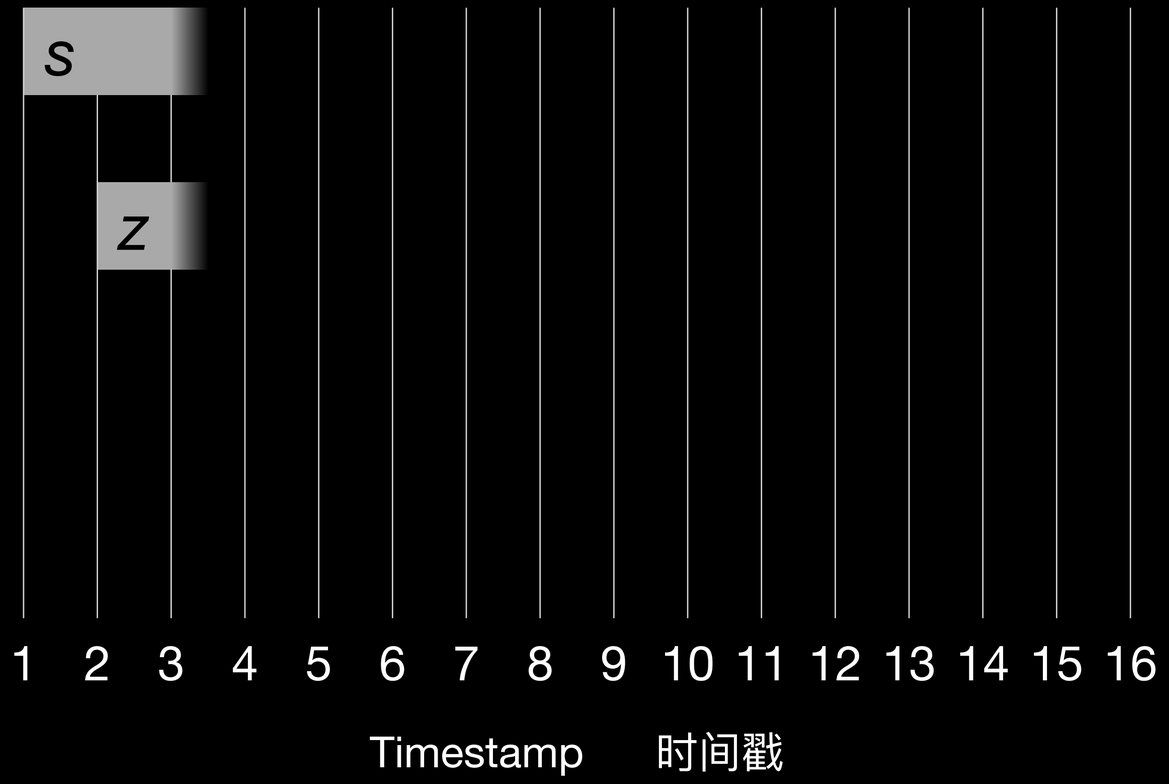


DFS(*G*) → DFS-VISIT(*G*,*s*)

Example

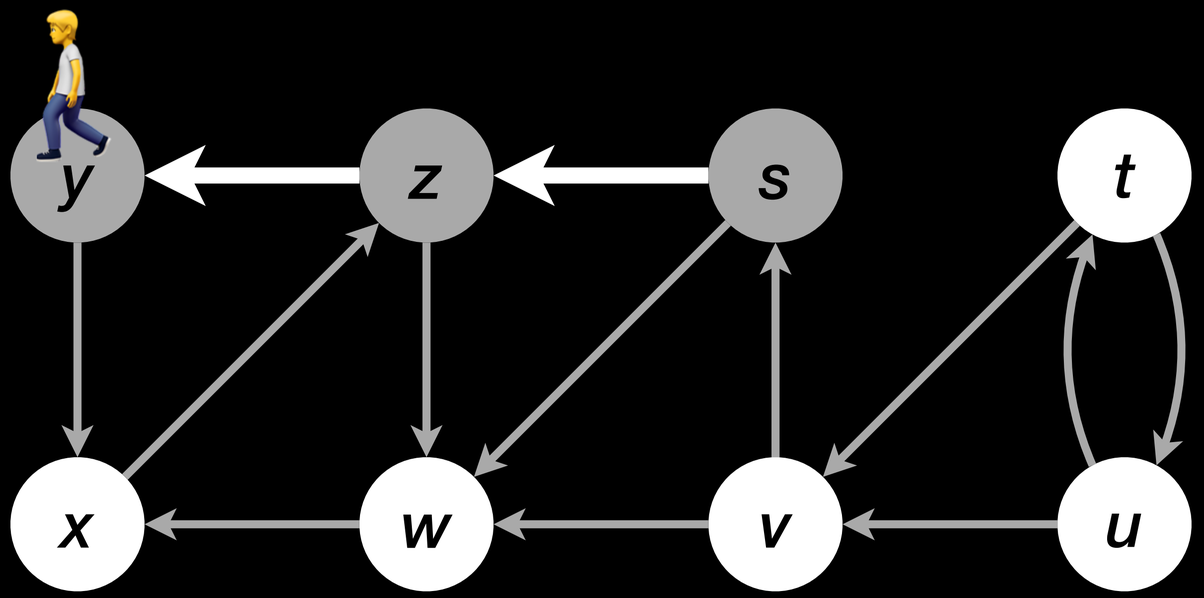
例子

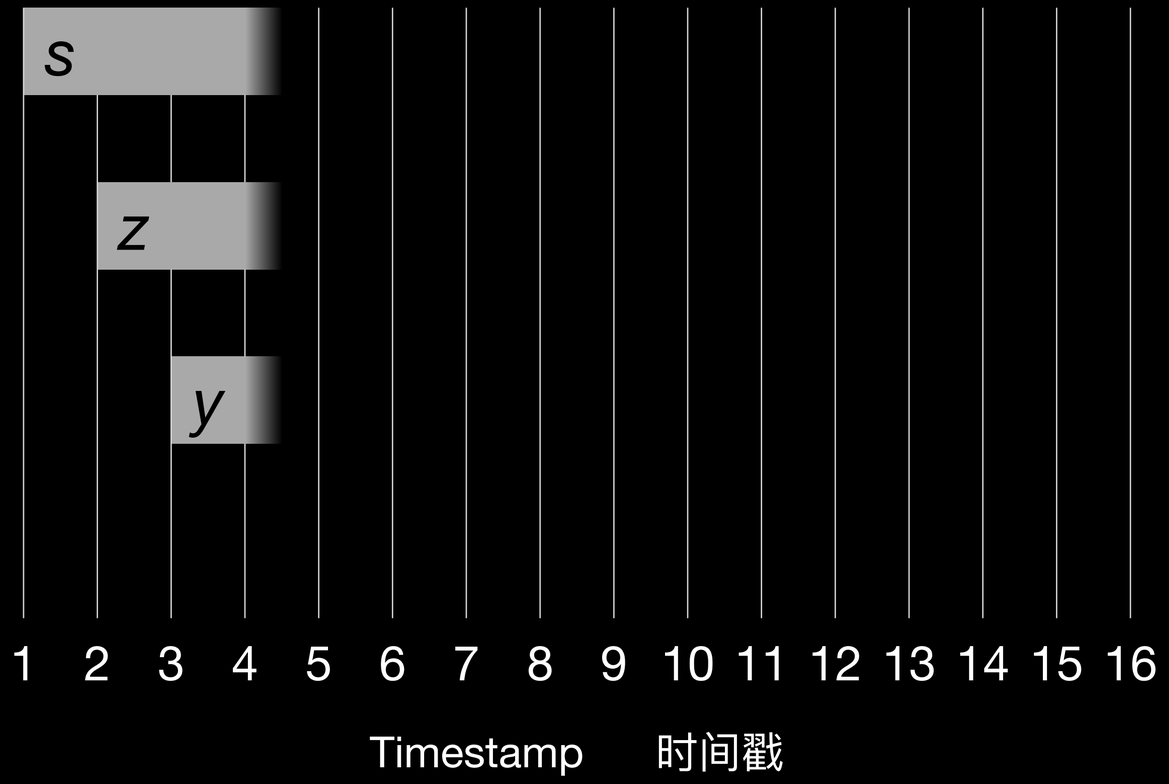




DFS(*G*) → DFS-VISIT(*G*,*s*) → DFS-VISIT(*G*,*z*)

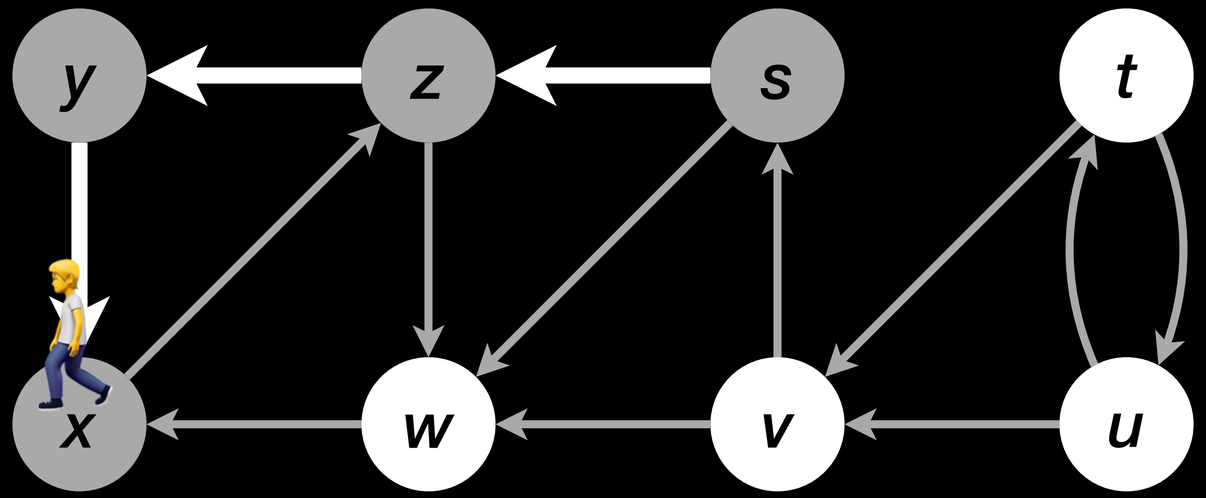
Example 例子

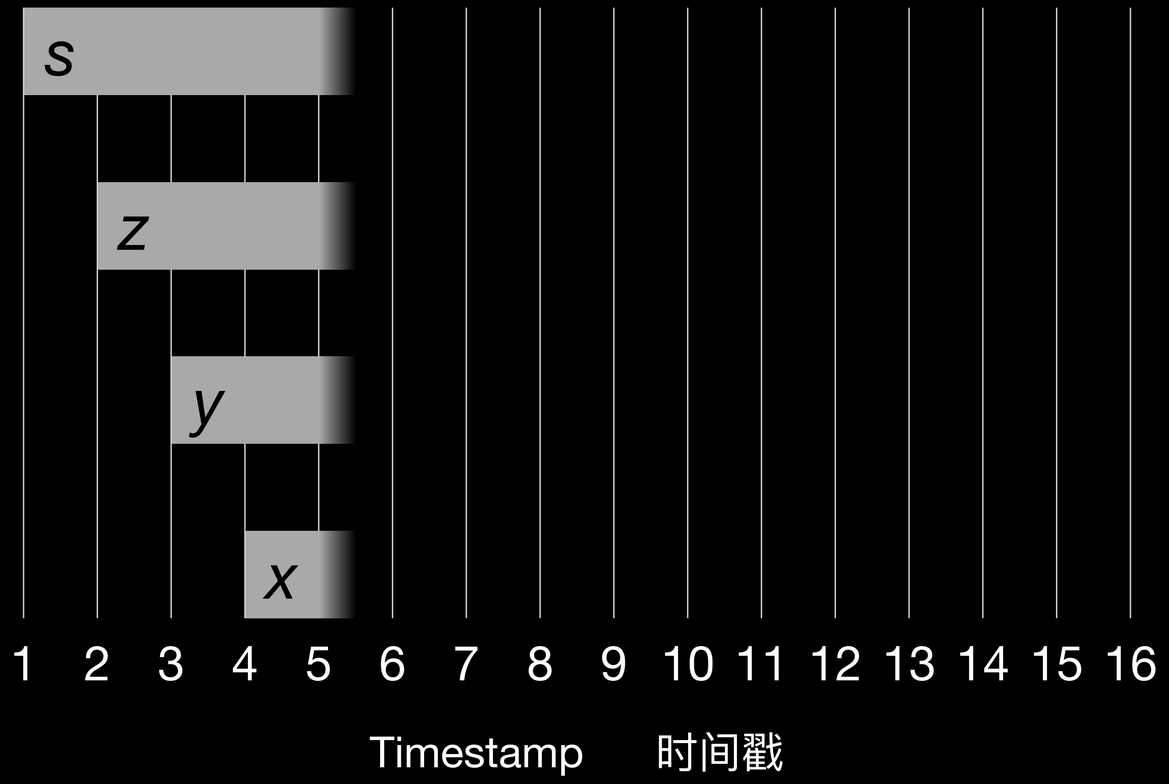




DFS(*G*) → DFS-VISIT(*G*,*s*) → DFS-VISIT(*G*,*z*) → DFS-VISIT(*G*,*y*)

Example 例子

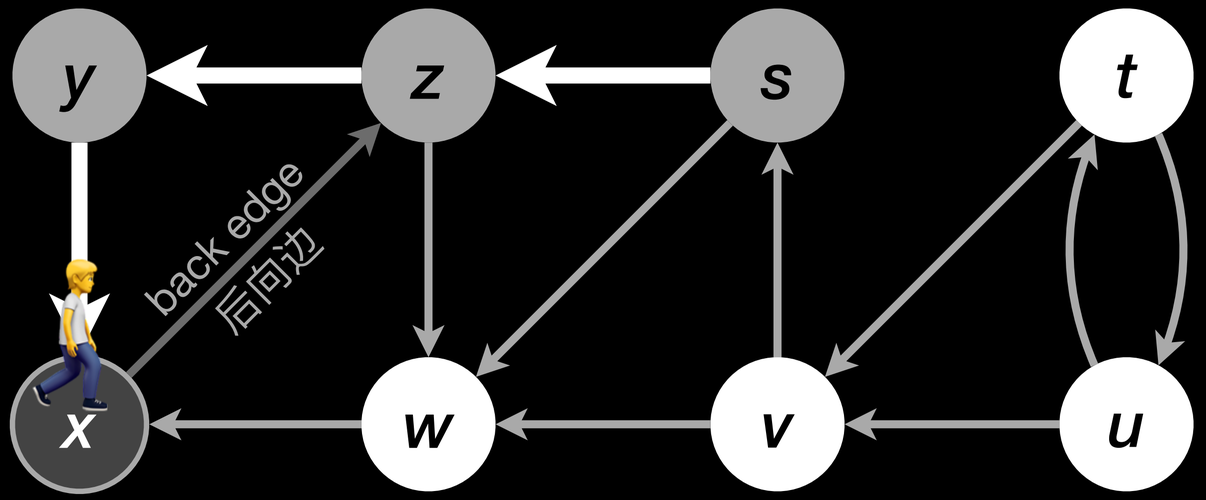


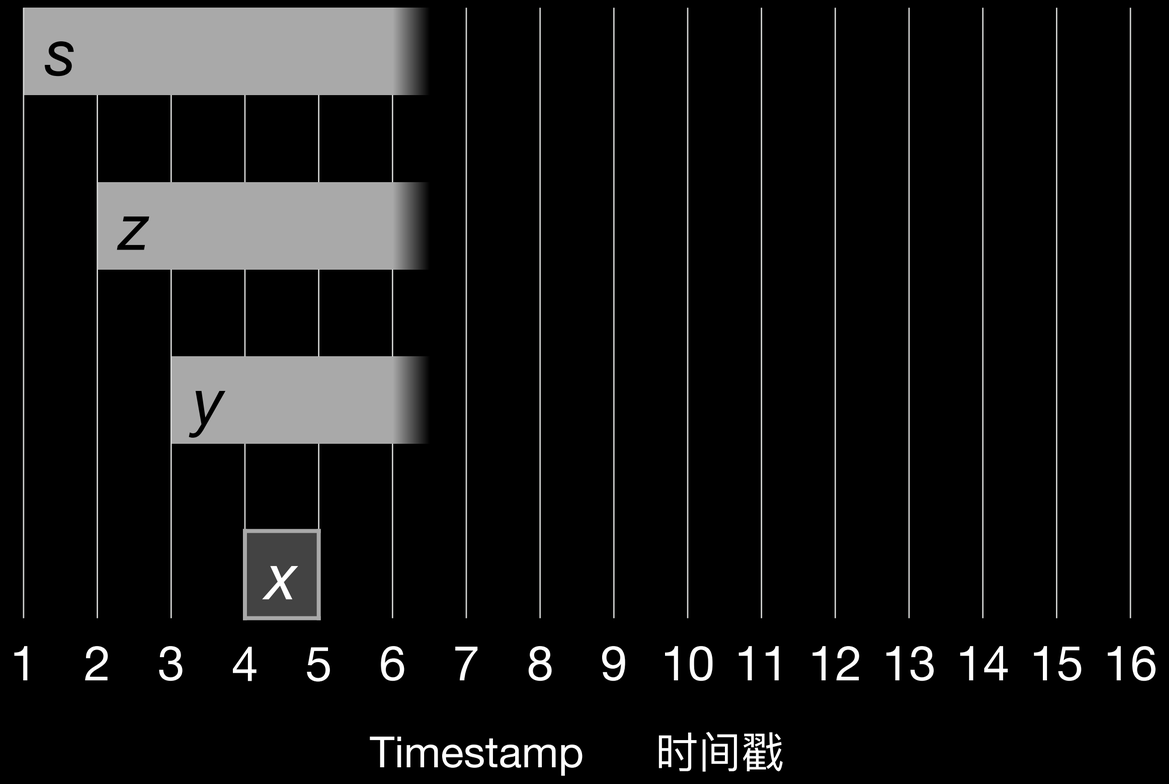


DFS(*G*) → DFS-VISIT(*G*,*s*) → DFS-VISIT(*G*,*z*) → DFS-VISIT(*G*,*y*) → DFS-VISIT(*G*,*x*)

Example

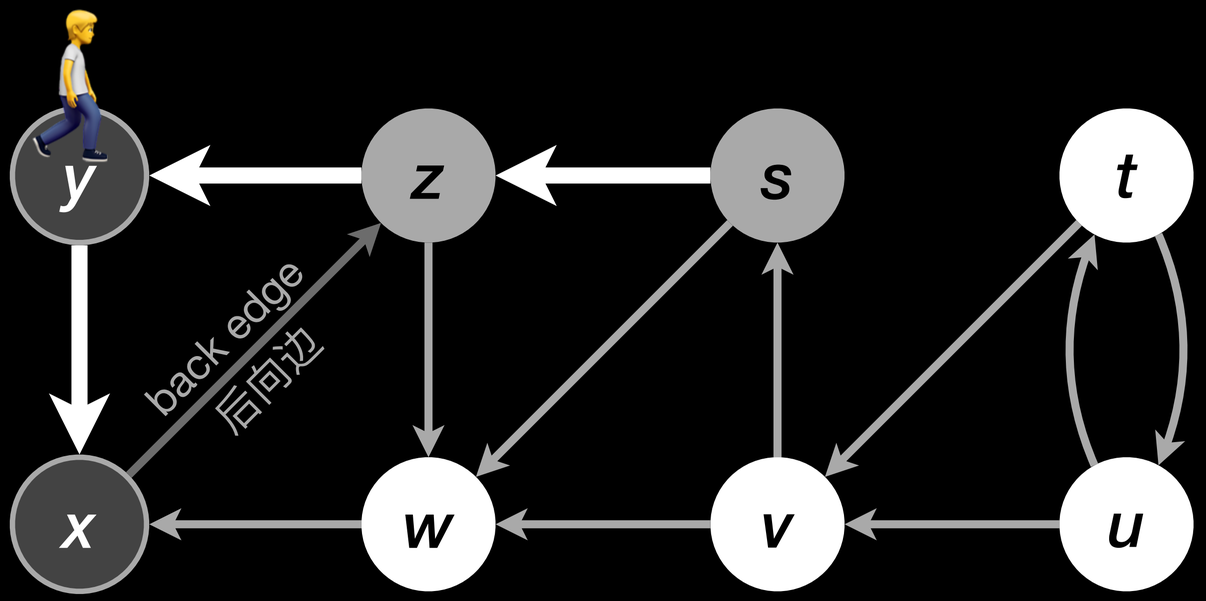
例子

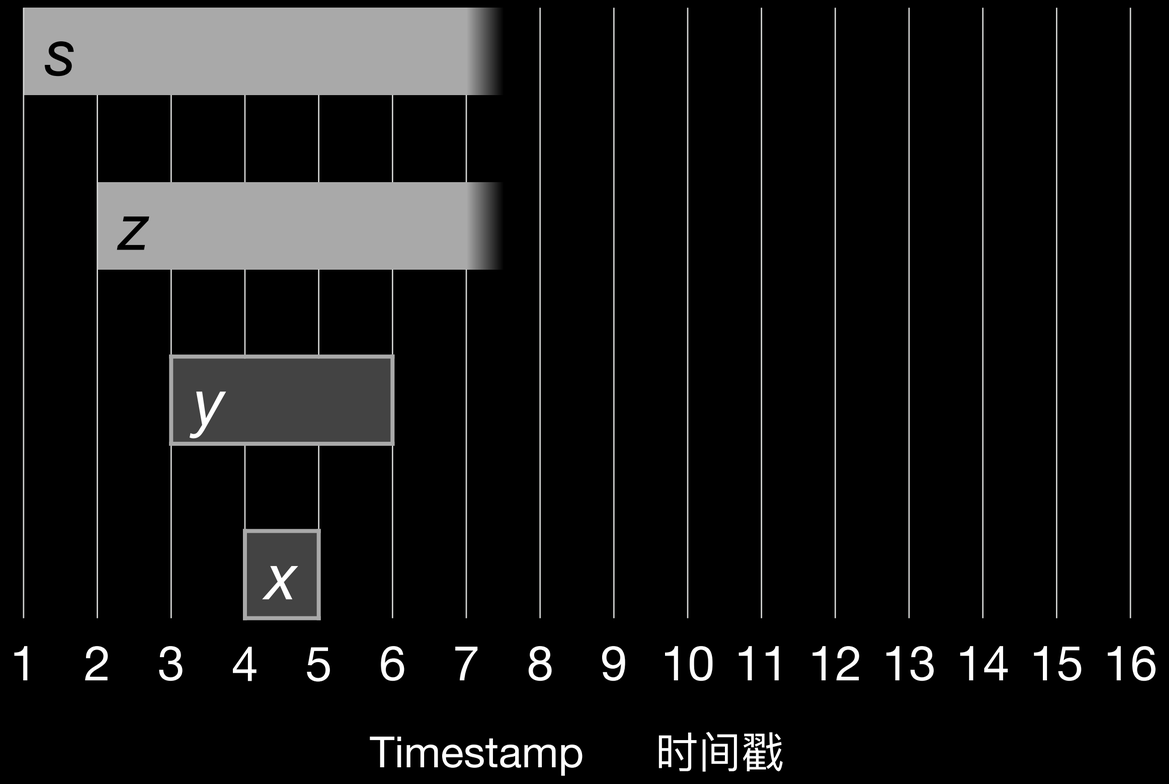




DFS(*G*) → DFS-VISIT(*G*,*s*) → DFS-VISIT(*G*,*z*) → DFS-VISIT(*G*,*y*) → DFS-VISIT(*G*,*x*)

Example 例子

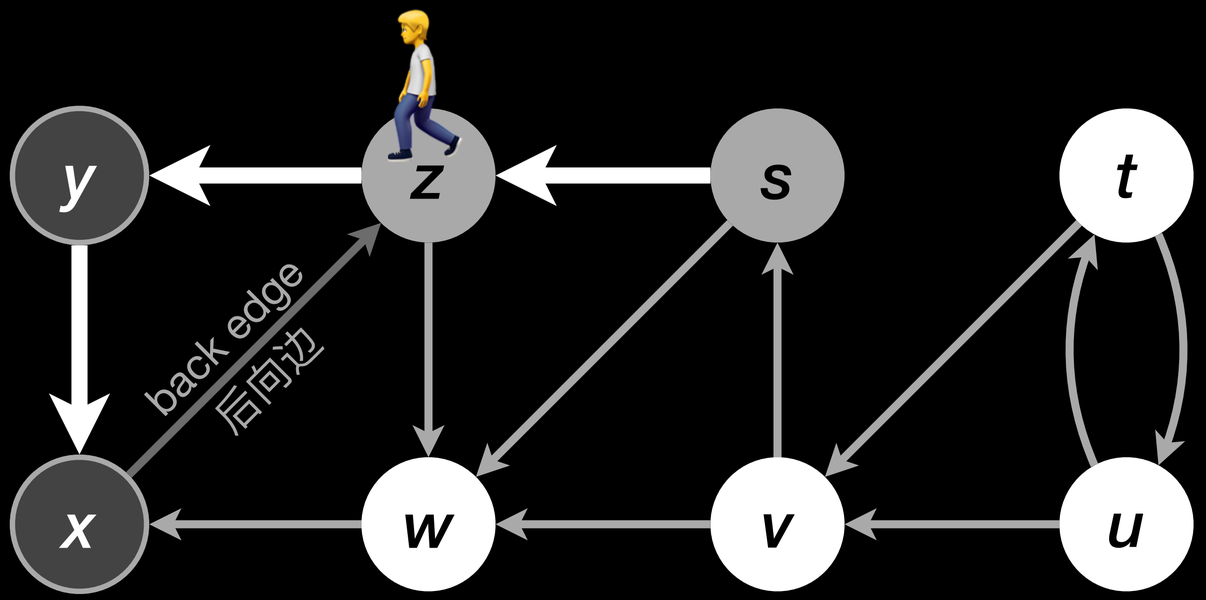


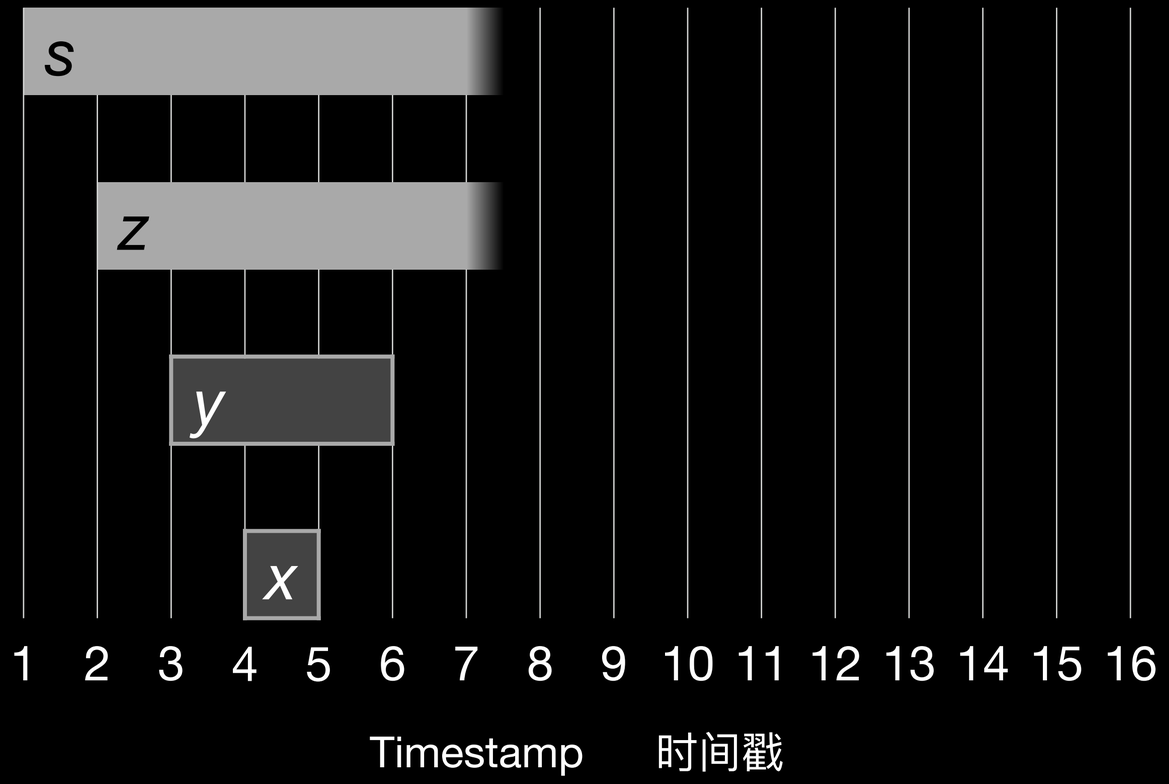


DFS(*G*) → DFS-VISIT(*G*,*s*) → DFS-VISIT(*G*,*z*) → DFS-VISIT(*G*,*y*)

Example

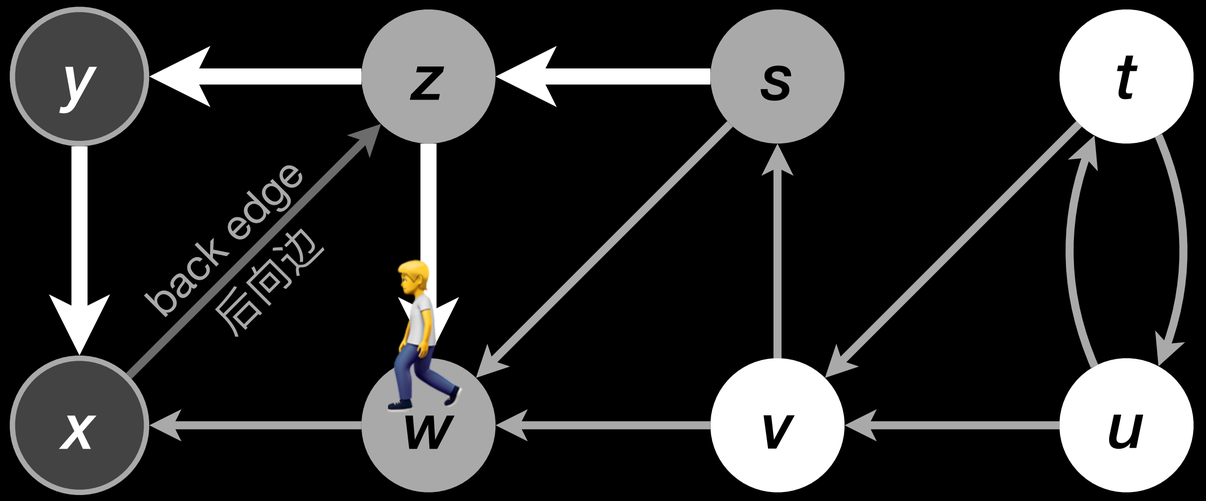
例子

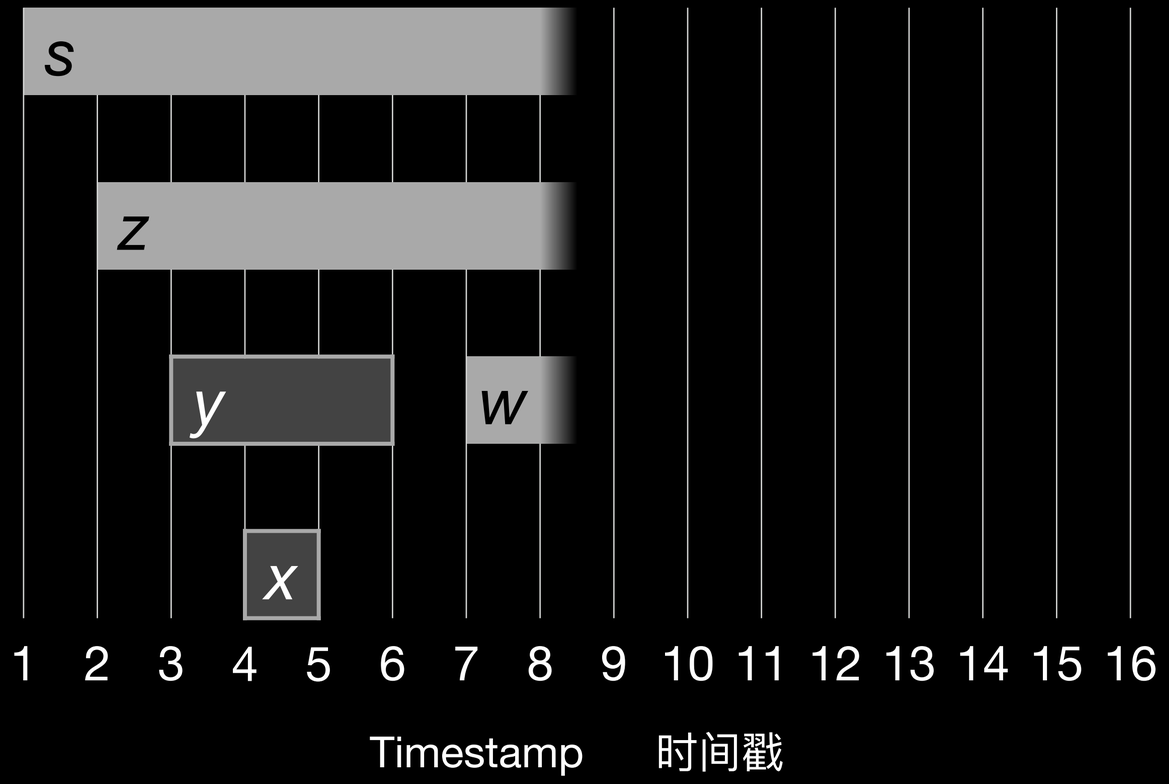




DFS(*G*) → DFS-VISIT(*G*,*s*) → DFS-VISIT(*G*,*z*)

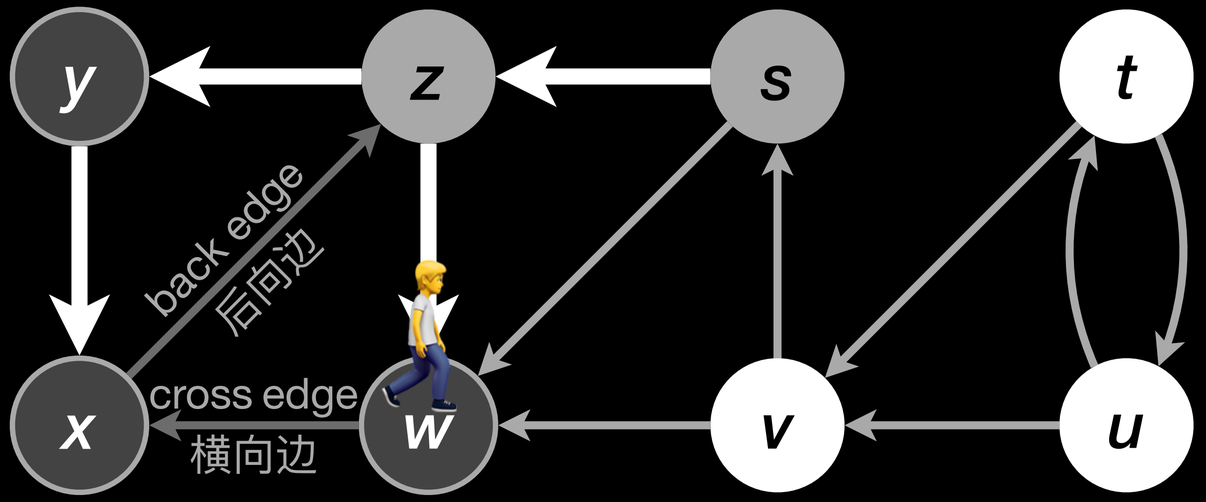
Example 例子

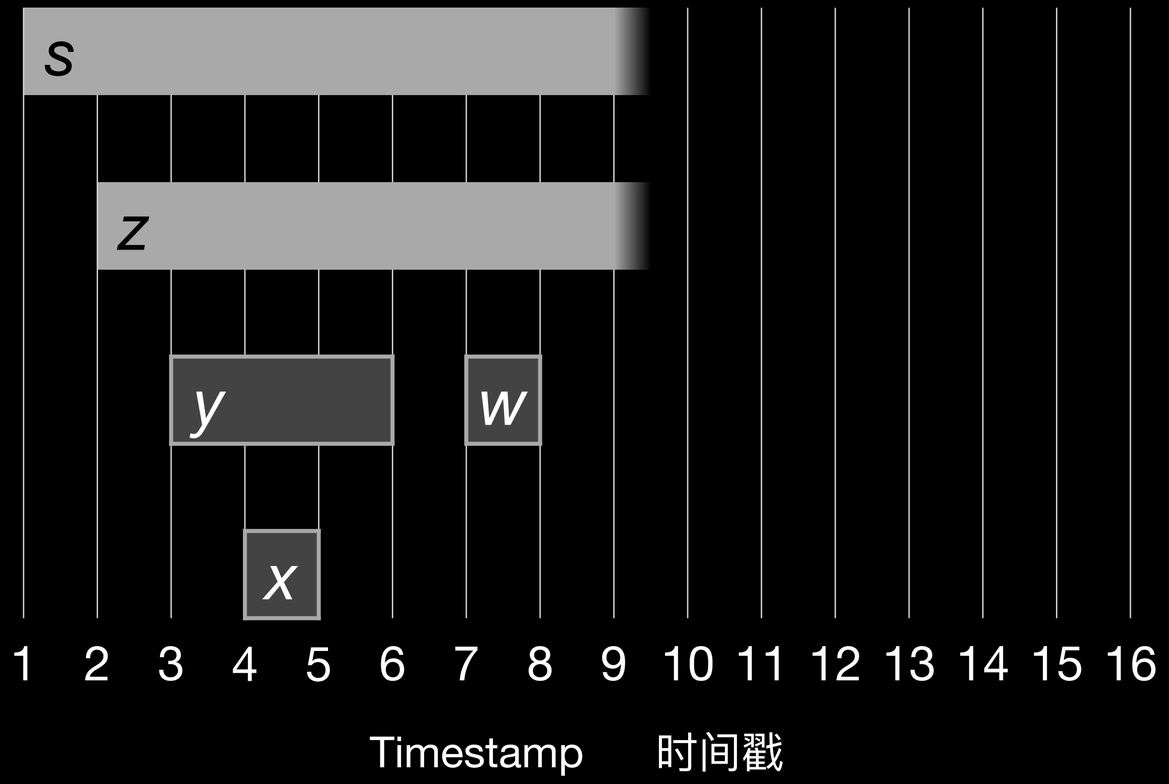




DFS(*G*) → DFS-VISIT(*G*,*s*) → DFS-VISIT(*G*,*z*) → DFS-VISIT(*G*,*w*)

Example 例子

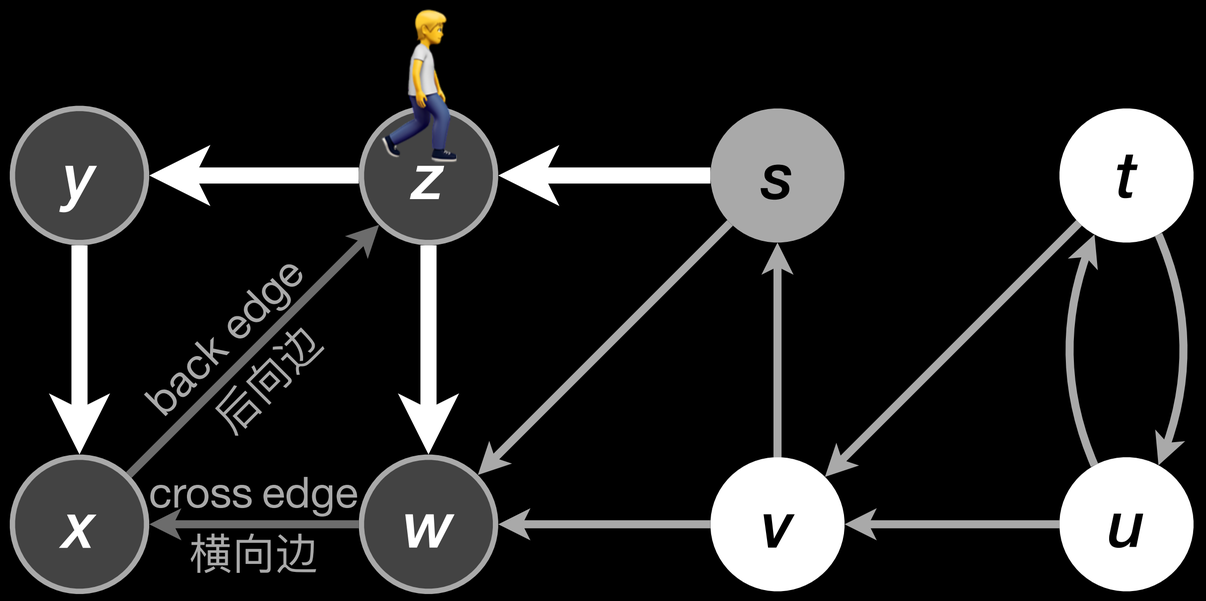


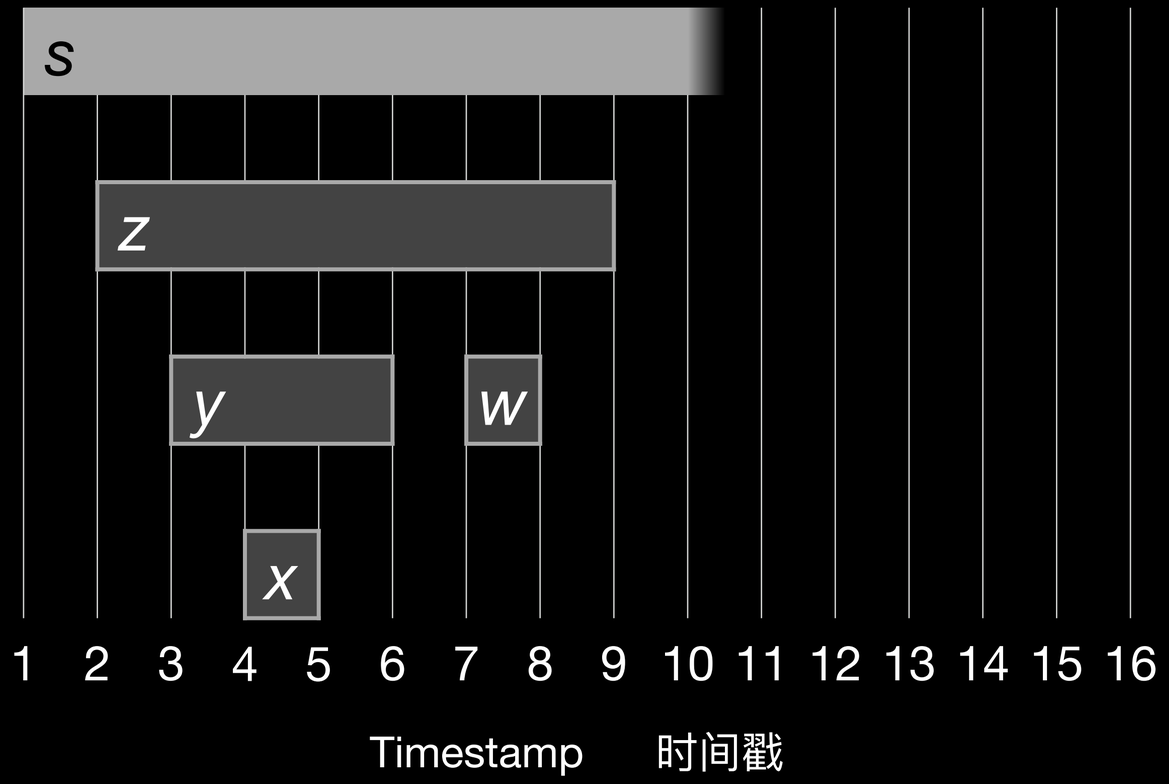


DFS(G) → DFS-VISIT(G,s) → DFS-VISIT(G,z) → DFS-VISIT(G,w)

Example

例子

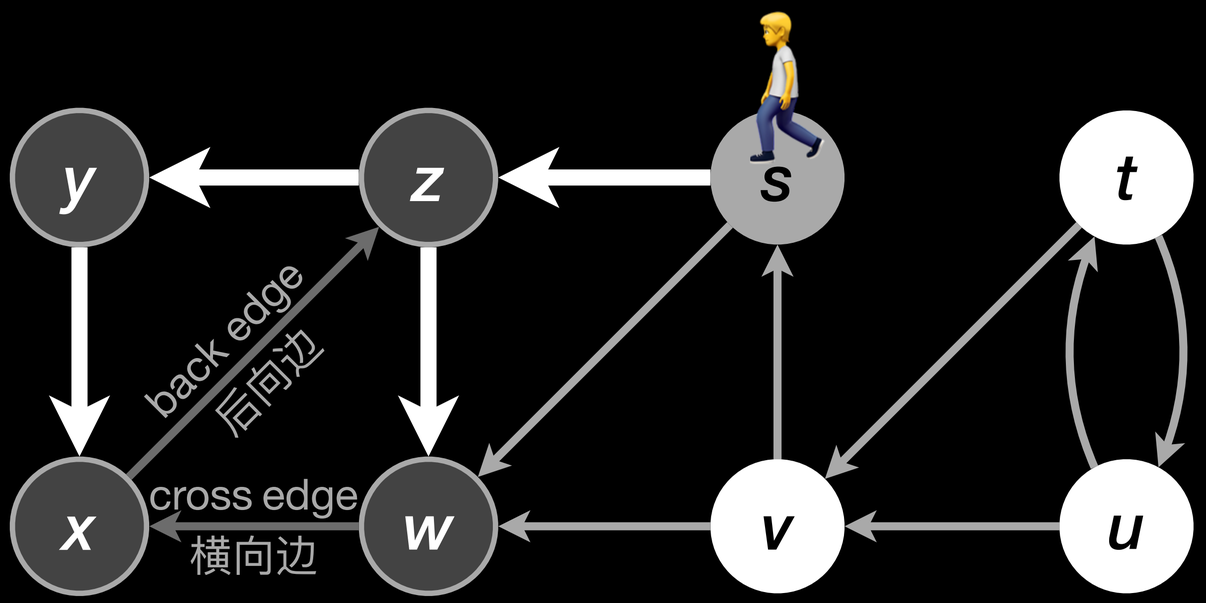


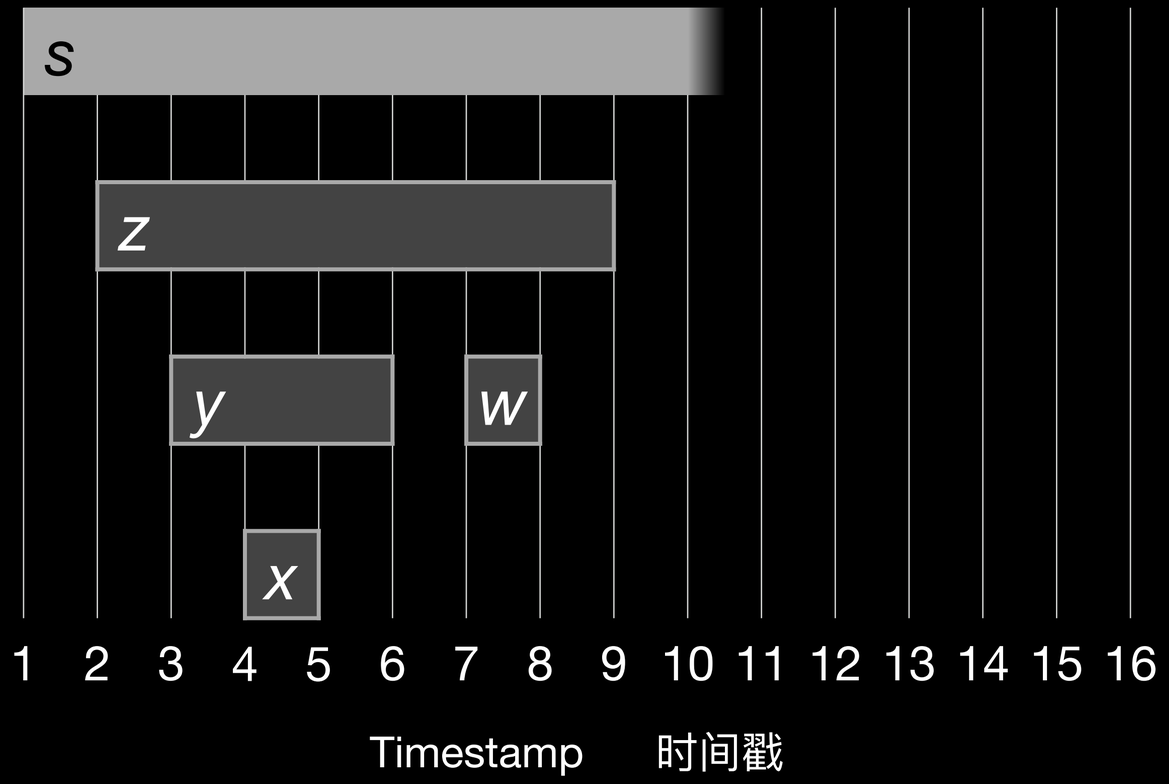


DFS(G) → DFS-VISIT(G,s) → DFS-VISIT(G,z)

Example

例子

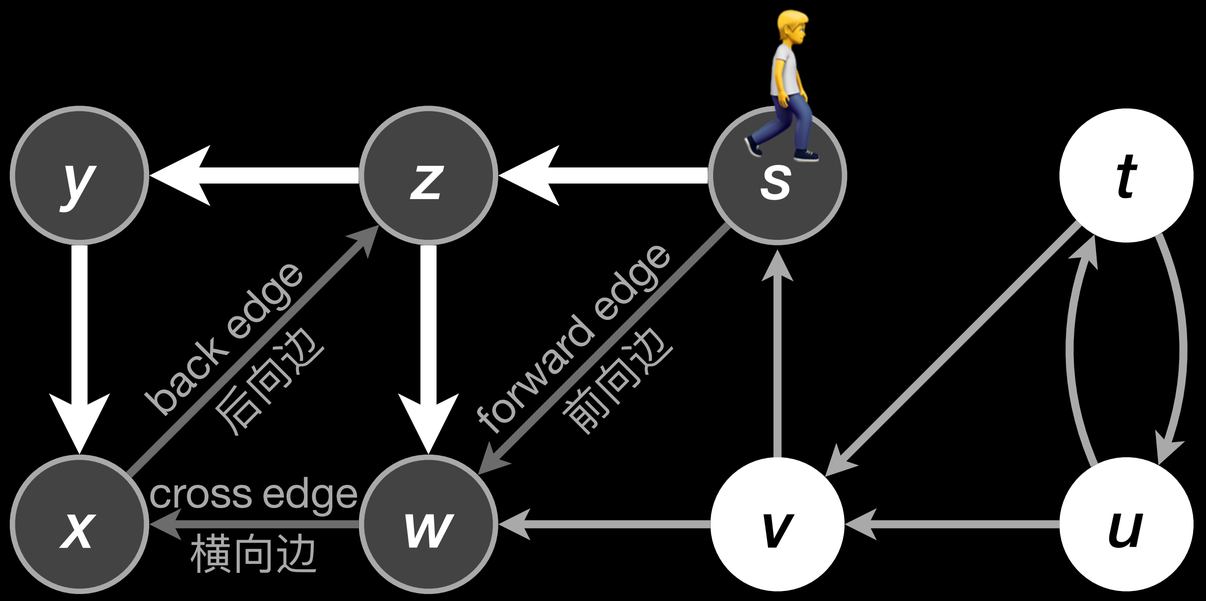


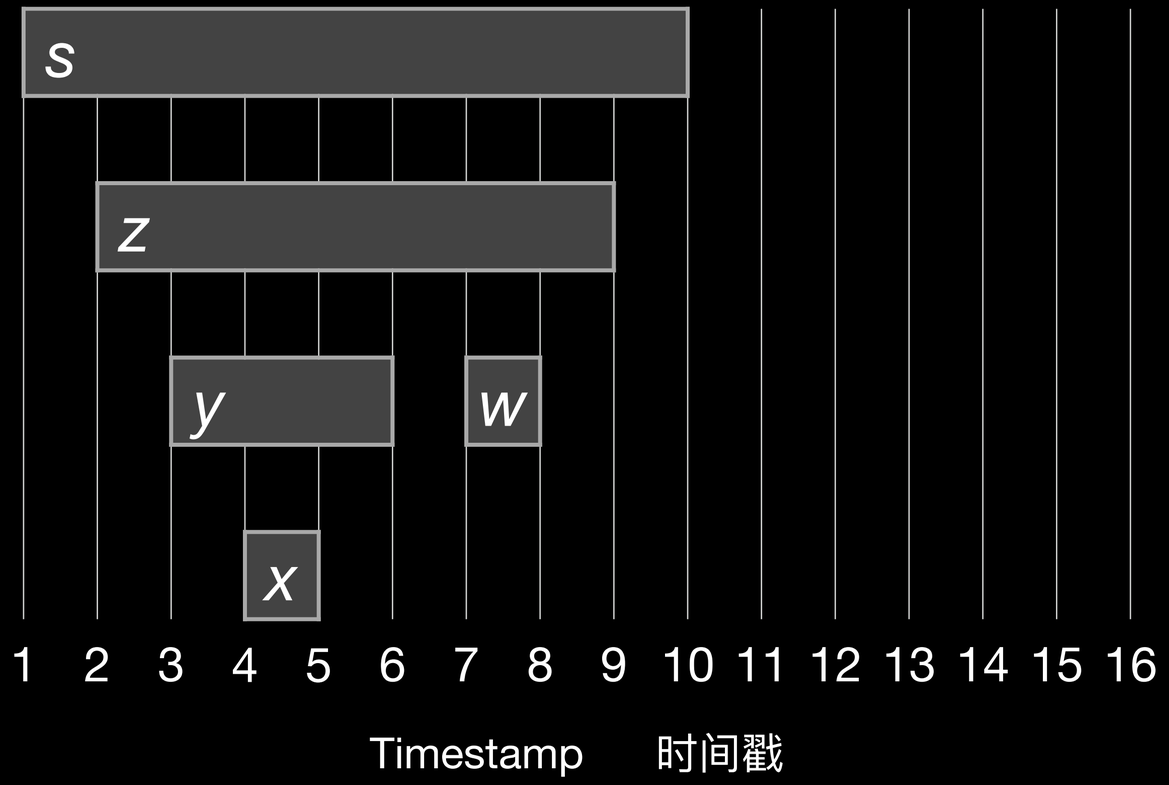


DFS(*G*) → DFS-VISIT(*G*,*s*)

