第三章 图

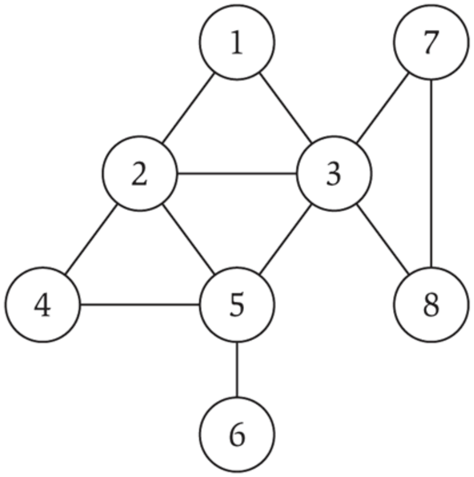
苏 明

3.1 基本定义与应用

- 无向图 G = (V, E)
  - V = 顶点
  - E = 边，反映顶点之间的关系
  - 图参数：n = |V|, m = |E|.

3.1 基本定义与应用

- 例子



V = { 1, 2, 3, 4, 5, 6, 7, 8 }

E = { 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 }

n = 8

m = 11

3.1 基本定义与应用