Example

例子

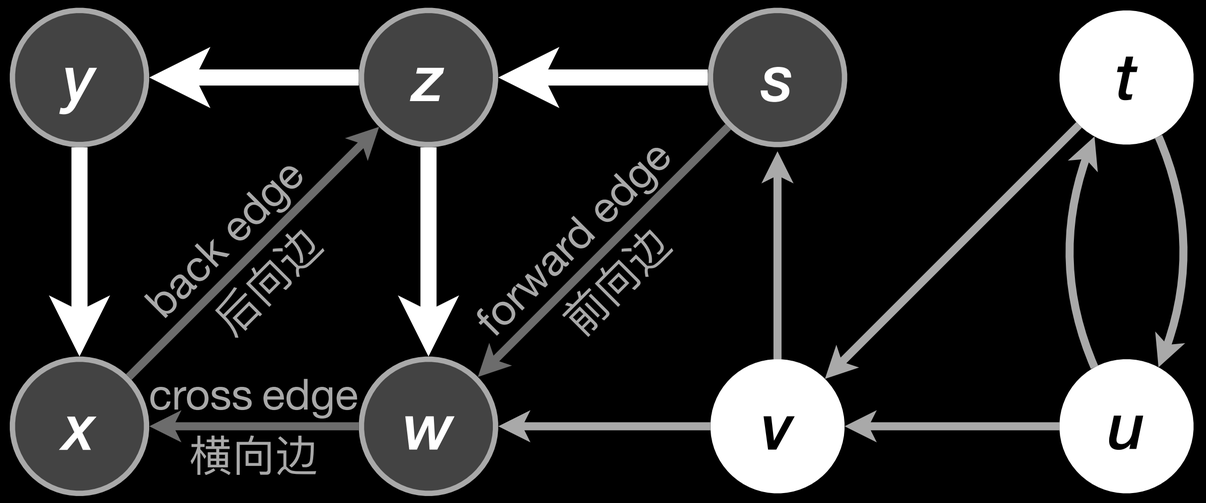


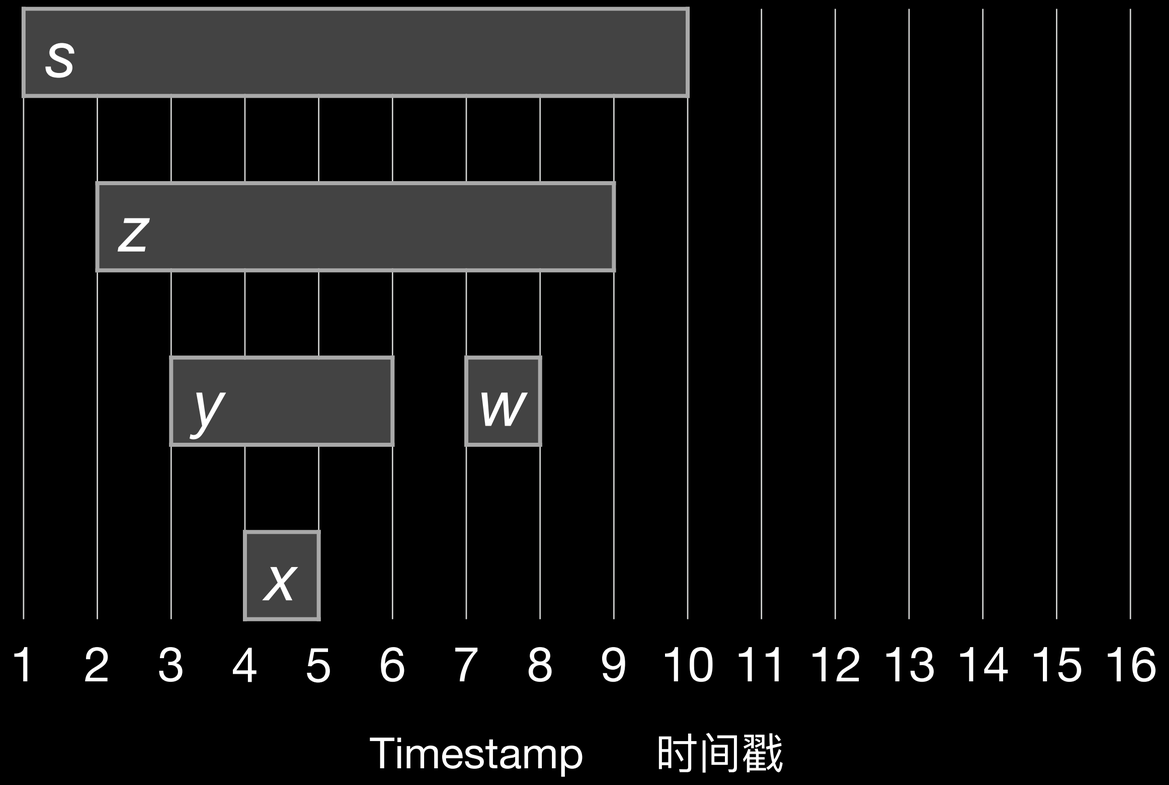


DFS(*G*) → DFS-VISIT(*G*,*s*)

Example

例子

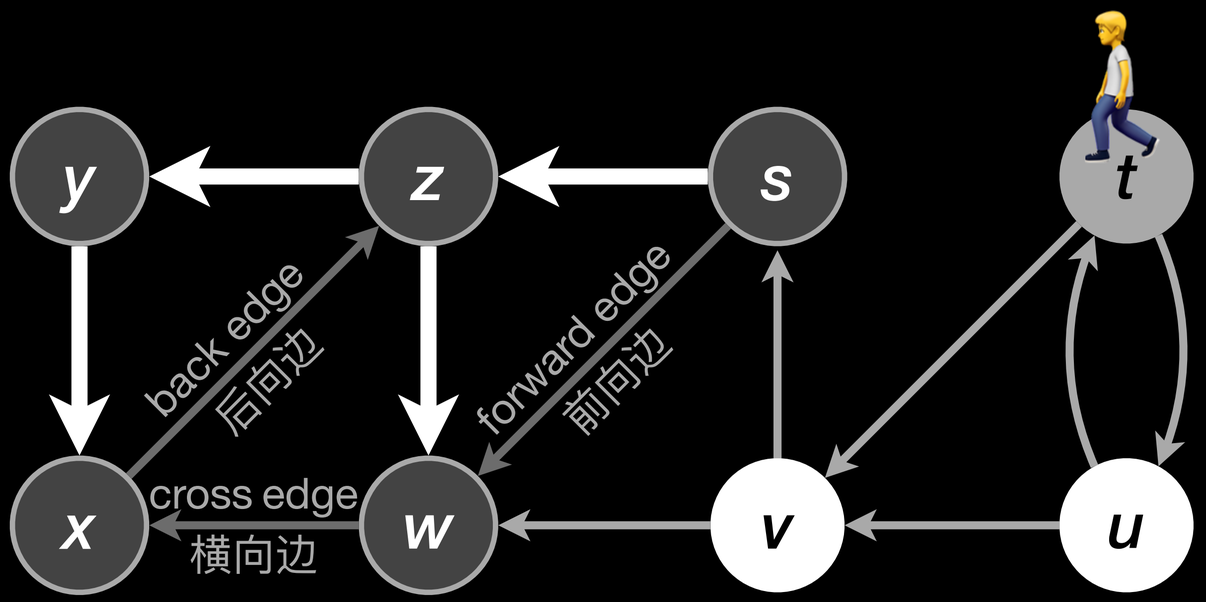


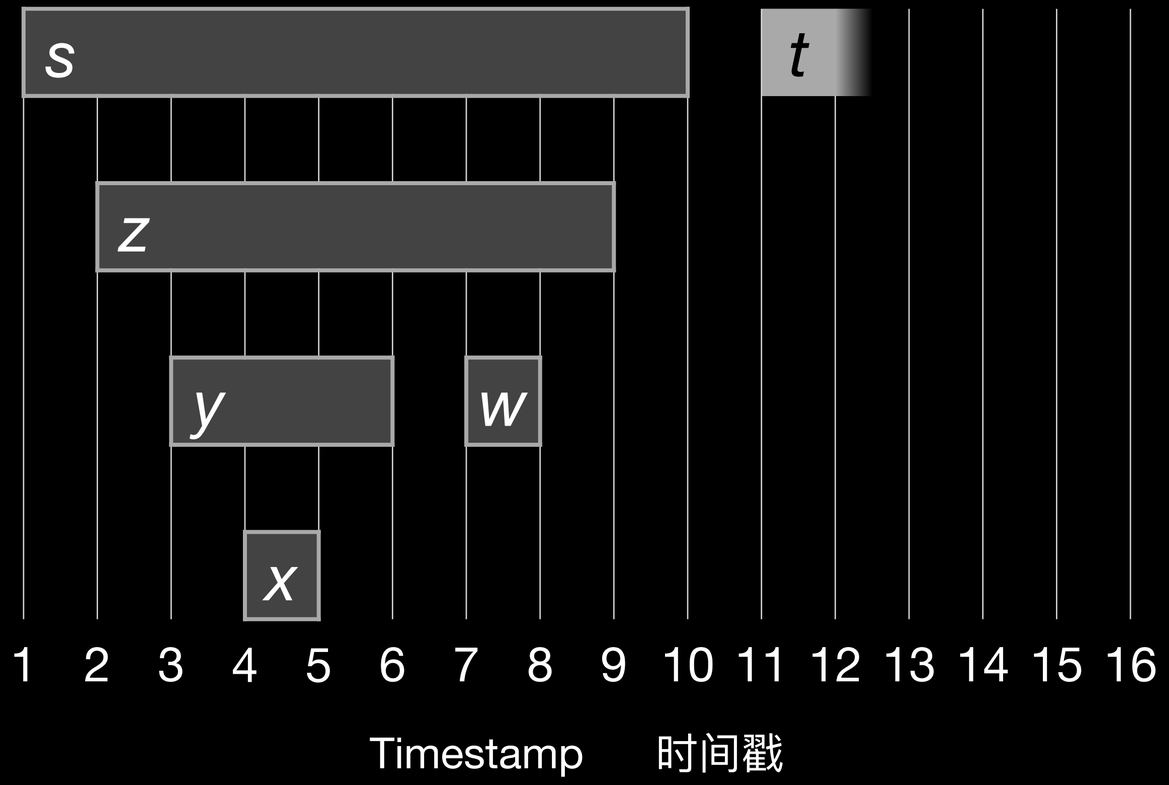


DFS(*G*)

Example

例子

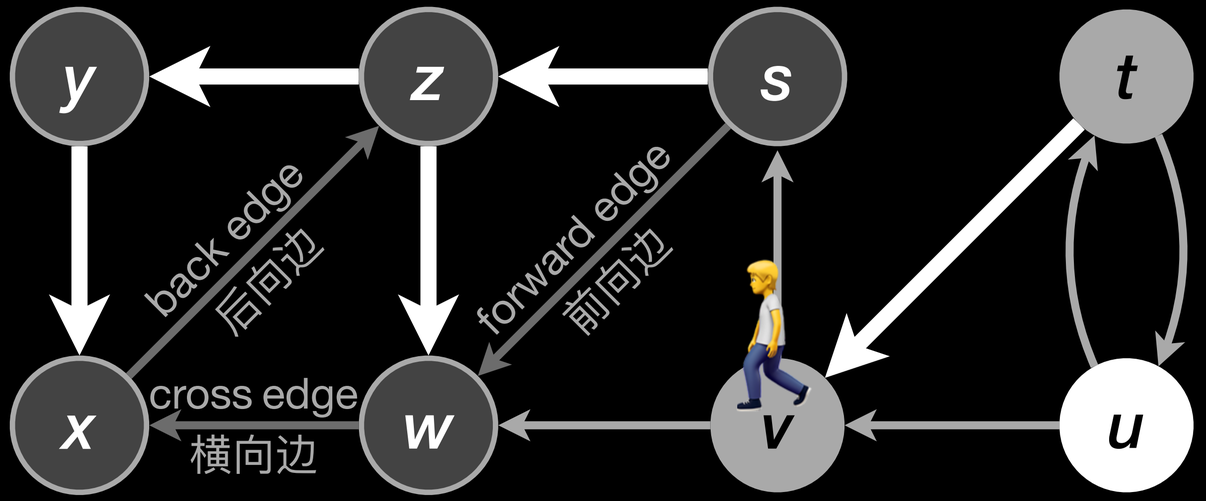


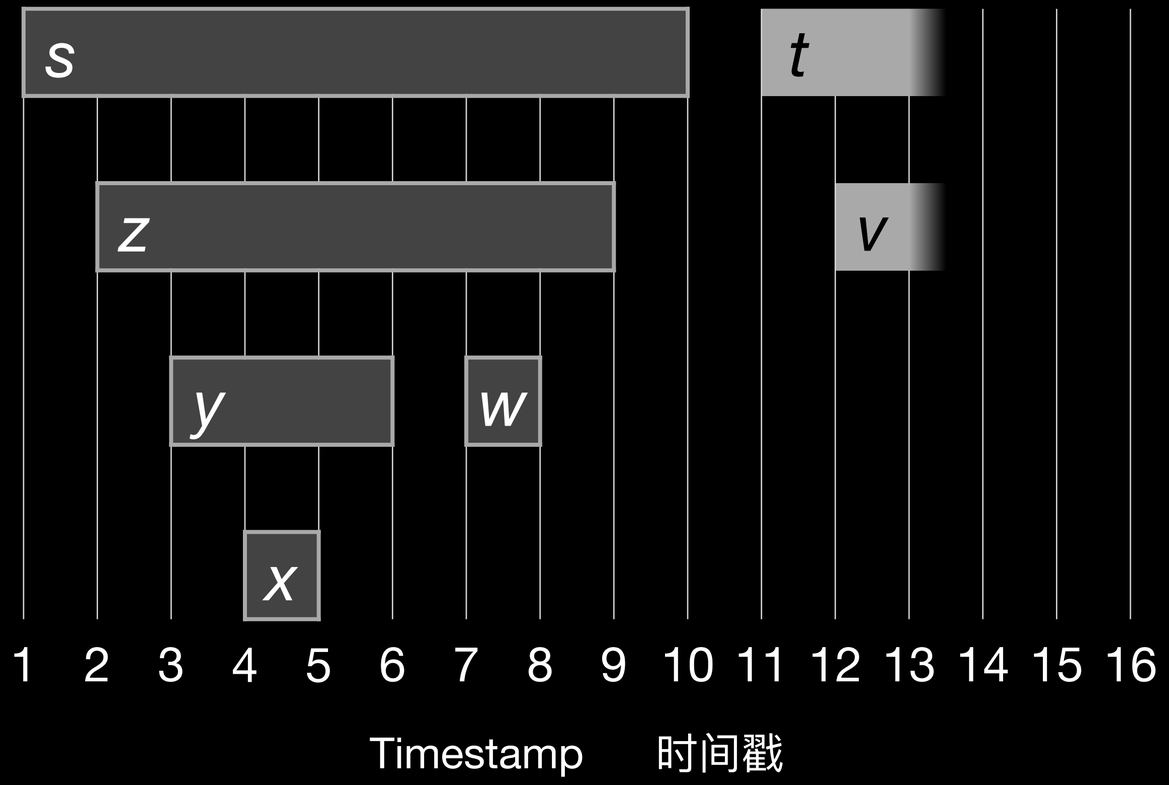


DFS(G) → DFS-VISIT(G,t)

Example

例子

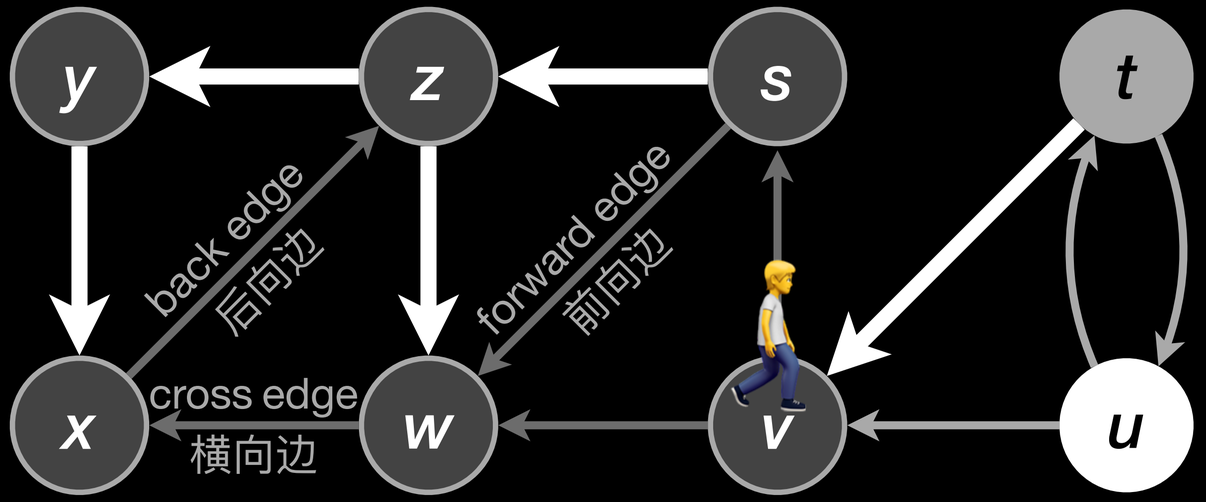


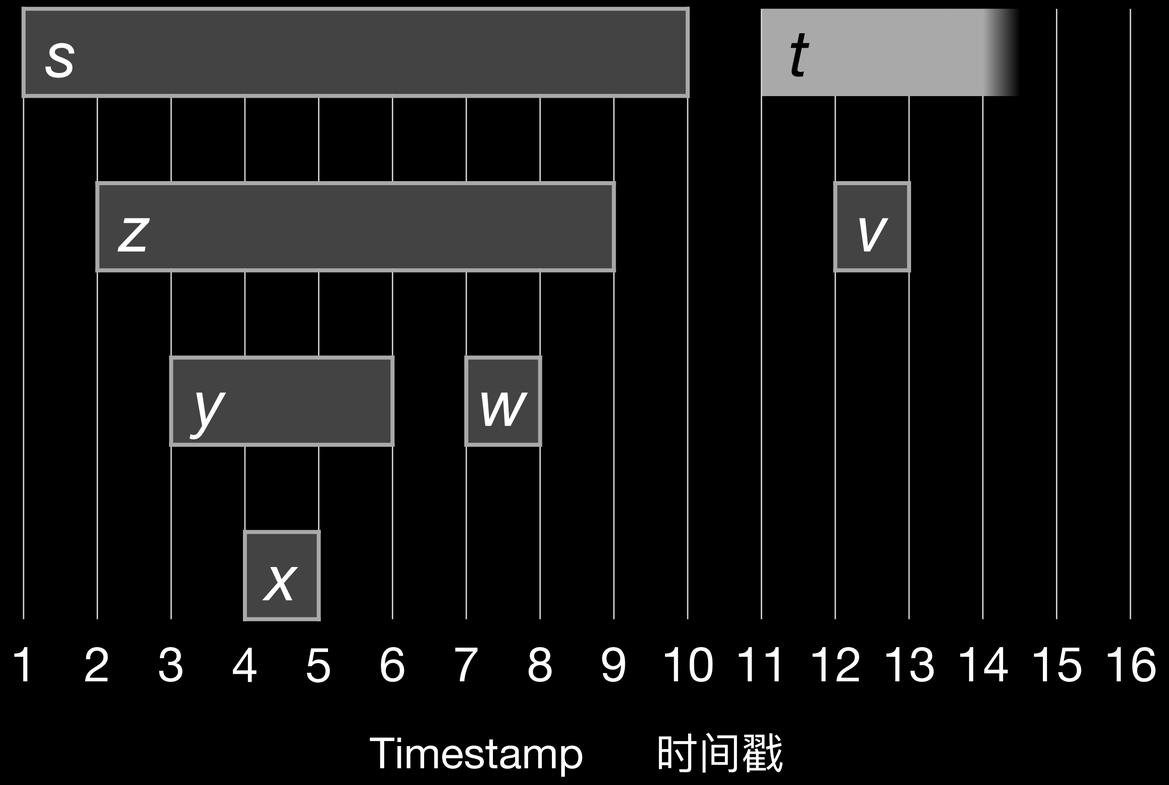


DFS(*G*) → DFS-VISIT(*G*,*t*) → DFS-VISIT(*G*,*v*)

Example

例子

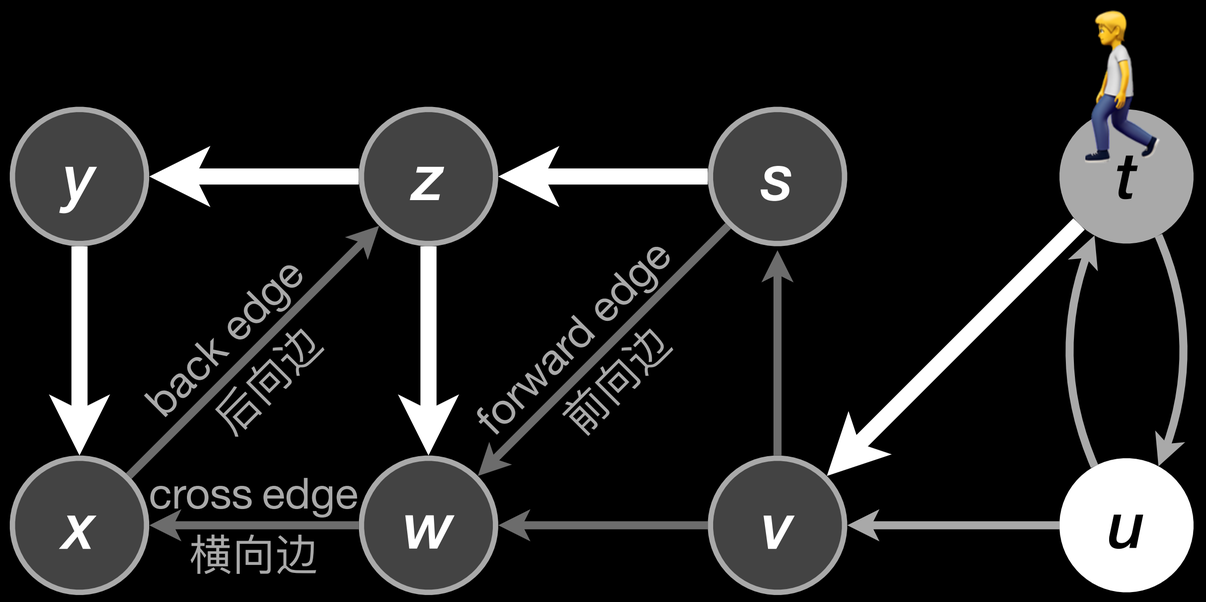


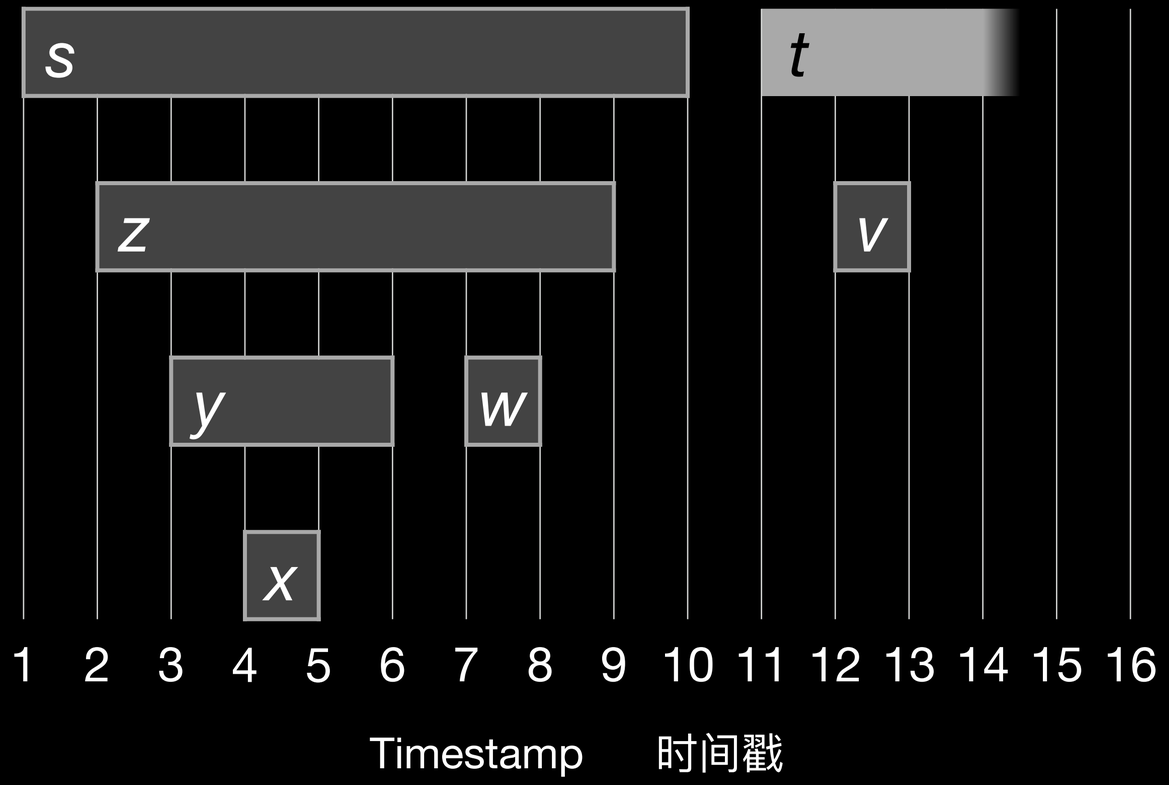


DFS(*G*) → DFS-VISIT(*G*,*t*) → DFS-VISIT(*G*,*v*)

Example

例子

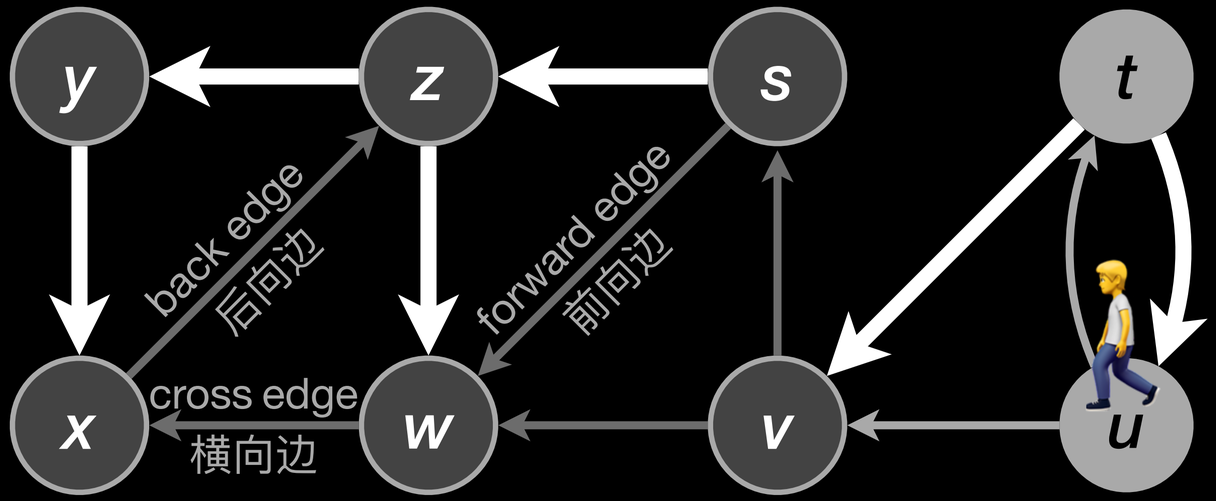


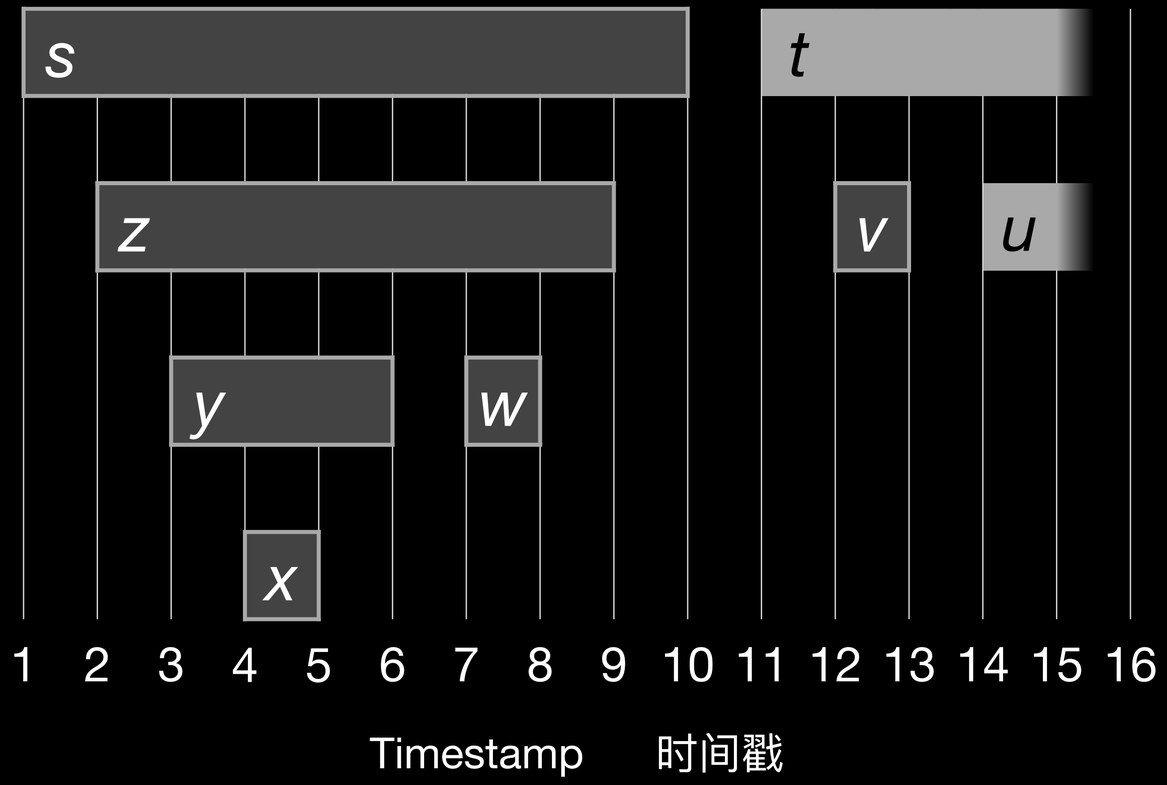


DFS(*G*) → DFS-VISIT(*G*,*t*)

Example

例子

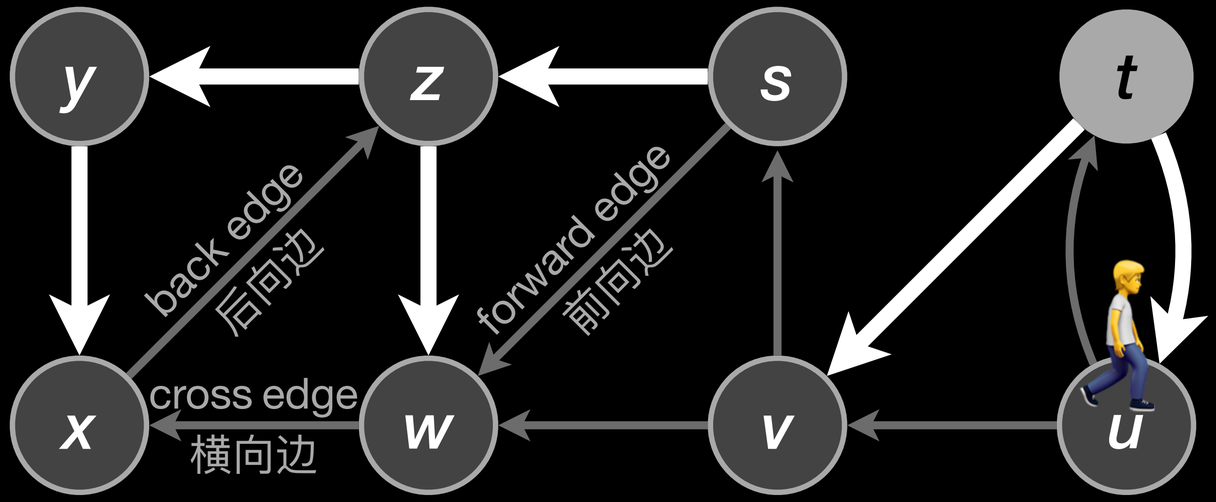


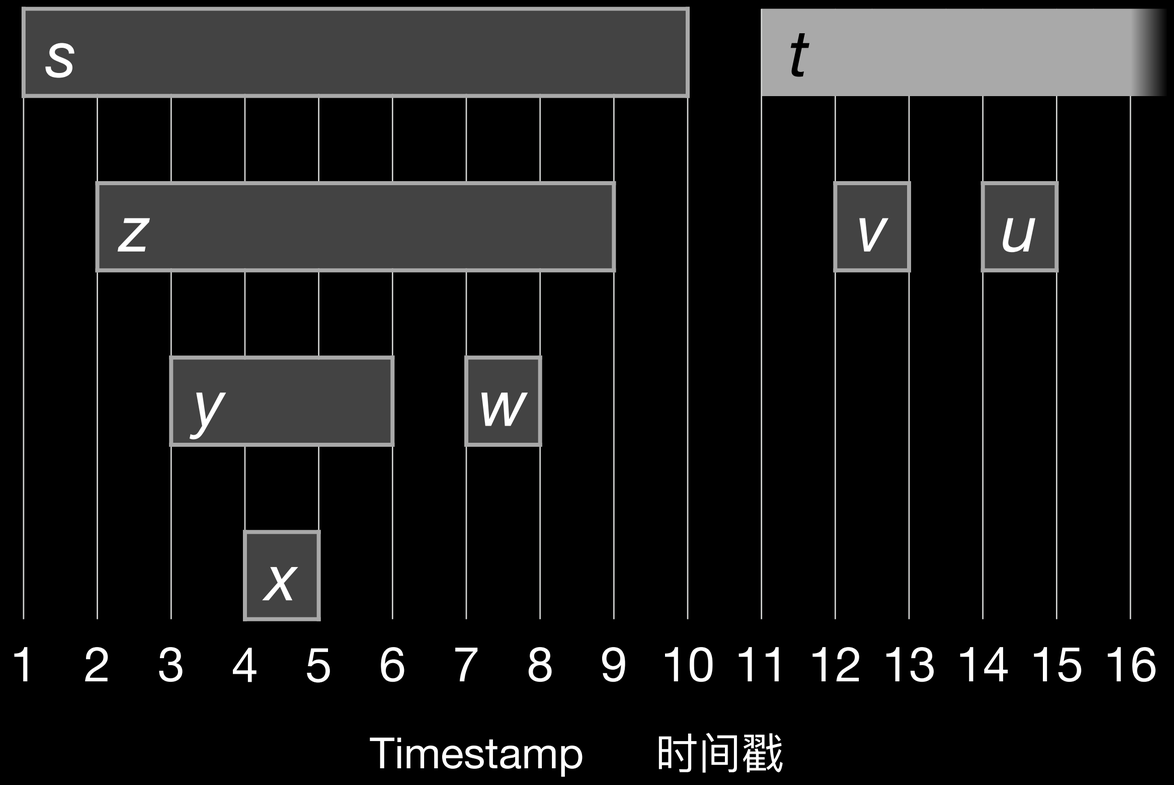


DFS(*G*) → DFS-VISIT(*G*,*t*) → DFS-VISIT(*G*,*u*)

Example

例子

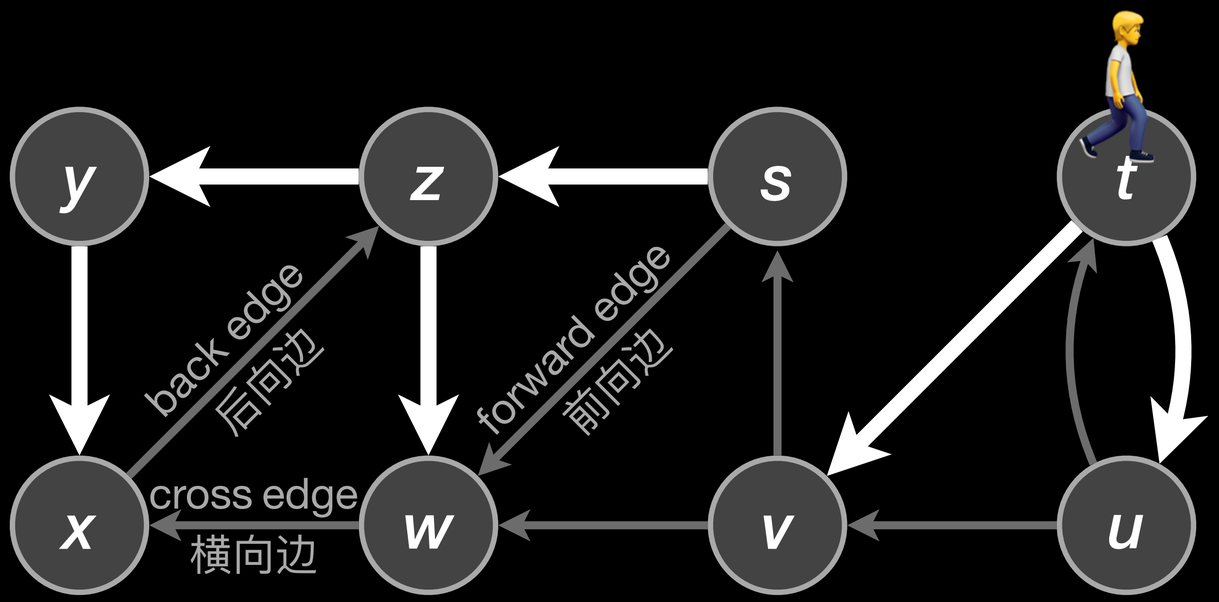


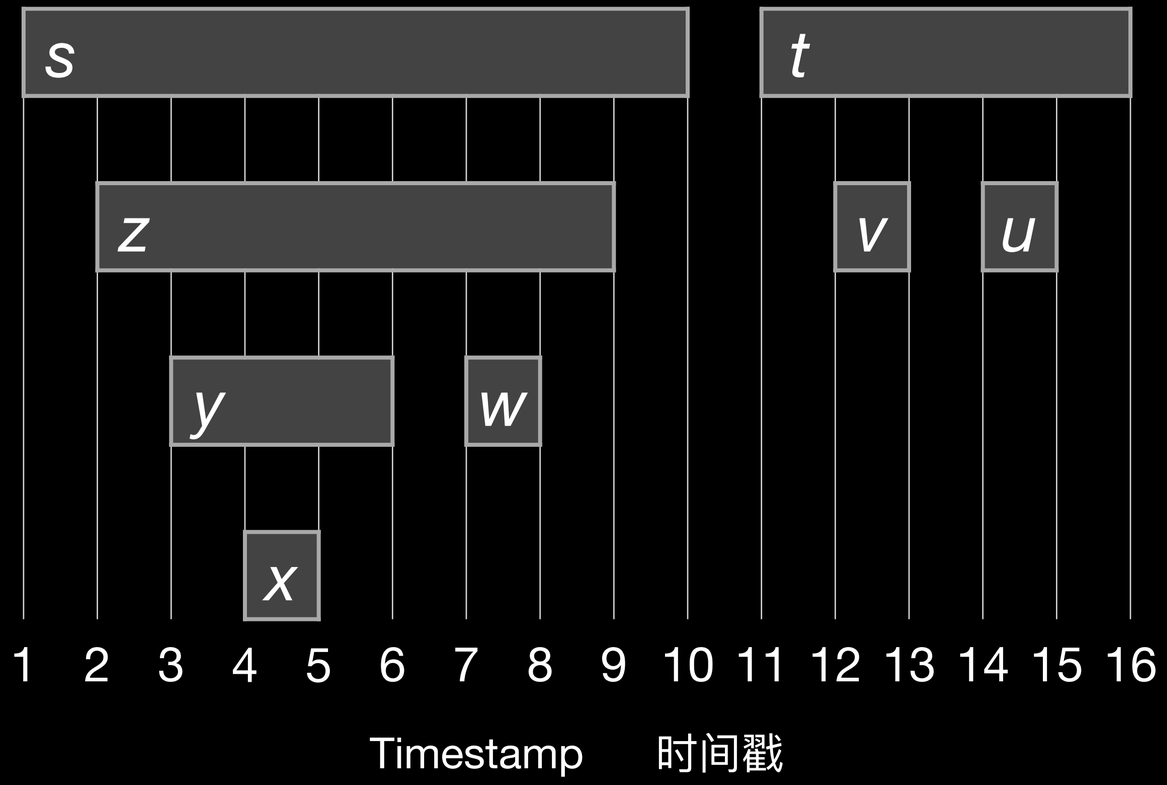


DFS(*G*) → DFS-VISIT(*G*,*t*) → DFS-VISIT(*G*,*u*)

Example

例子

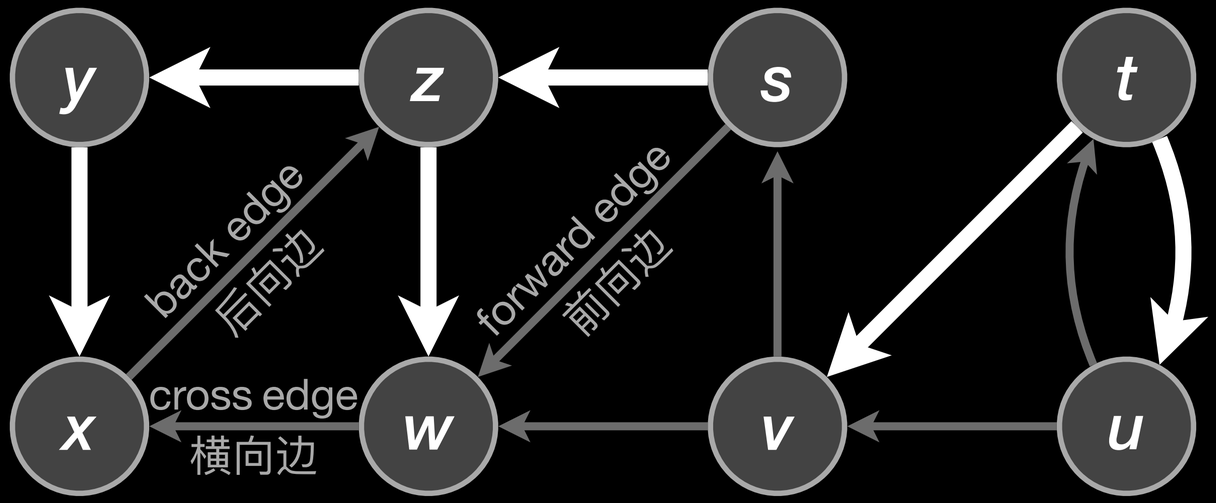


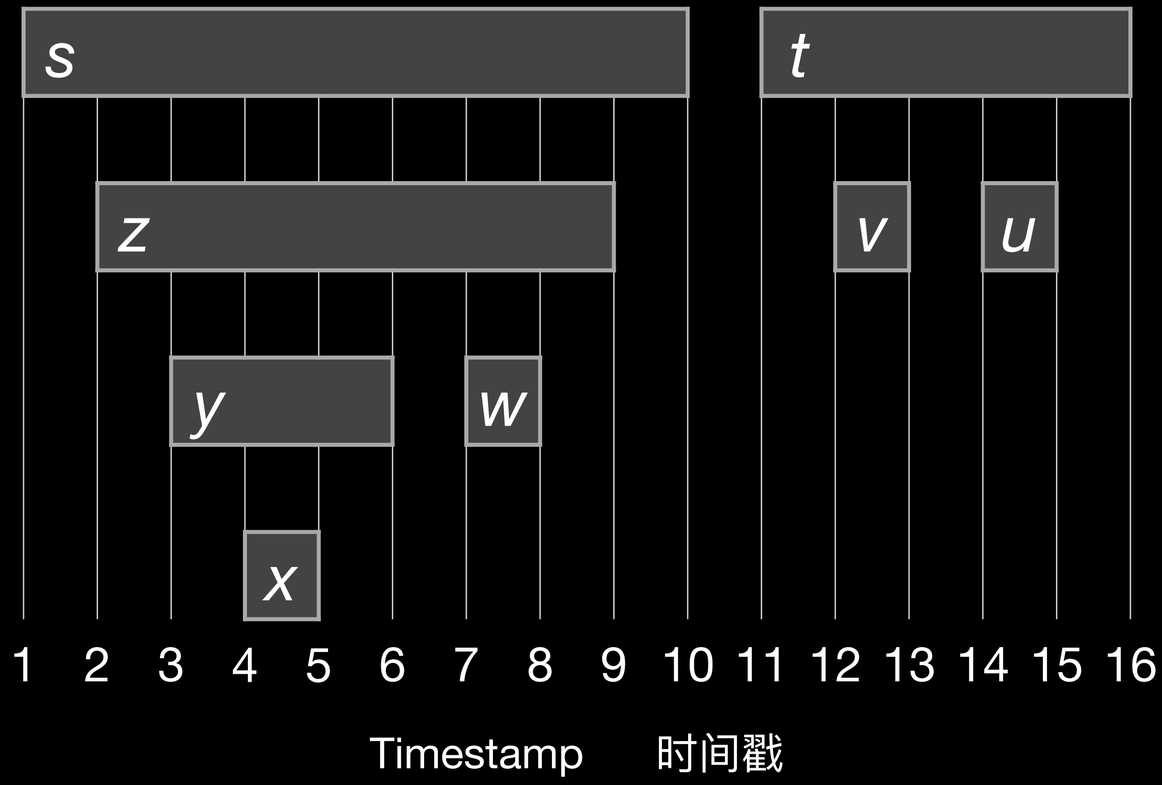


DFS(*G*) → DFS-VISIT(*G*,*t*)

Example

例子

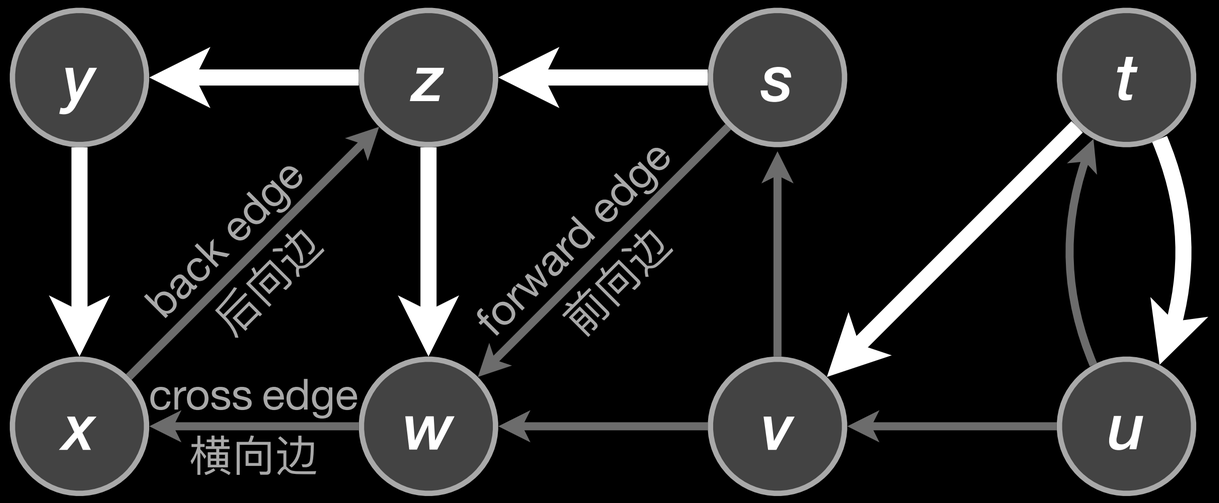


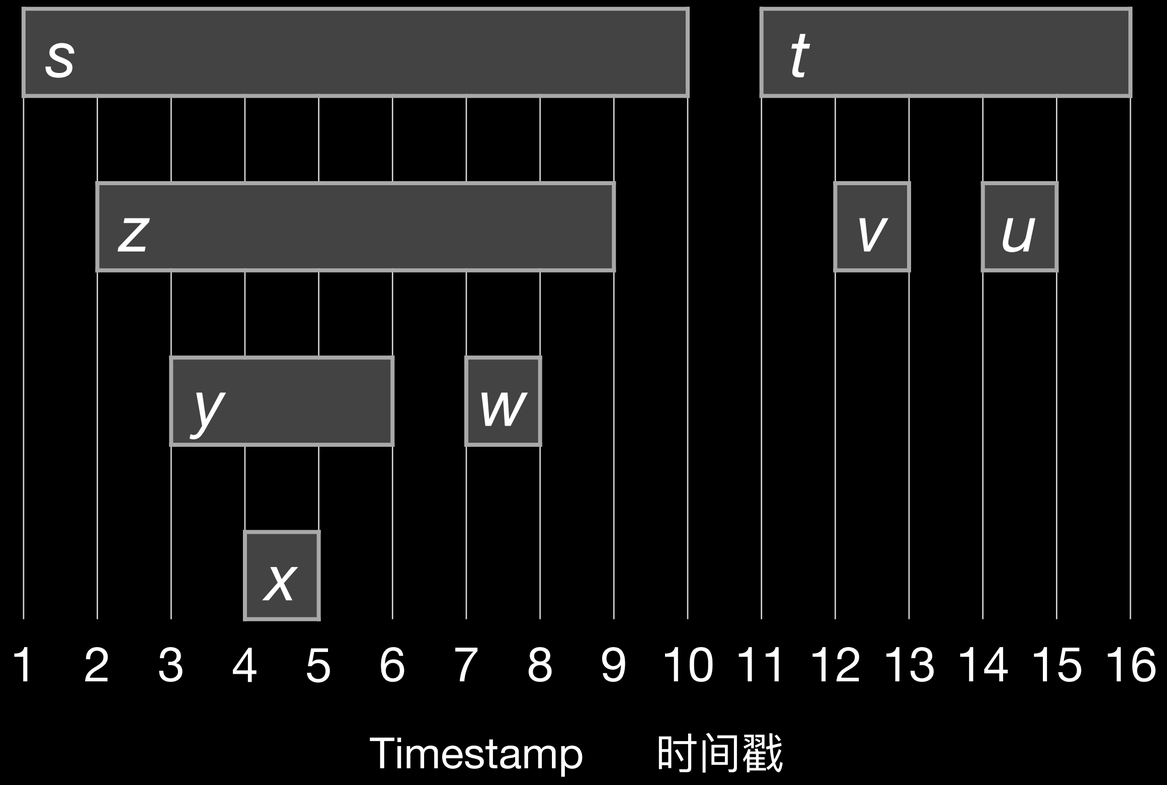


DFS(*G*)

Example

例子


y
z
s
t
x
w
v
u
back edge
后向边
forward edge
前向边
cross edge
横向边

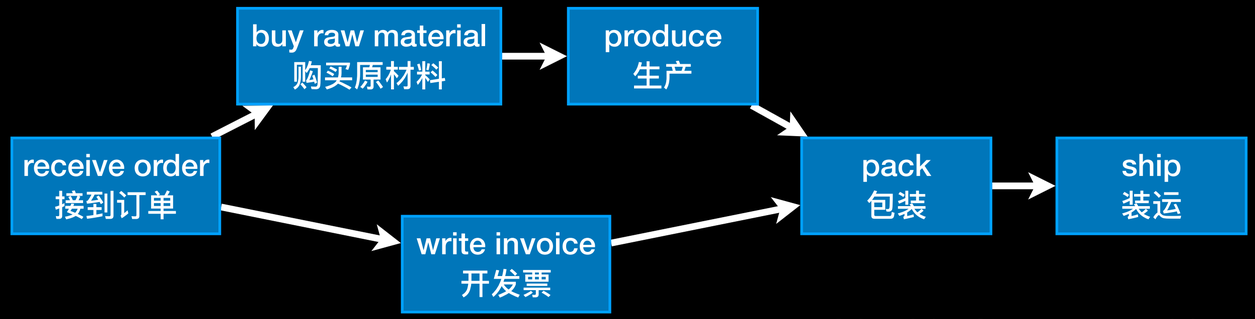

s
t
z
v
u
y
w
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Topological Sort 拓扑排序

Topological Sort

拓扑排序

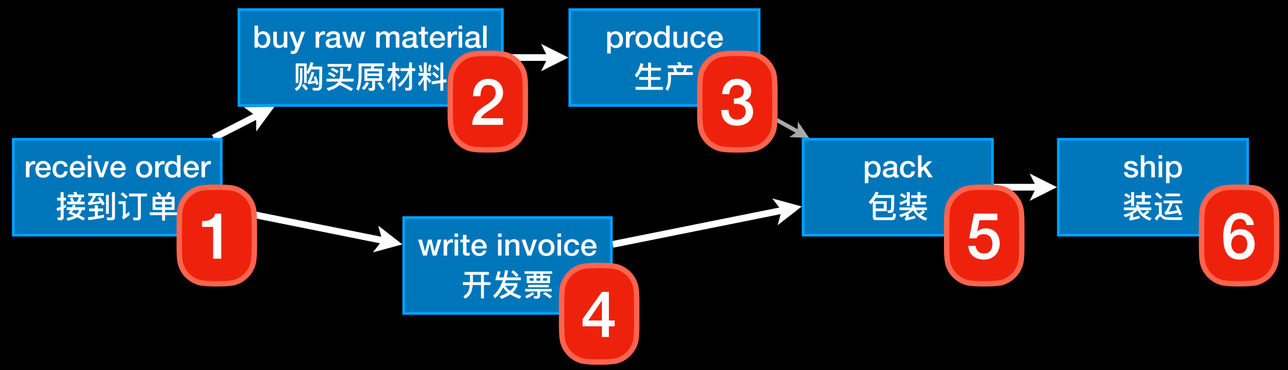
- **Problem:** linearize a process
- 问题：将过程线性化

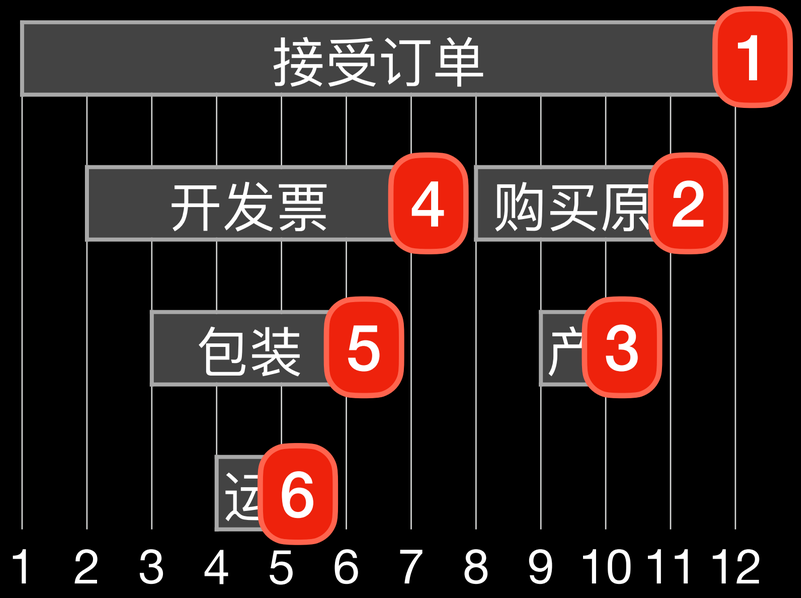


- in mathematical terms:
  find a total order
  compatible with the partial order
- 用数学术语来说：
  查找总偏序
  与偏序相容

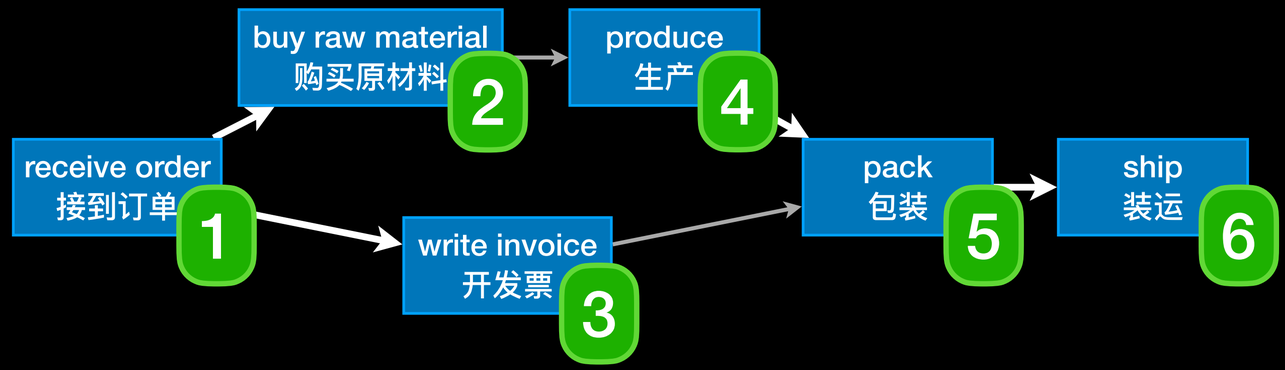
- **Solution:** use DFS and order according to *v.f* (time when examination ends)
- 解决方案：使用DFS，
  排序根据*v.f*（检查结束时间）

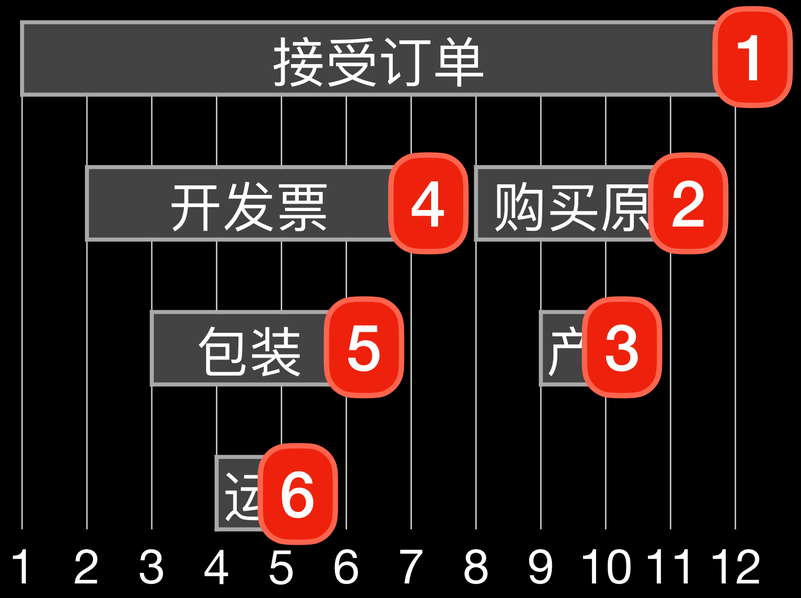
Topological Sort 拓扑排序





Topological Sort 拓扑排序





Topological Sort 拓扑排序

TOPOLOGICAL-SORT(*G*)
L = empty list
Call DFS(*G*), and as each vertex is finished,
insert it onto the front of *L*
return *L*

DFS-VISIT(*G*,*u*)
u.*color* = gray, *time* = *time* + 1, *u*.*d* = *time*
for each *v* adjacent to *u*
if *v*.*color* == white
v.π = *u*
DFS-VISIT(*G*,*v*)
u.*color* = black, *time* = *time* + 1, *u*.*f* = *time*
insert *u* onto the front of *L*

Correctness

- Does this algorithm skip any vertex? Does it duplicate any vertex?
  — No, every vertex *v* is visited exactly once by DFS-VISIT(*G*,*v*).
- Does this procedure order the vertices correctly?
  — See Theorem 22.12.

正确性

- 该算法是否跳过任何顶点? 它是否复制任何顶点?
  — 不，DFS-VISIT(*G*)只访问每个顶点一次。
- 此过程是否正确排列顶点?
  — 见定理22.12。

Correctness

正确性

Lemma 22.11:
A directed graph *G* is acyclic iff
a depth-first search yields no back edges.

引理22.11：
有向图G是非循环当且仅当
深度优先搜索不会产生后缘。

Theorem 22.12:
TOPOLOGICAL-SORT(*G*) produces a
topological sort of the directed acyclic G.

定理22.12：
TOPOLOGICAL-SORT(*G*)产生
有向无环*G*的拓扑排序。

Running Time

- TOPOLOGICAL-SORT(G)
  is a small modification of DFS(G)
  and runs in the same time complexity.

- DFS(G) runs in time $O(|V| + |E|)$.

运行时间

- TOPOLOGICAL-SORT(G)
  是DFS(G)的一个小修改
  并在相同的时间复杂度下运行。

- DFS(G)在时间$O(|V| + |E|)$中运行。

Strongly Connected Components

強連通分量

Strongly Connected Components

- In a directed graph there may be cycles.
- Vertices *u*,*v* that can reach each other (*u* ⇝ *v* ⇝ *u*) are *strongly connected.*
- A *strongly connected component* is a maximal set of strongly connected vertices.

强连通分量

- 在有向图中可能有圈。
- 可以相互接触的顶点*u*、*v*（*u* ⇝ *v* ⇝ *u*） 它们之间有很强连通。
- 强连通分量是一组最大的强连通顶点。

Strongly Connected Components

- Lemma 22.13
  There cannot be a bidirectional link between two distinct SCCs.

强连通分量

- 引理22.13
  两个不同的SCC之间不能存在双向链路。

Use of SCCs

- example from research
- Which states can reach the target state?
  If one state in a SCC can reach,
  then the whole SCC can reach
  the target state.

强连通分量的使用

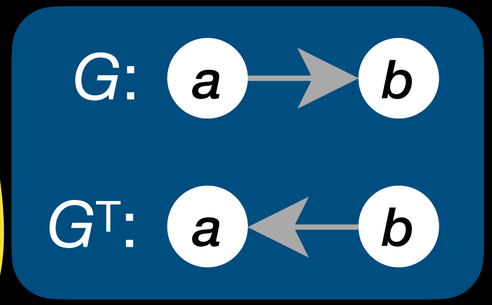
- 研究中的例子
- 哪些状态可以达到目标状态?

  如果SCC中的一个状态可以达到,
  那么整个SCC都可以达到
  目标状态。

Jansen/Groote/Timmers/Yang: **A Near-Linear-Time Algorithm for Weak Bisimilarity on Markov Chains.**
In: *CONCUR 2020.* Dagstuhl: Leibniz-Zentrum für Informatik, 2020 (LIPIcs 171).
https://doi.org/10.4230/LIPIcs.CONCUR.2020.8

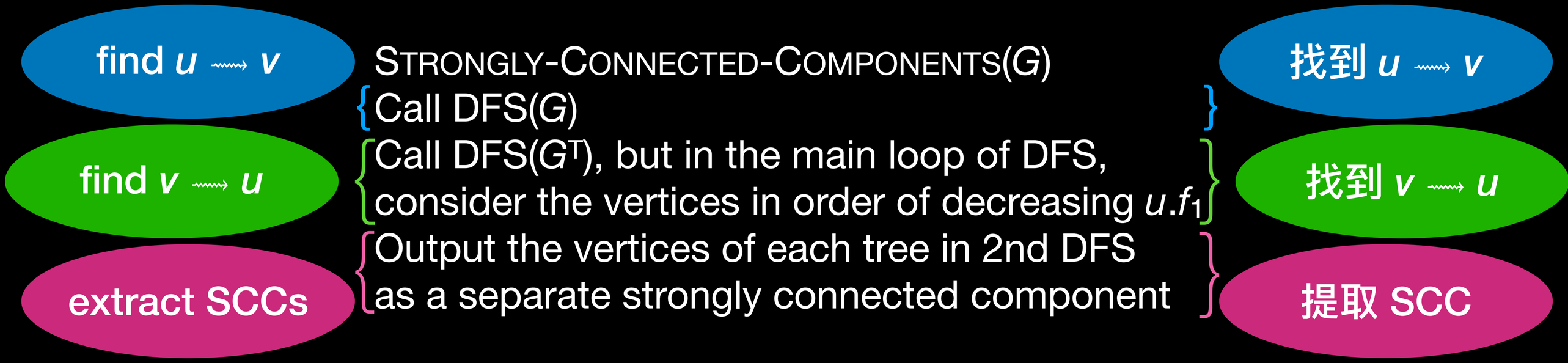
How to find SCCs　如何找到强连通分量

- If two vertices u,v are in the same SCC, then there are paths $u \rightsquigarrow v$ and $v \rightsquigarrow u$.
- Use double depth-first search:
  1. find a path $u \rightsquigarrow v$
  2. find a path $v \rightsquigarrow u$ (if $\exists\, u \rightsquigarrow v$)
- In the 2nd DFS:
  - Use G^T = transposed graph
  - Select roots of depth-first trees in order to ensure the condition.



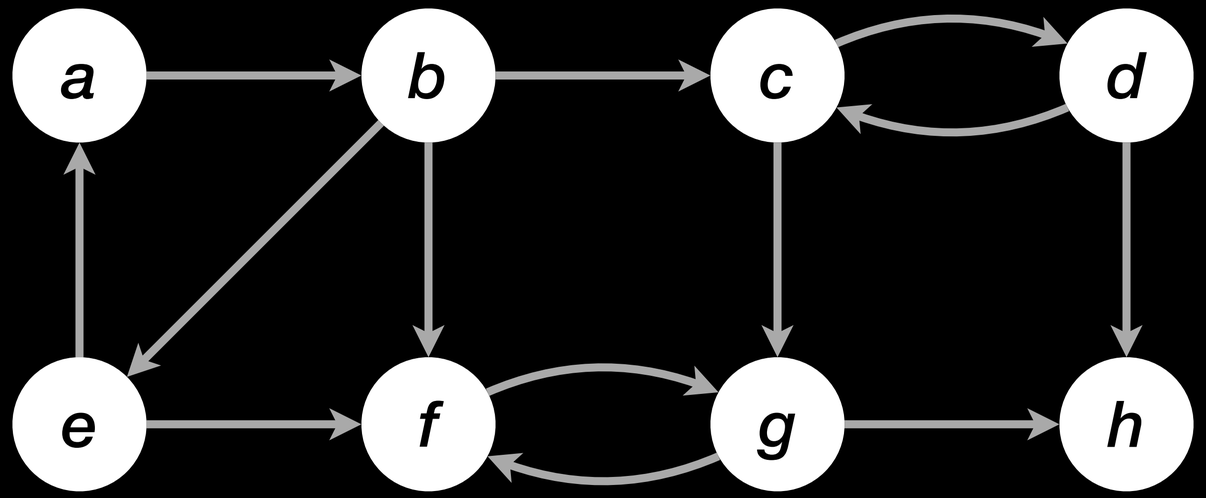
- 如果两条点u、v在一样的SCC，那么存在路行$u \rightsquigarrow v$和$v \rightsquigarrow u$。
- 用两次深度优先搜索：
  1. 找到路径$u \rightsquigarrow v$
  2. 找到路径$v \rightsquigarrow u$（如果存在$u \rightsquigarrow v$）
- 在第二个深度优先搜索：
  - 用G^T = 图的专制
  - 选择深度优先树的根为了保证条件。

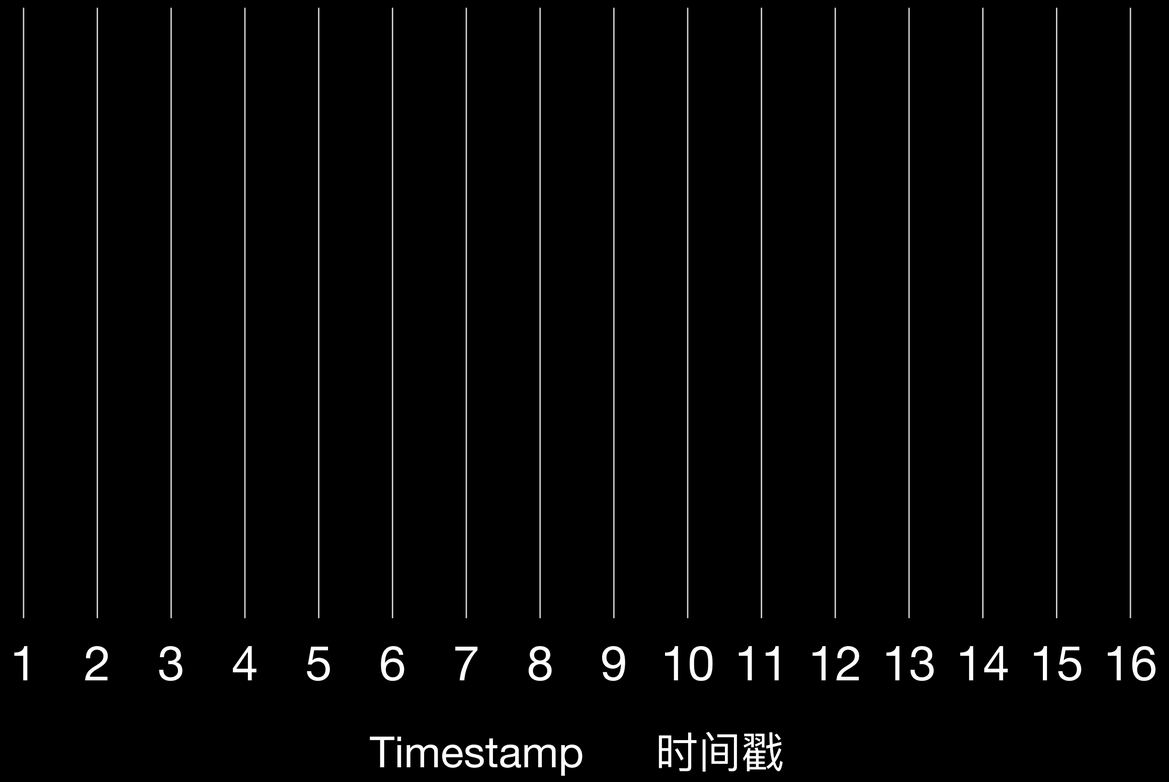
How to find SCCs　如何找到强连通分量



Example

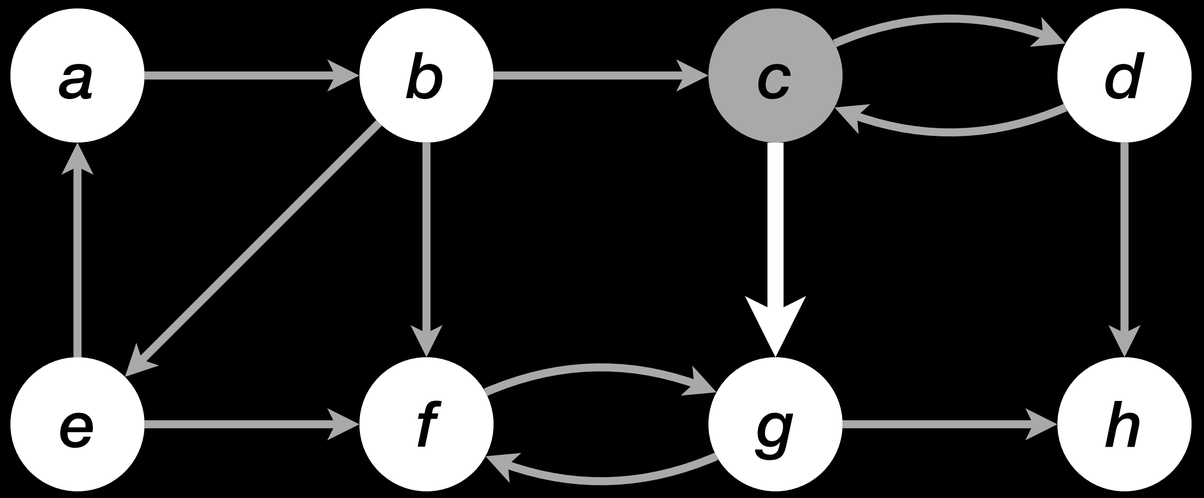
例子

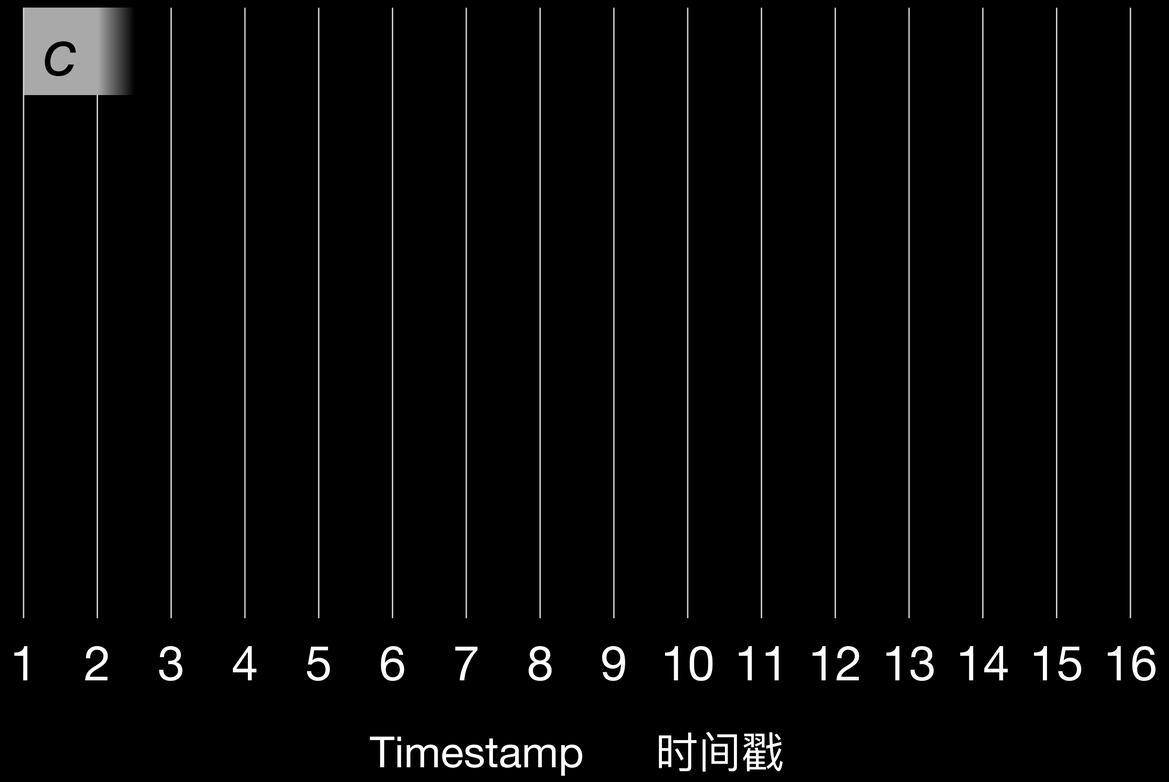




Example

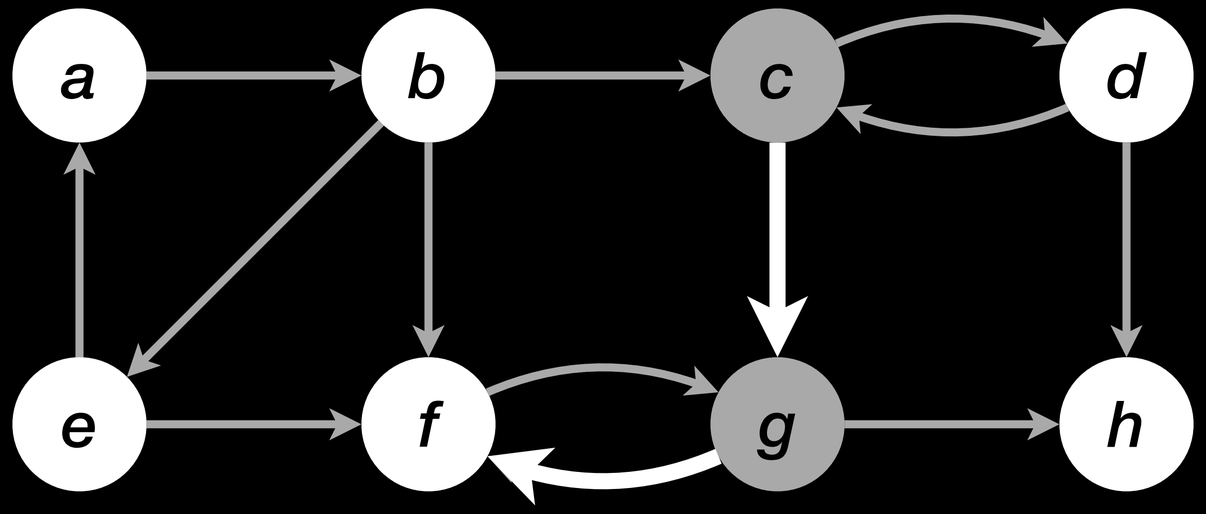
例子

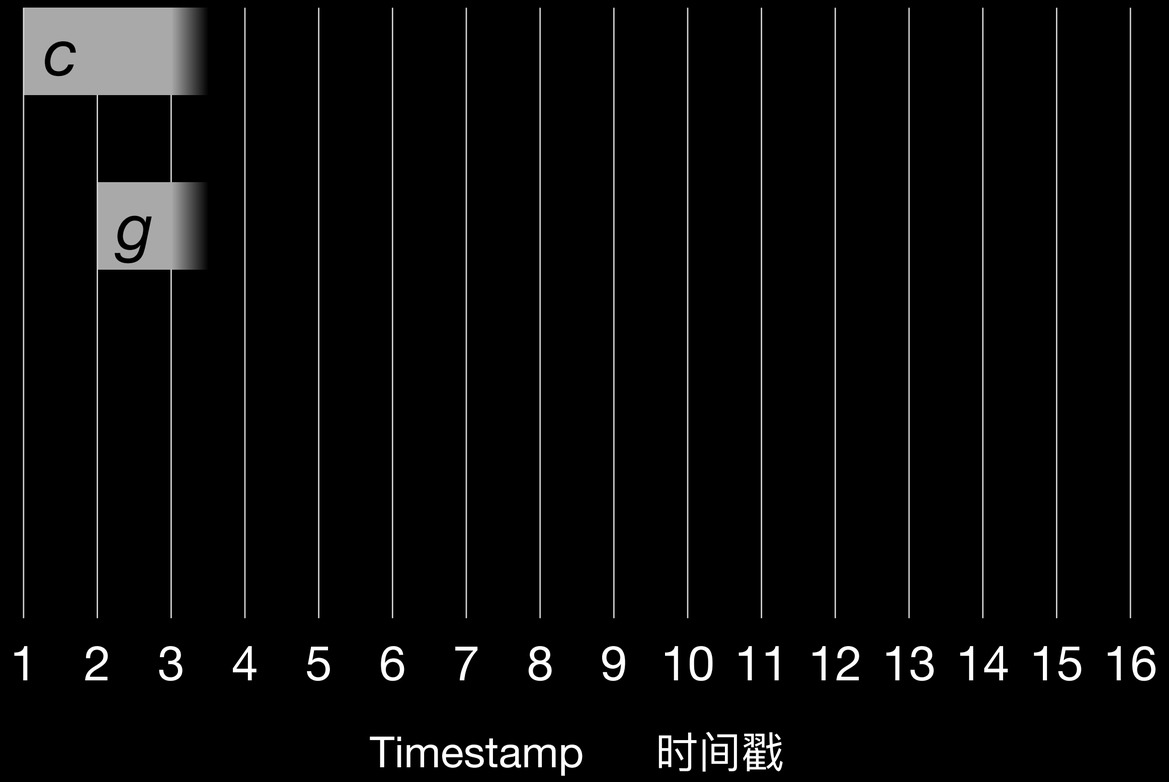




Example

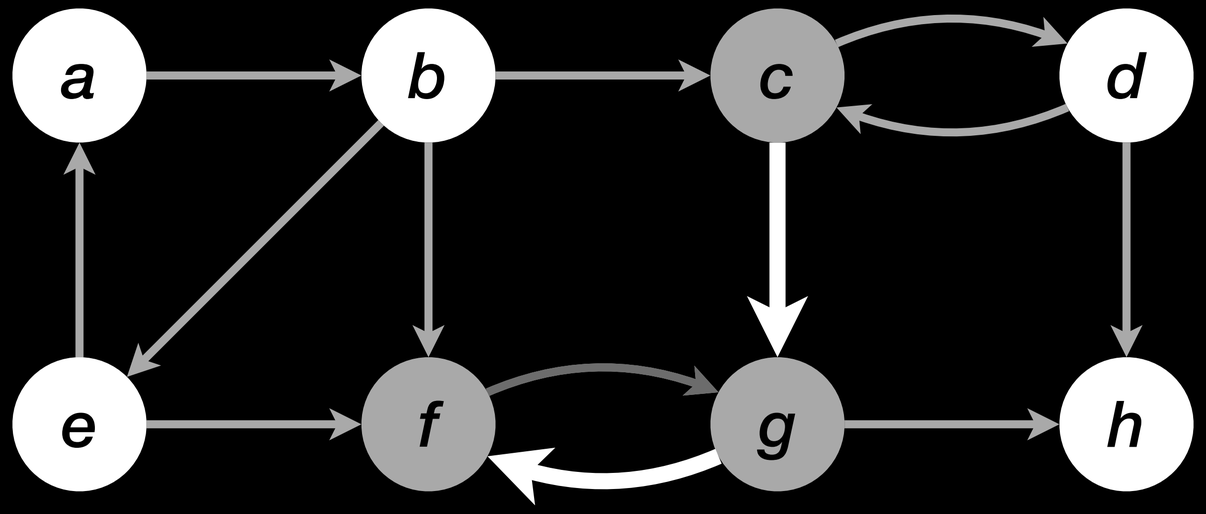
例子

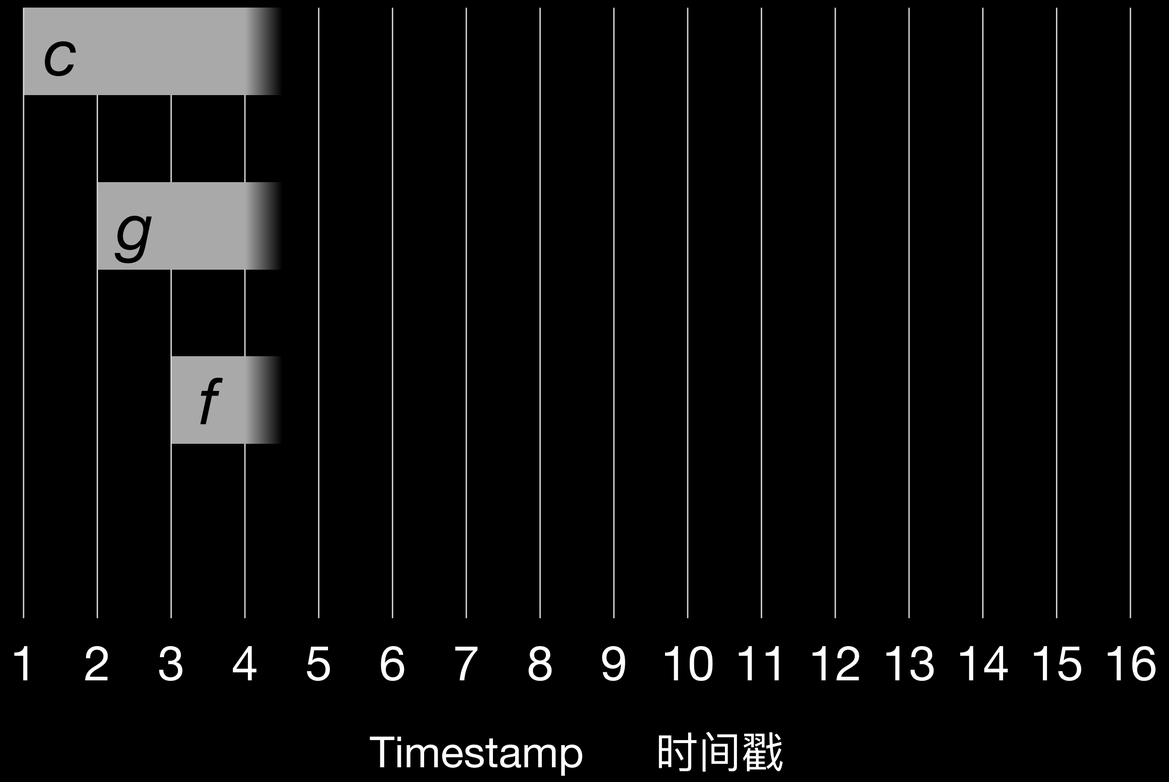




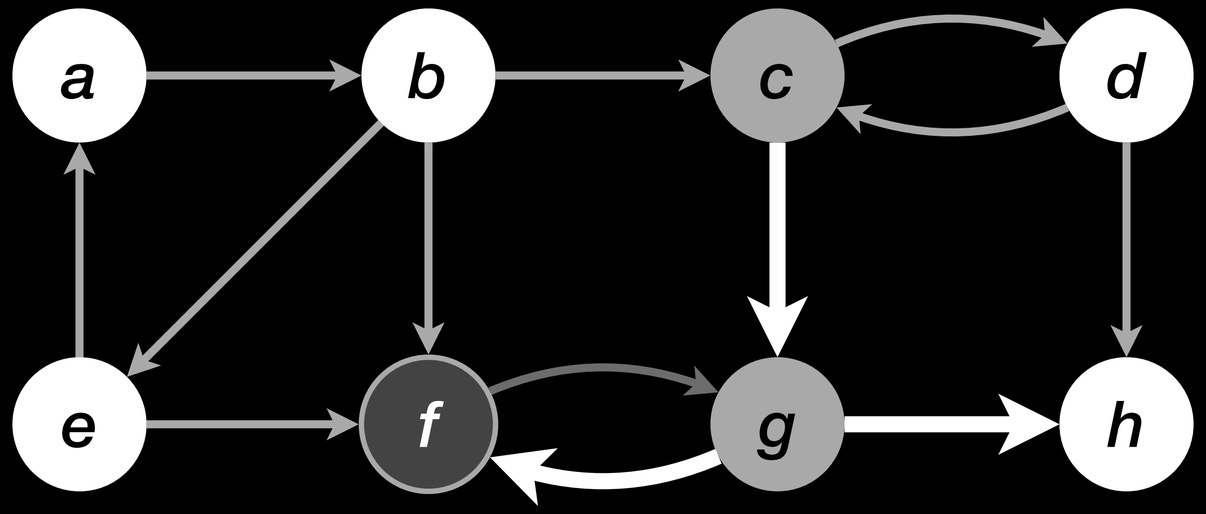
Example

例子

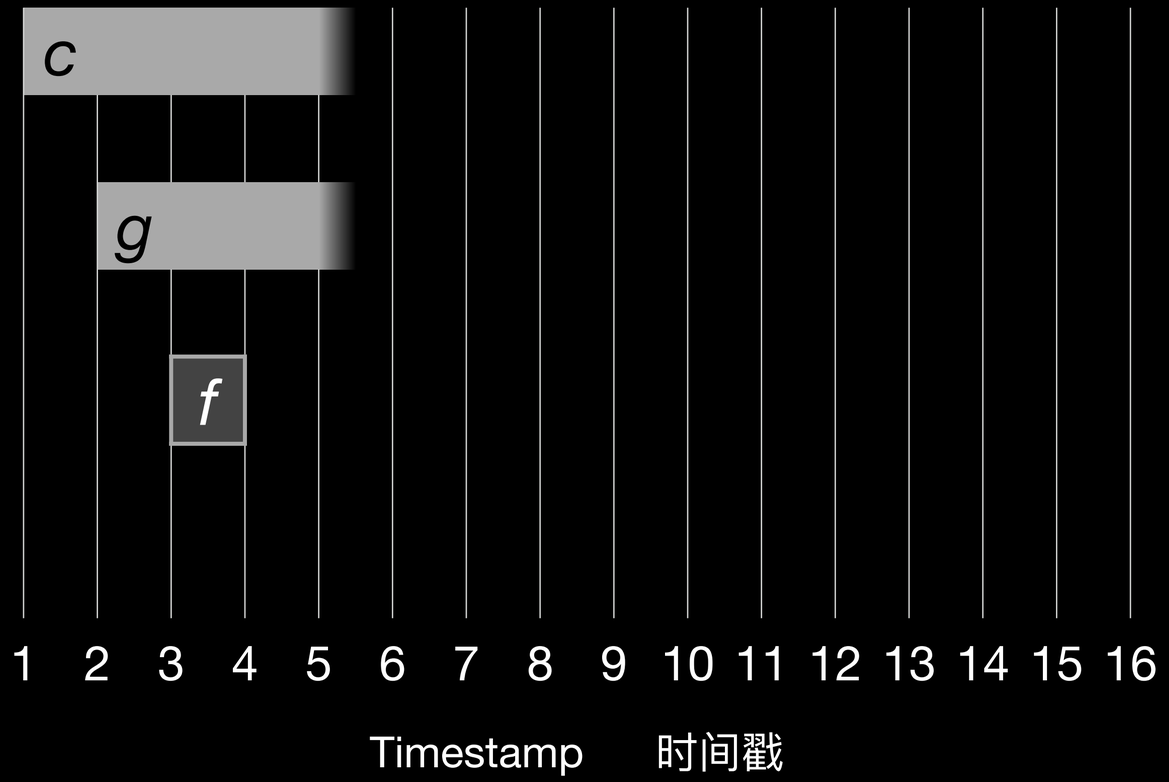




Example

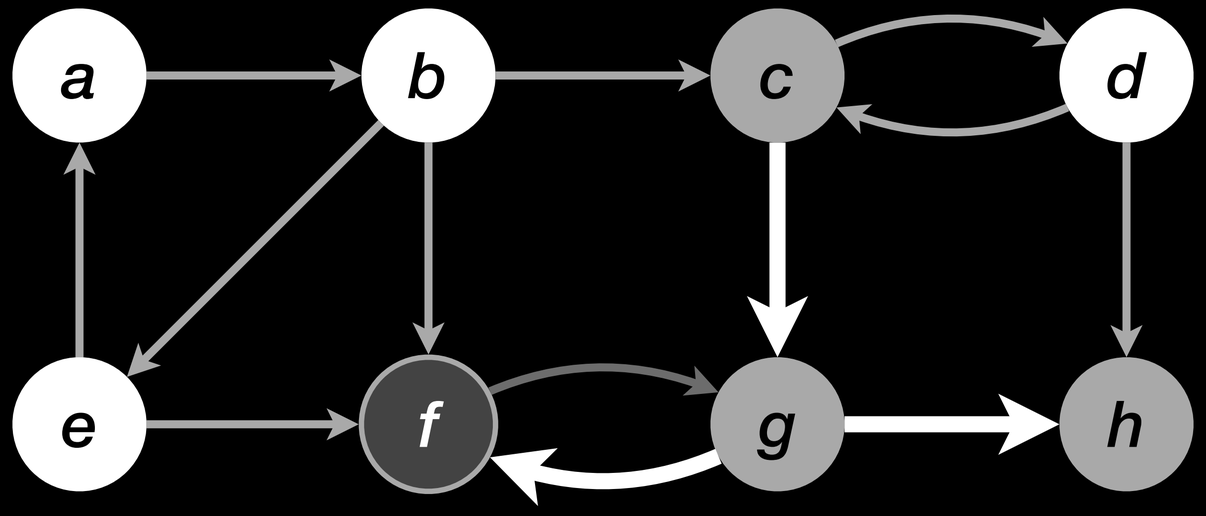


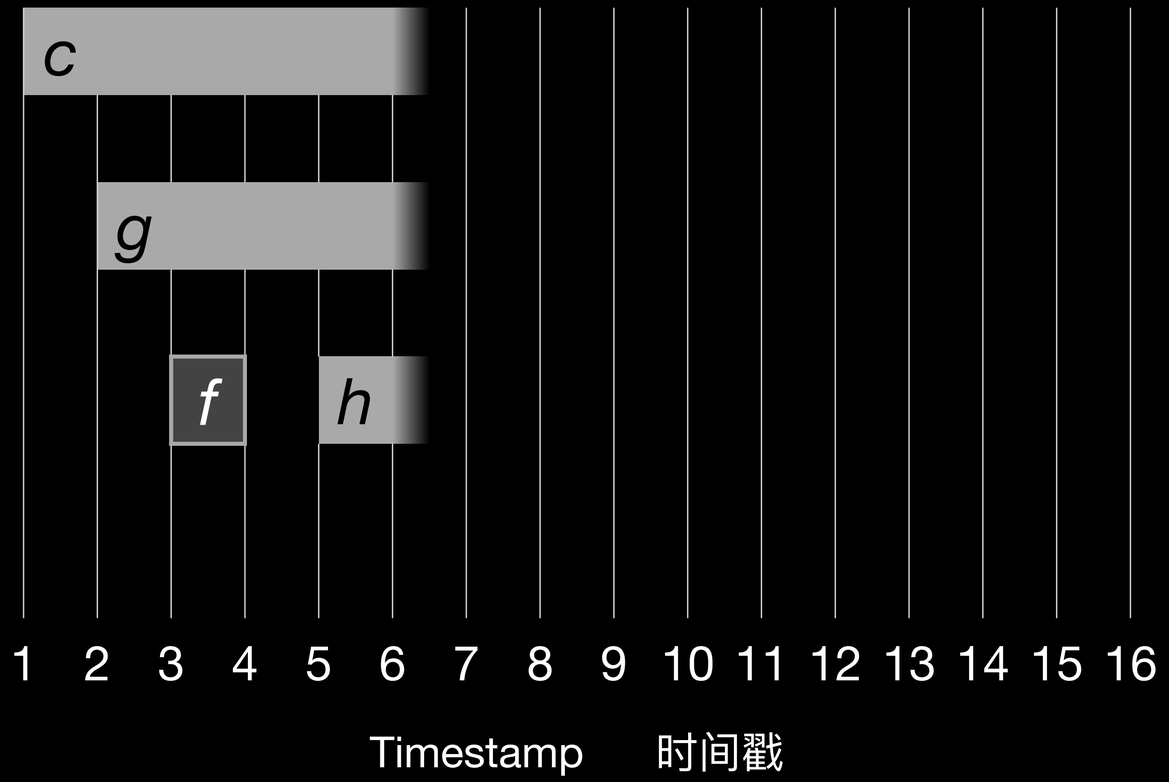
例子



Example

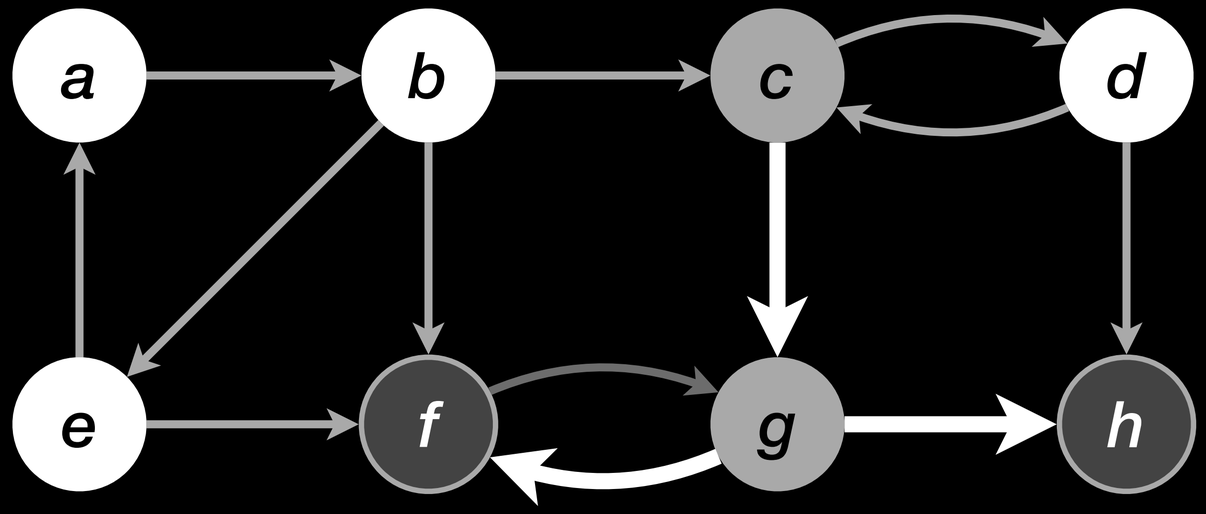
例子

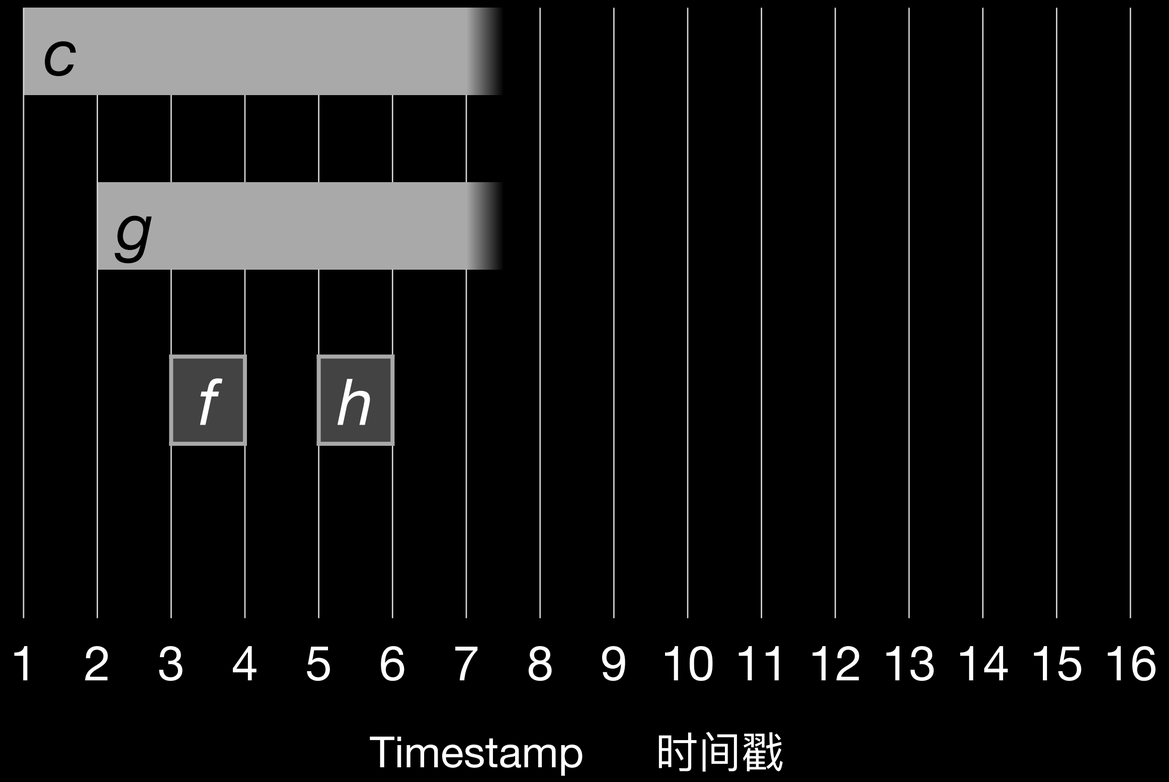




Example

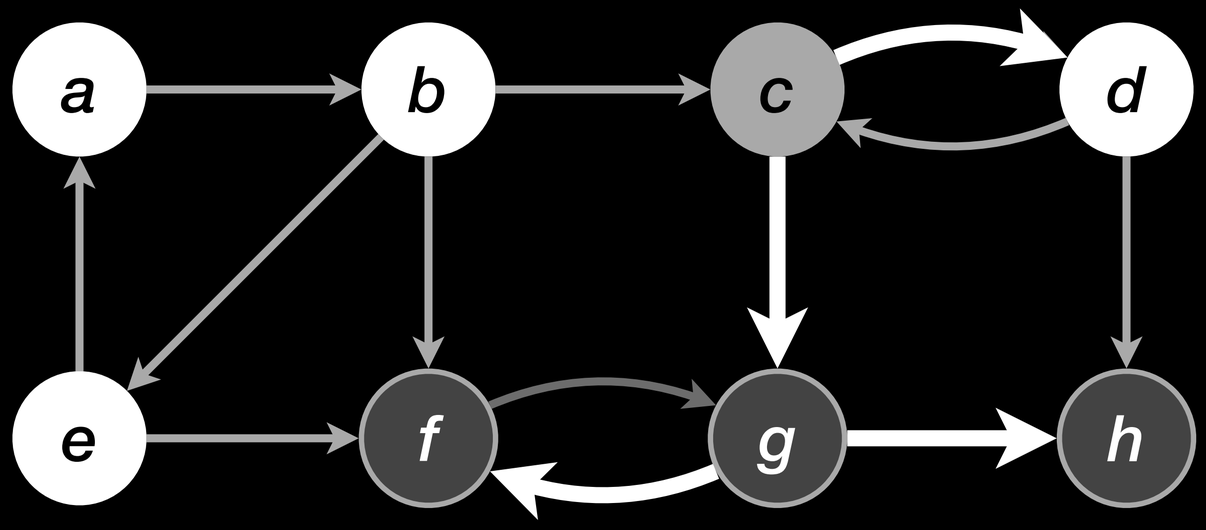
例子

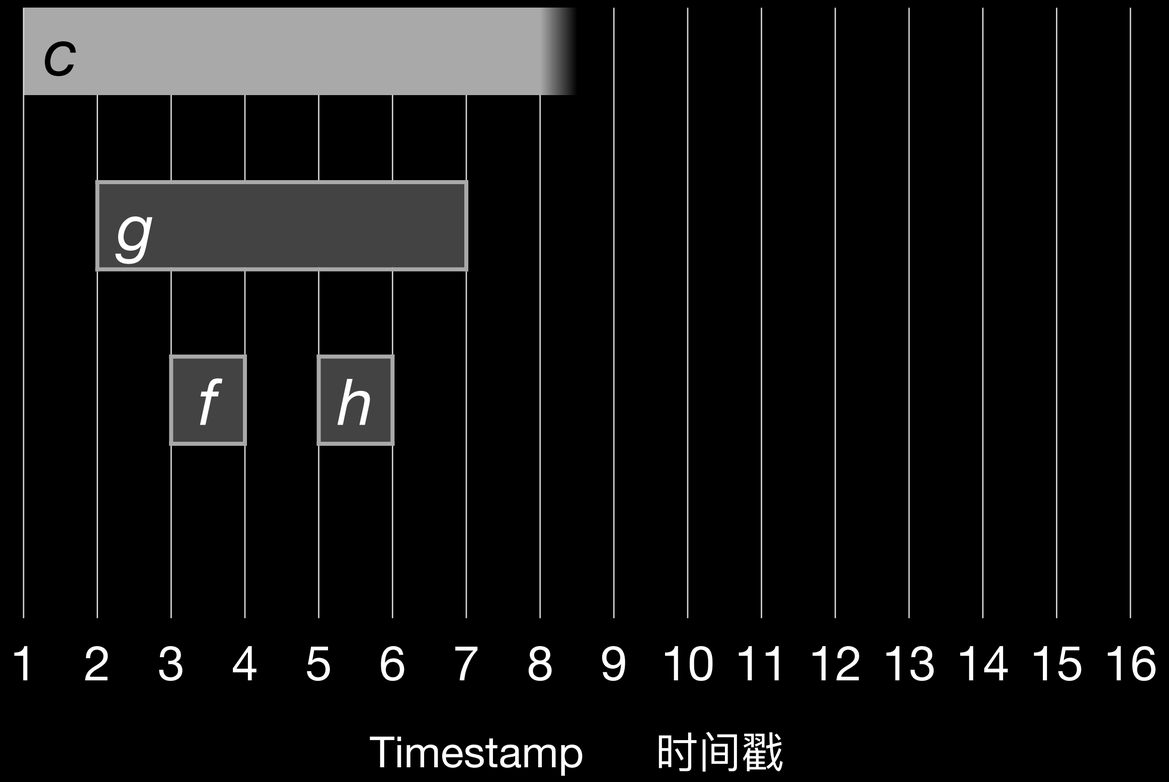




Example

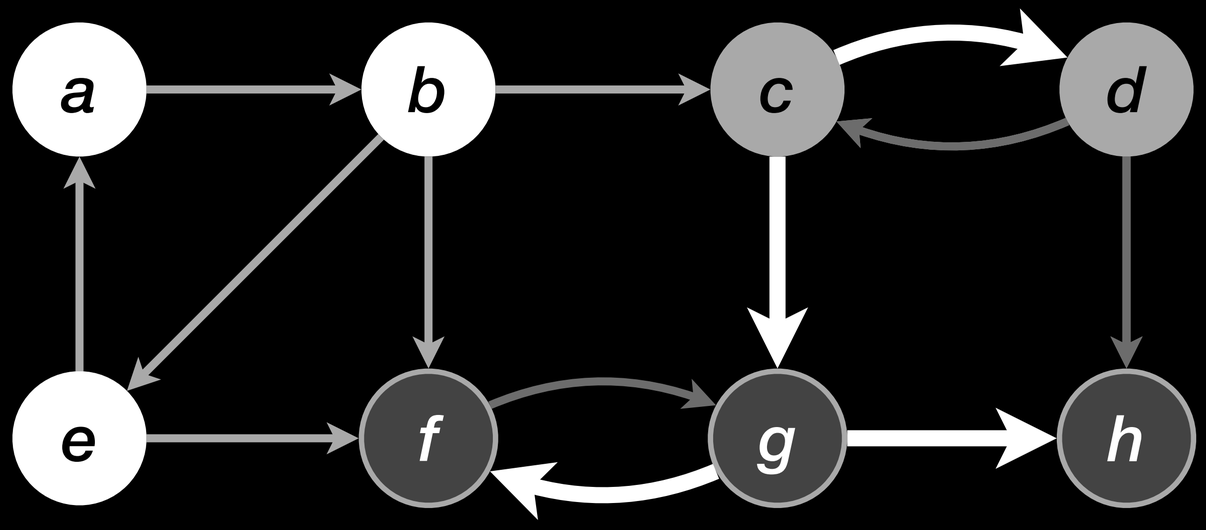
例子

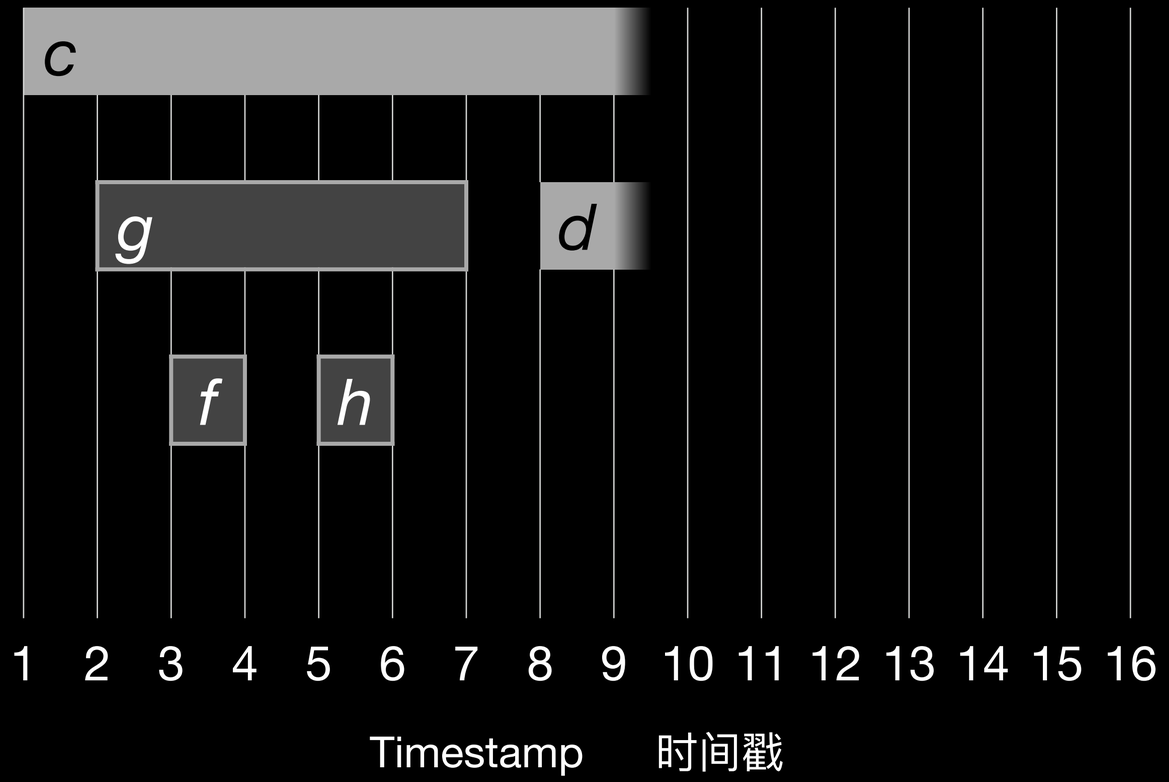




Example

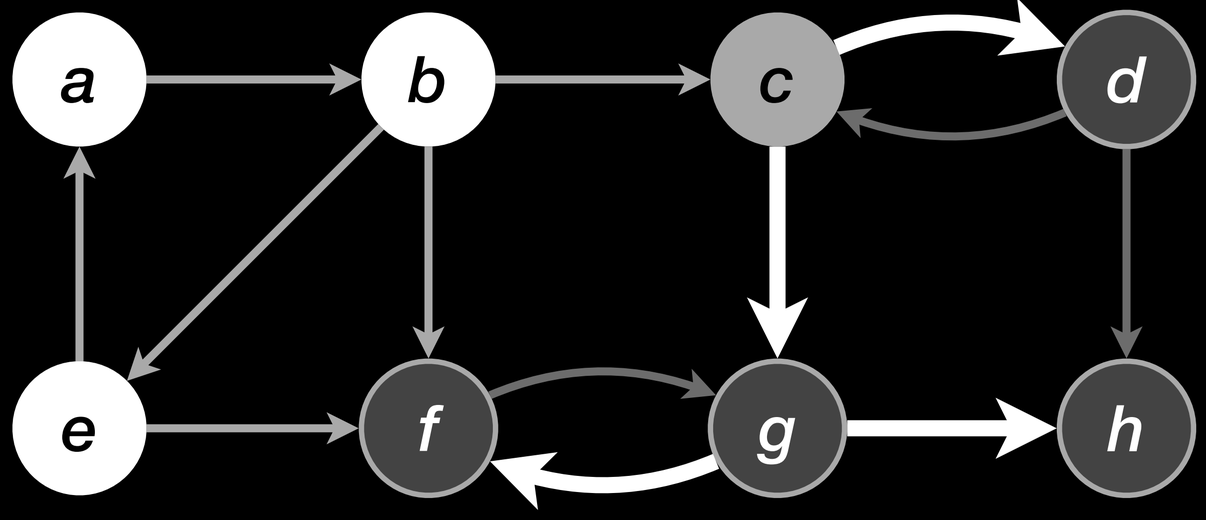
例子

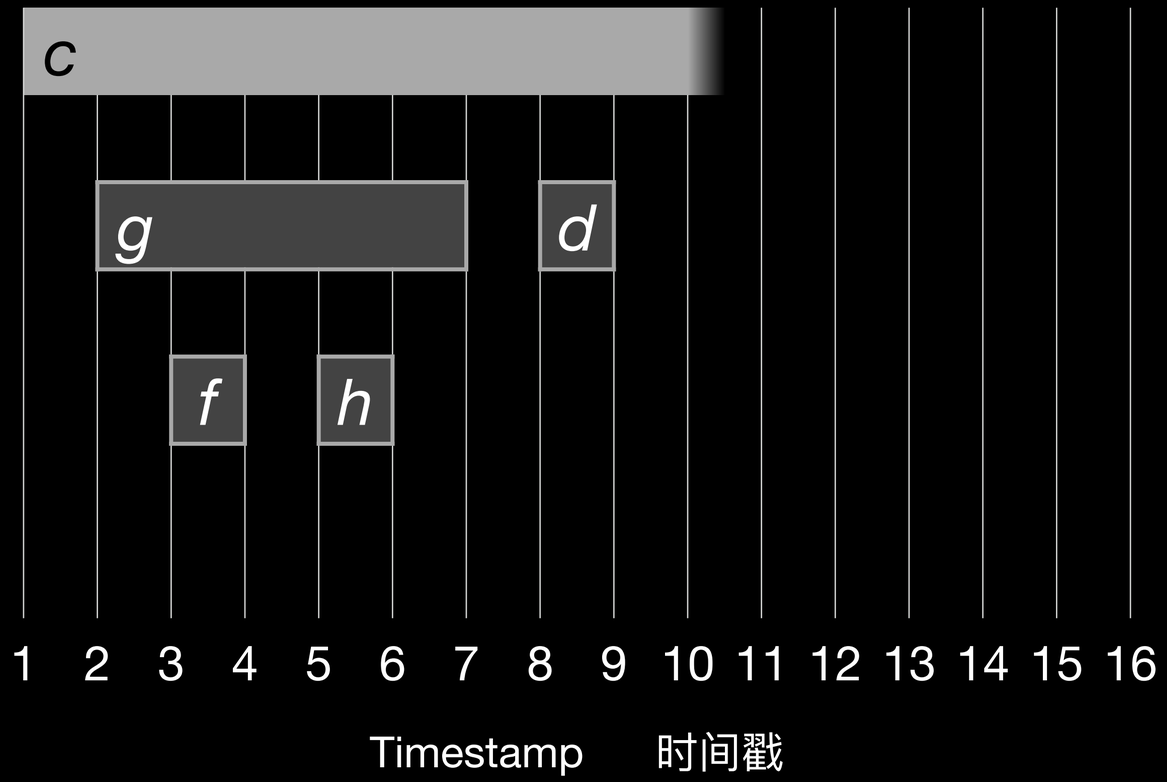




Example

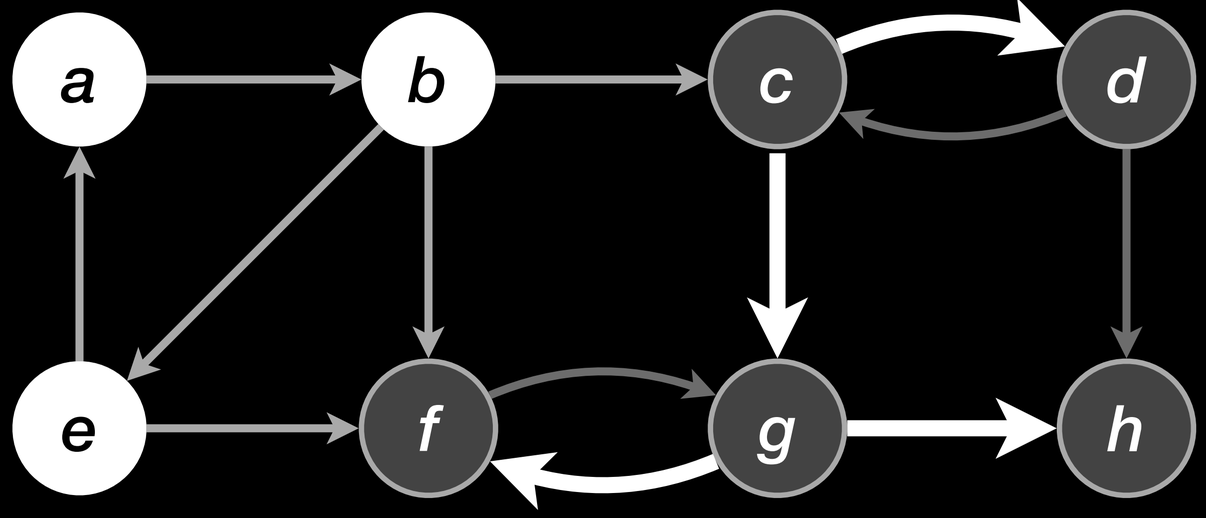
例子

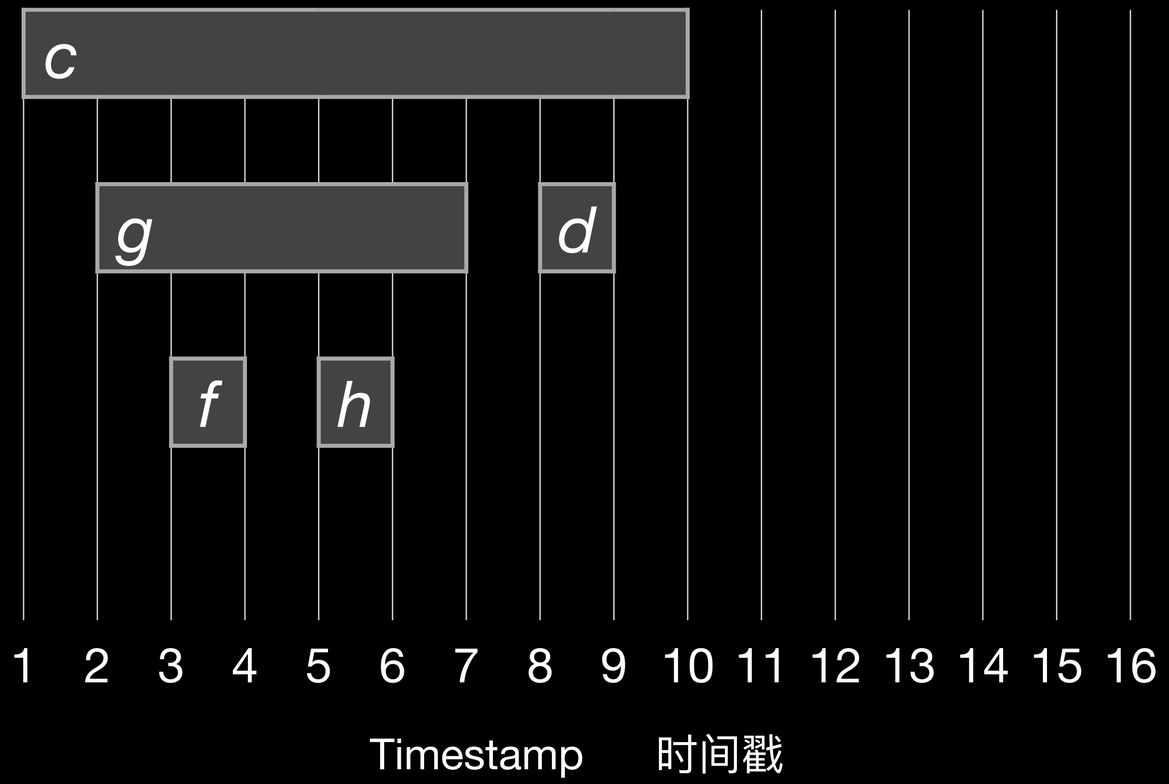




Example

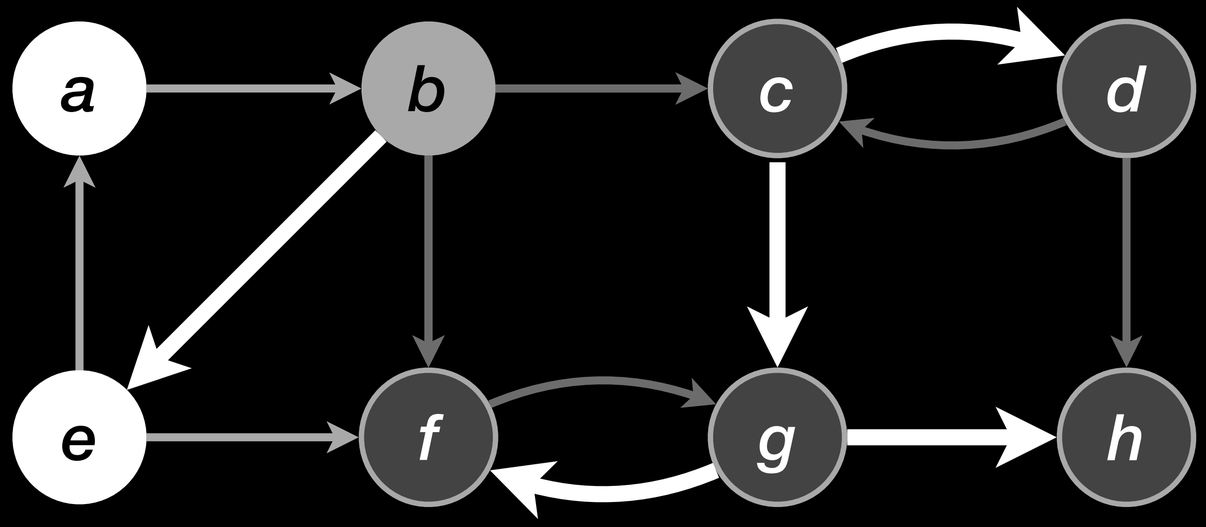
例子

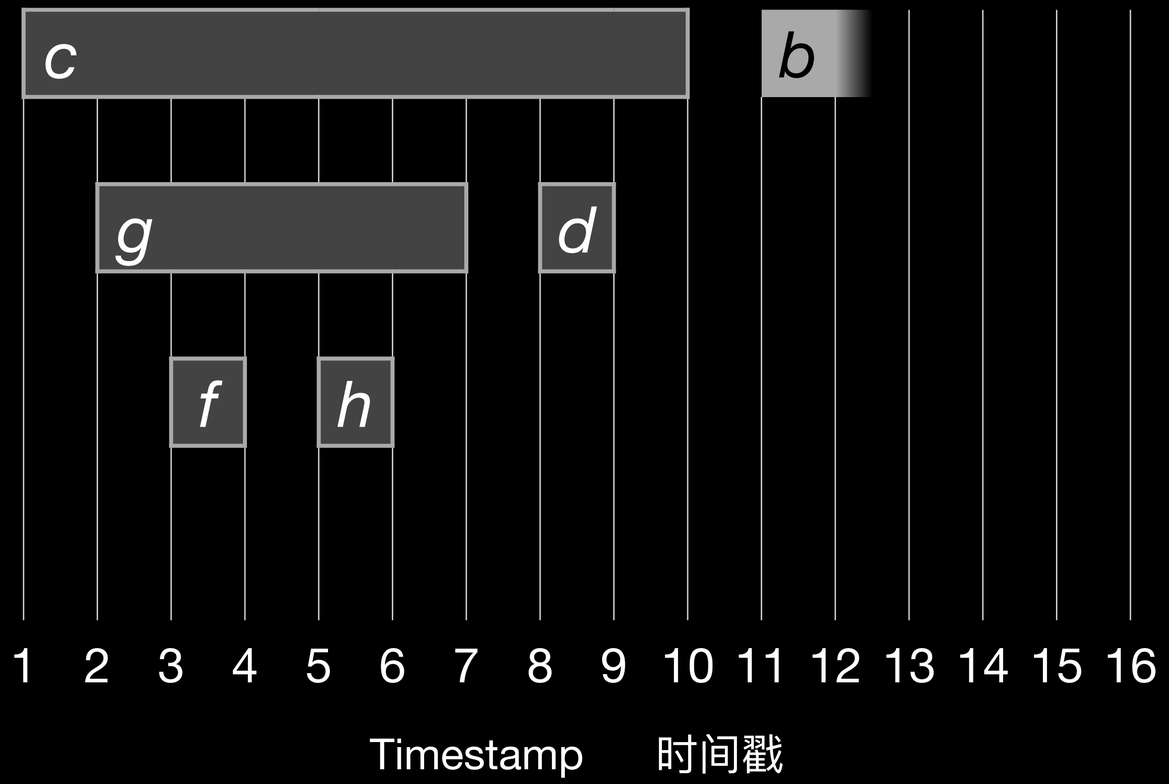




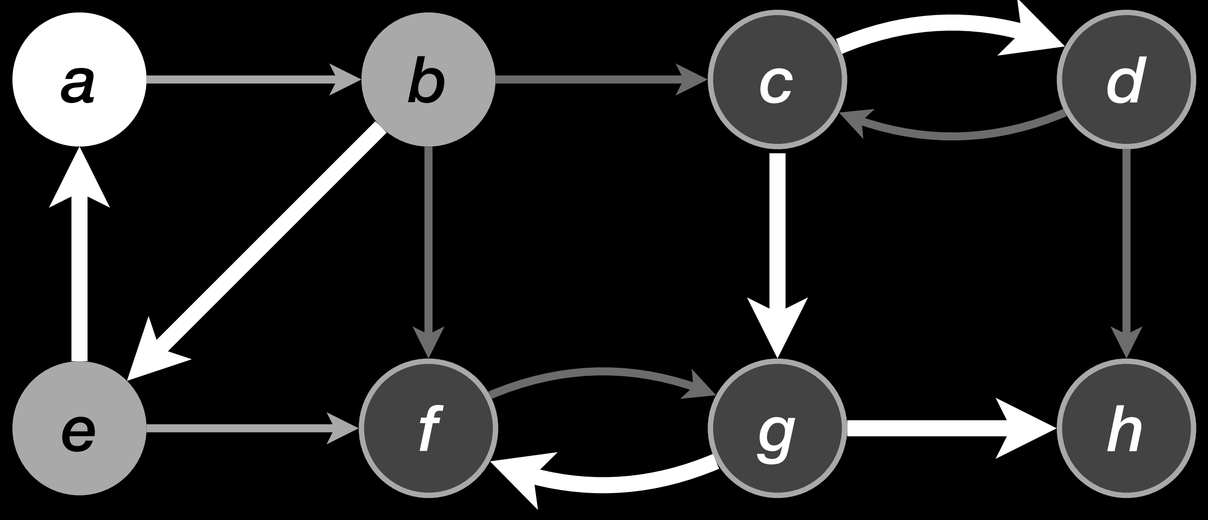
Example

例子

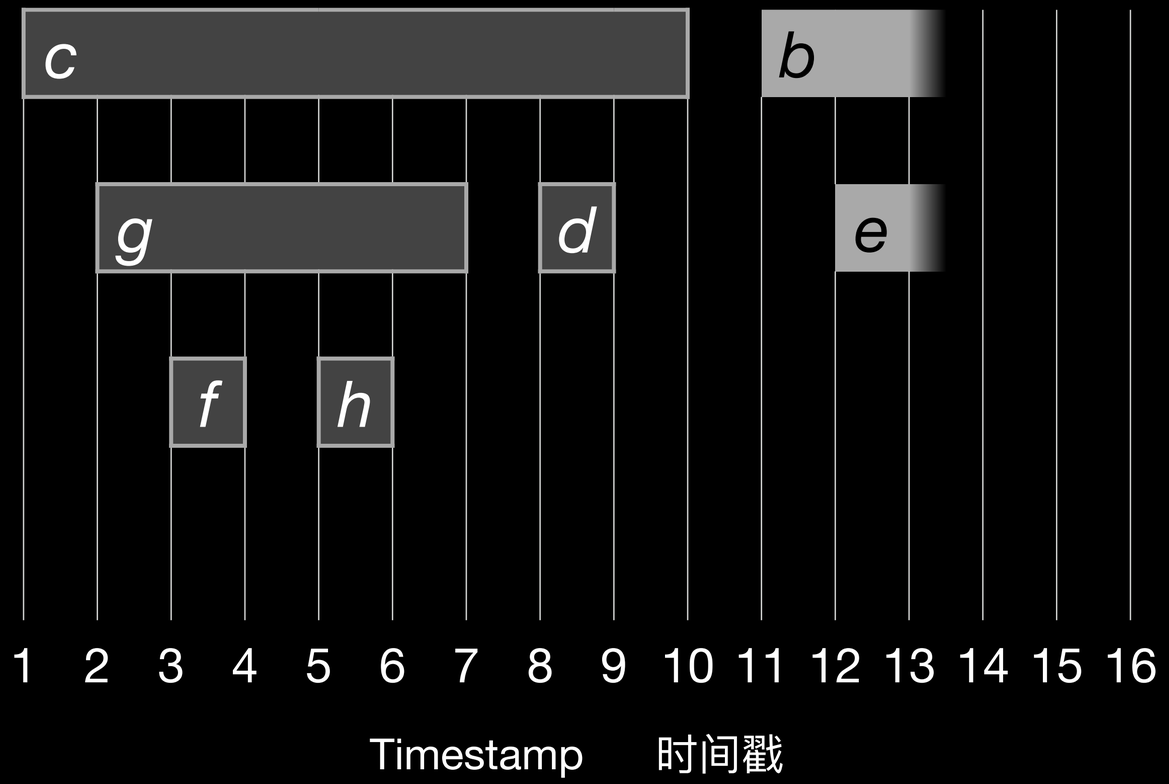




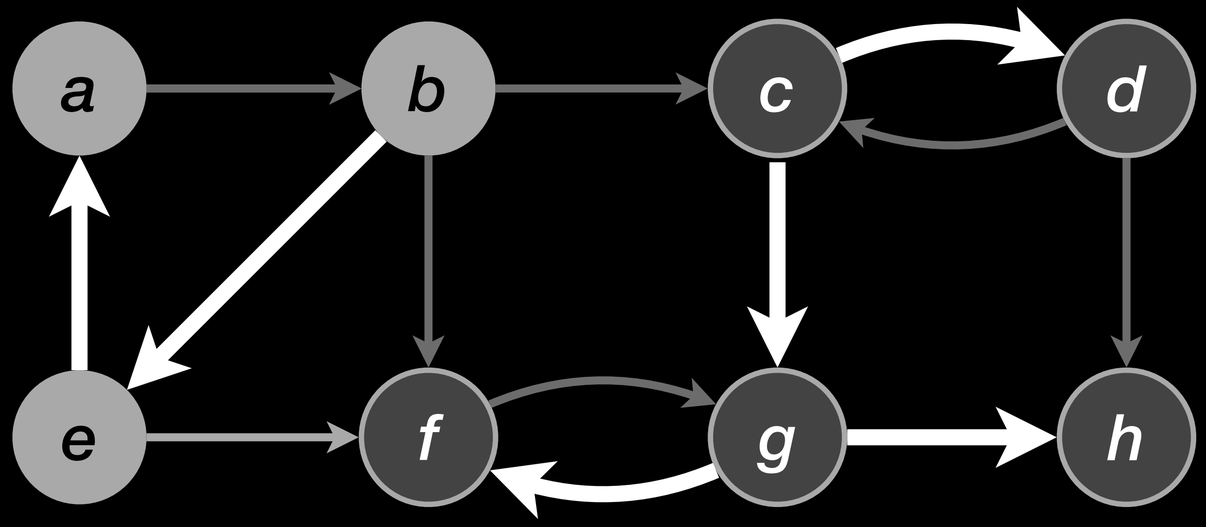
Example


a
b
c
d
e
f
g
h

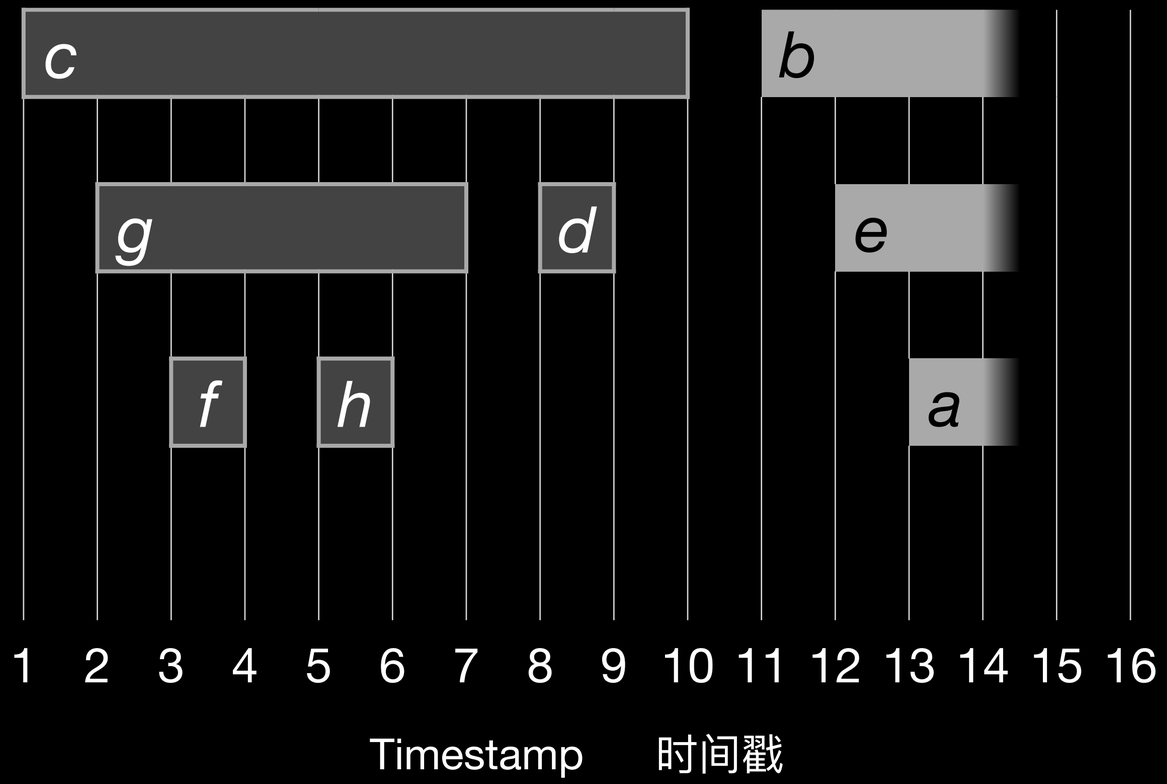
例子


c
b
g
d
e
f
h
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

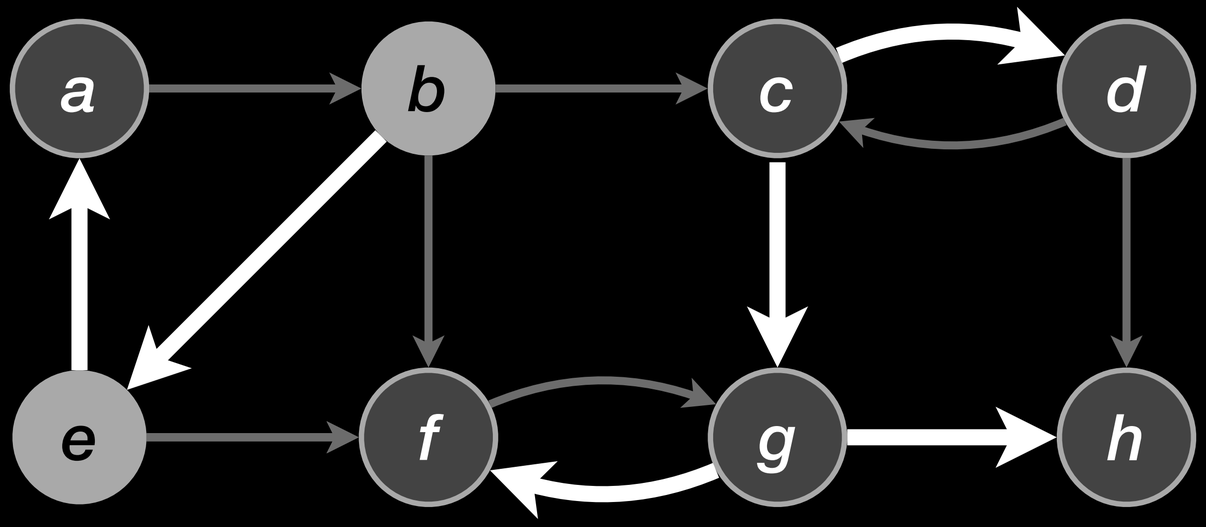
Example


a
b
c
d
e
f
g
h

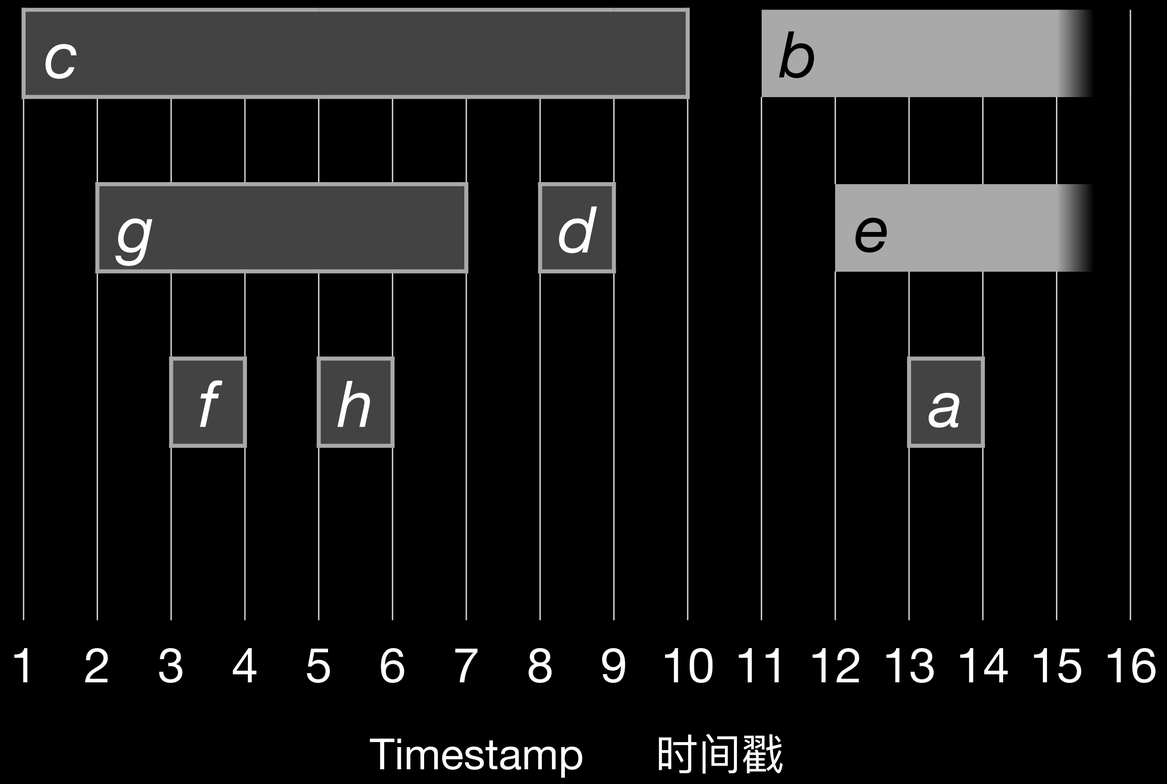
例子


c
b
g
d
e
f
h
a
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

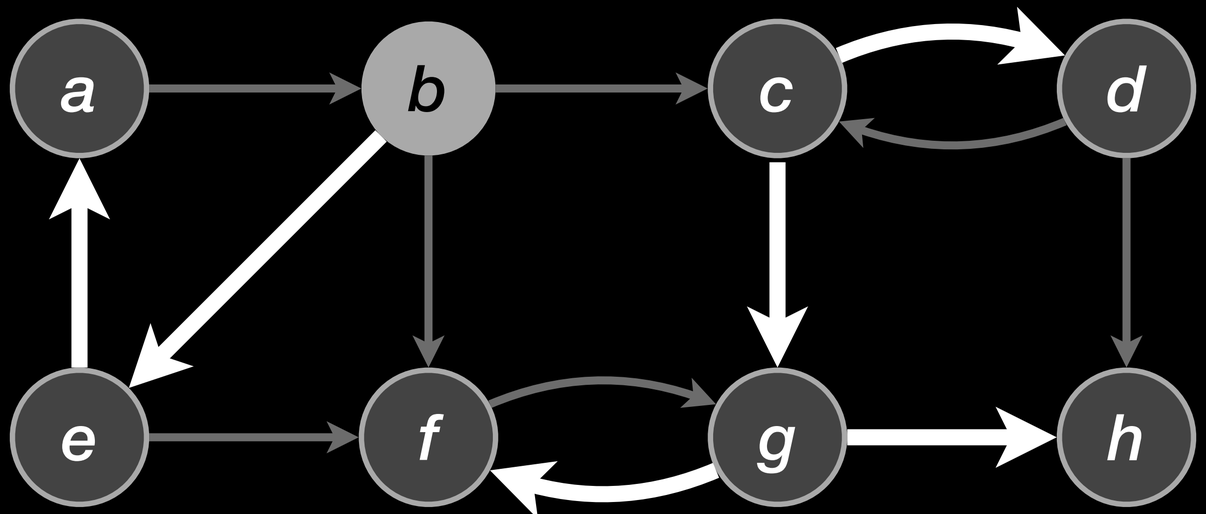
Example


a
b
c
d
e
f
g
h

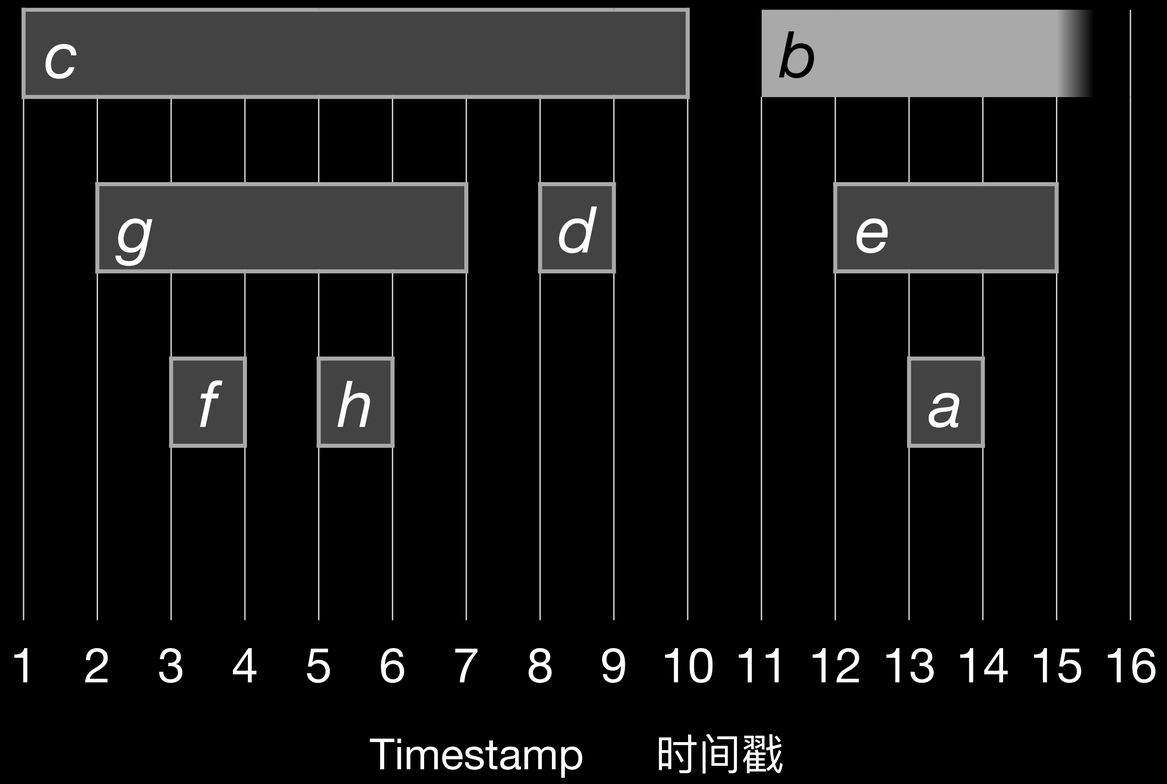
例子


c
b
g
d
e
f
h
a
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

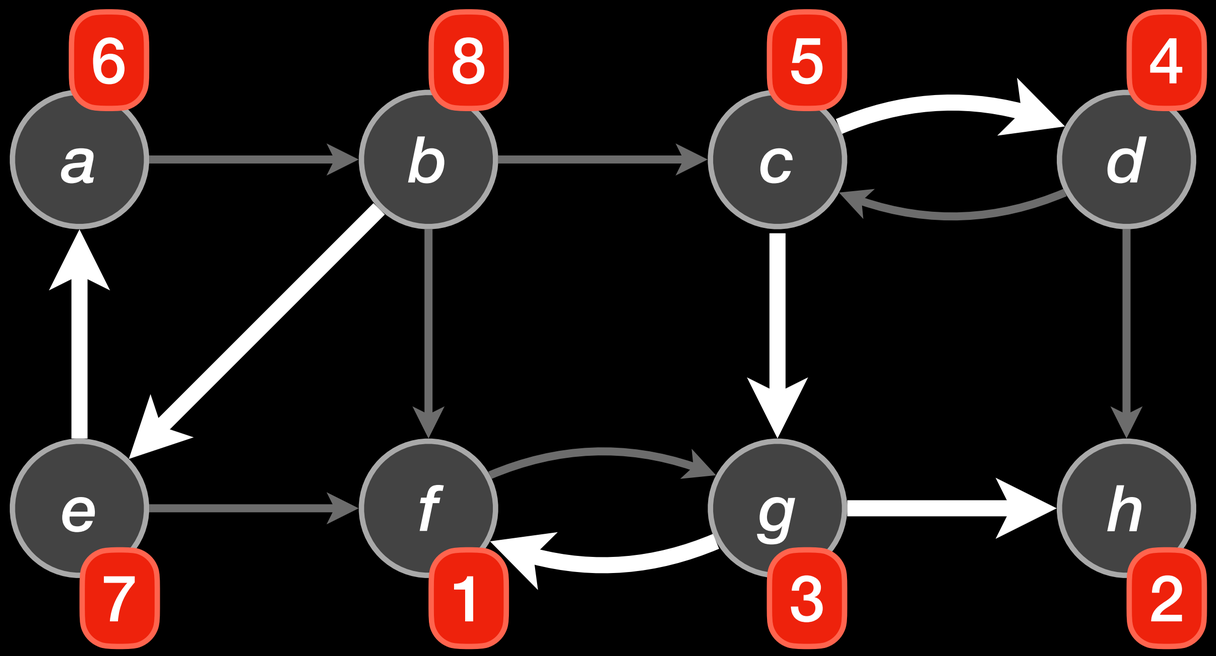
Example


a
b
c
d
e
f
g
h

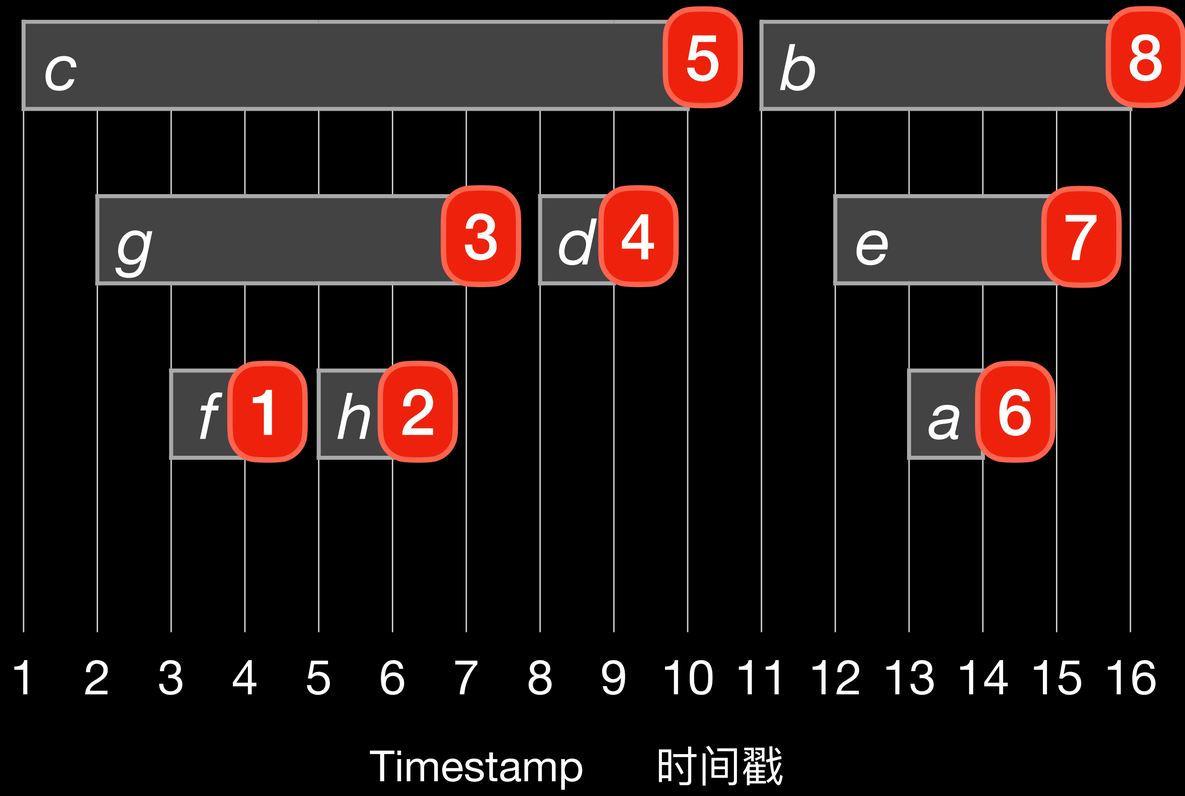
例子


c
b
g
d
e
f
h
a
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

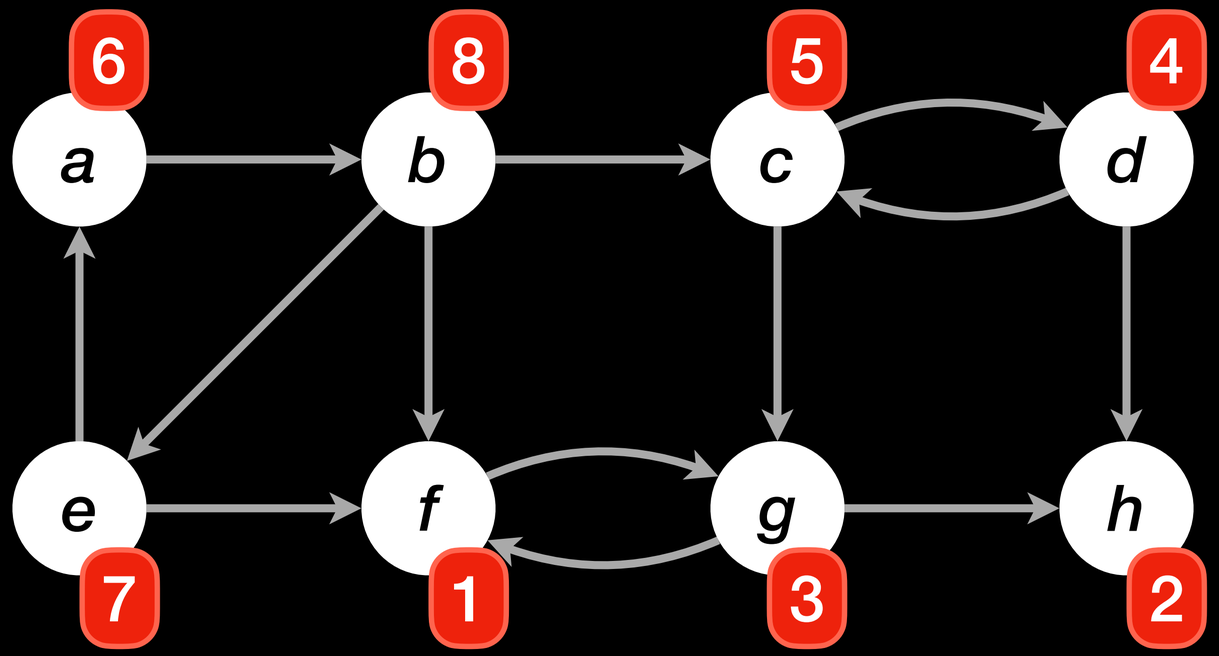

6
8
5
4
a
b
c
d
e
f
g
h
7
1
3
2

例子


c
5
b
8
g
3
d
4
e
7
f
1
h
2
a
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

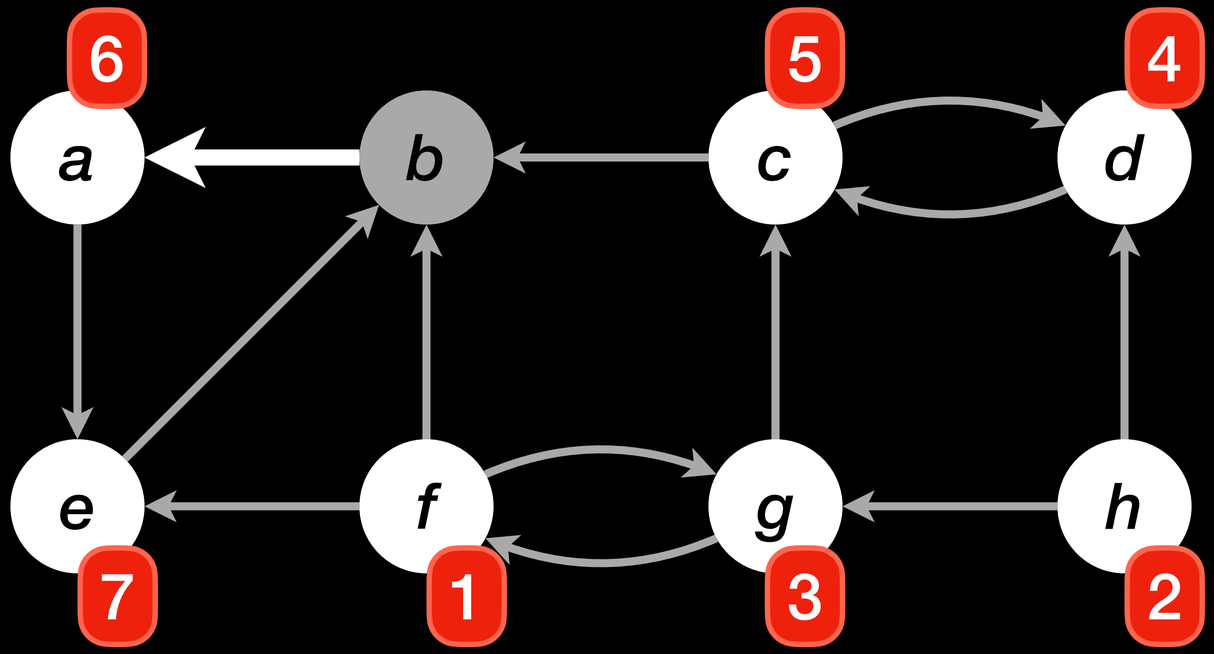
Example

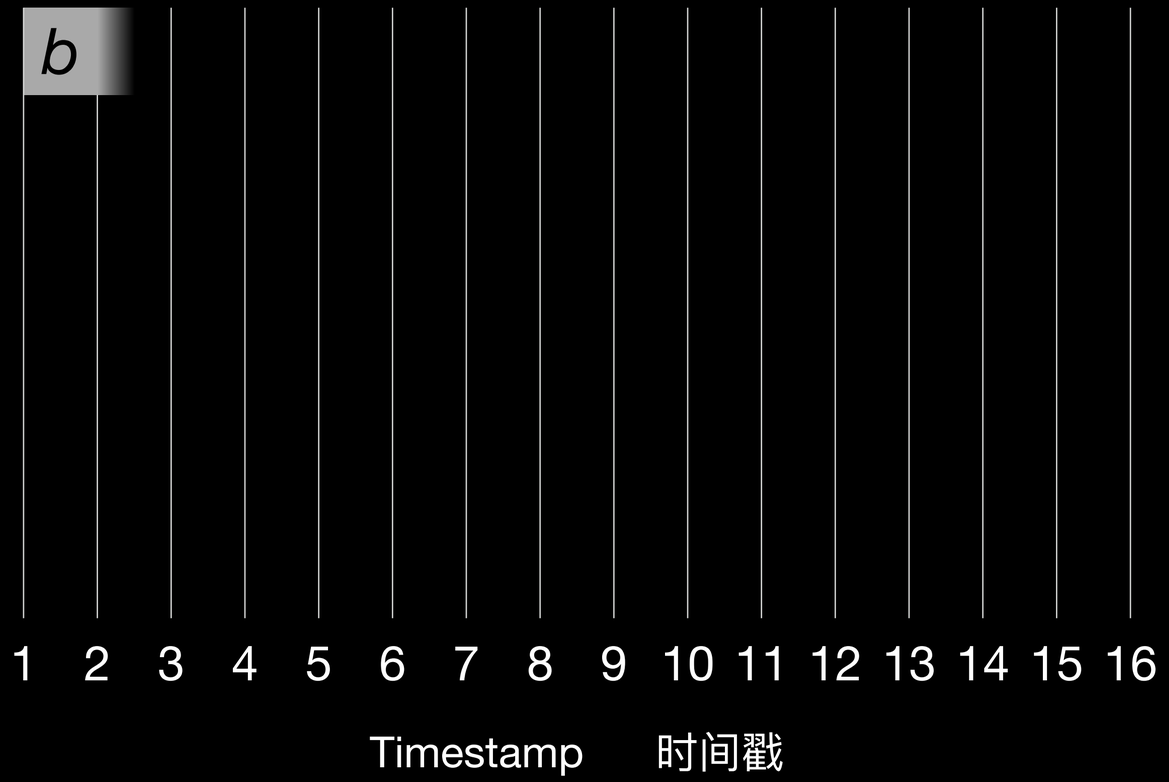
例子



Example

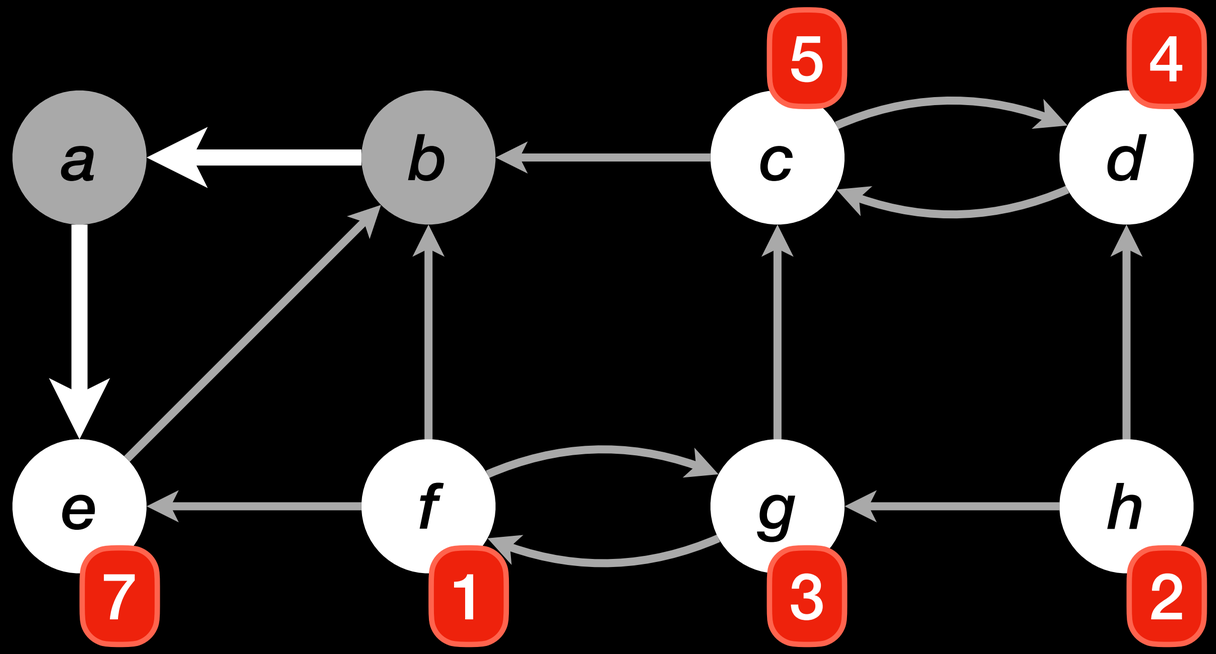
例子

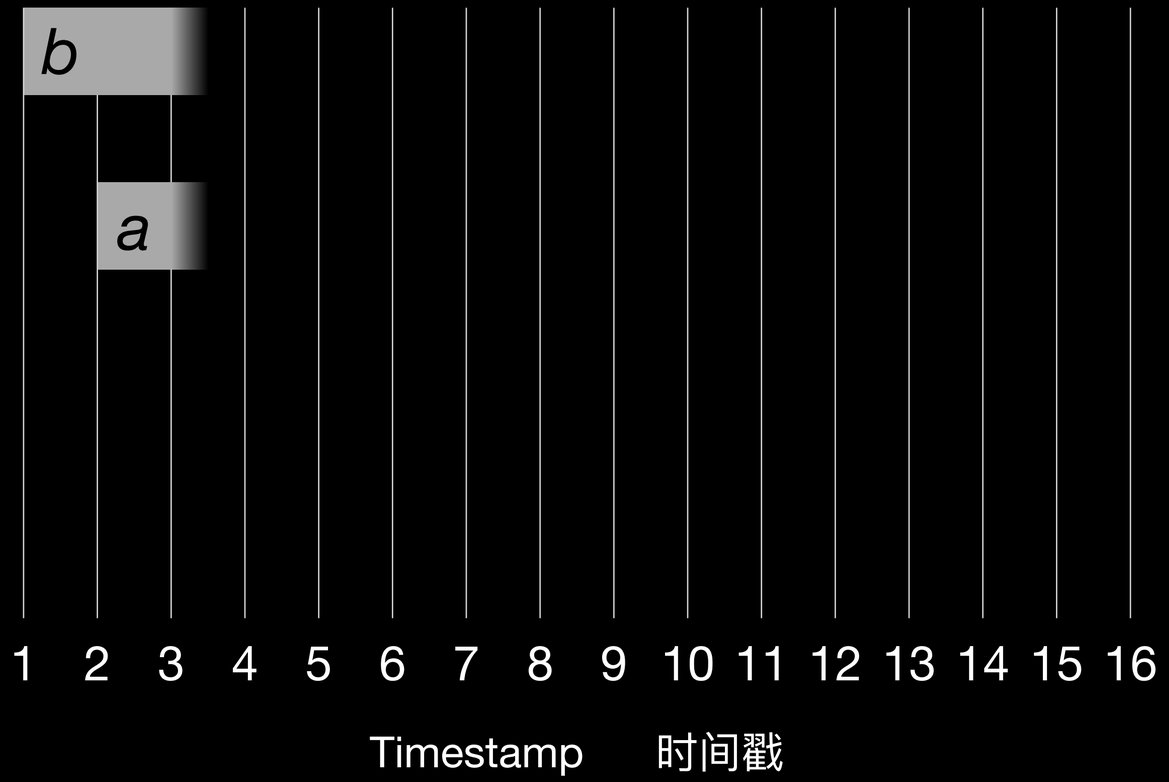

6
5
4
a
b
c
d
e
f
g
h
7
1
3
2


b
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

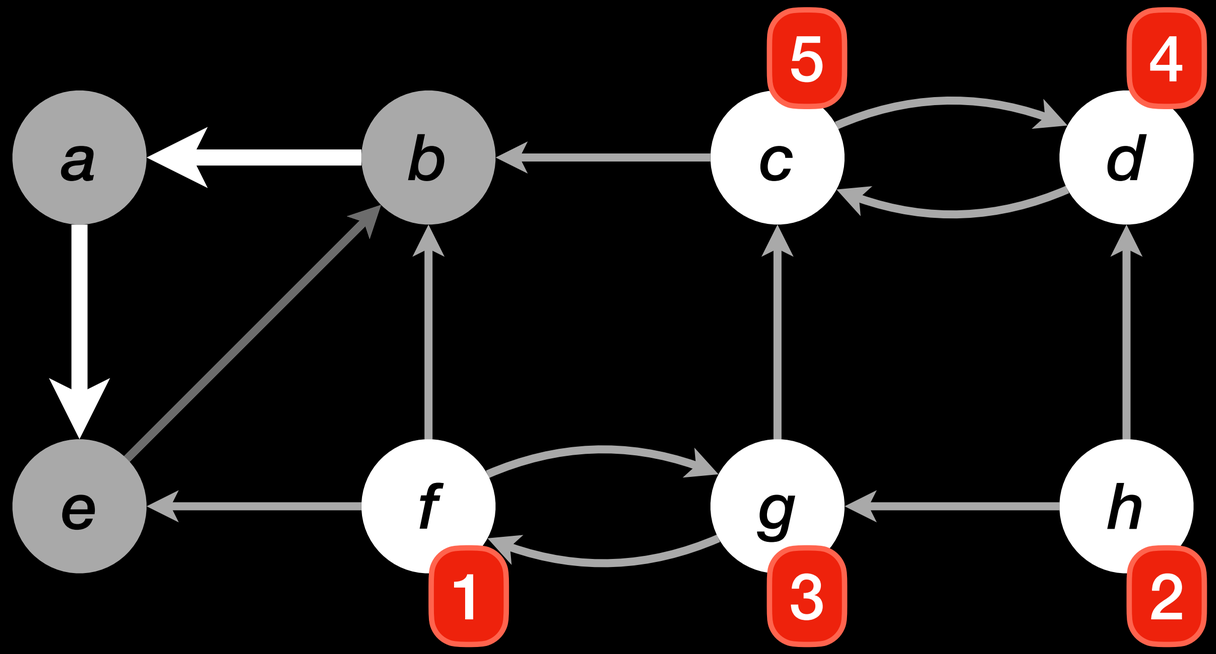
例子

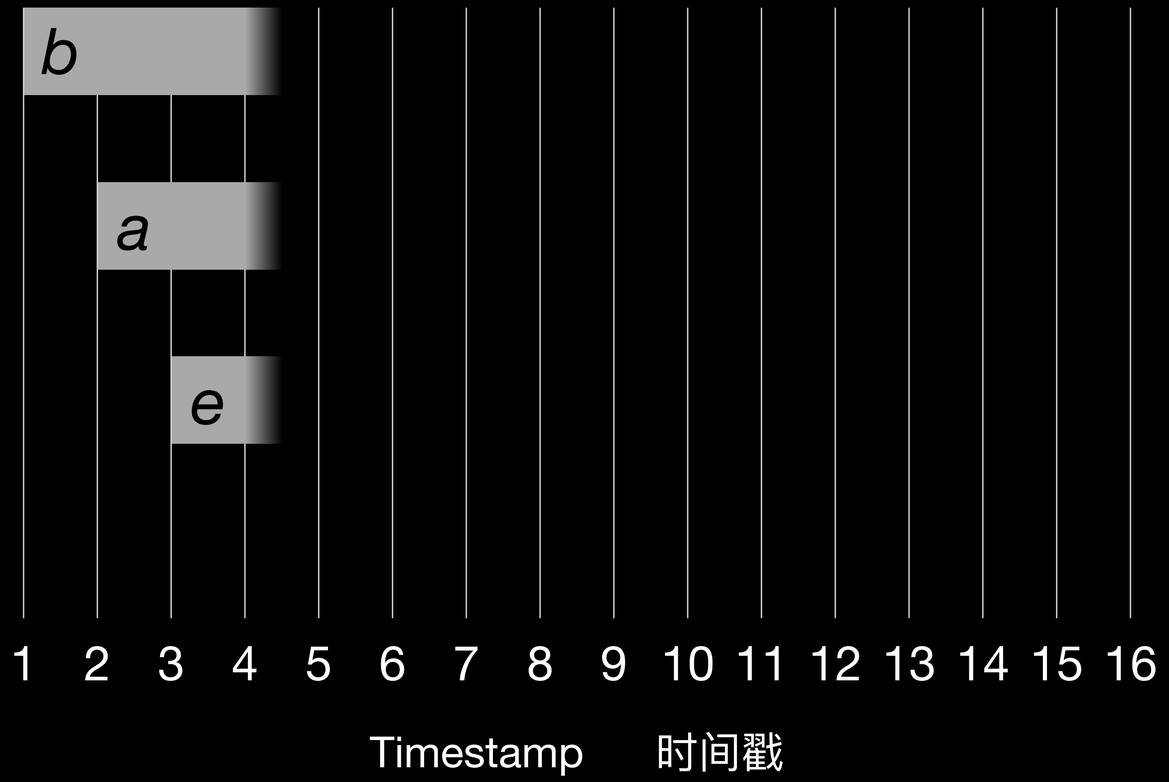

a
b
c
d
e
f
g
h
5
4
7
1
3
2


b
a
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

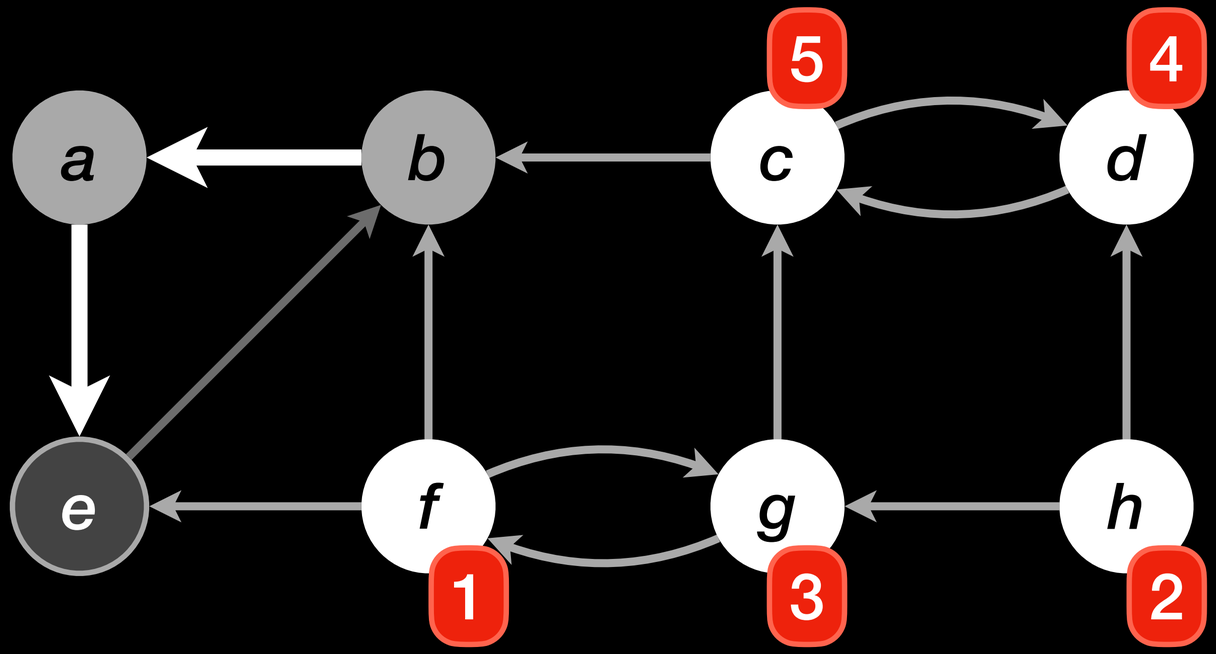
例子

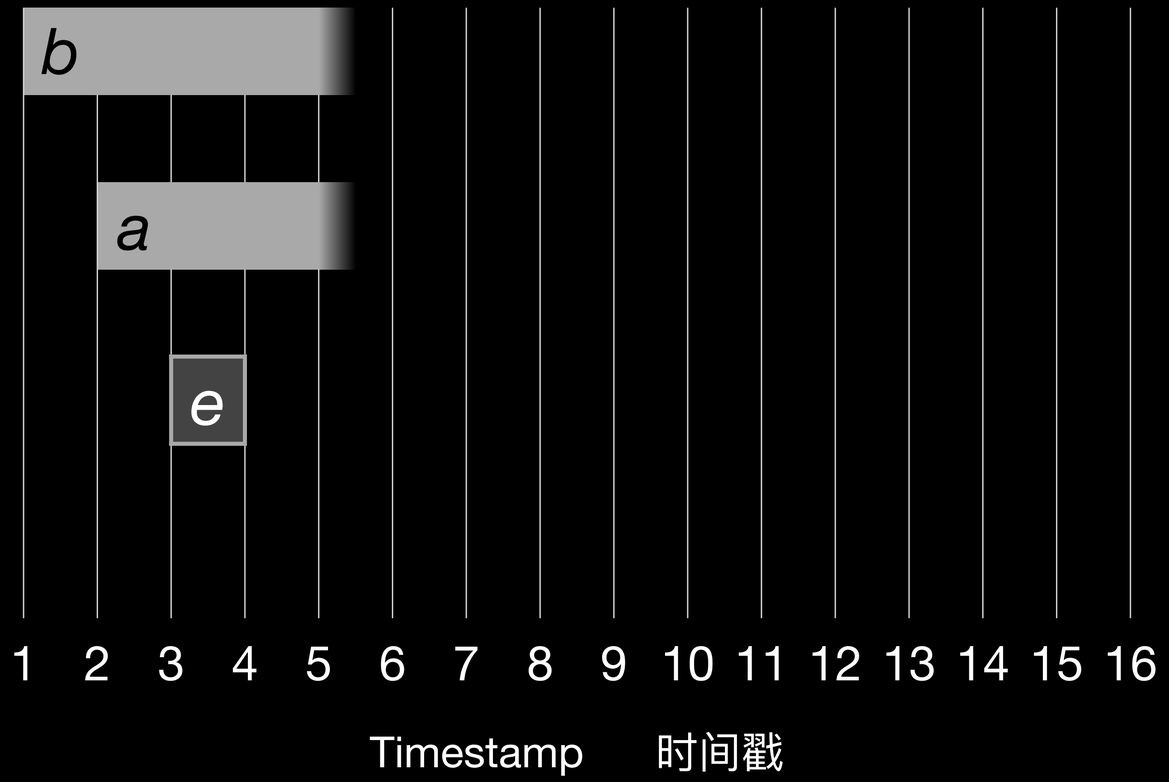

a
b
c
d
e
f
g
h
5
4
1
3
2


b
a
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

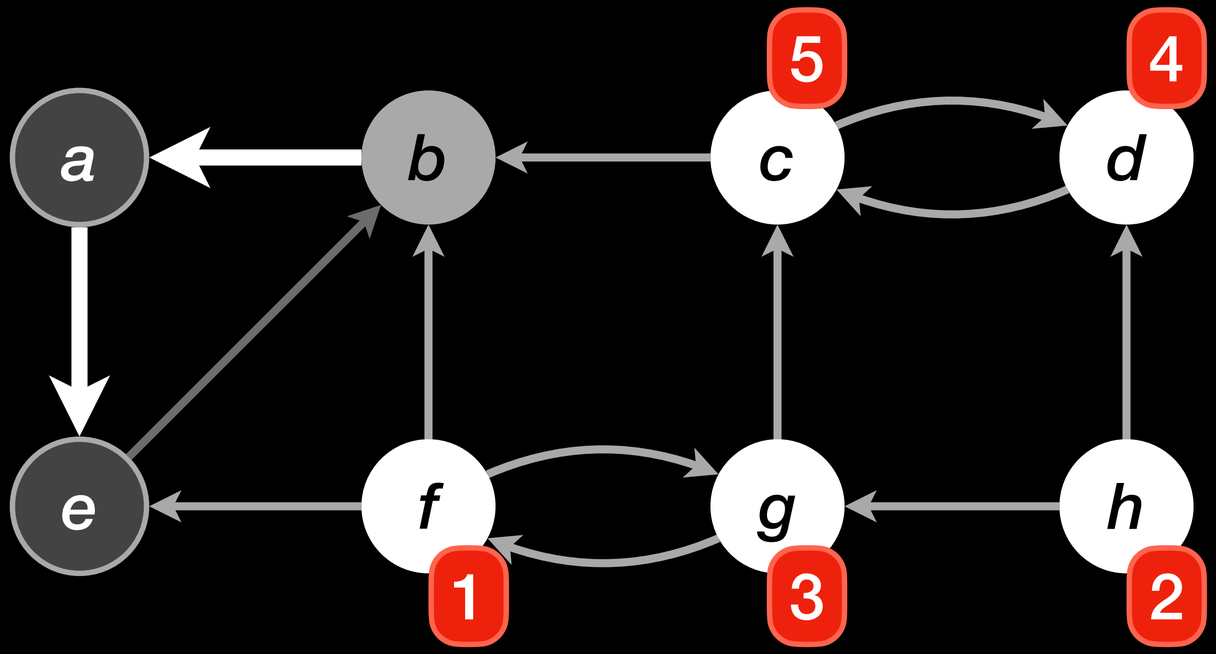
例子

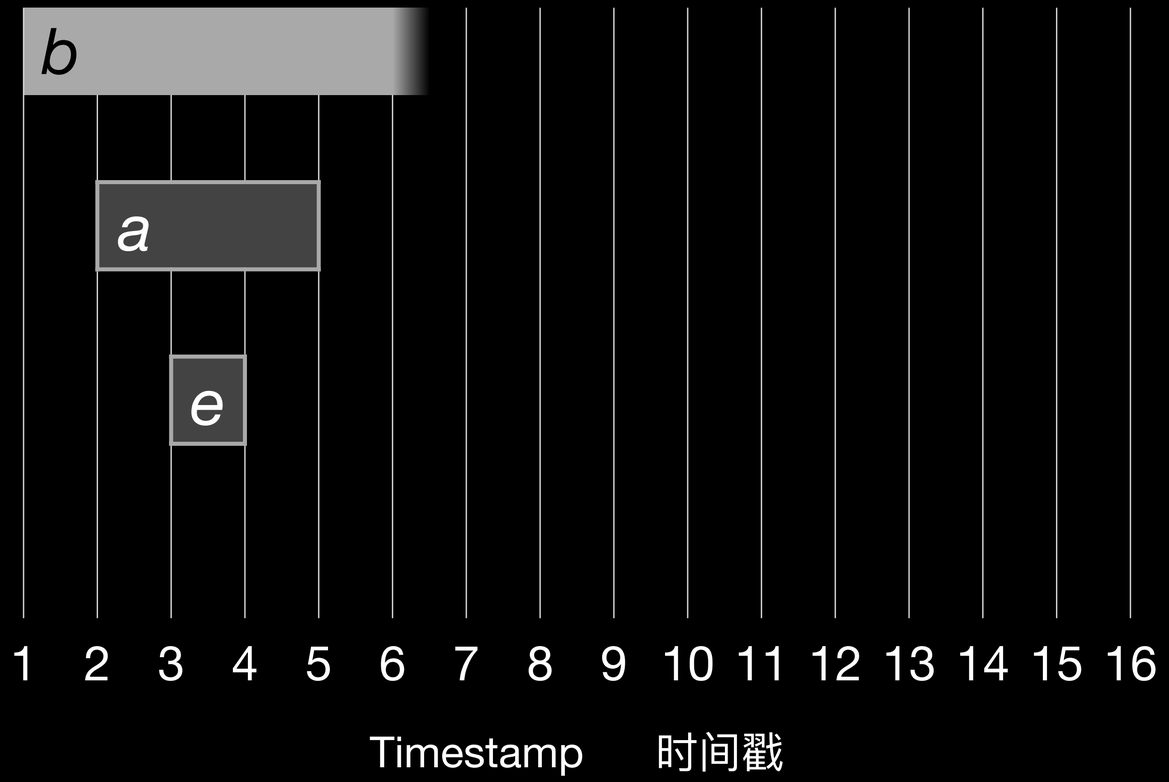

a
b
c
d
e
f
g
h
5
4
1
3
2


b
a
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

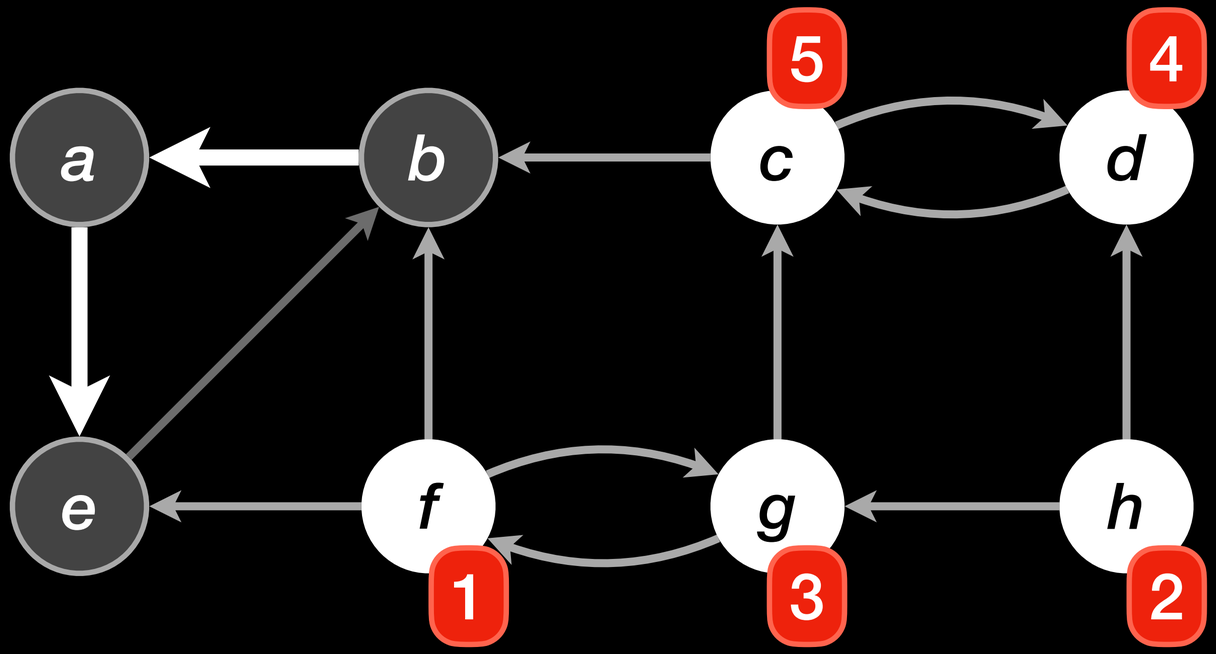
例子

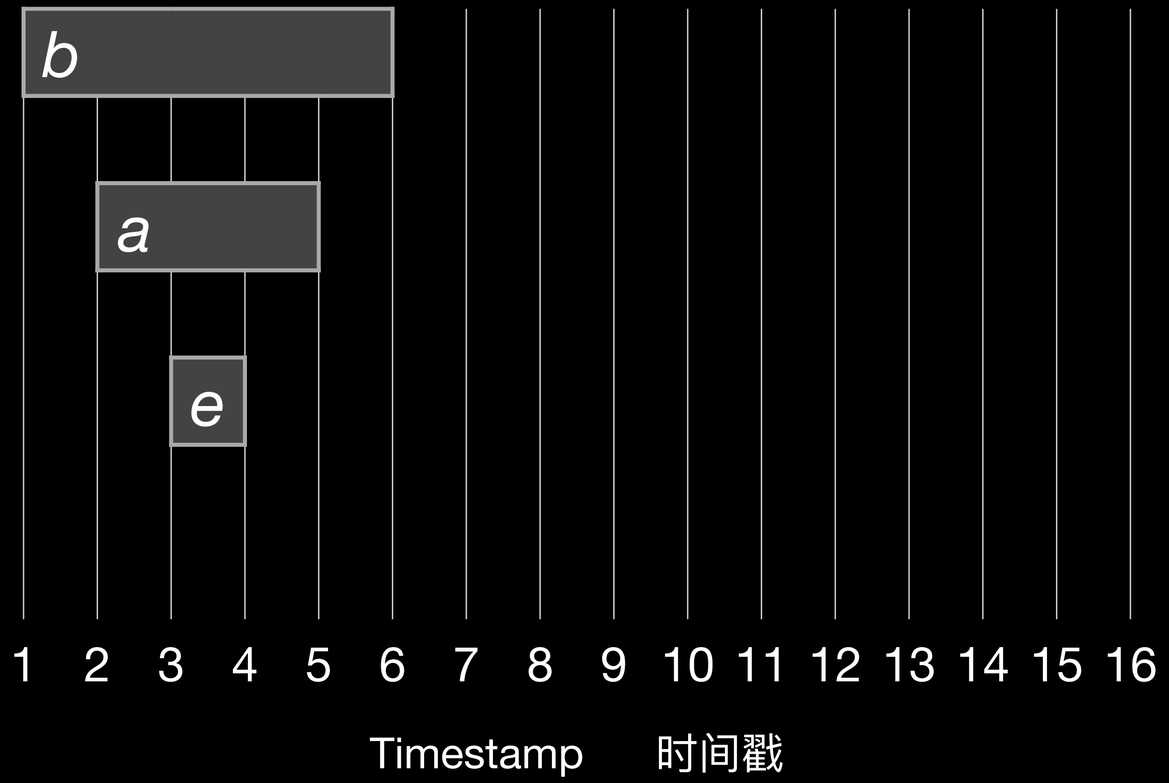

a
b
c
d
e
f
g
h
5
4
1
3
2


b
a
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

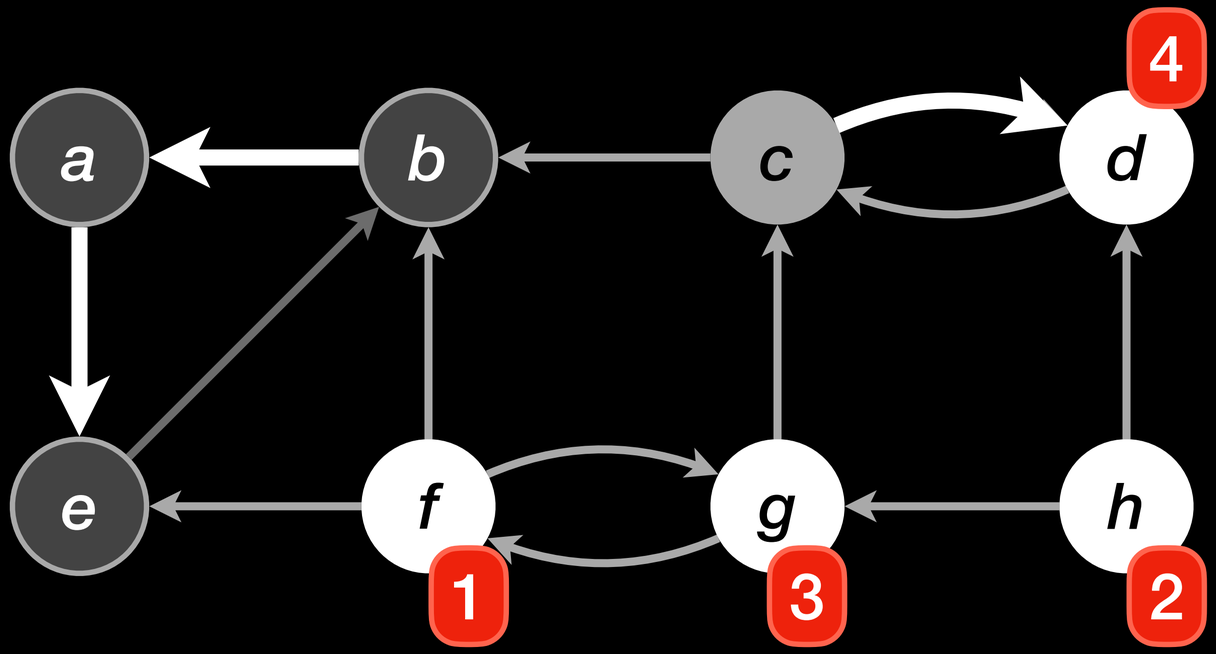
例子

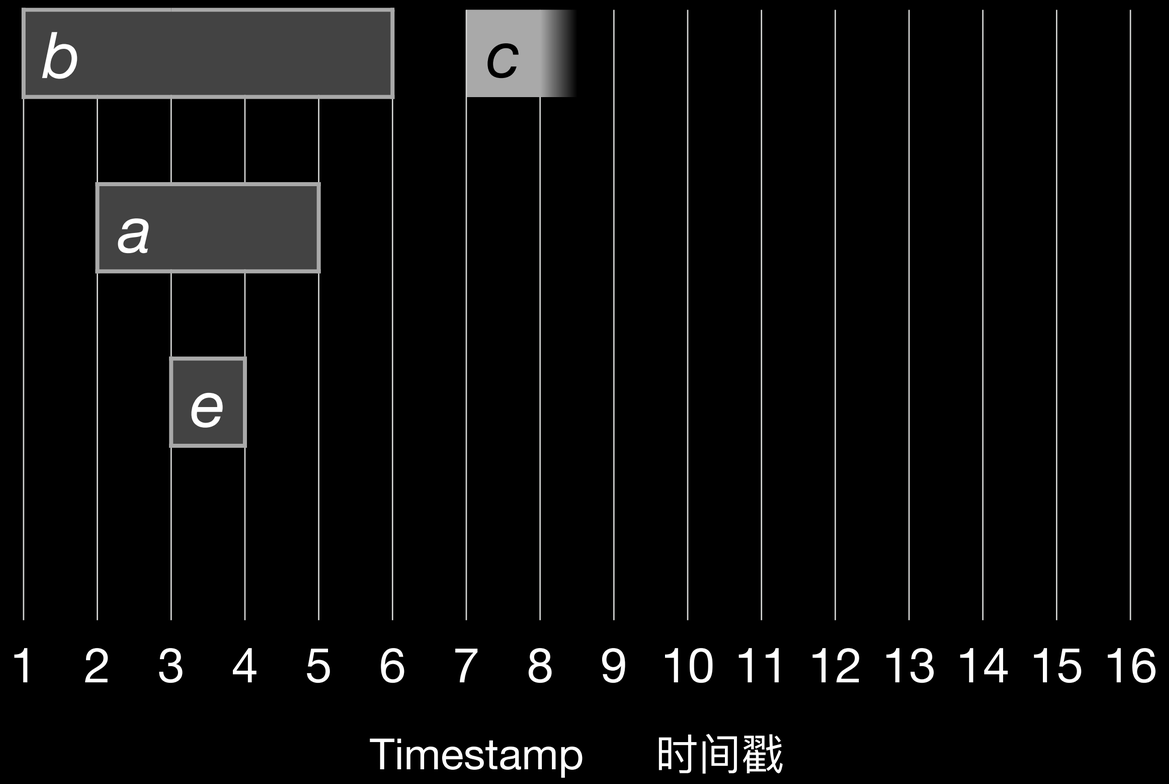

a
b
c
d
e
f
g
h
5
4
1
3
2


b
a
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

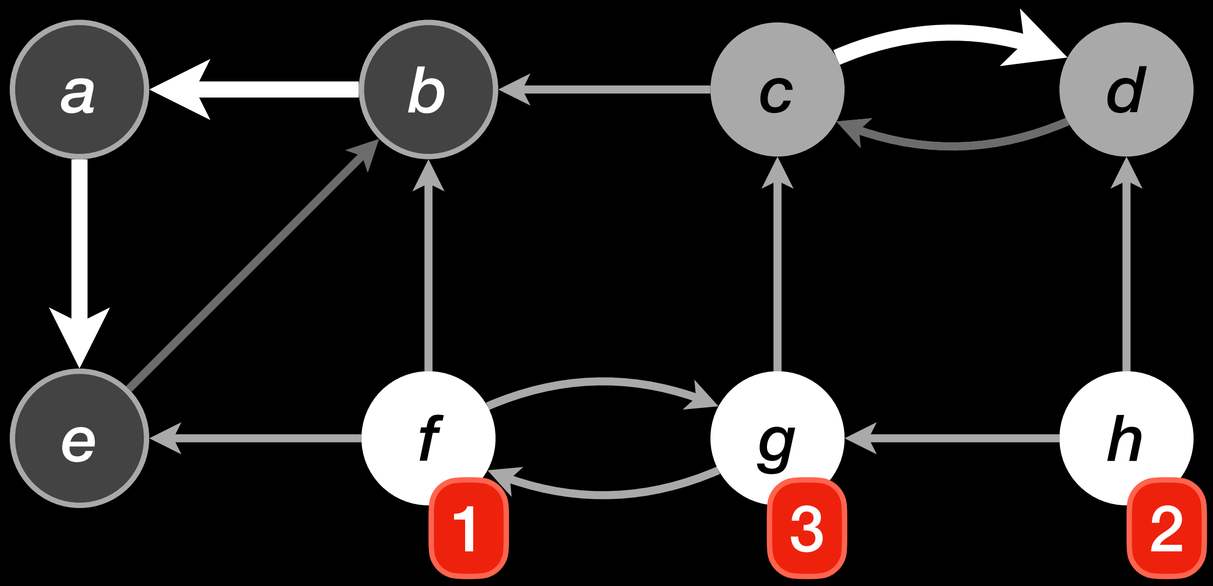
例子

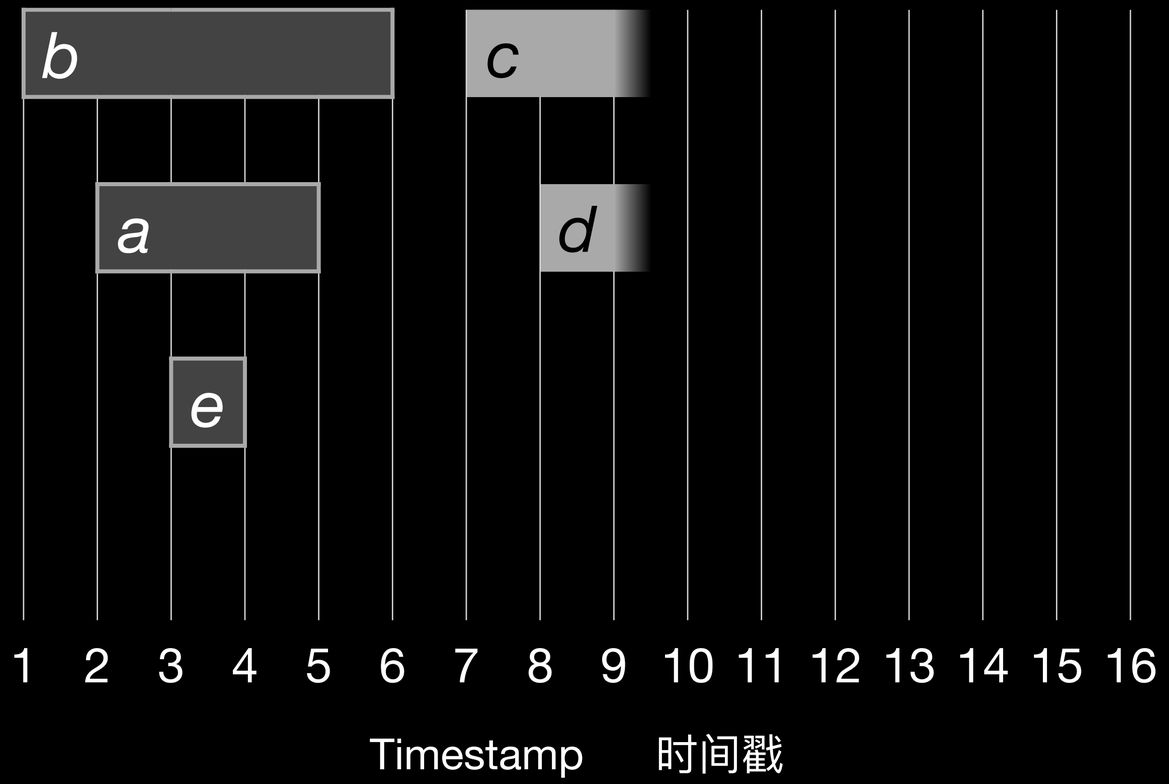

a
b
c
d
e
f
g
h
4
1
3
2


b
c
a
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

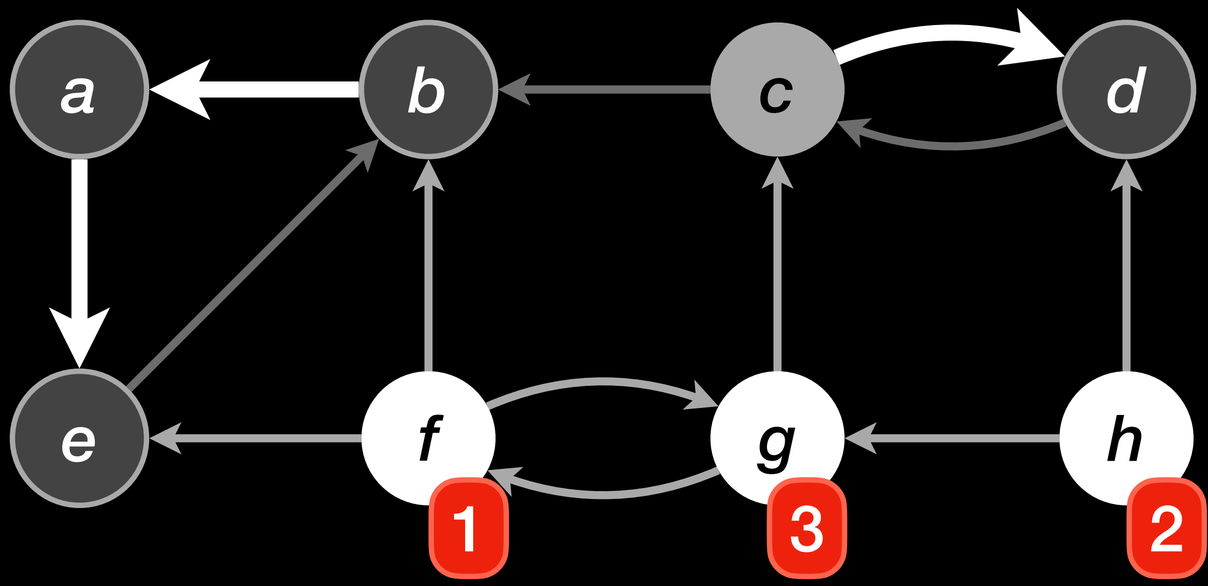
例子

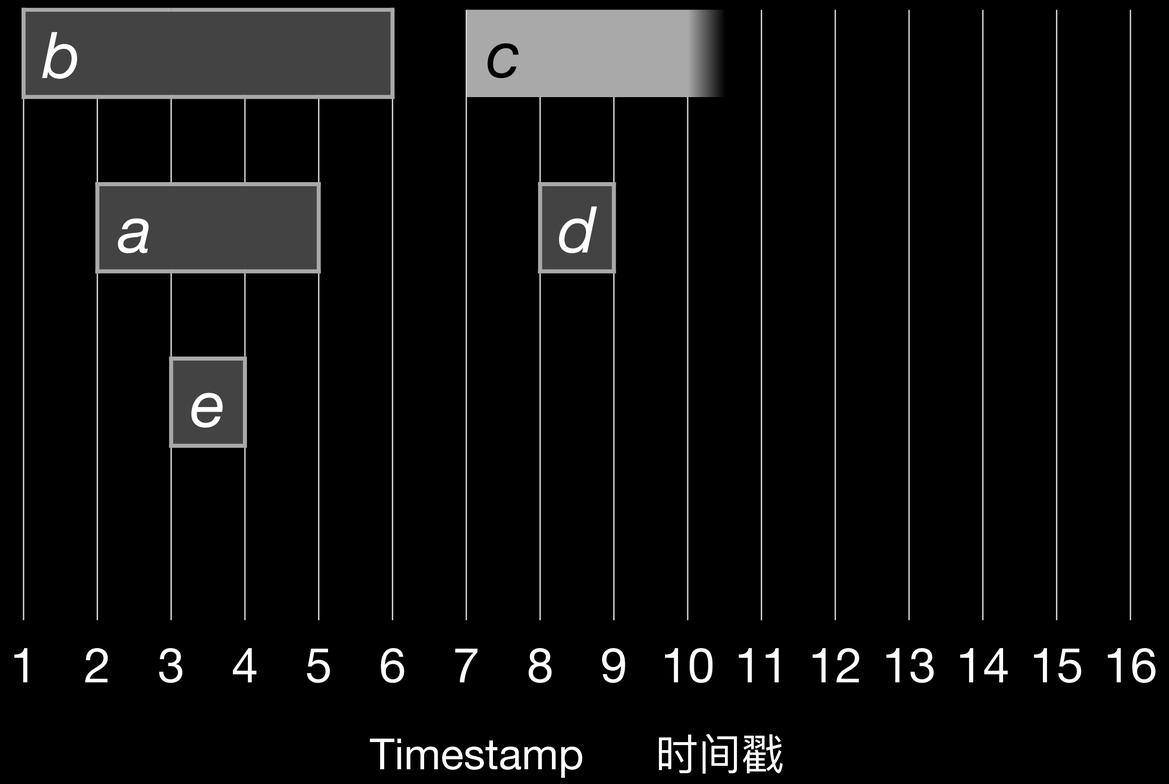

a
b
c
d
e
f
g
h
1
3
2


b
c
a
d
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

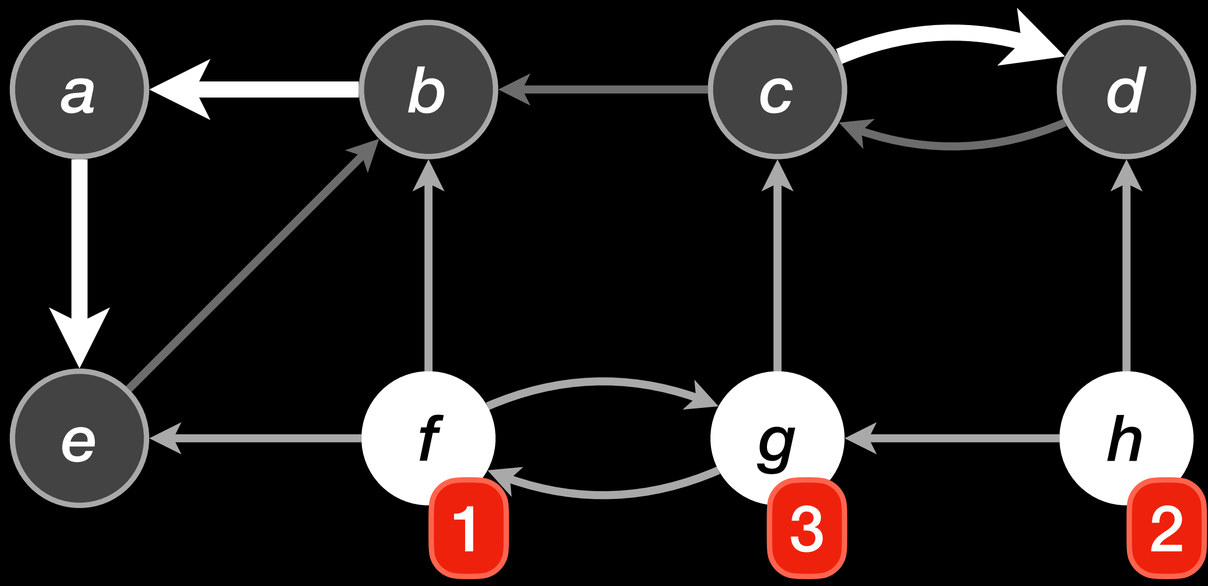
例子

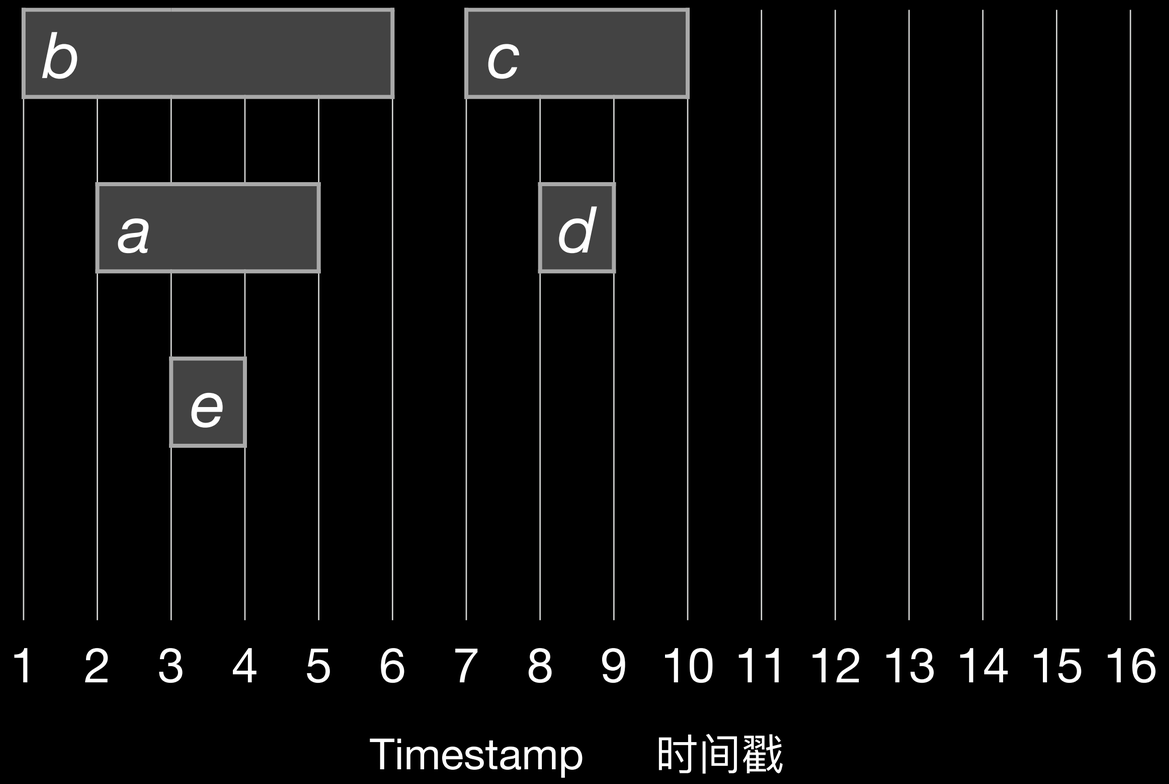

a
b
c
d
e
f
g
h
1
3
2


b
c
a
d
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

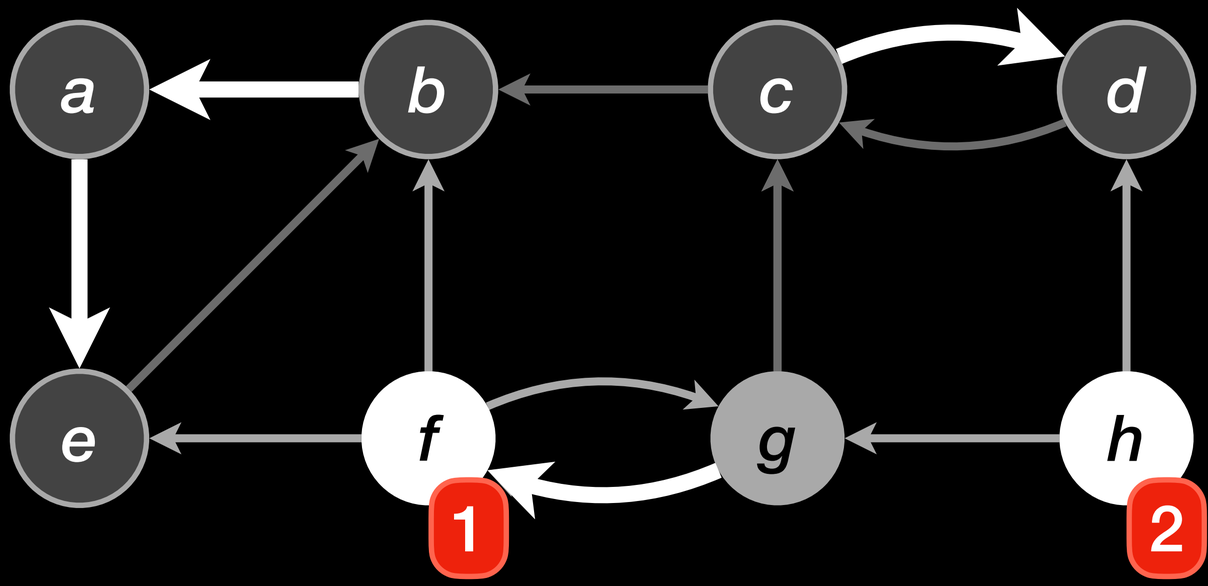
Example

例子

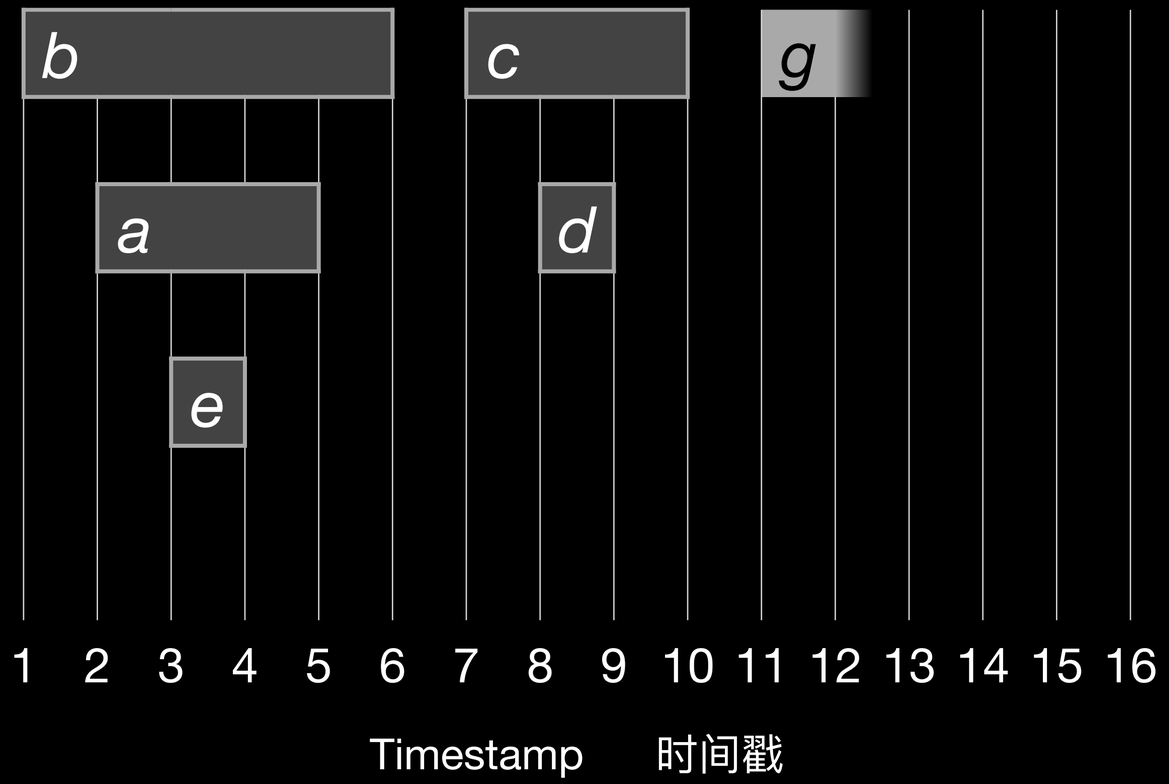

a
b
c
d
e
f
g
h
1
3
2


b
c
a
d
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

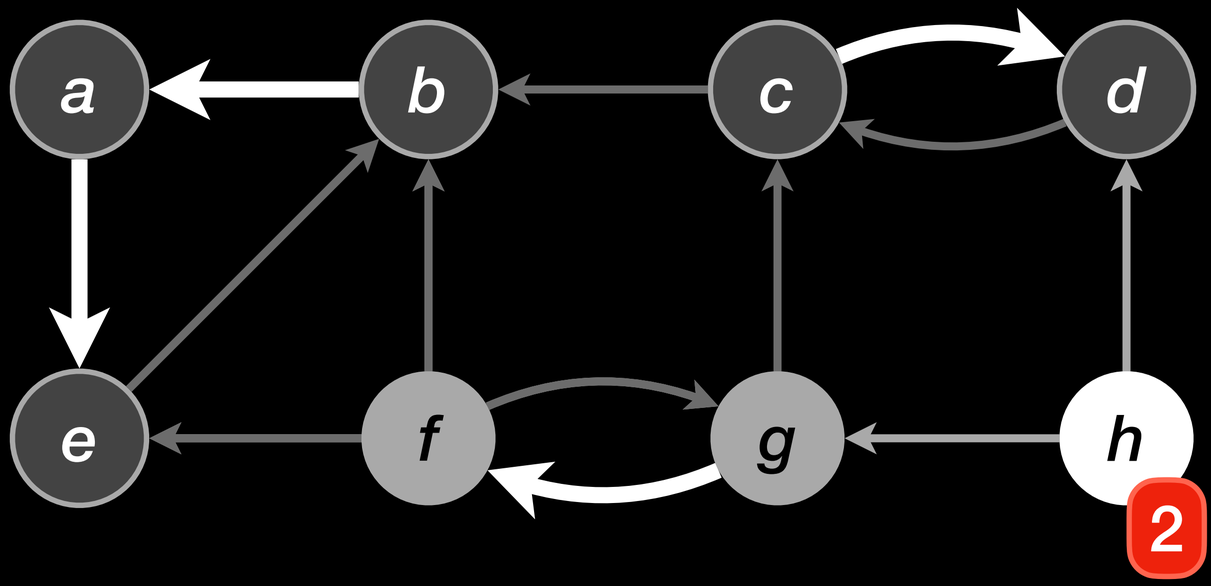

a
b
c
d
e
f
g
h
1
2

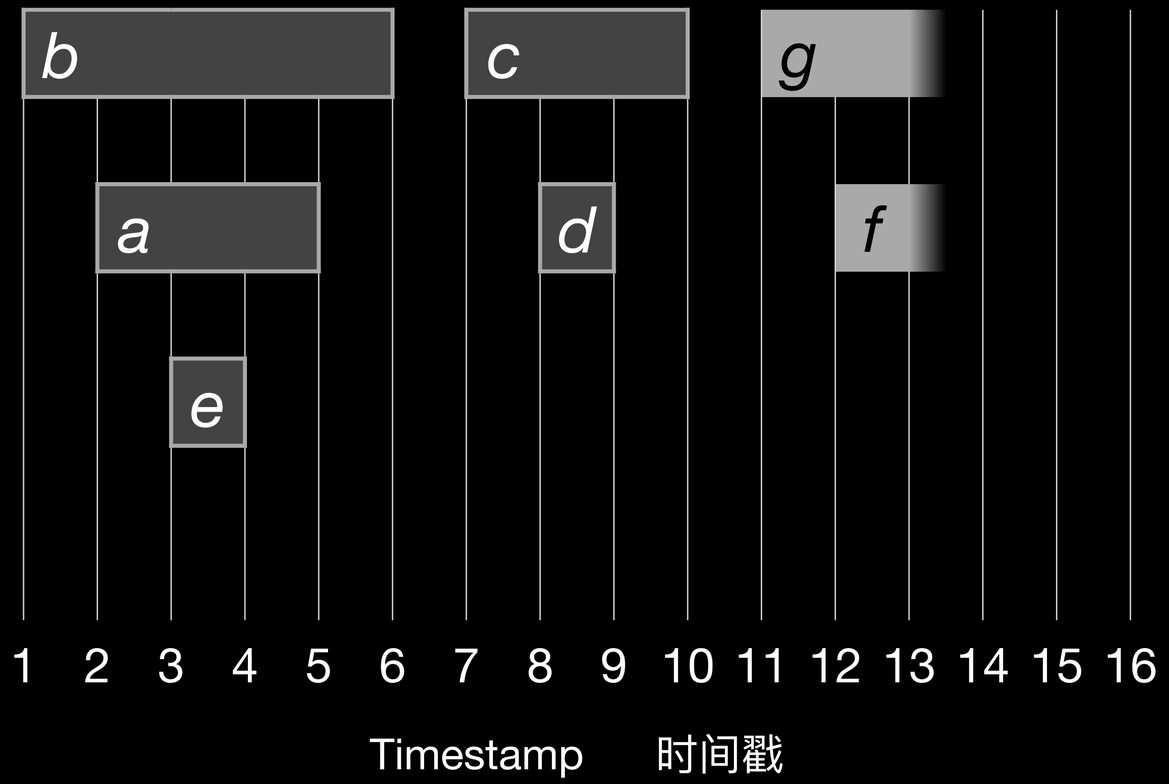
例子


b
c
g
a
d
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

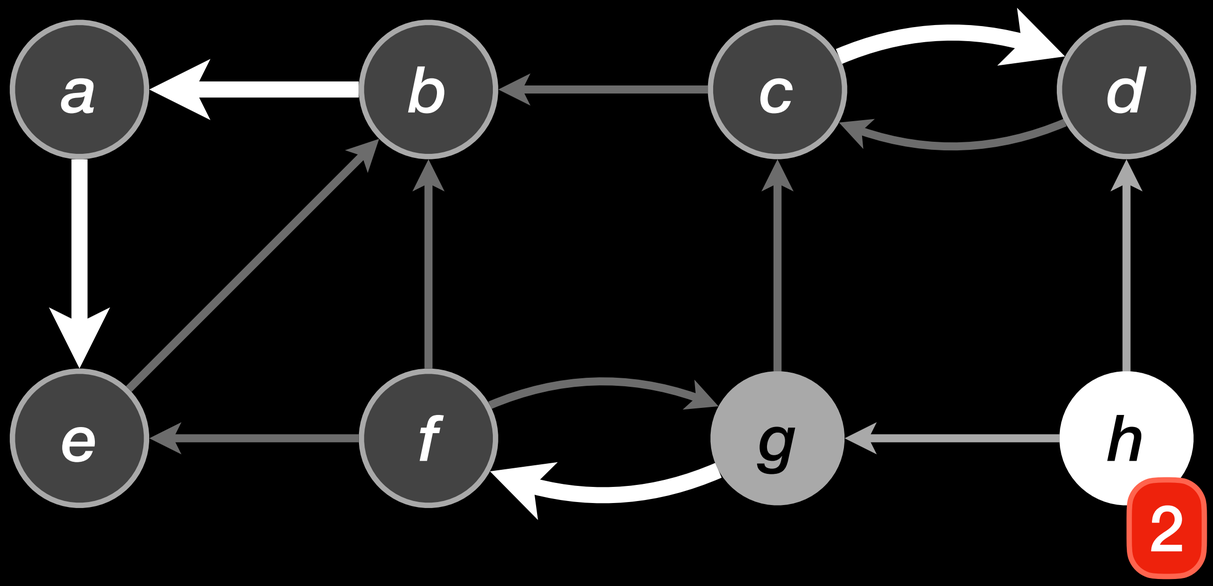
Example

例子

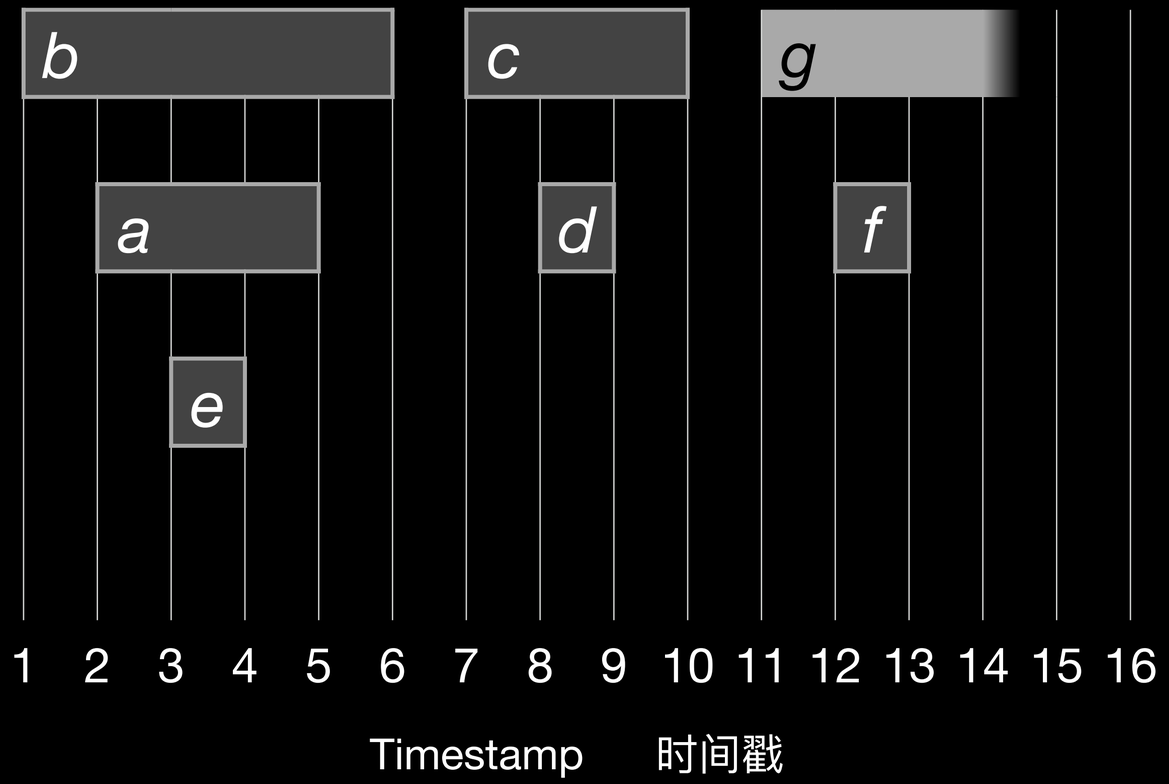

a
b
c
d
e
f
g
h
2


b
c
g
a
d
f
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

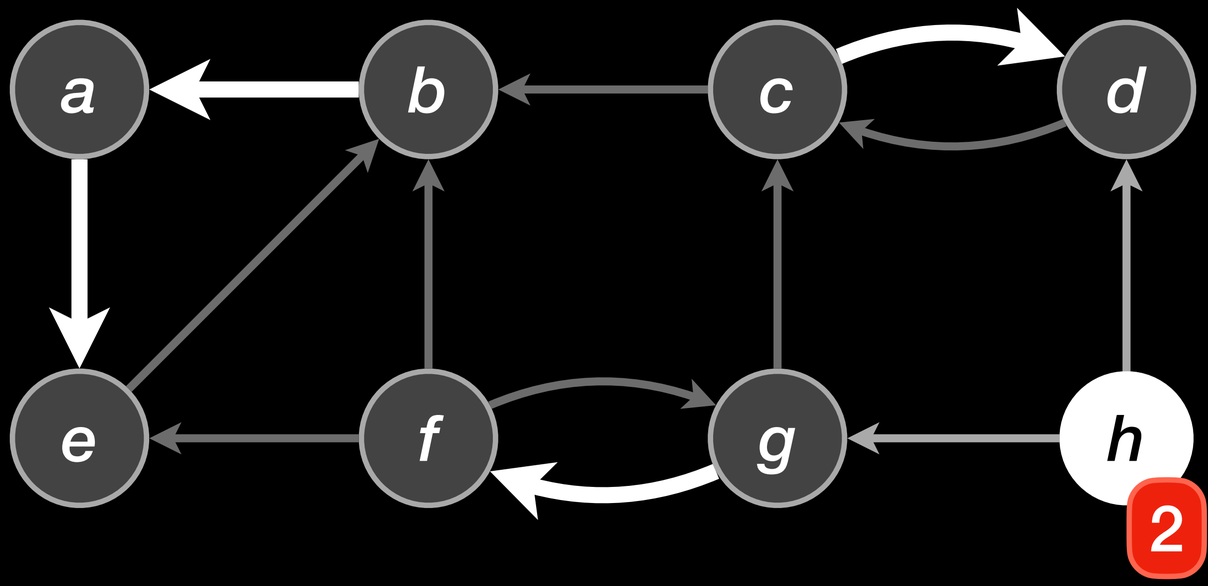

a
b
c
d
e
f
g
h
2

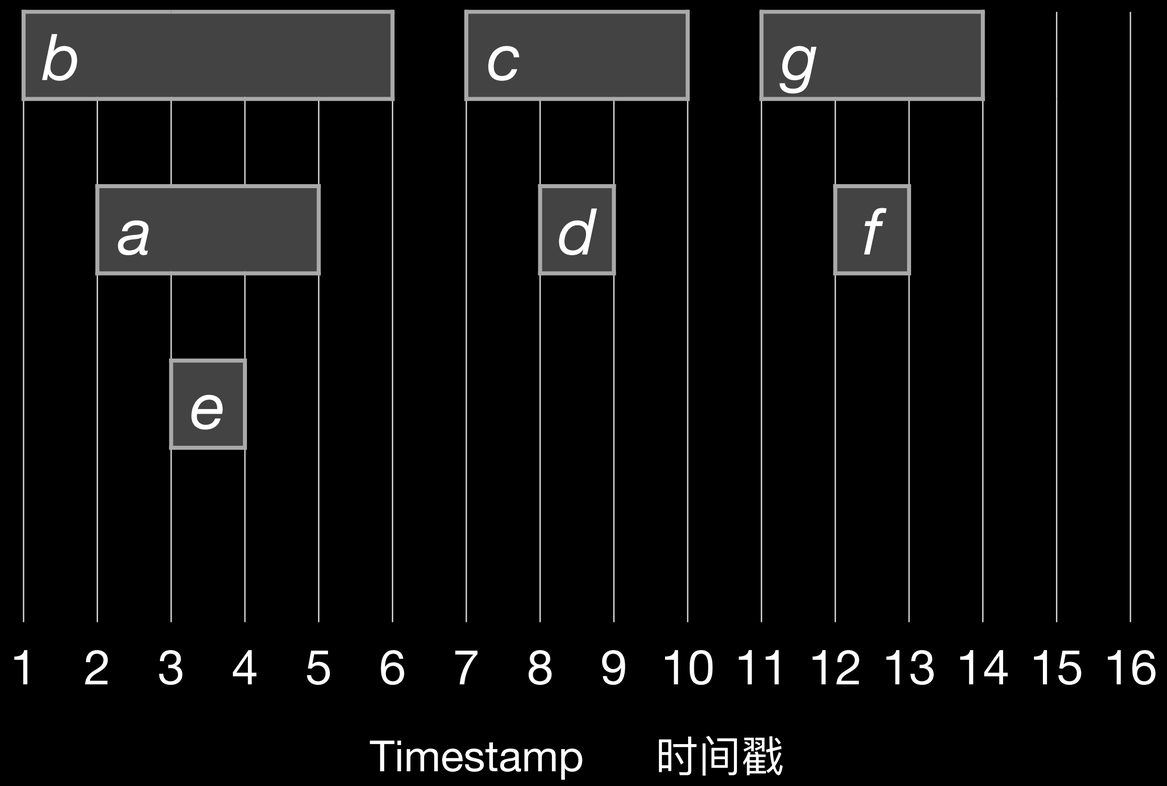
例子


b
c
g
a
d
f
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

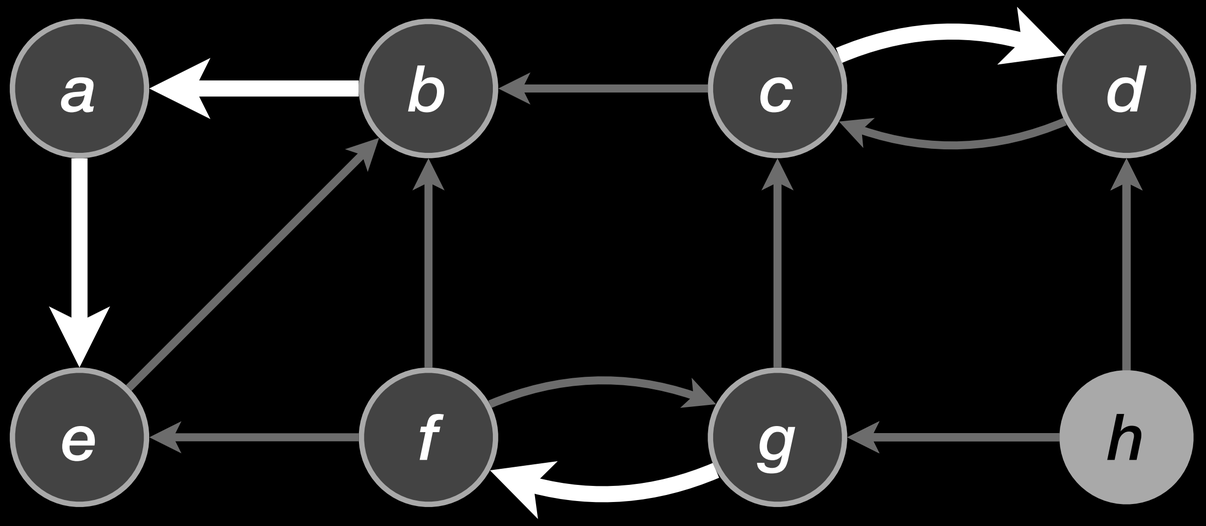
Example

例子

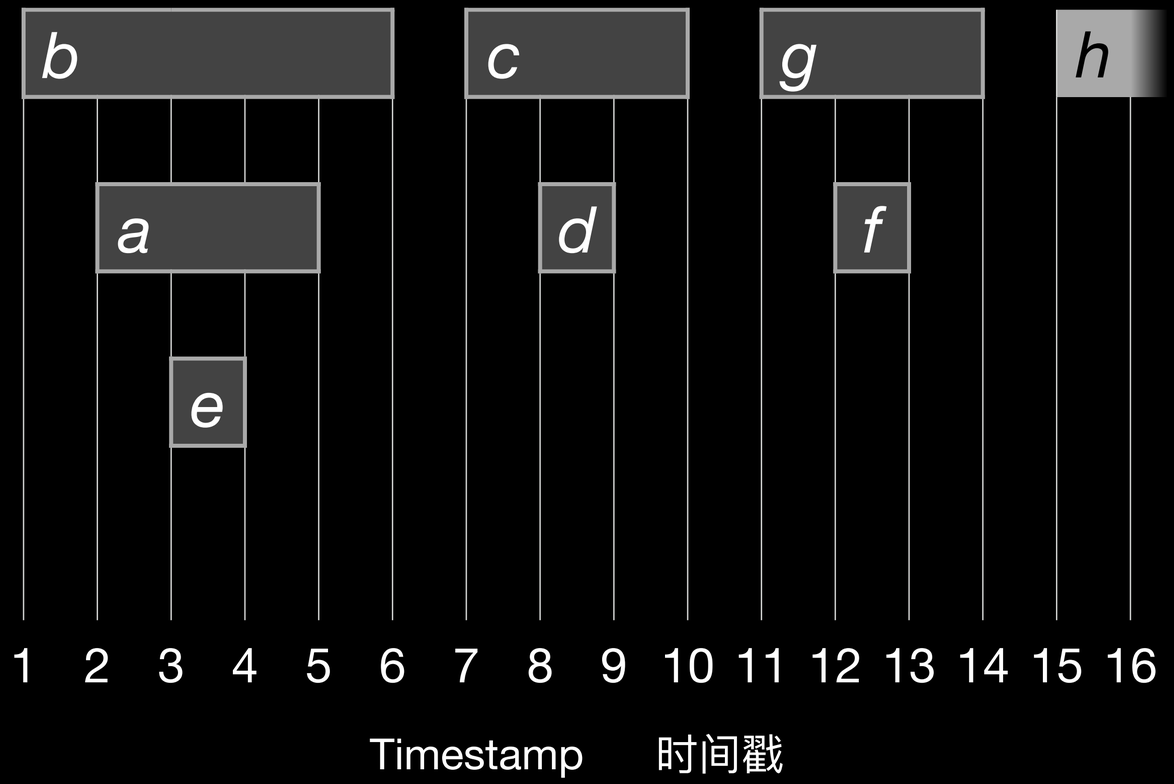

a
b
c
d
e
f
g
h
2


b
c
g
a
d
f
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

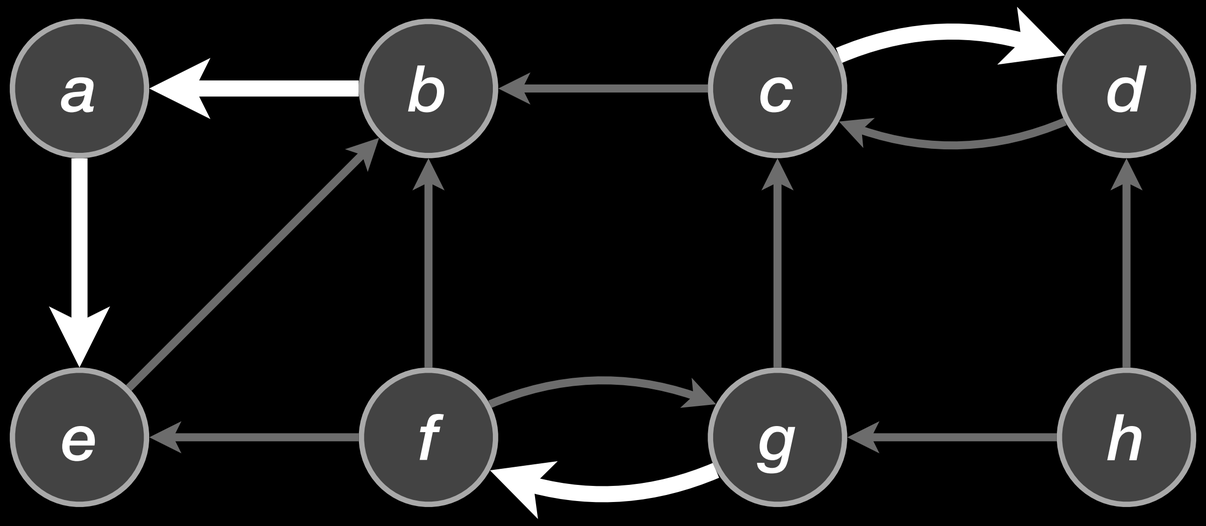
Example


a
b
c
d
e
f
g
h

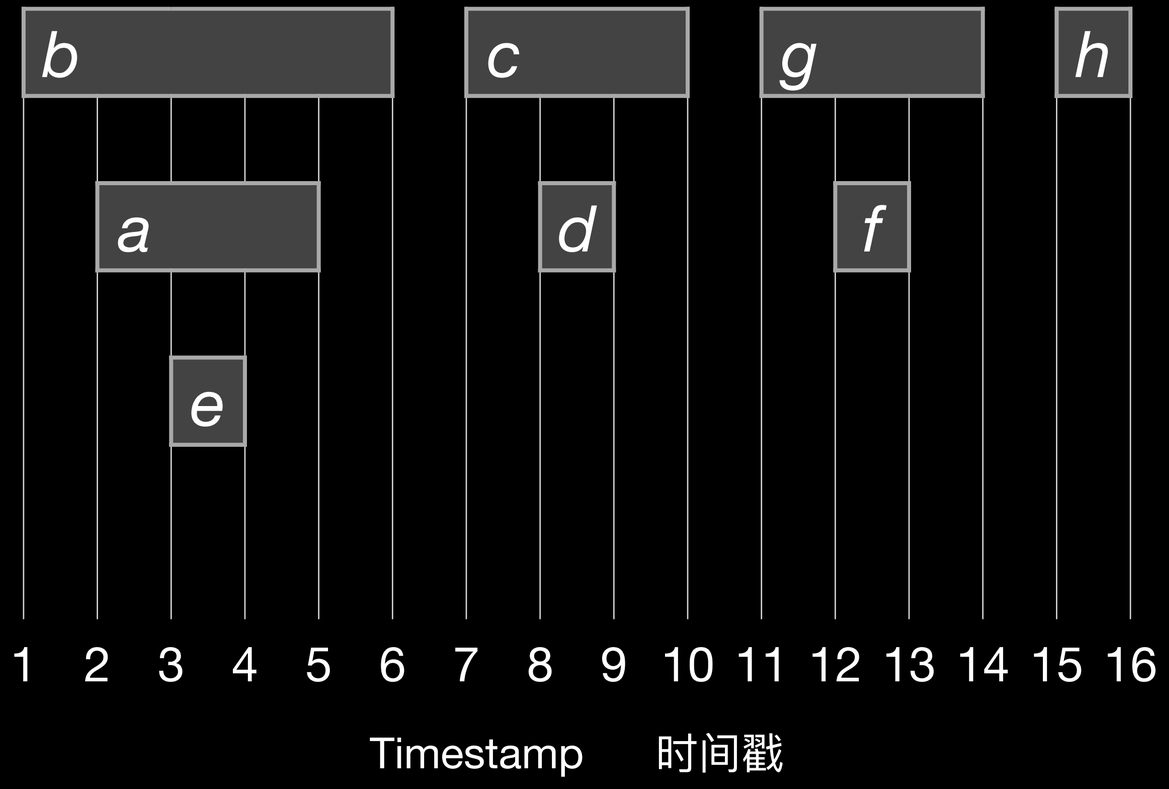
例子


b
c
g
h
a
d
f
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

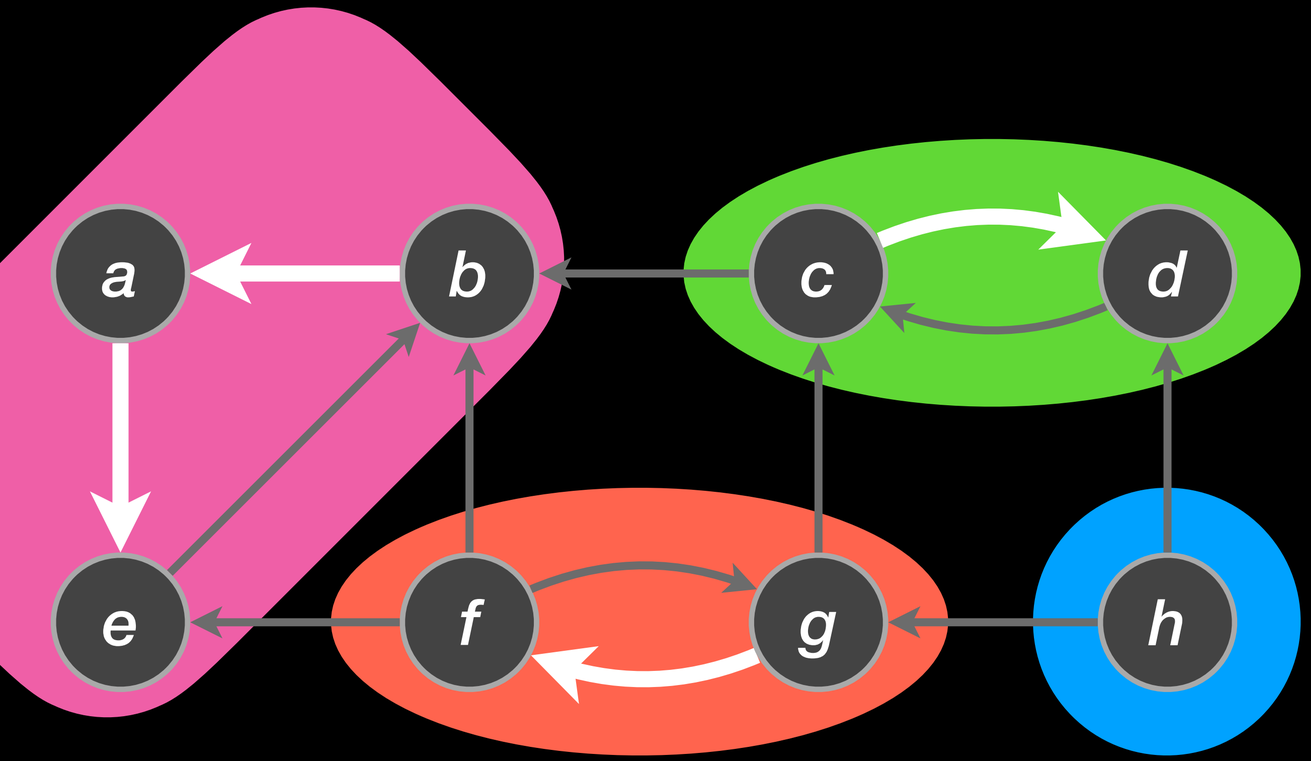
Example


a
b
c
d
e
f
g
h

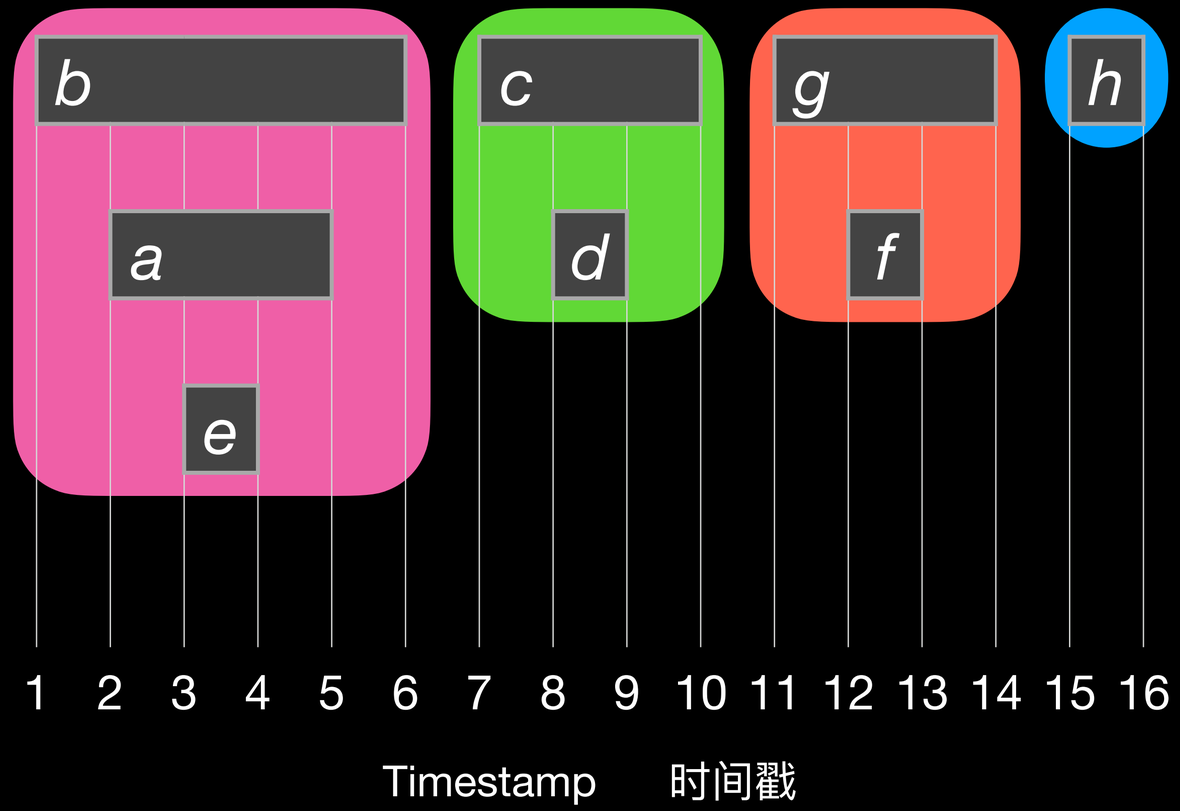
例子


b
c
g
h
a
d
f
e
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

Example

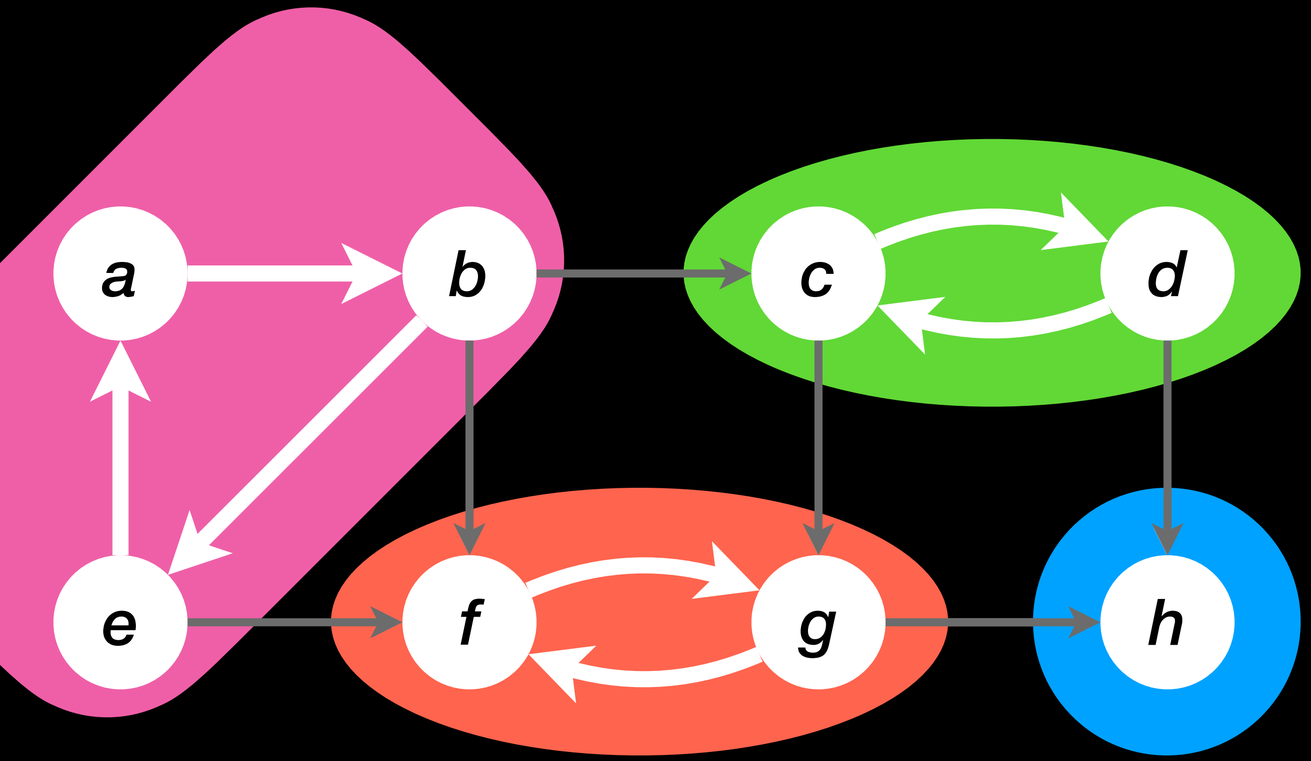

a
b
c
d
e
f
g
h

例子


b
a
e
c
d
g
f
h
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Timestamp 时间戳

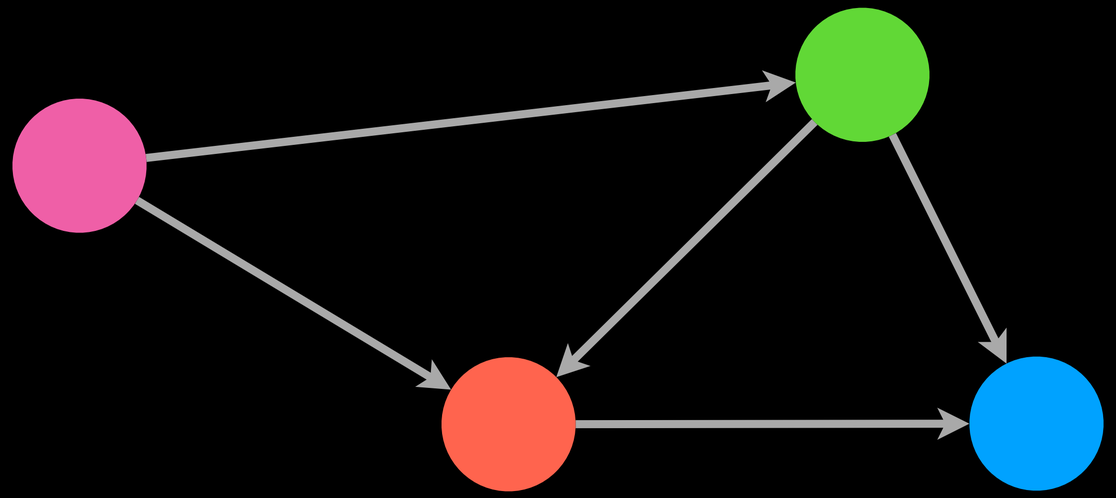
Example

例子



Example

例子



component graph
分量图

always acyclic
总是无环

Partial Correctness

- For a set of vertices $C \subseteq V$,
  let $d(C) = \min \{u.d \mid u \in C\}$ and
  $f(C) = \max \{u.f \mid u \in C\}$

- **Lemma 22.14:**
  Let C and C' be distinct SCCs.
  If there is and edge from C to C',
  then $f(C) > f(C')$.

部分正确性

- 对于一组顶点$C \subseteq V$
  设$d(C) = \min \{u.d \mid u \in C\}$及
  设$f(C) = \max \{u.f \mid u \in C\}$

- 引理22.14：
  设C和C'是图的不同强连通分量。
  如果从C到C'之间存在边和边，
  则$f(C) > f(C')$。

Partial Correctness

- **Corollary 22.15:** Let C and C' be distinct SCCs of a graph G. If there is and edge from C to C' in the transposed graph G^T, then $f(C) < f(C')$.

部分正确性

- 推论22.15：
  设C和C'是图G的不同强连通分量。
  如果转置图G^T中存在从C到C'的边，
  则$f(C) < f(C')$。
-

Partial Correctness

- **Theorem 22.16:** STRONGLY-CONNECTED-COMPONENTS computes the SCCs of the directed graph provided as its input.

- We prove "The first k depth-first trees output by DFS(G^T) are SCCs" by induction over k.

部分正确性

- 定理22.16：STRONGLY-CONNECTED-COMPONENTS计算作为其输入的有向图的强连通分量。

- 通过对k的归纳，证明"DFS(G^T)输出的前k个深度优先树是强连通分量"。

Running Time

- The algorithm STRONGLY-CONNECTED-COMPONENTS runs two DFSes.
- Every DFS requires time $O(|V| + |E|)$.
- Sorting the vertices according to $u.f_1$ does not take additional time, as the first DFS can create a list respecting this order (like for topological sort).
- So the total running time is in $O(|V| + |E|)$.

运行时间

- STRONGLY-CONNECTED-COMPONENTS 算法运行两次深度游向搜索。
- 每个深度游向搜索需要时间$O(|V| + |E|)$。
- 根据$u.f_1$对顶点进行排序不需要额外的时间，因为第一次搜索可以创建一个与此顺序相关的列表（如拓扑排序）。
- 因此，总运行时间为$O(|V| + |E|)$。

Minimum Spanning Tree

最小生成树

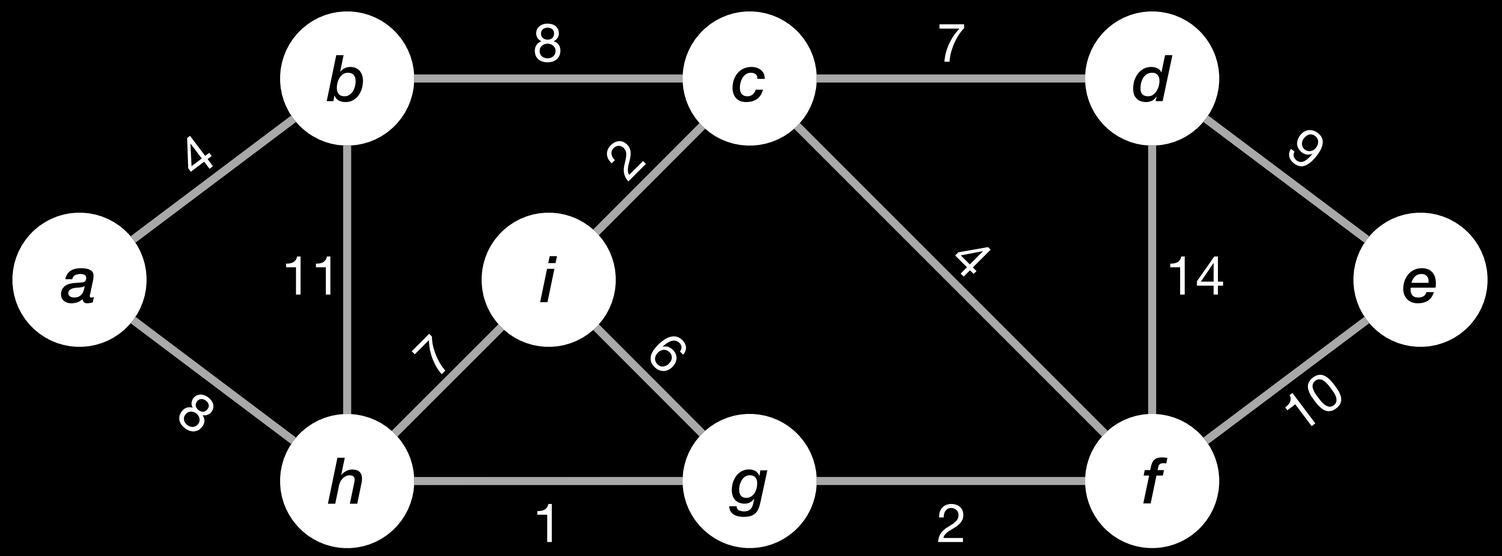
Cheapest Network

最便宜的网络

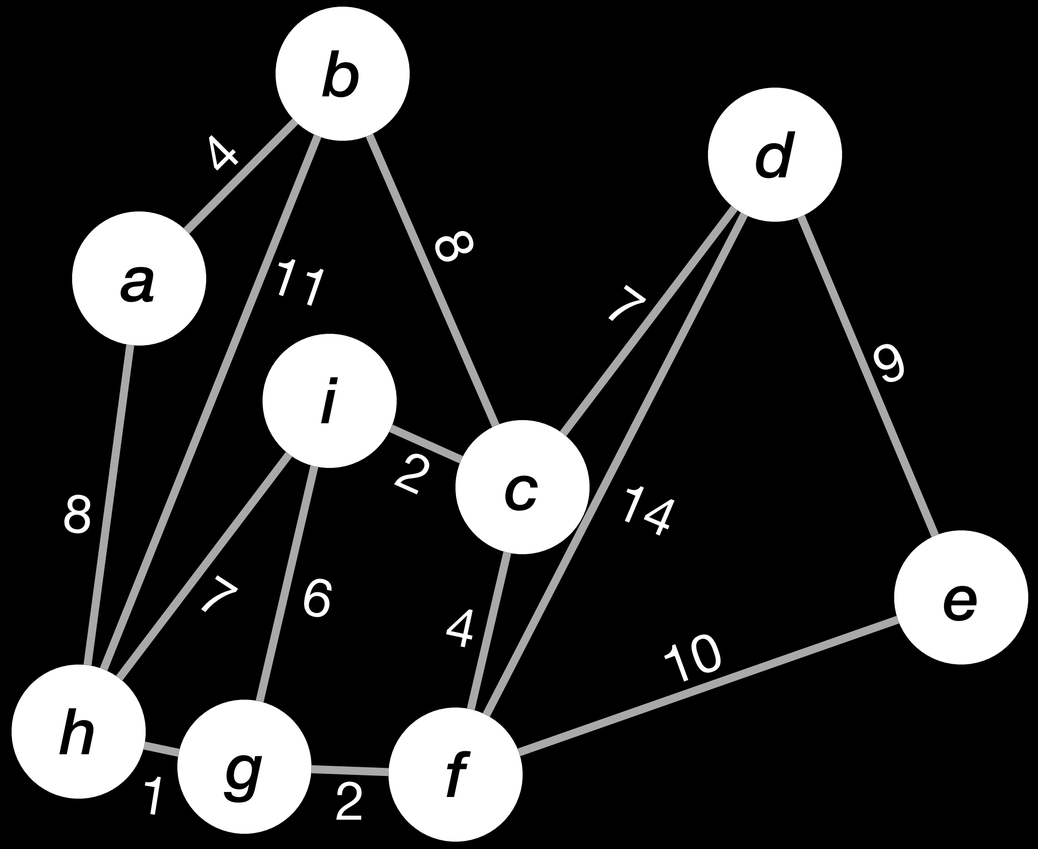
- Example: A number of computers are connected in a network. What is the cheapest network?
- Solution: minimum spanning tree
  - shortest edges
  - all computers / vertices connected
  - no cycles

- 示例：许多计算机在网络中连接。最便宜的网络是什么?
- 解决方案：最小生成树
  - 最短边
  - 连接的所有计算机 / 顶点
  - 没有周期

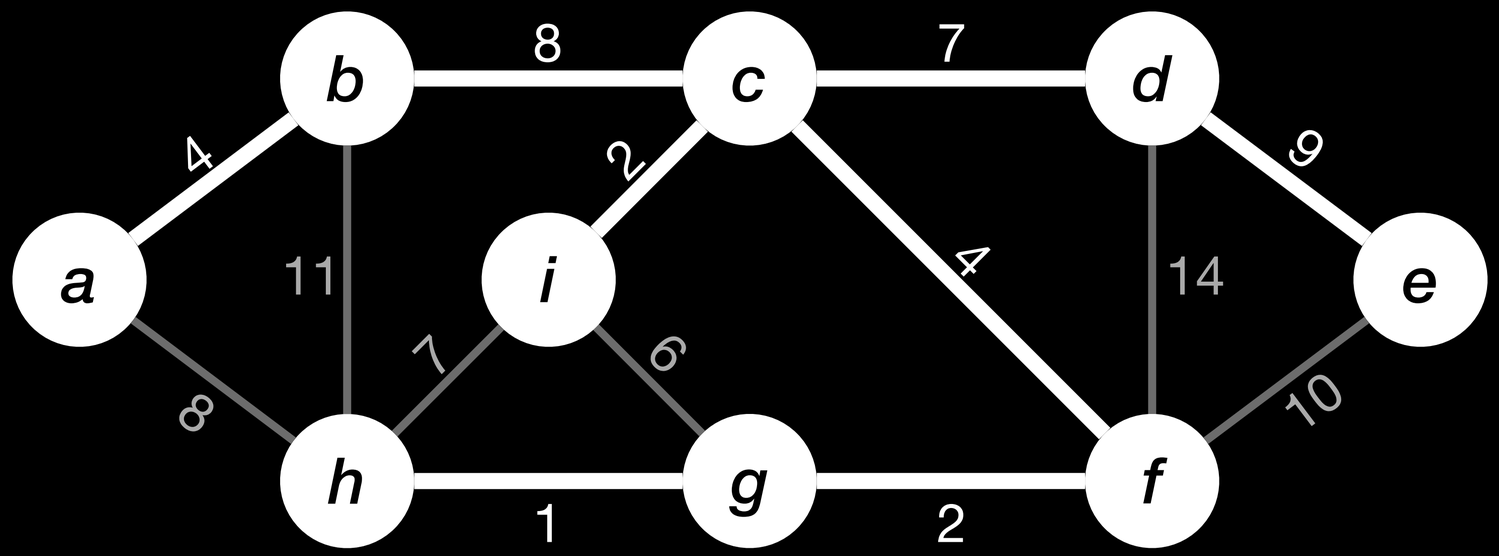
Cheapest Network 最便宜的网络



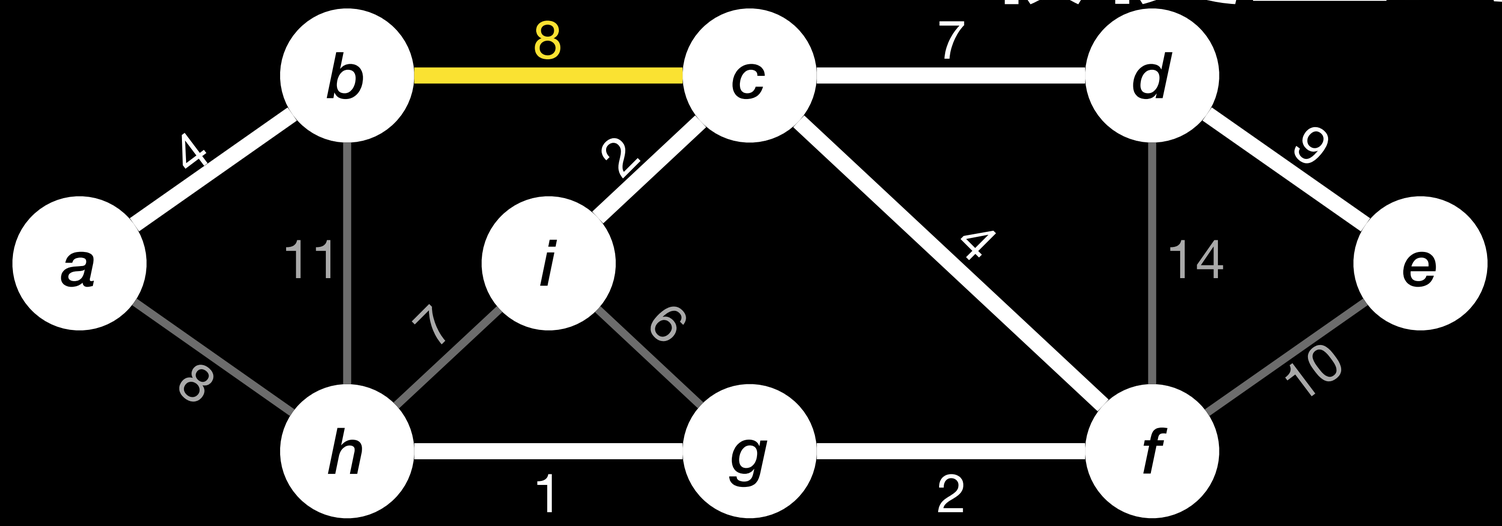
Cheapest Network 最便宜的网络

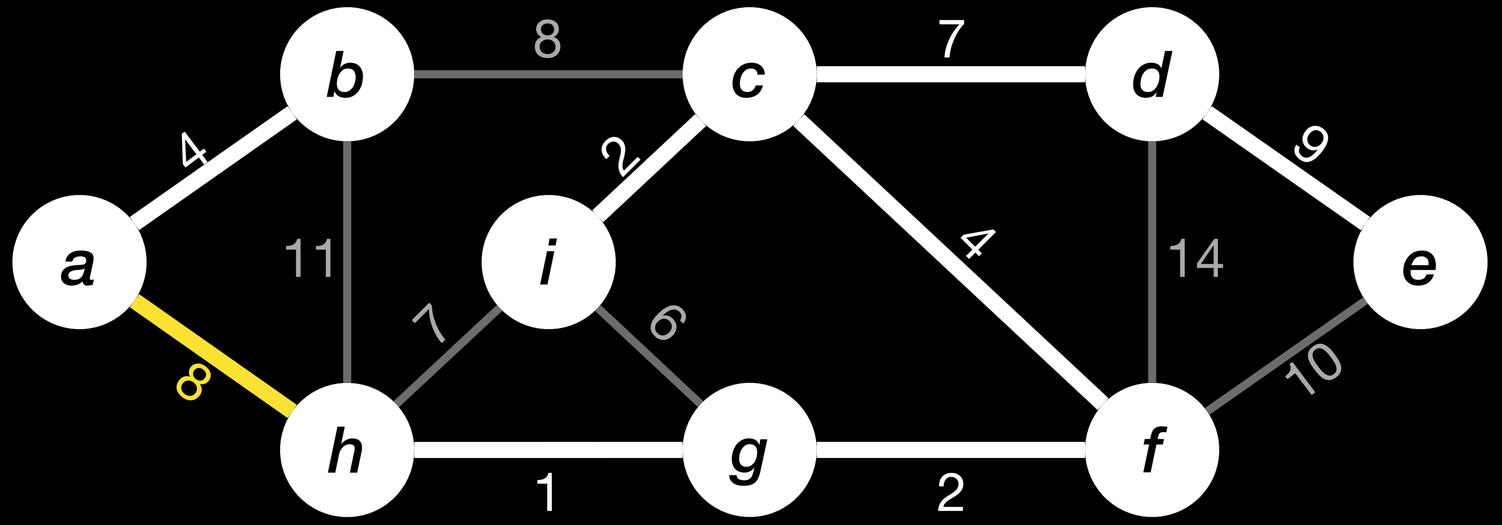


Cheapest Network 最便宜的网络



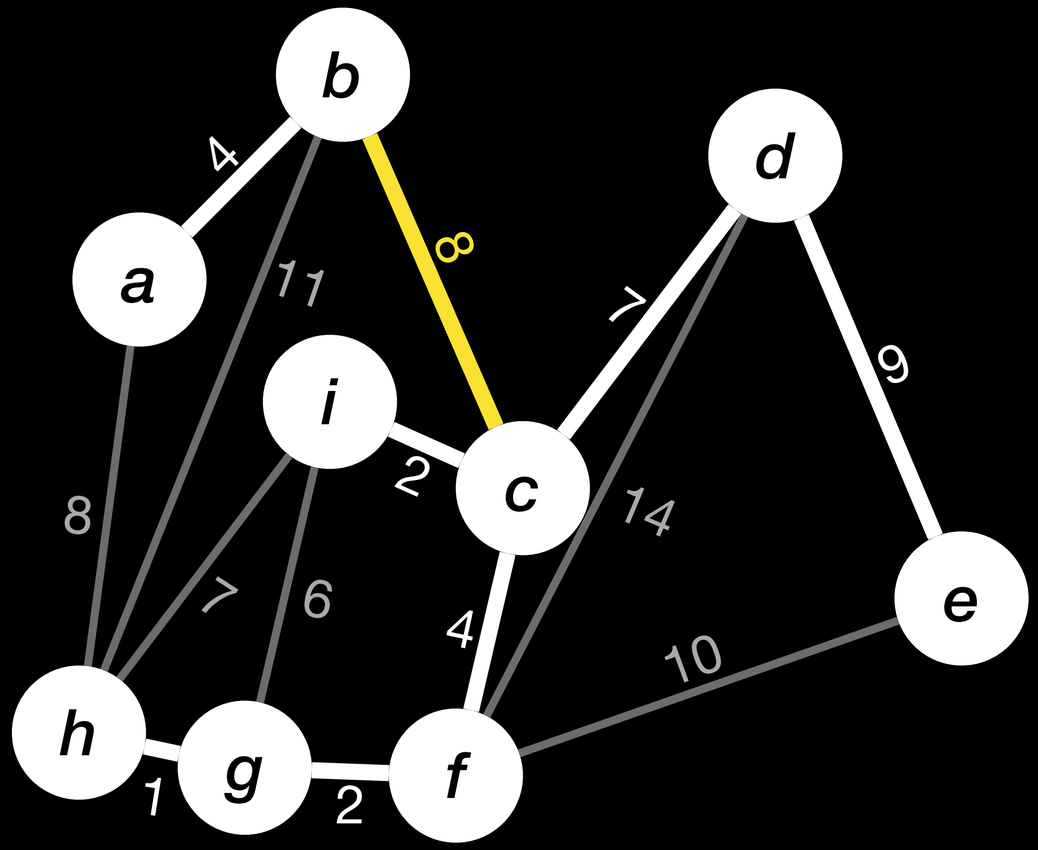
Cheapest Network 最便宜的网络

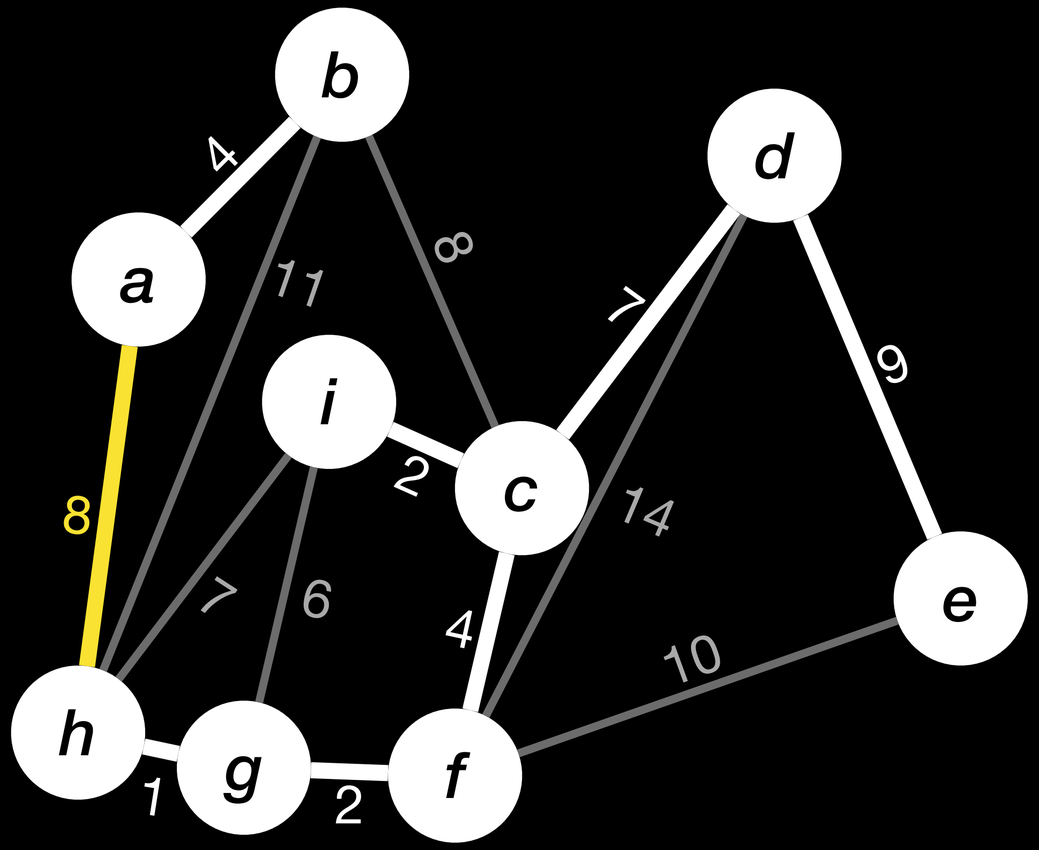




Cheapest Network

最便宜的网络





Generic Method

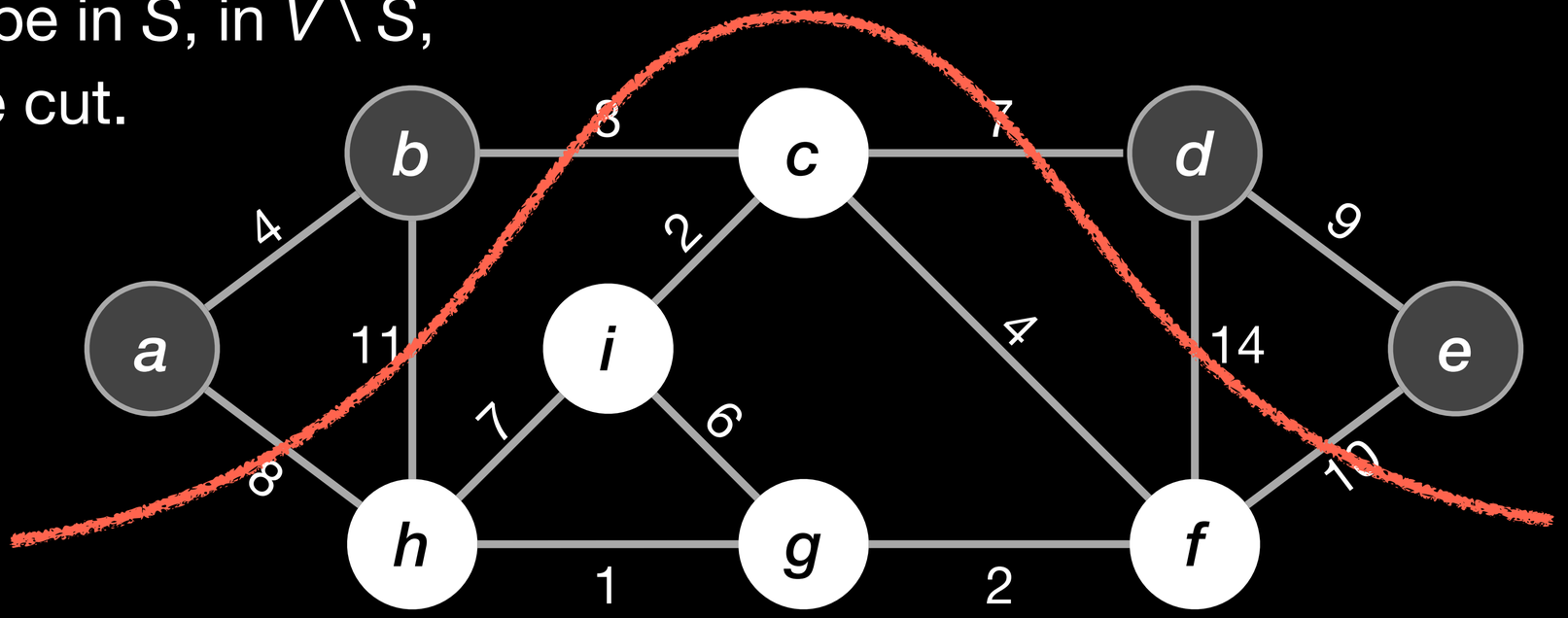
- Idea for a greedy algorithm:
  - Start with a subset of the MST
  - In every step, add one “safe” edge, i.e. one edge that is ok to be added
- Q: What is a “safe” edge?
  A: Theorem 23.1

通用方法

- 贪婪算法的想法：
  - 从最小生成树的子集开始
  - 在每一步中，添加一个“安全”边缘，i.e.可以添加的一条边
- 问：什么是“安全”边?
  答：定理23.1

Cut of a Graph 图的切割

- A cut of a graph $G = (V, E)$ separates the vertices into two subsets S and $V \setminus S$.
- Edges can be in S, in $V \setminus S$, or cross the cut.



Cut of a Graph

- A cut of a graph $G = (V, E)$ separates the vertices into two subsets S and $V \setminus S$.
- Edges can be in S, in $V \setminus S$, or cross the cut.
- A cut *respects* a set of edges $A \subseteq E$ if no edge in A crosses the cut.

图的切割

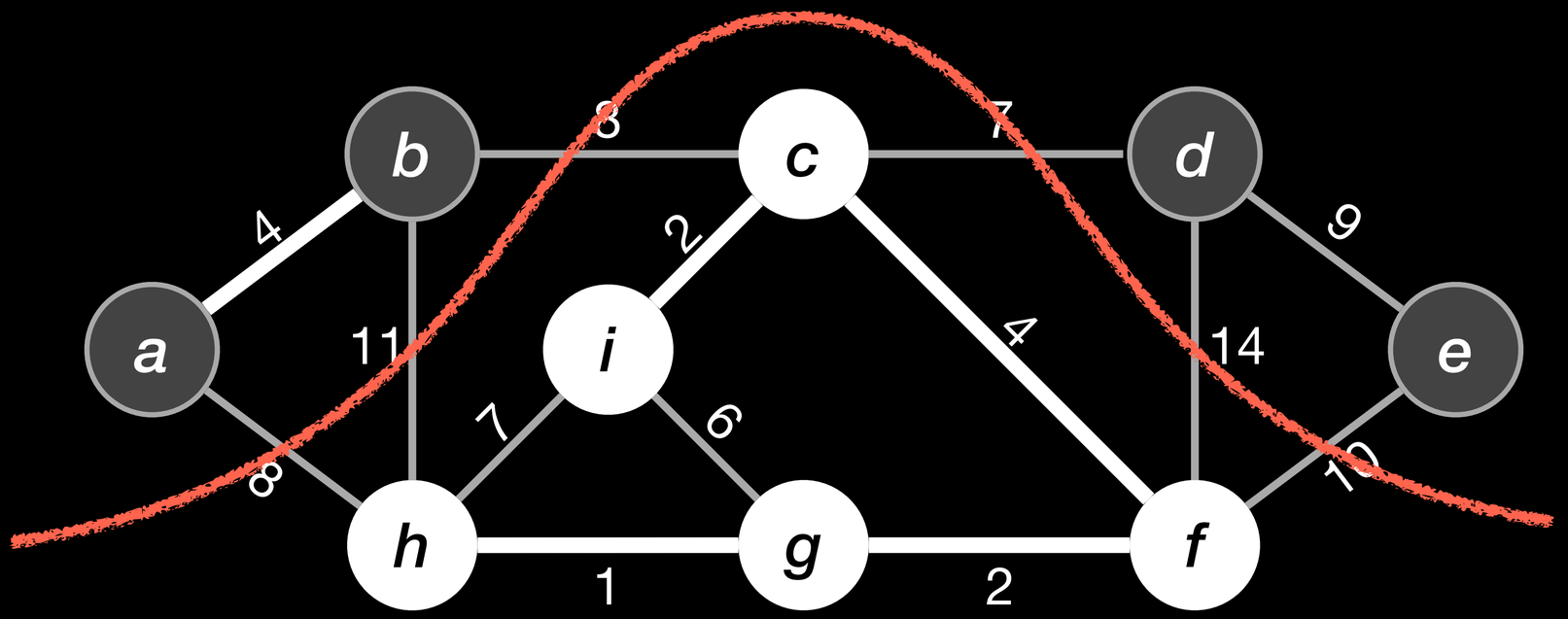
- 图$G = (V, E)$的切割分开顶点成两个子集S和$V\setminus S$。
-

Example: Respect a cut

例子

- Cut $S = \{a, b, d, e\}$
  $A = \{(a,b), (c,f), (c,i), (f,g), (g,h)\}$

- The cut S respects the set of edges A.



Safe Edge for MST

- **Theorem 23.1**
  Let A be a subset of the edges of a graph that is included in some MST.
  Let $(S, V \setminus S)$ be any cut of the graph that respects A.
  Then every light edge crossing the cut is safe for A.

- (light edge = edge with minimal weight)

Kruskal's Algorithm

- Idea:
  Always add the lightest/shortest edge
  that is allowed.

- Allowed are edges
  that do not form a cycle.

- Q: How do we test
  whether an edge is allowed?
  A: Check whether
  endpoints are in the same tree.

Kruskal的算法

- 想法：
  始终添加允许的最轻/最短边。

- 允许有边
  这不会形成一个循环。

- 问：我们如何测试
  是否允许边缘?
  答：检查一下
  端点在同一棵 树 中。

Disjoint Sets

- a data structure to describe
  a partition of a set
- operations:
  - UNION: join two subsets into one
  - FIND-SET: find the representative
    of the subset
- sometimes called
  “Union–Find data structure”

不相交集合

- 要描述的数据结构
  集合的划分
- 操作：
  - UNION：将两个子集合并为一个
  - FIND-SET：查找子集的代表
- 有时叫
  “Union–Find数据结构”

Representative

- Every subset is identified
  by a representative element.
  When the set does not change,
  the representative stays the same.

- Operations in detail:
  - MAKE-SET(x): create structure for $\{x\}$
  - UNION(x,y): join the subsets containing
    x and y into one
  - FIND-SET(x): find the representative
    of the subset containing x

代表

- 每个子集被识别
  由一个代表的成员。
  当集合不变时，
  代表保持不变。

- 具体操作：
  - MAKE-SET(x)：为$\{x\}$创建结构
  - UNION(x,y)：将包含
    x和y合为一
  - FIND-SET(x)：查找代表
    包含x的子集的

Example: Connected Components

例子：连通分量

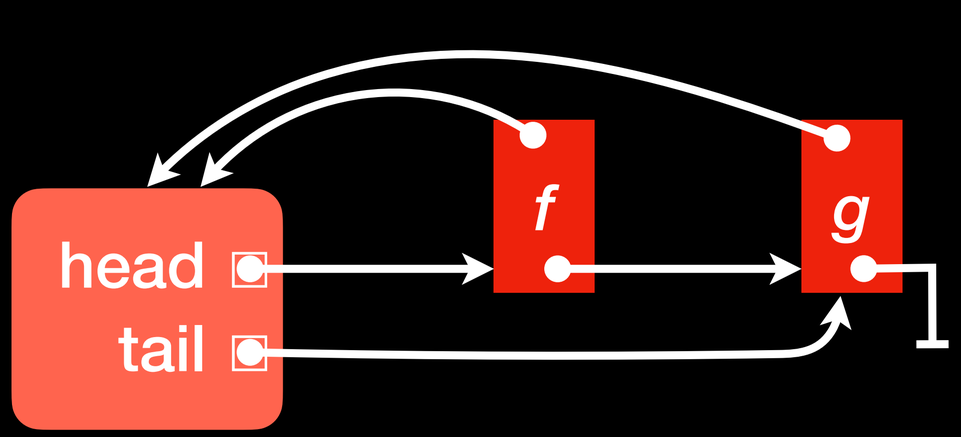
- **Idea:**
  Start with every vertex in its own set.
  If two sets are connected, unify them.

- 想法：
  从其自身集合中的每个顶点开始。
  如果两组连接，则将它们统一起来。

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
        MAKE-SET(v)
for each edge (u,v) ∈ G.E
        if FIND-SET(u) ≠ FIND-SET(v)
                UNION(u,v)
return the partition of vertices
```

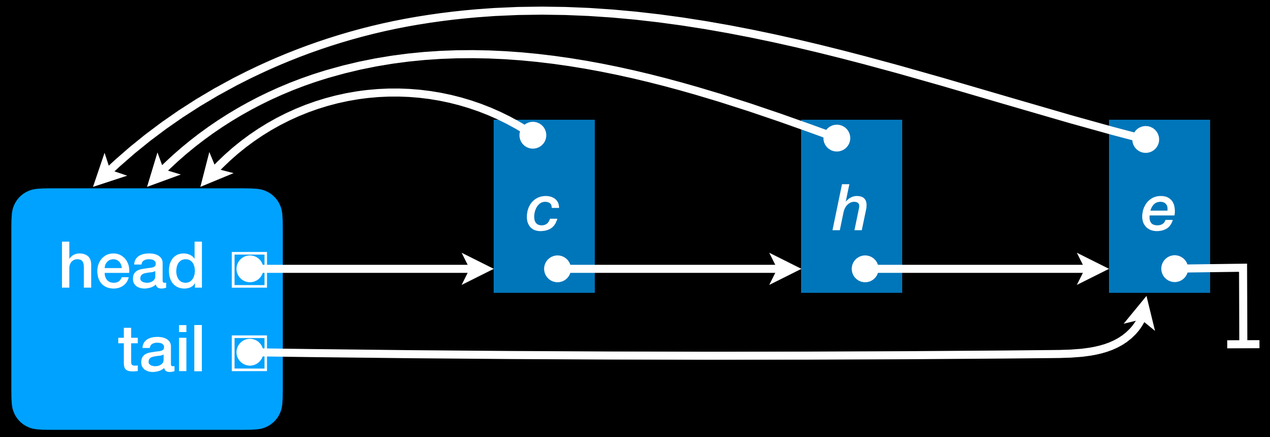
Simple Implementation: Linked Lists

- Store subset as linked list of elements
  representative = first element
- Example: sets {*f*,*g*} and {*c*,*h*,*e*}
- UNION(*g*,*c*)



简单实现：链表

- 为元素的链接列表
  代表是第一要素
- 示例：集合 {*f*,*g*}和{*c*,*h*,*e*}
- UNION(*g*,*c*)

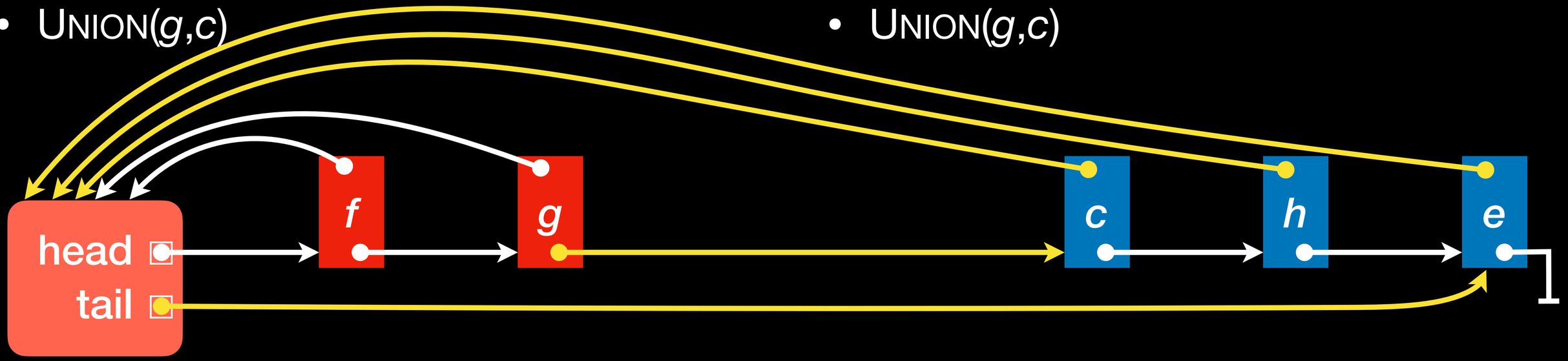


Simple Implementation: Linked Lists

简单实现：链表

- Store subset as linked list of elements
  representative = first element
- Example: sets {*f*,*g*} and {*c*,*h*,*e*}
- UNION(*g*,*c*)

- 为元素的链接列表
  代表是第一要素
- 示例：集合 {*f*,*g*}和{*c*,*h*,*e*}
- UNION(*g*,*c*)



Time Complexity

- Changing head/tail pointers is in $O(1)$, but changing pointers to the set object needs time Ω(size of the second subset).
- The aggregate time needed for a sequence of n operations may be up to $\Omega(n^2)$.

时间复杂性

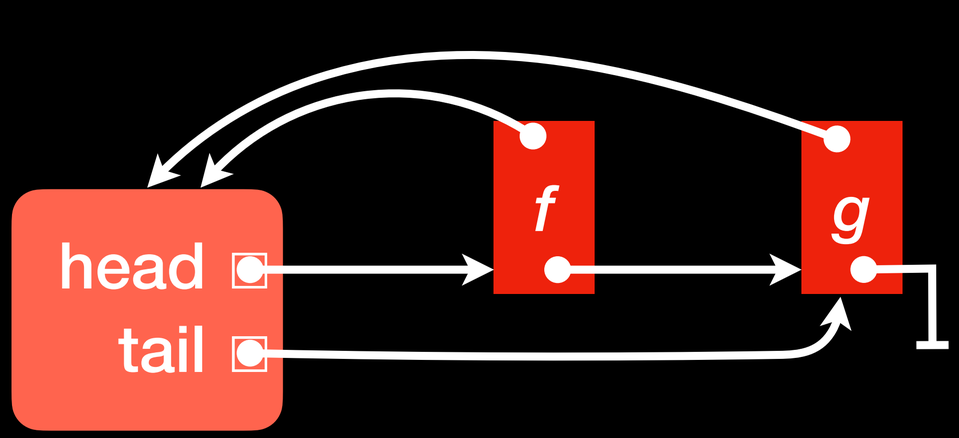
- 改变头/尾指针在$O(1)$中，但是更改指向集合对象的指针需要时间Ω(第二个子集的大小)。
- 所需的总时间对于n个操作序列可能高达$\Omega(n^2)$。

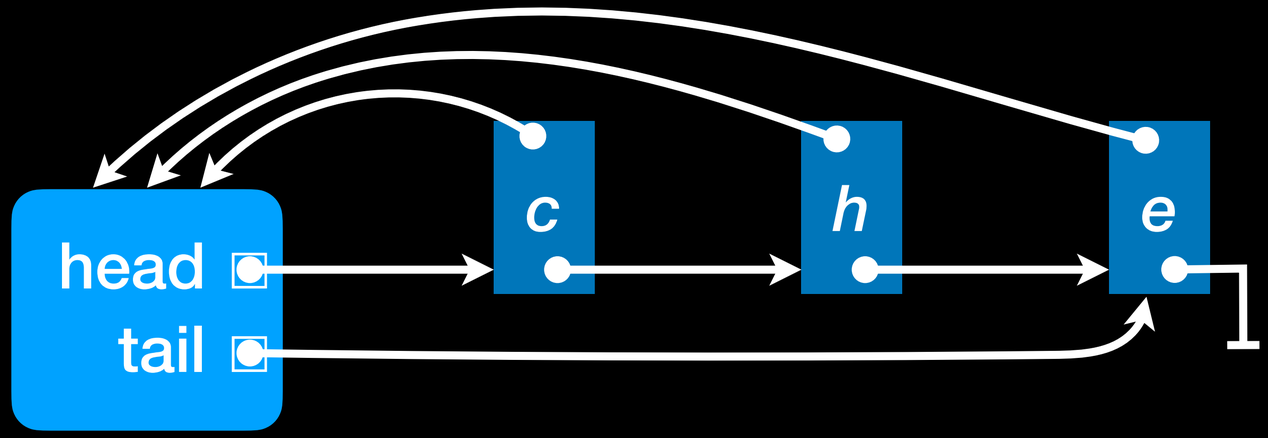
Improved Version

改进版

- Always add the smaller subset to the larger subset.

- 始终添加较小的子集到更大的子集。

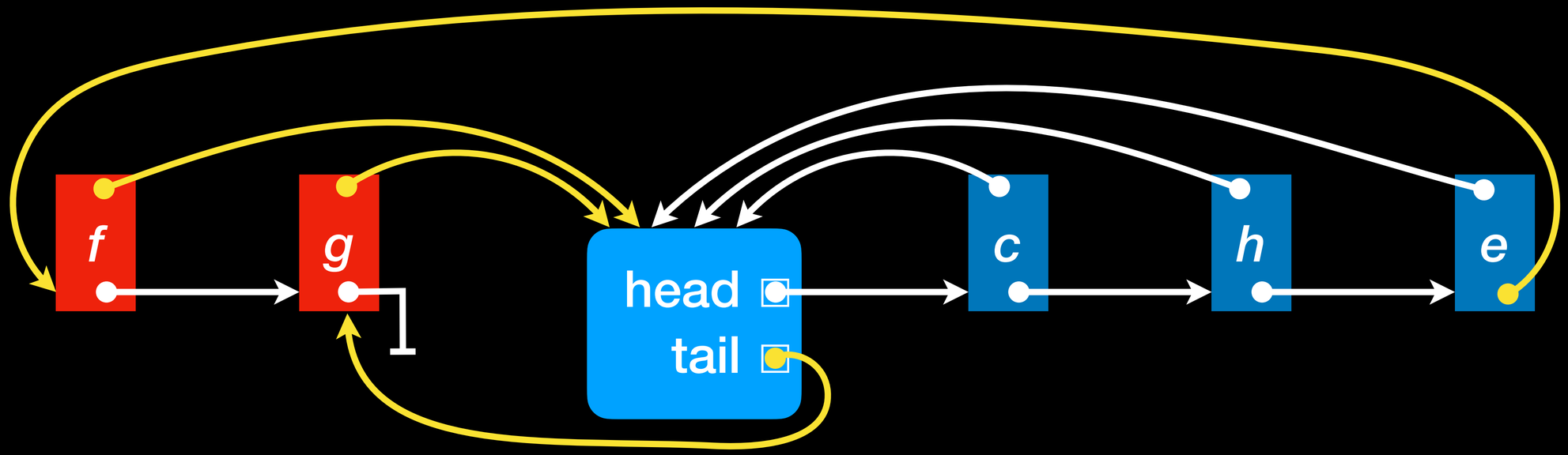




Improved Version

改进版

- Always add the smaller subset to the larger subset.
- 始终添加较小的子集到更大的子集。



Timing Anlysis of Improved version

- How often does the pointer of one element need to be changed?
- If x was in subset S_x and its pointer is changed to become part of subset S'_x, then
  $1 \le |S_x| \le \frac{1}{2}|S'_x| \le$ total number of elements
- x's pointer can be changed at most $\lfloor \lg(\text{total number of elements}) \rfloor$ times.
- n operations take time $O(n \log n)$.

改进版的时间复杂度

- 一个元素的指针需要多久更改一次?
- 如果x在子集S_x中，且其指针更改为子集S'_x的一部分，
  则
  $1 \le |S_x| \le \frac{1}{2}|S'_x| \le$ 元素总数
- x的指针最多可以更改 $\lfloor \lg(\text{元素总数}) \rfloor$ 次。
- n个操作需要时间$O(n \log n)$。

More Improvements

- The book presents a more sophisticated data structure to reach almost linear time $O(n\ \alpha(n))$.
- Not a topic of our course.
- We will use the disjoint-set data structure only as part of an algorithm that runs in time $\Theta(n \log n)$.

更多改进

- 这本书提出了一个更复杂的数据结构，以达到几乎线性时间$O(n\ \alpha(n))$。
- 这不是我们课程的主题。
- 我们只使用不相交集数据结构作为在时间$\Theta(n \log n)$中运行的算法的一部分。

Kruskal's Algorithm

- Idea:
  Always add the lightest/shortest edge that is allowed.
- Allowed are edges that do not form a cycle.
- Q: How do we test whether an edge is allowed?
  A: Check whether endpoints are in the same **subset.**

Kruskal的算法

- 想法：
  始终添加允许的最轻/最短边。
- 允许有边
  这不会形成一个循环。
- 问：我们如何测试是否允许边缘?
  答：检查一下端点在同一棵 **子集** 中。

Kruskal's Algorithm Kruskal的算法

```
MST-KRUSKAL(G,w)
A = ∅
for each vertex v ∈ G.V
        MAKE-SET(v)
sort the edges in G.E by weight w
for each edge (u,v) ∈ G.E in order of weight
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u,v)}
                UNION(u,v)
return A
```

every vertex forms a trivial tree

每个顶点形成一棵平凡的树

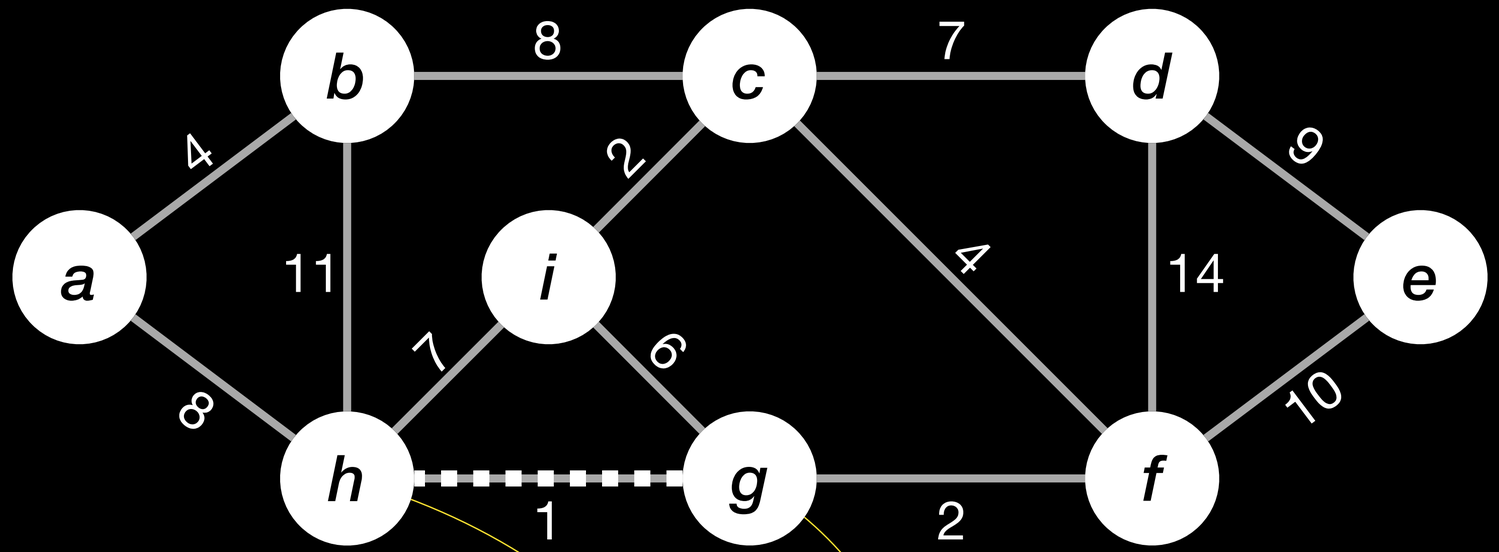
join trees until connected

连接树到连通起来

trees of *u* and *v* are not connected

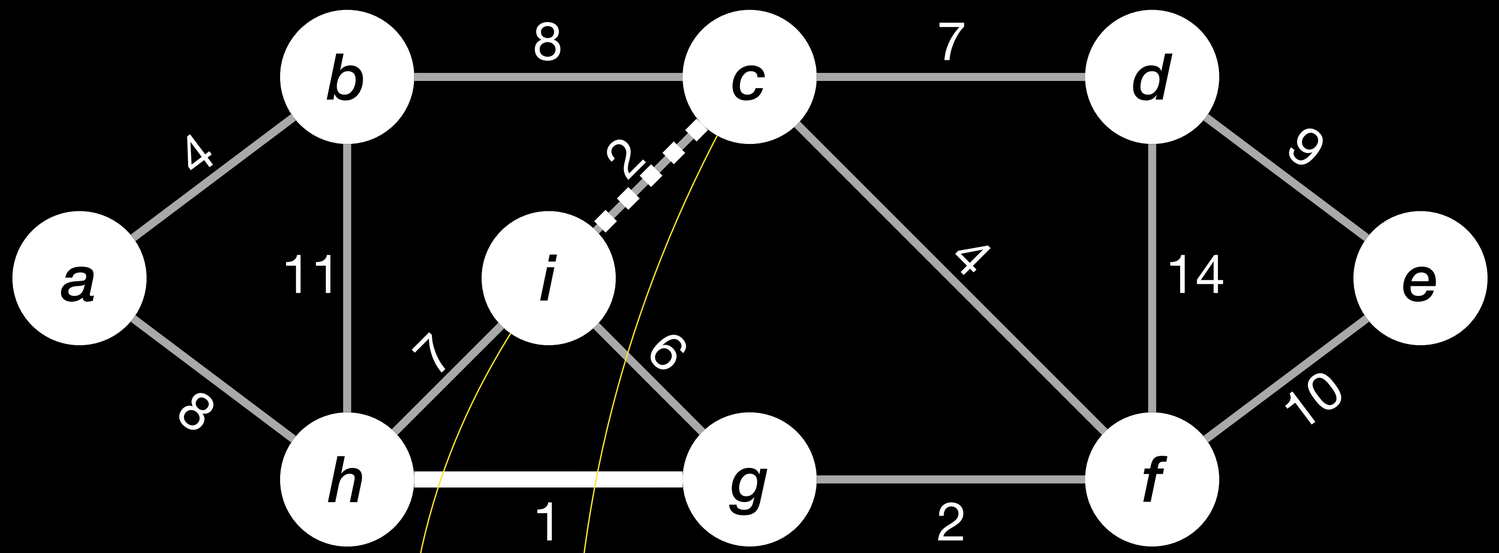
*u*和*v*的树不连通

Example 例子



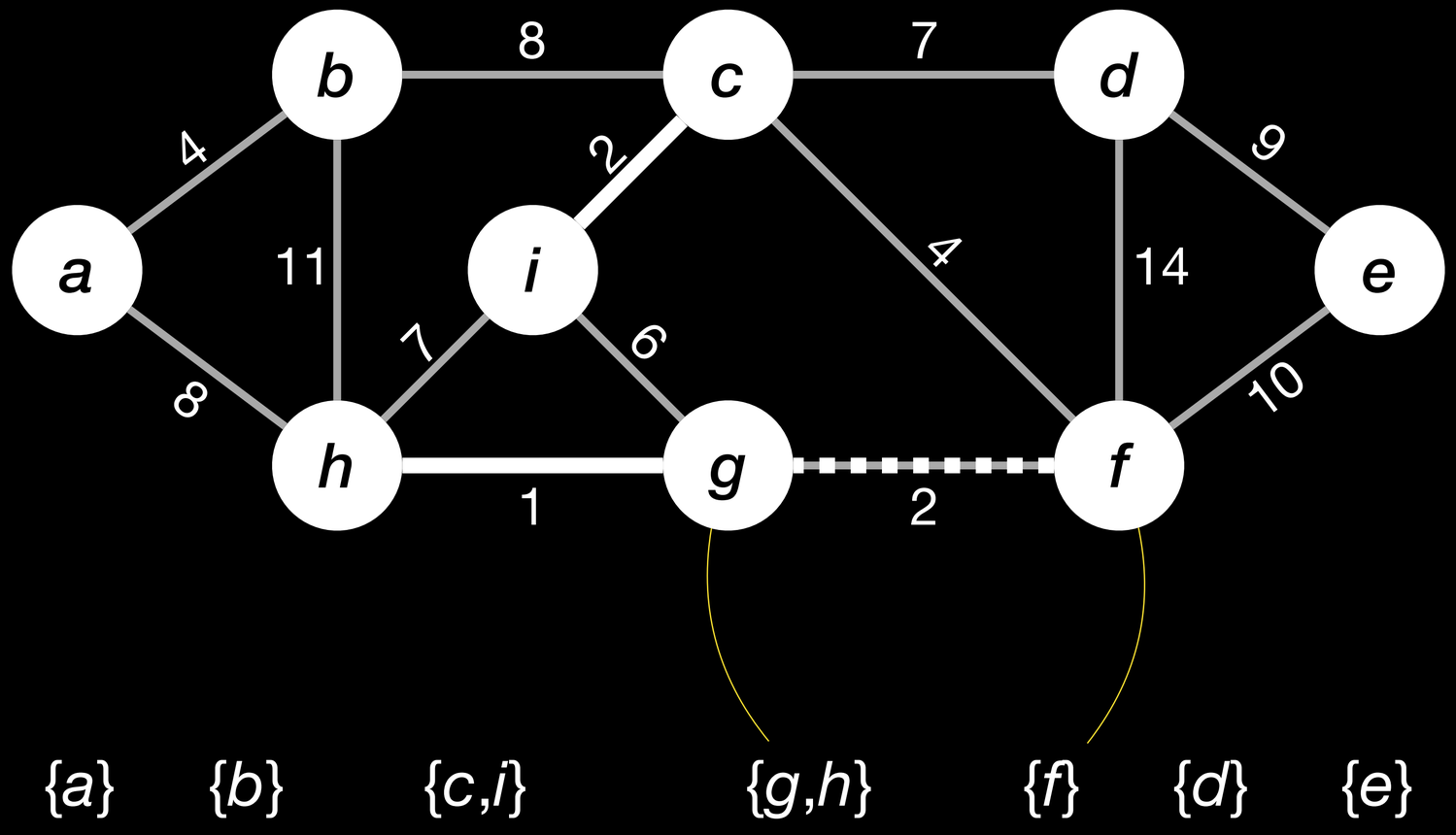
{*a*} {*b*} {*i*} {*c*} {*h*} {*g*} {*f*} {*d*} {*e*}

Example 例子

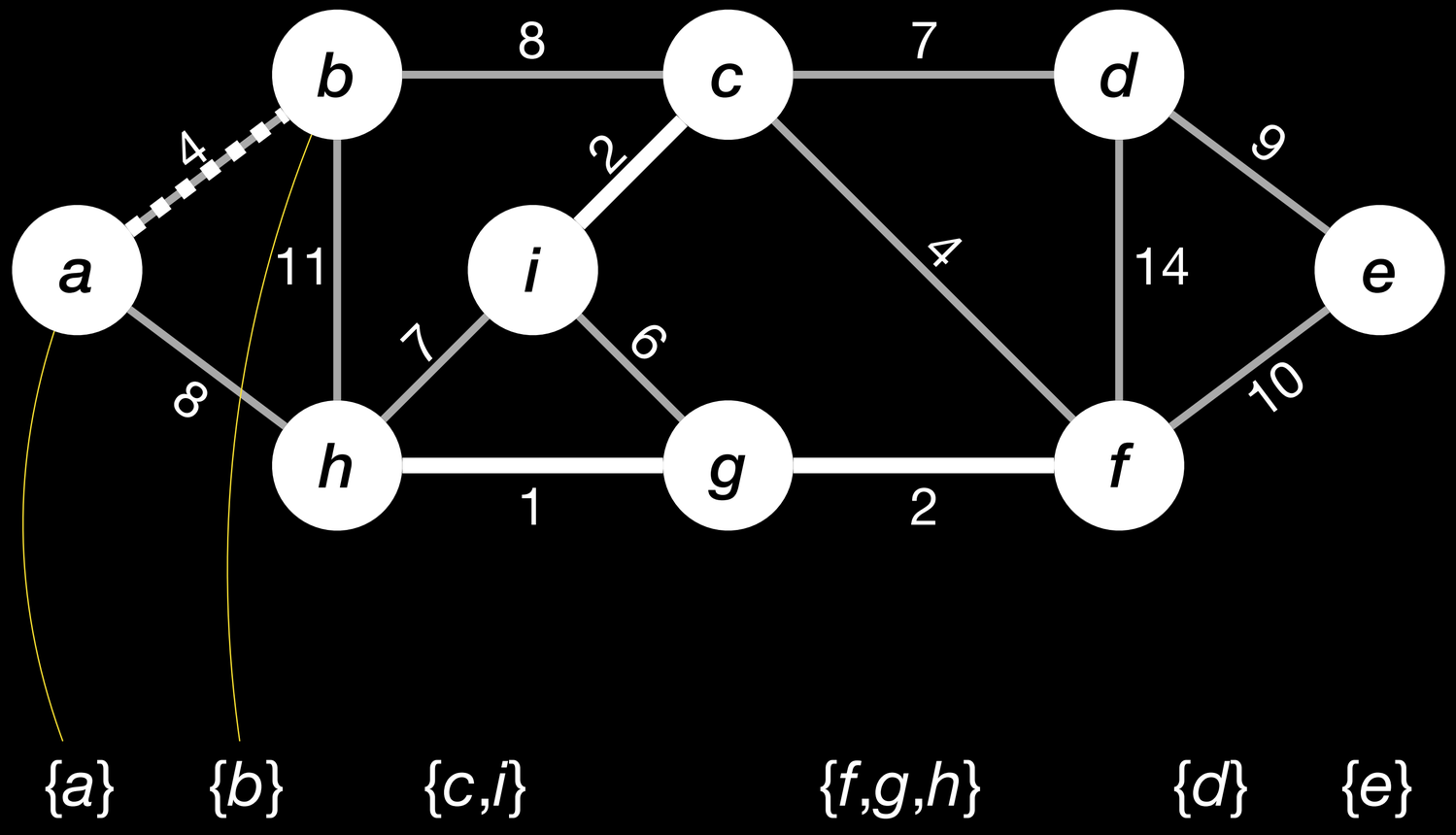


{*a*} {*b*} {*i*} {*c*} {*g*,*h*} {*f*} {*d*} {*e*}

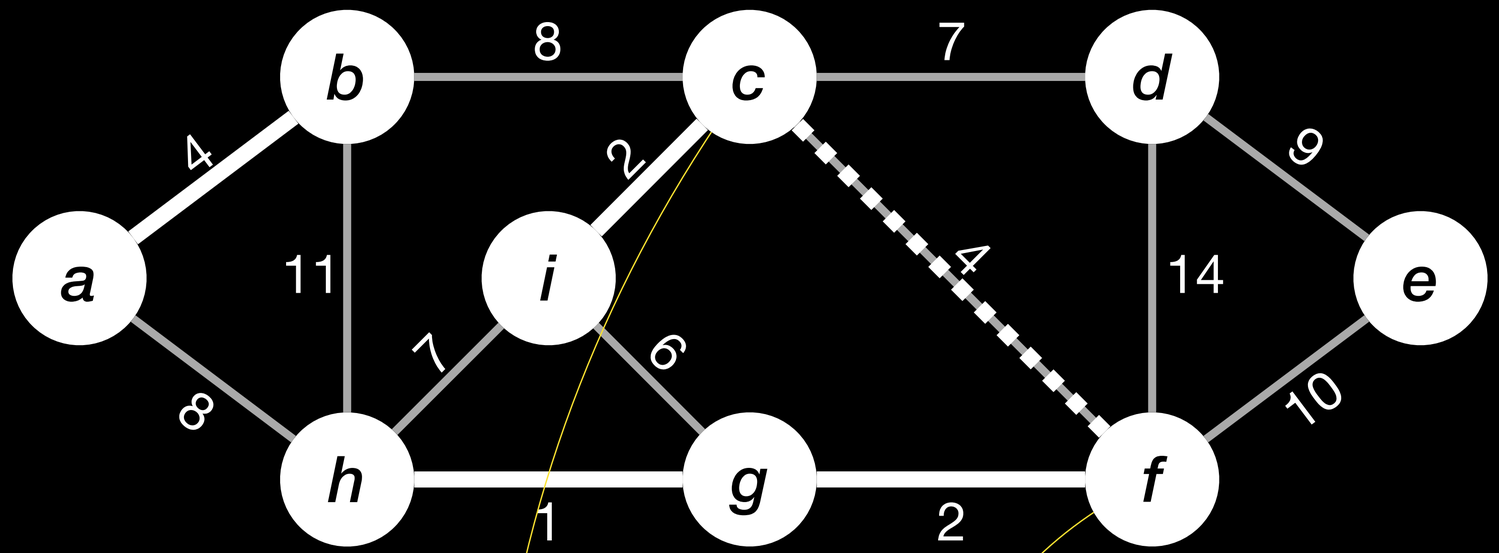
Example 例子



Example 例子

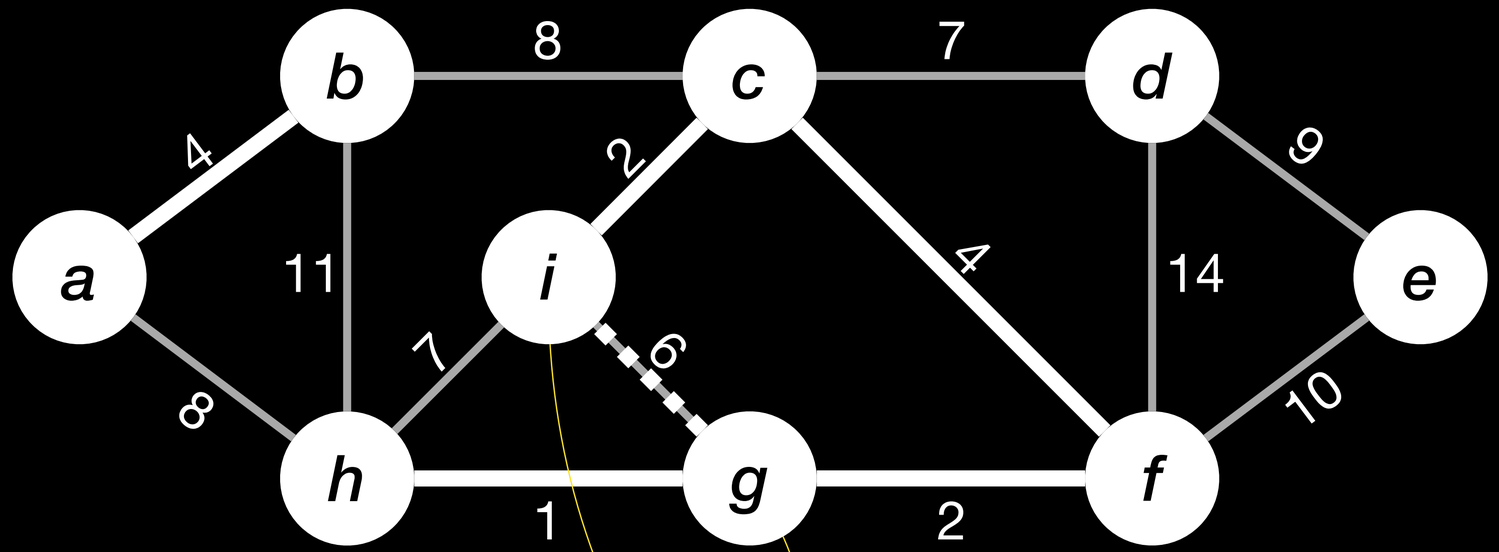


Example 例子



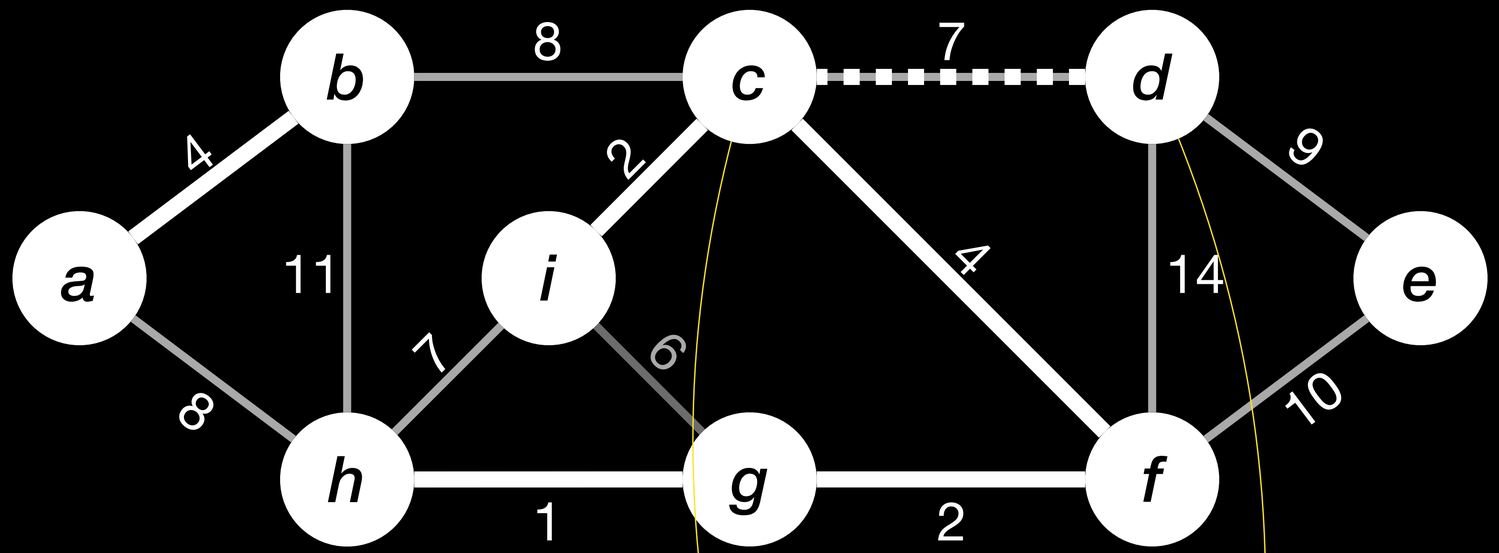
{*a*,*b*} {*c*,*i*} {*f*,*g*,*h*} {*d*} {*e*}

Example 例子



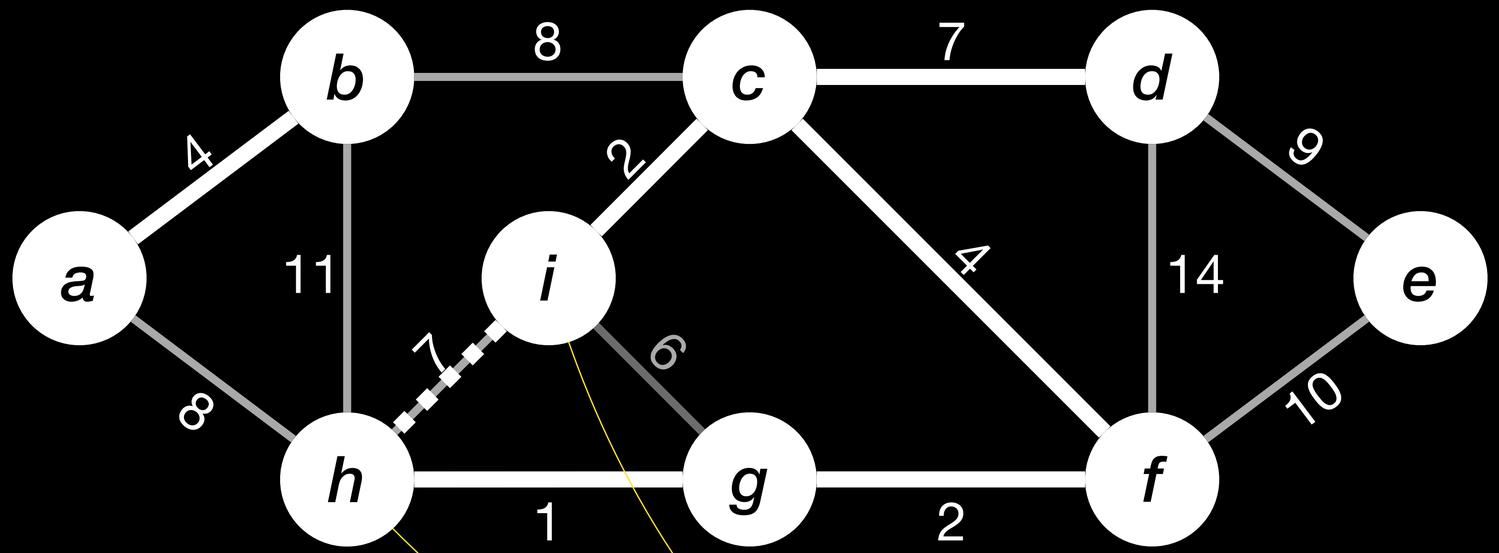
{*a*,*b*} {*c*,*f*,*g*,*h*,*i*} {*d*} {*e*}

Example 例子



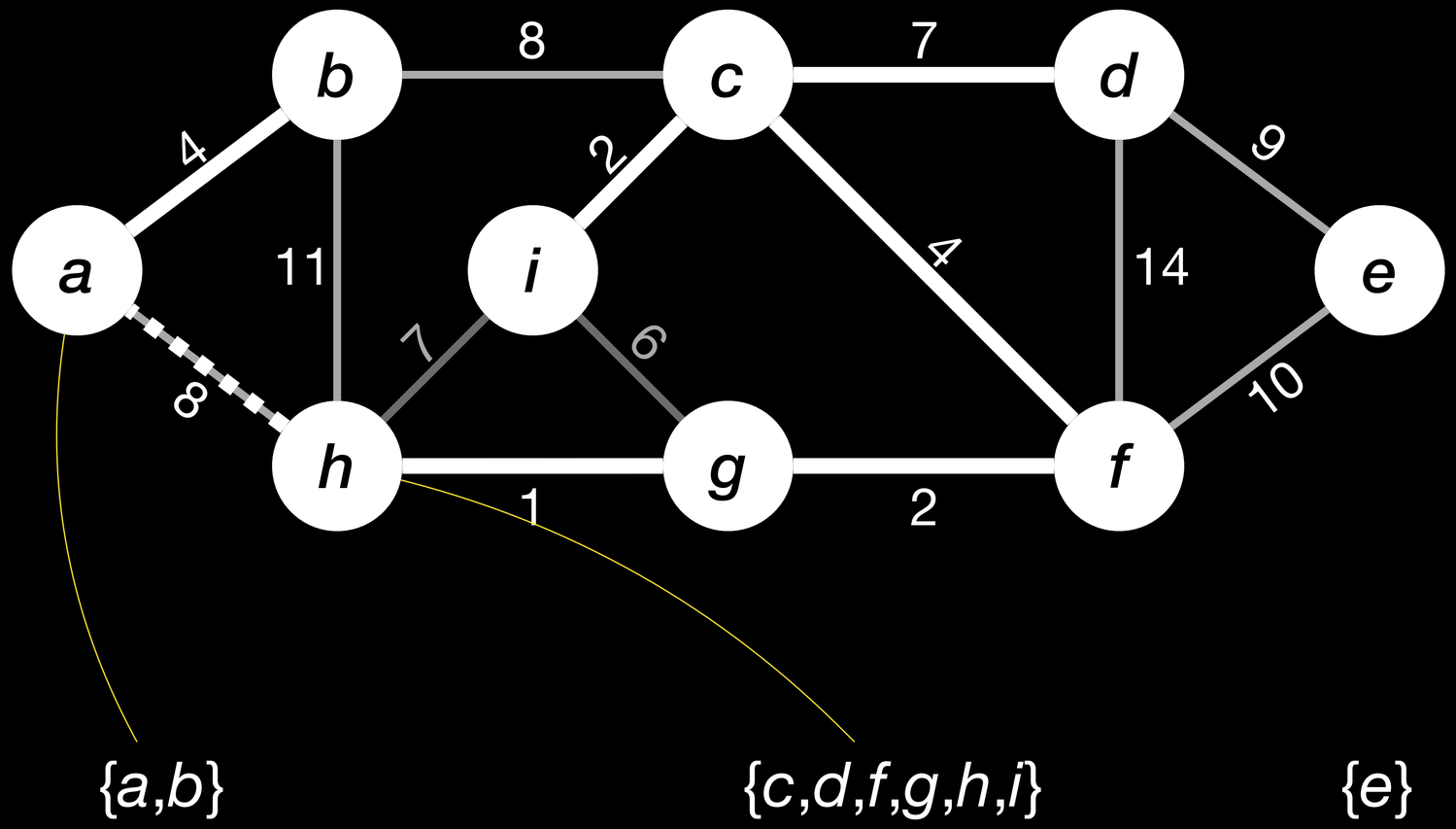
{*a*,*b*} {*c*,*f*,*g*,*h*,*i*} {*d*} {*e*}

Example 例子

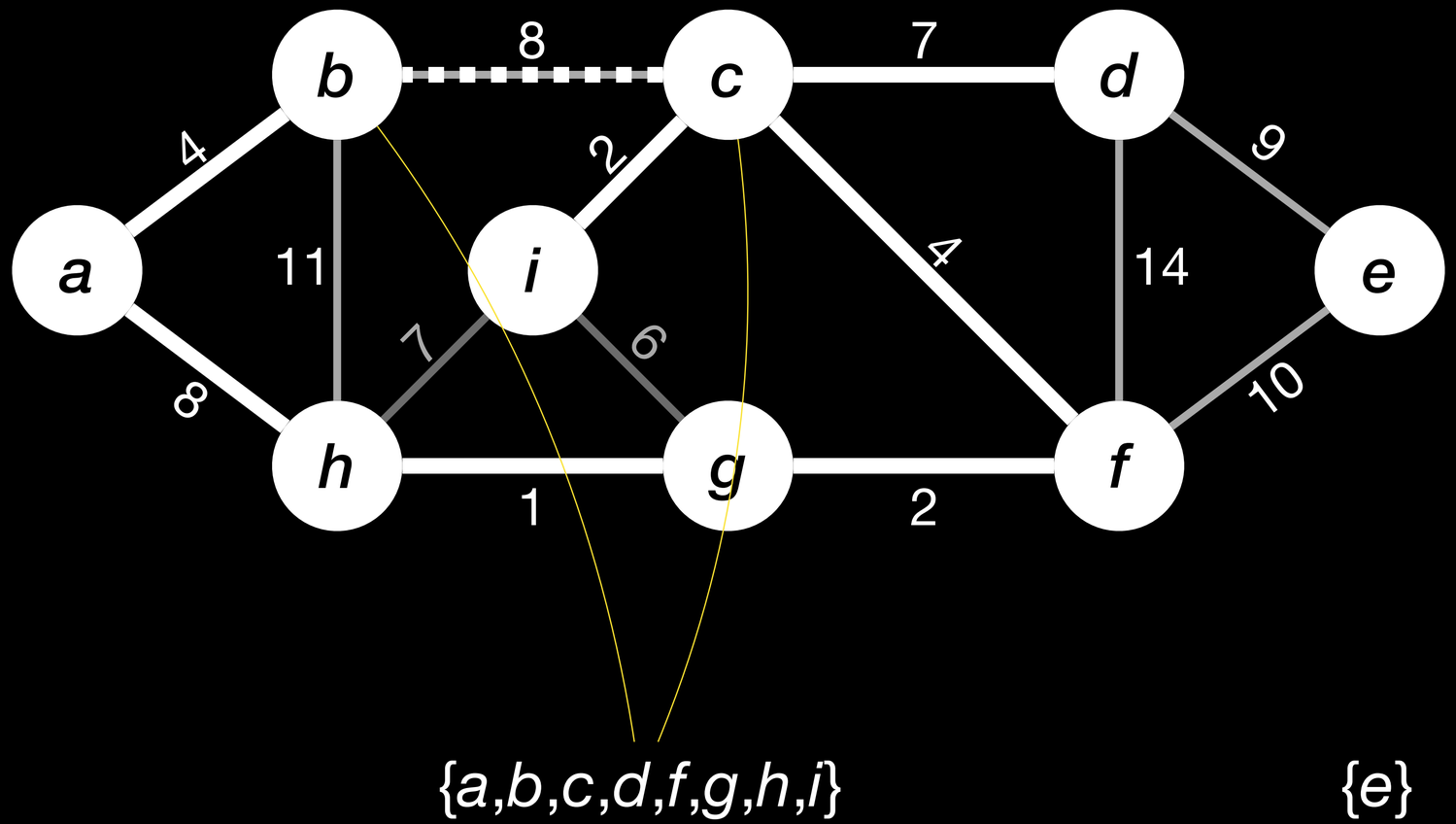


{*a*,*b*} {*c*,*d*,*f*,*g*,*h*,*i*} {*e*}

Example 例子

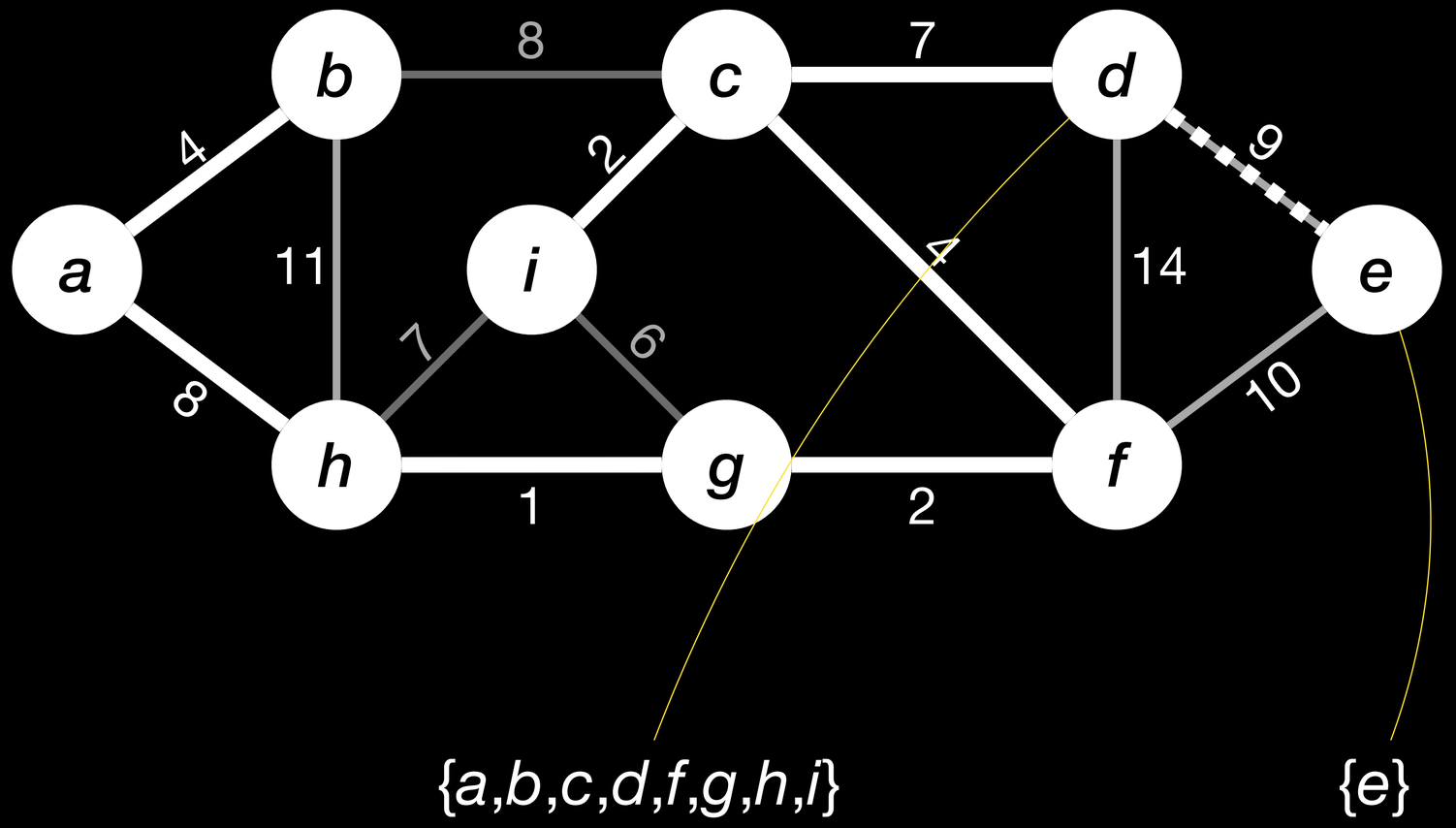


Example 例子

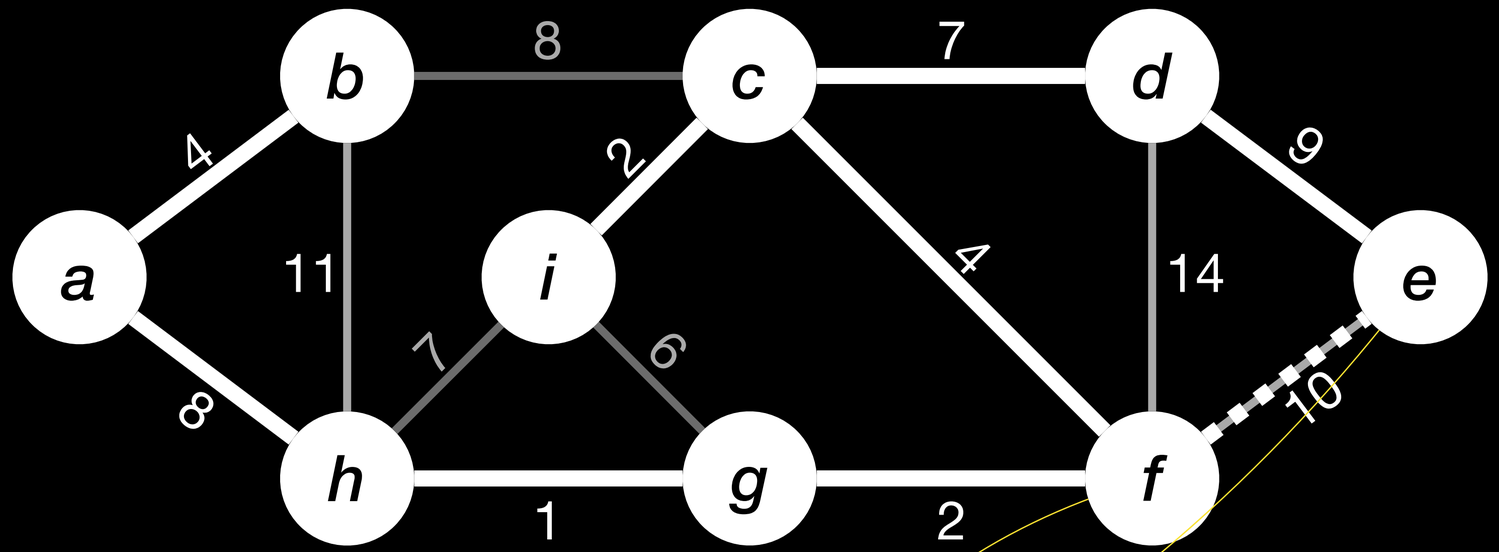


{*a*,*b*,*c*,*d*,*f*,*g*,*h*,*i*} {*e*}

Example 例子

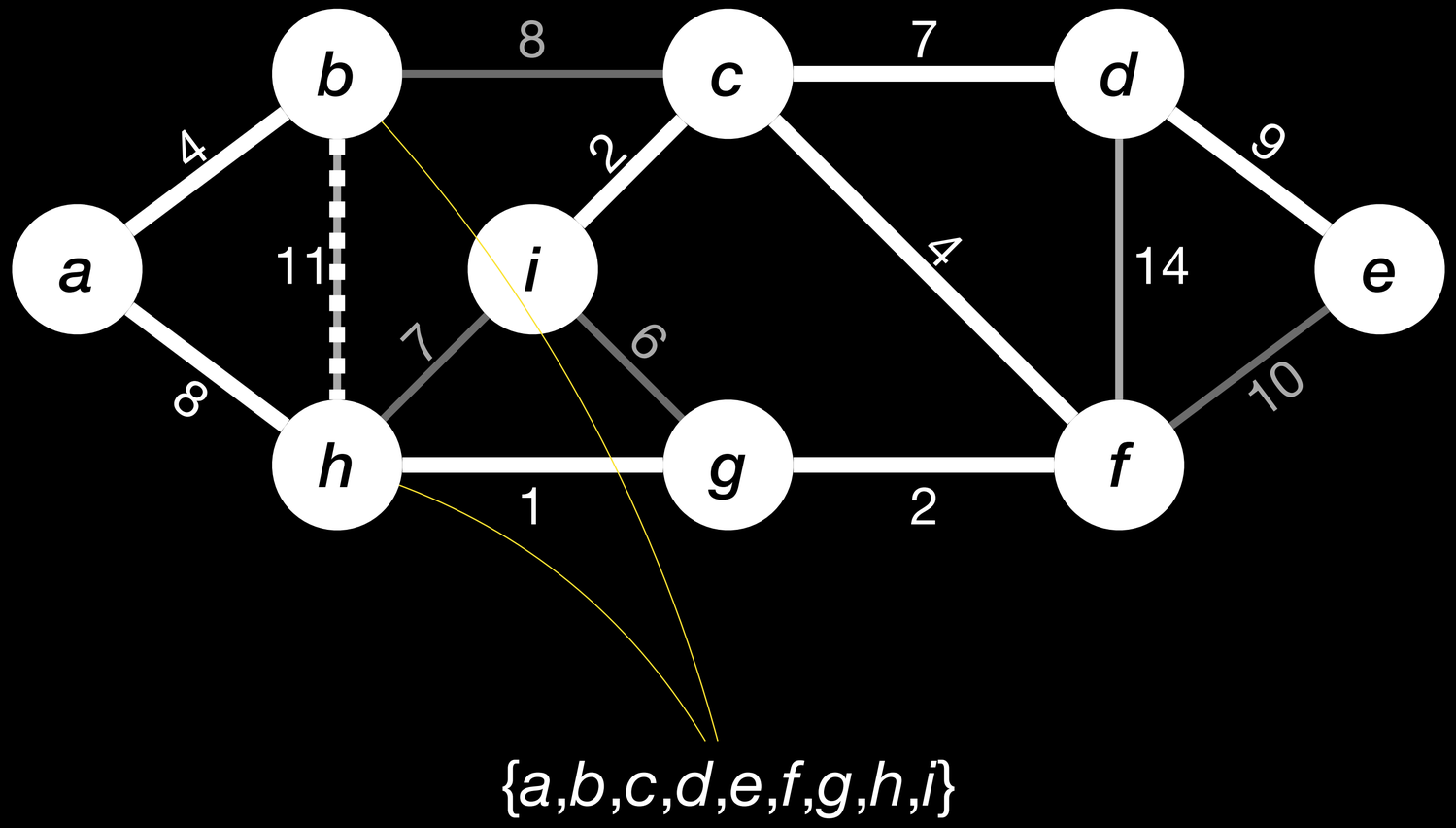


Example 例子



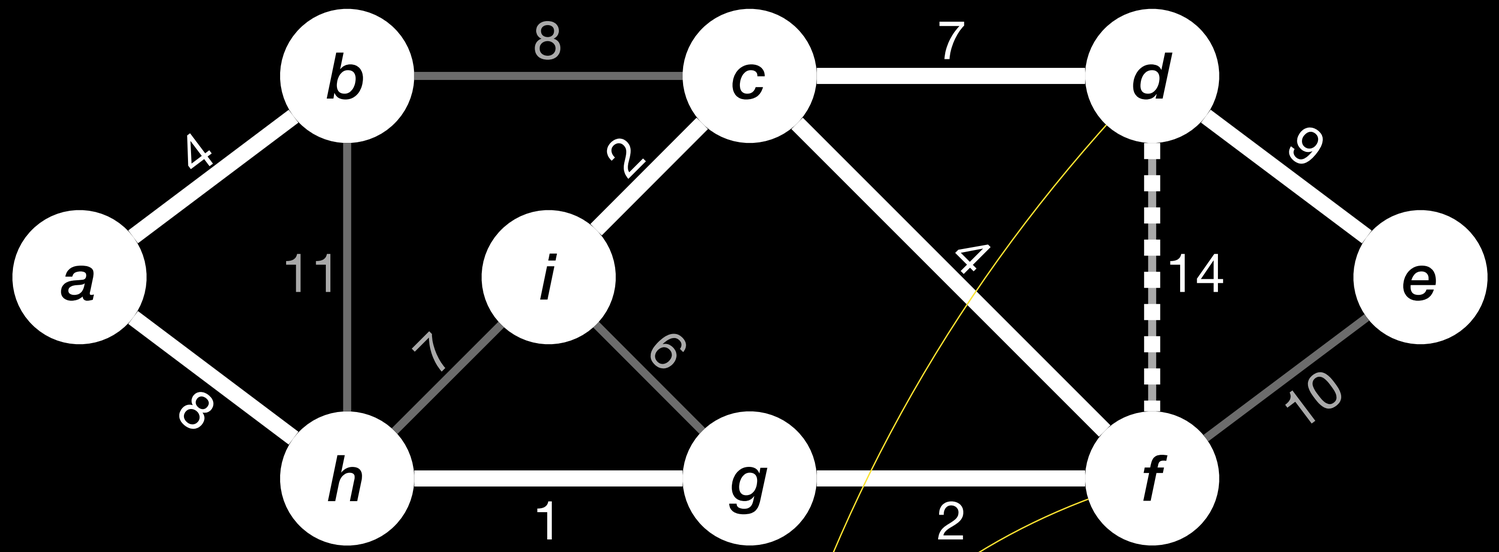
{*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*}

Example 例子



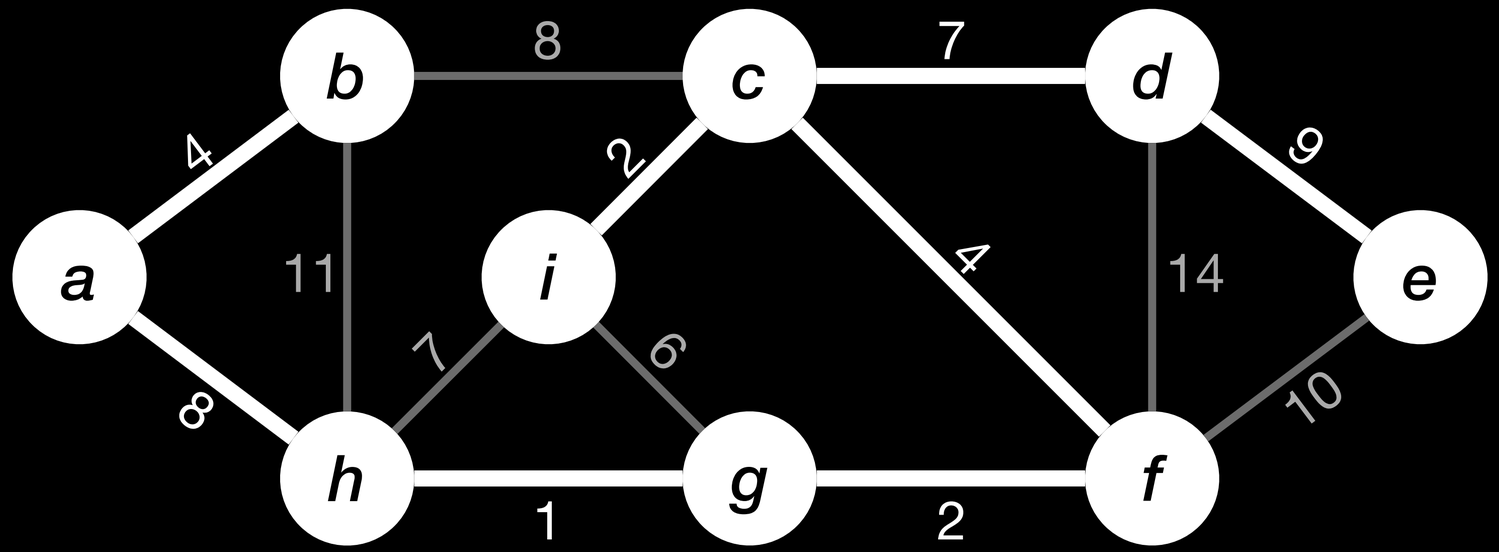
{*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*}

Example 例子



{*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*}

Example 例子



{*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*}

Partial Correctness

- immediate consequence of the partial correctness of the generic method.
- Find a cut that respects *A*:
  If edge endpoints are in different trees, let *S* = one of the trees.

  If edge endpoints are in the same tree, there is no cut that respects *A*.

部分正确性

- 通用方法部分正确性的直接后果。
- 找到符合以下条件的切口：
  如果边端点位于不同的树中，
  则设 *S* = 其中一棵树。

  如果边端点位于同一棵树中，
  没有一个切口是尊重*A*的。

Running Time

- Sorting edges takes time $O(|E| \log |E|)$.
  Note that $|E| \leq |V|^2$, so $\log |E| \leq 2 \log |V|$.
  We can also write:
  Sorting edges takes time $O(|E| \log |V|)$.

- We execute $O(|V| + |E|)$ disjoint set operations over the set V.
  Note that $|V| \leq |E|+1$ (if G is connected).
  Together, they take time in $O(|E| \log |V|)$.

- The total running time is in $O(|E| \log |V|)$.

运行时间

- 对边进行排序需要时间$O(|E| \log |E|)$。
  注意$|E| \leq |V|^2$，所以$\log |E| \leq 2 \log |V|$。
  我们还可以写：
  对边进行排序需要时间$O(|E| \log |V|)$。

- 我们在集合V上执行$O(|V| + |E|)$不相交集合运算。
  注意$|V| \leq |E|+$（如果G已连接）。
  它们一起在$O(|E| \log |V|)$中花费时间。

- 总运行时间以$O(|E| \log |V|)$为单位。

Prim's Algorithm

- Idea:
  Only keep one (rooted) tree.
  Always extend the tree
  by the lightest edge that is allowed.
- Allowed are edges
  that do not form a cycle.
- To find the lightest edge,
  use a priority queue.

Prim的算法

- 想法：
  只保留一棵（有根的）树。
  始终延伸树
  在允许的最轻边缘。
- 允许有边
  这不会形成一个循环。
- 寻找最轻的边缘，
  使用优先队列。

Priority Queue

- Wu Zhilin taught this topic on September 15/16.
- A min-priority queue has the operations:
  - INSERT(Q,x) inserts the element x into the queue Q
  - EXTRACT-MIN(Q) returns an element of Q with minimal key
  - DECREASE-KEY(Q,x,k) decreases the value of x's key to the new value k

优先队列

- 吴志志林老师在9月15、16日讲授了这个主题。
- 最小优先级队列具有以下操作：
  - INSERT(Q,x) 插入元素x进入队列Q
  - EXTRACT-MIN(Q) 返回一个元素具有最小密钥的Q
  - DECREASE-KEY(Q,x,k) 将x的关键点的值减少为新值k

Prim's Algorithm　　Prim的算法

```
MST-PRIM(G,w,r)
Q = ∅
for each vertex u ∈ G.V \ {r}
        u.key = ∞, INSERT(Q,u)
        u.π = NIL
r.key = 0, INSERT(Q,r)
r.π = NIL
while Q ≠ ∅
    {   u = EXTRACT-MIN(Q)                    }
        for each v ∈ G.V adjacent to u
            if v ∈ Q and w(u,v) < v.key
                v.π = u
                DECREASE-KEY(Q,v,w(u,v))
```

initialize priority queue

预置优先队列

ensure that the tree starts with *r*

确保树以*r*开头

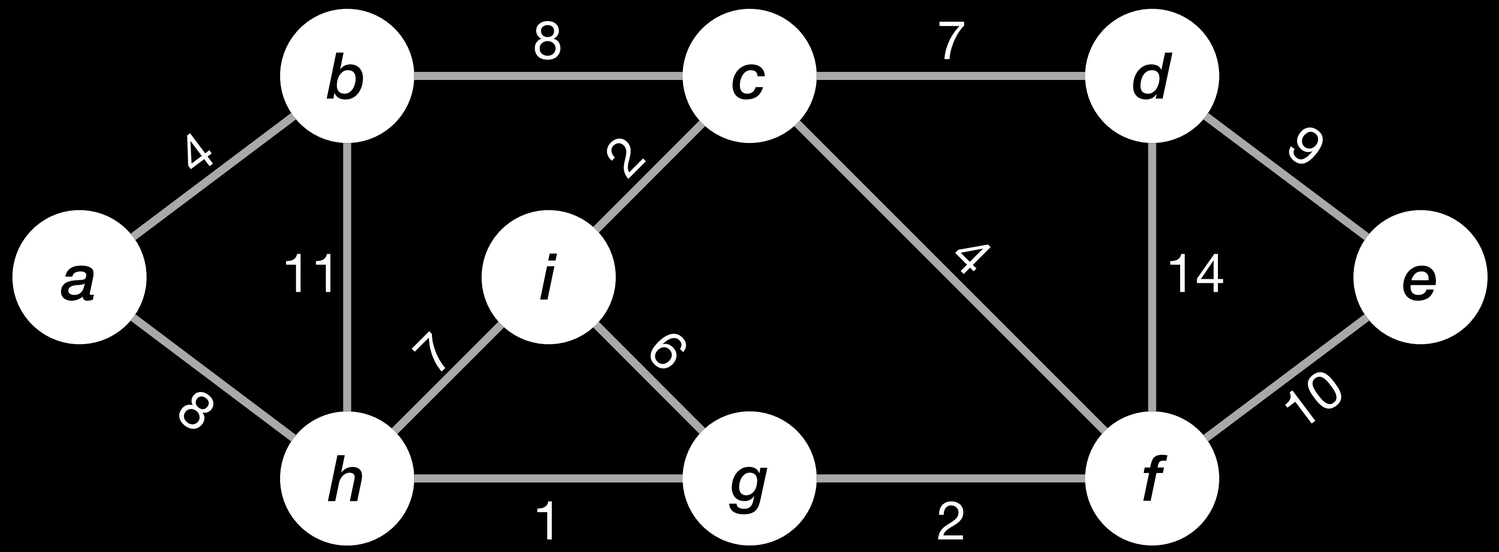
add *u* to the tree

把*u*添加到树中

correct keys of vertices adjacent to *u*

更正与*u*相邻顶点的关键

Example 例子



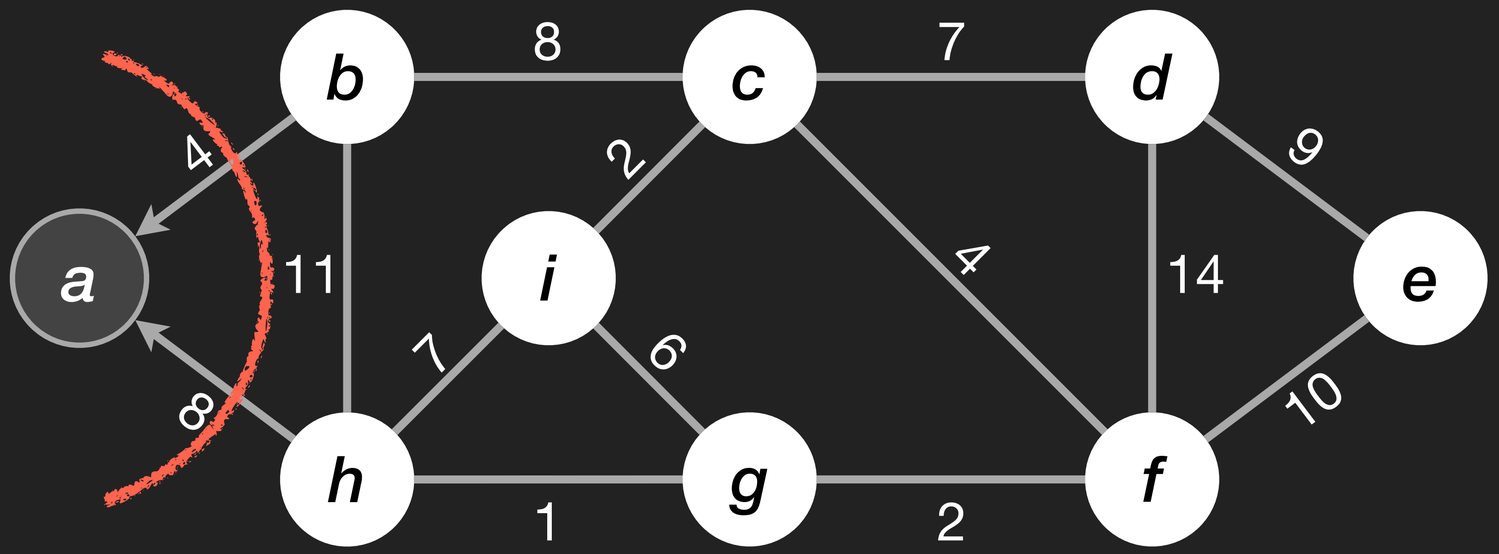
0

a

∞

b c d e f g h i

Example 例子

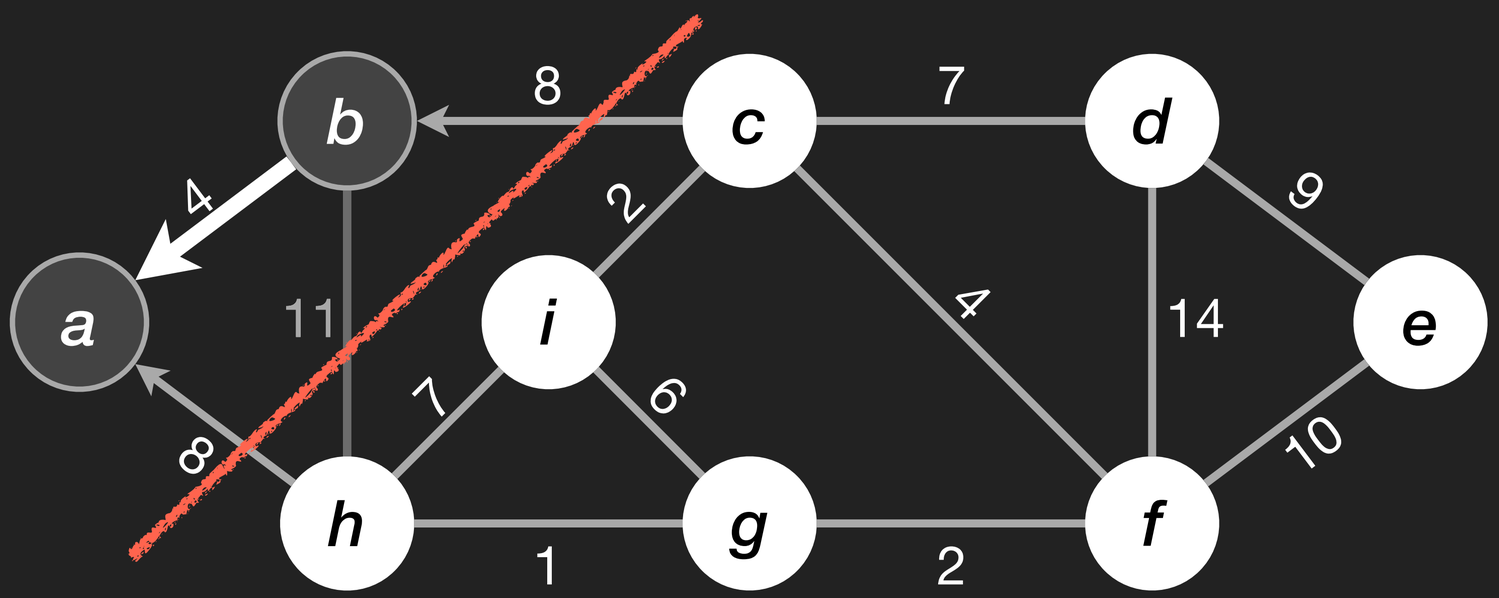


4
b

8
h

∞
c d e f g i

Example 例子



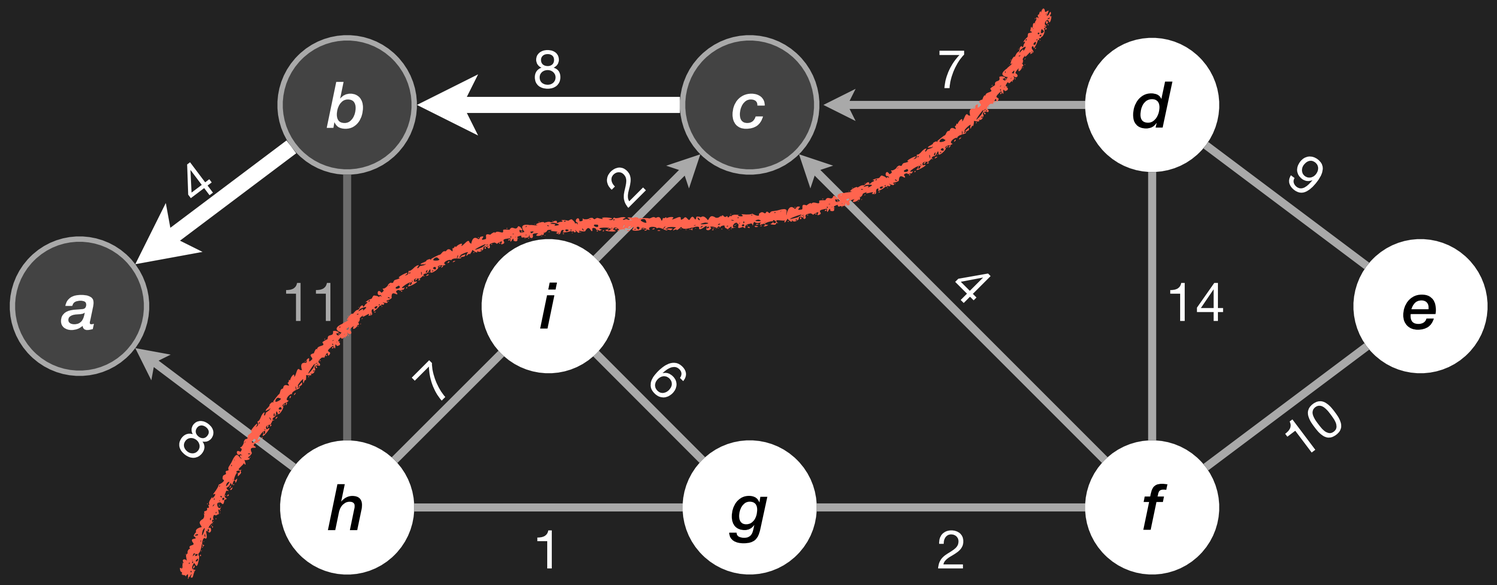
8

c h

∞

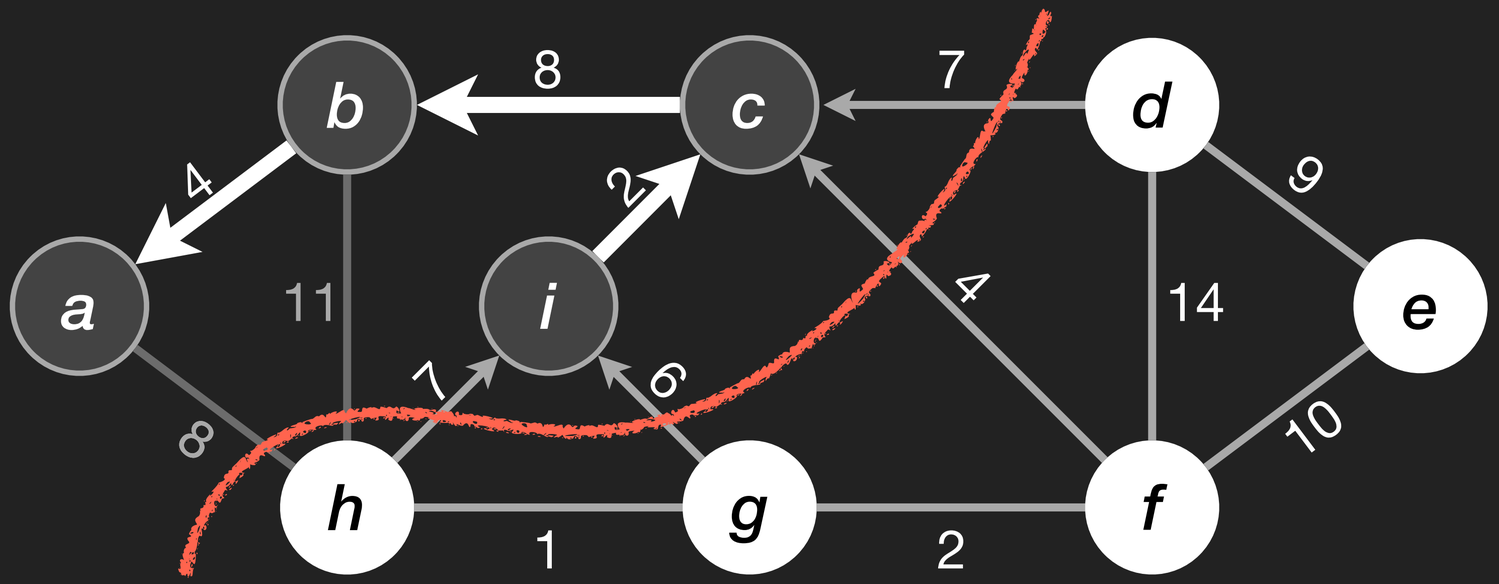
d e f g i

Example 例子



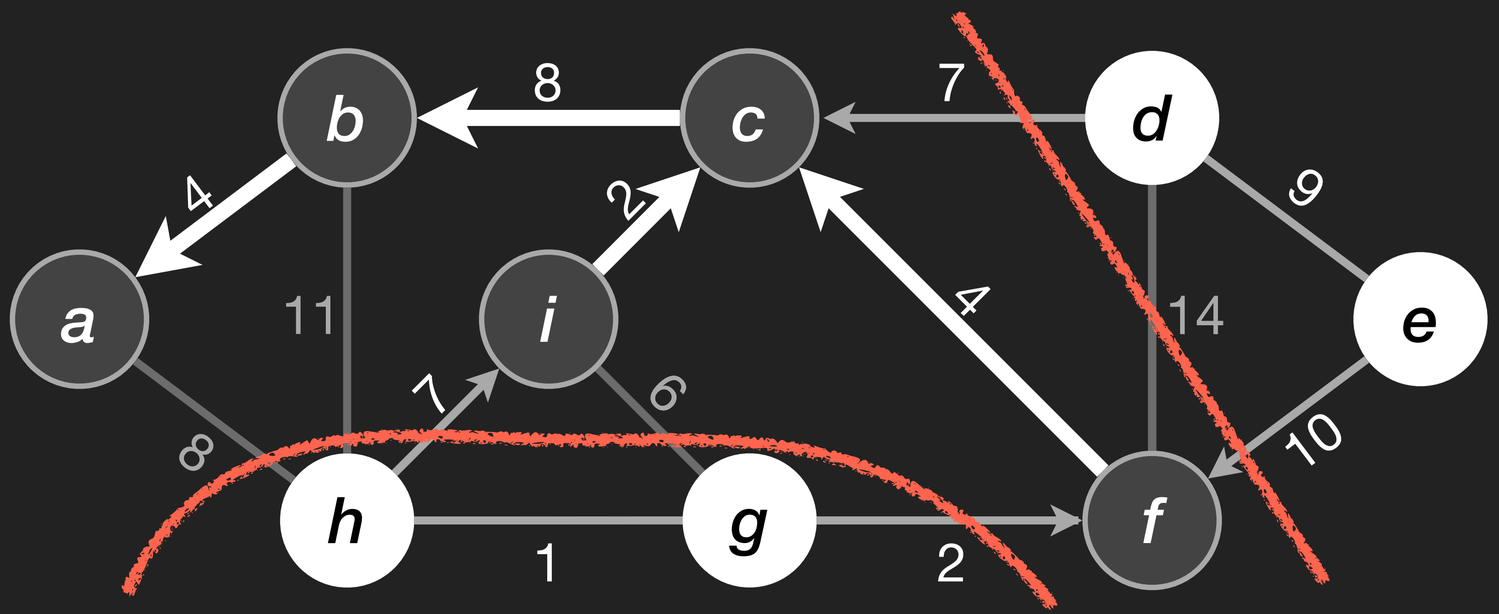
2	**4**	**7**	**8**	**∞**
i	*f*	*d*	*h*	*e* *g*

Example 例子



4		**6**	**7**	**∞**
f		*g*	*d* *h*	*e*

Example 例子

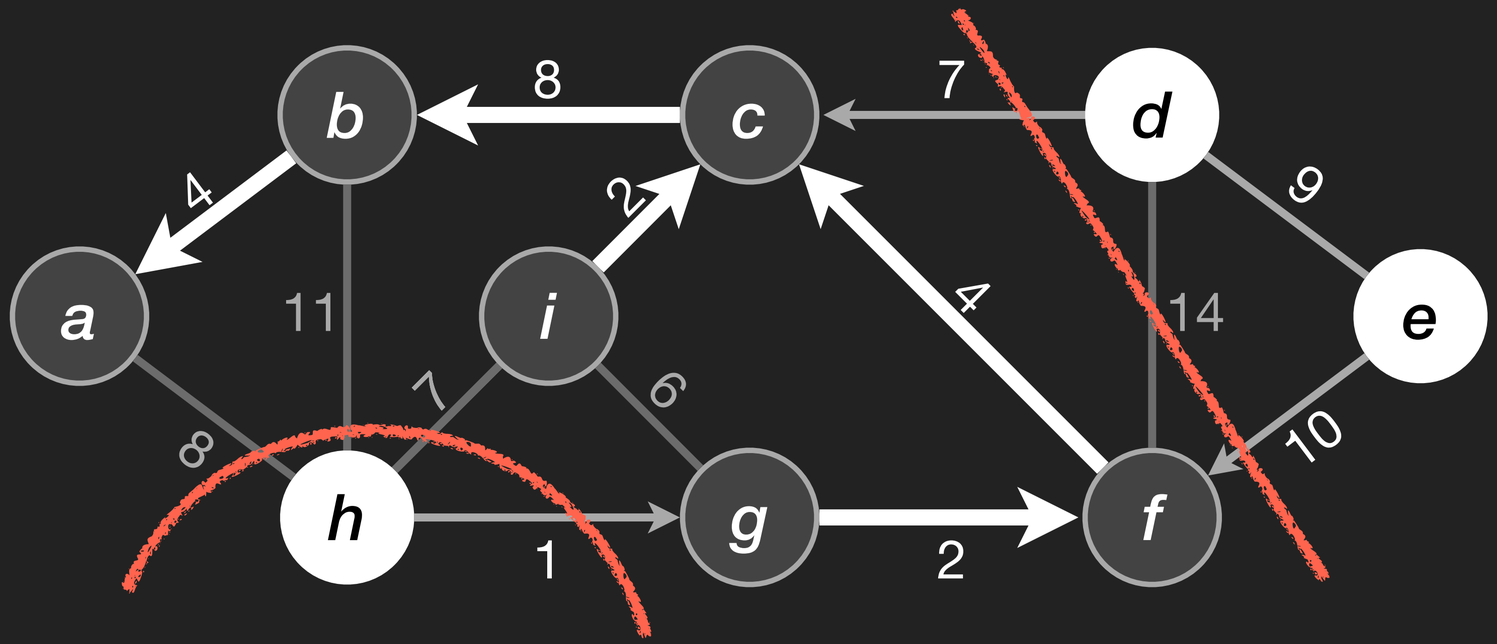


2
g

7
d *h*

10
e

Example 例子

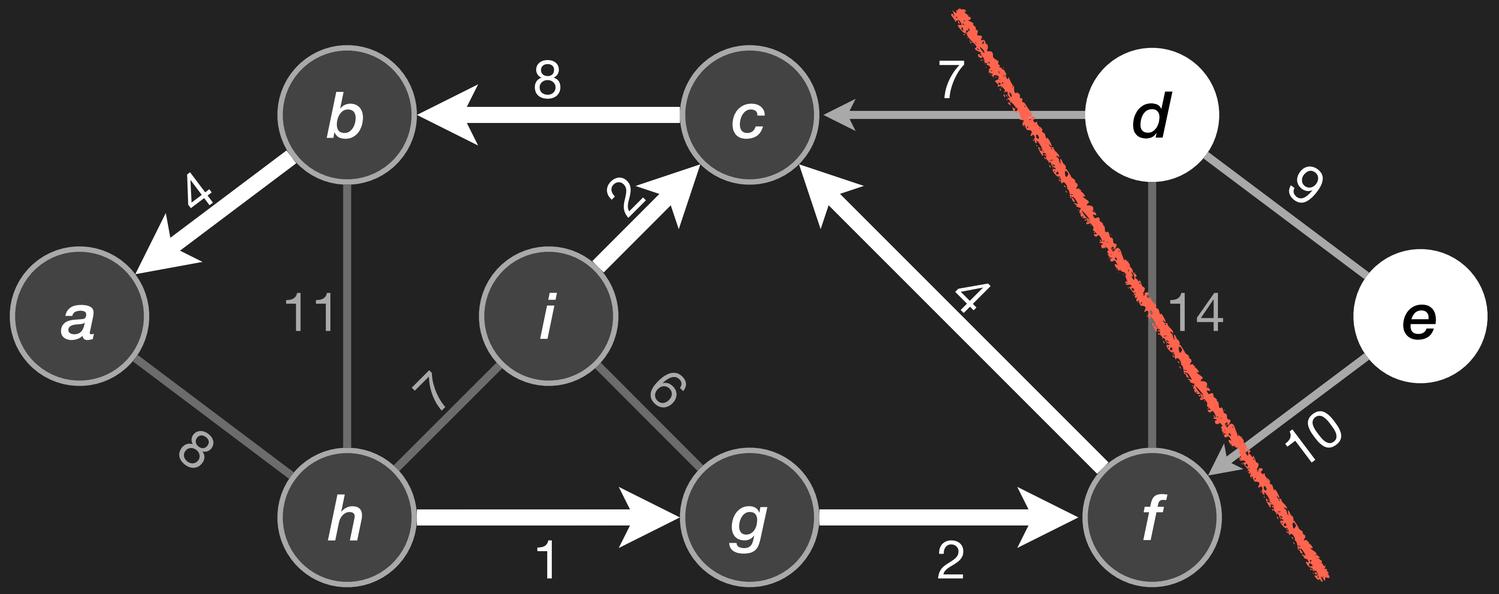


1
h

7
d

10
e

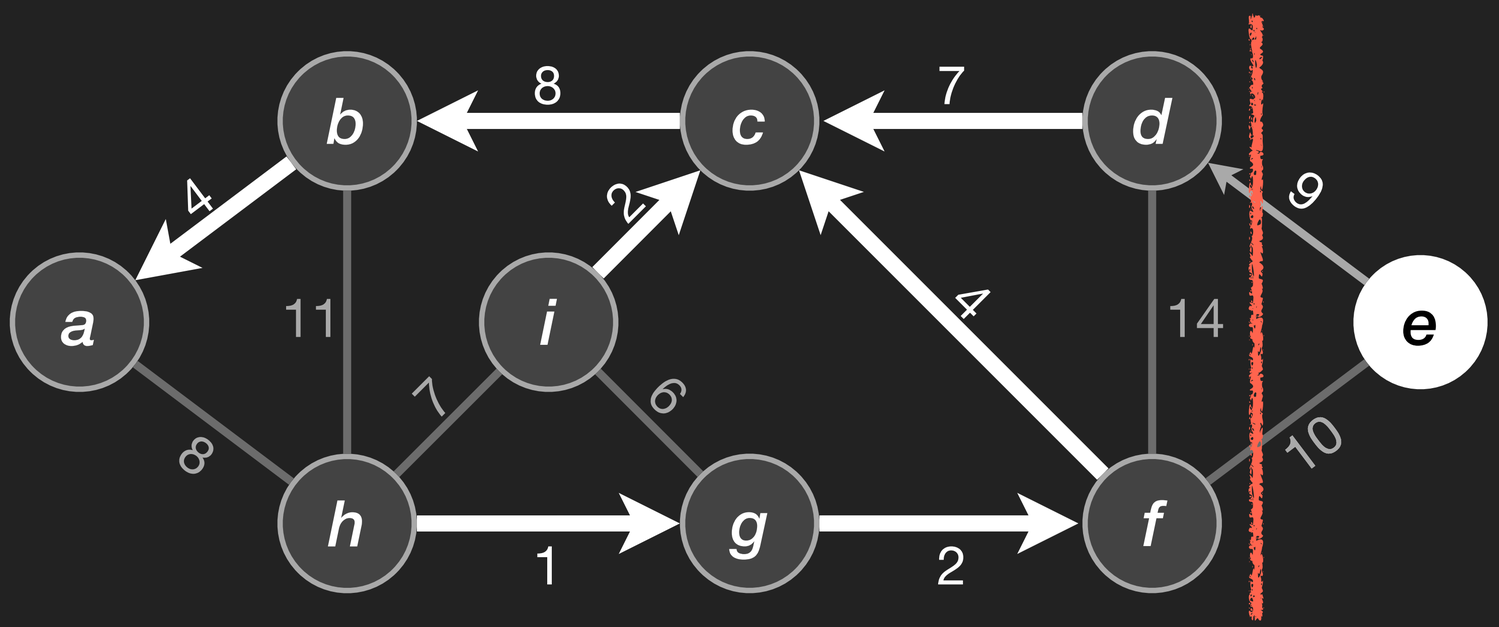
Example 例子



7
d

10
e

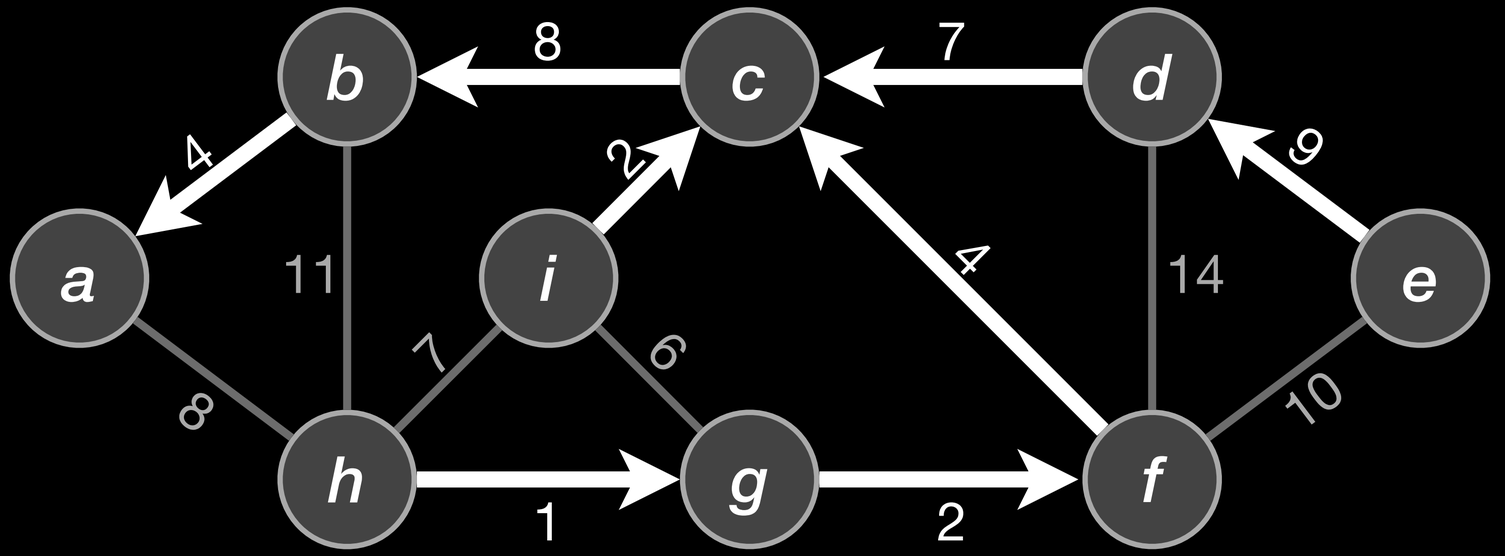
Example 例子



9

e

Example 例子



$Q = \varnothing$

Partial Correctness

- immediate consequence of the partial correctness of the generic method.
- The (partial) tree consists of
  the vertices $V \setminus Q$ and
  the edges $(v, v.\pi)$ for $v \in V \setminus Q$, $v \neq r$.
- Find a cut that respects A:
  S = the vertices that are in the tree
  $= V \setminus Q$.

部分正确性

- 通用方法部分正确性的直接后果。
- （部分）树包括
  顶点$V \setminus Q$和
  的 $(v, v.\pi)$ for $v \in V \setminus Q$，$v \neq r$
- 找到符合以下条件的切口：
  S = 树中的顶点
  $= V \setminus Q$。

Running Time

- Assume that the priority queue uses a min-heap.
- The initialization of the (very simple) priority queue takes time in $O(|V|)$.
- We execute $|V|$ times EXTRACT-MIN, which take total time $O(|V| \log |V|)$.
- We execute $O(|E|)$ times DECREASE-KEY, which take total time $O(|E| \log |V|)$.
- The total running time is in $O(|E| \log |V|)$.

运行时间

- 假设优先级队列使用最小堆。
- （非常简单）优先级队列的初始化需要$O(|V|)$的时间。
- 我们执行 $|V|$ 次EXTRACT-MIN，它占用总时间$O(|V| \log |V|)$。
- 我们执行 $O(|E|)$ 次DECREASE-KEY，总时间为$O(|E| \log |V|)$。
- 总运行时间以$O(|E| \log |V|)$为单位。

Quiz

5. What kind of graph should be stored using an adjacency matrix?

6. What data structure does breadth-first search use to get the correct order of examination?

测验

5. 应该使用邻接矩阵存储哪种图?

6. 广度优先搜索使用什么数据结构来获得正确的检查顺序?

Quiz

7. Why do we calculate a BFS tree but a DFS forest?
8. What is the relation between topological sorting of graphs and topological sorting of partial orders?
9. How did we calculate the strongly connected components of a graph?
10. Explain the concept of "cut respecting a set of edges" and its relation to MSTs.

测检

7. 为什么BFS计算一棵树但是DFS计算森林（多棵树）？
8. 图的拓扑排序与偏序的拓扑排序有什么关系?
9. 我们怎么样计算图的强连通分量?
10. 解释什么是"切割尊重边的集合"和它根最小生成树的关系。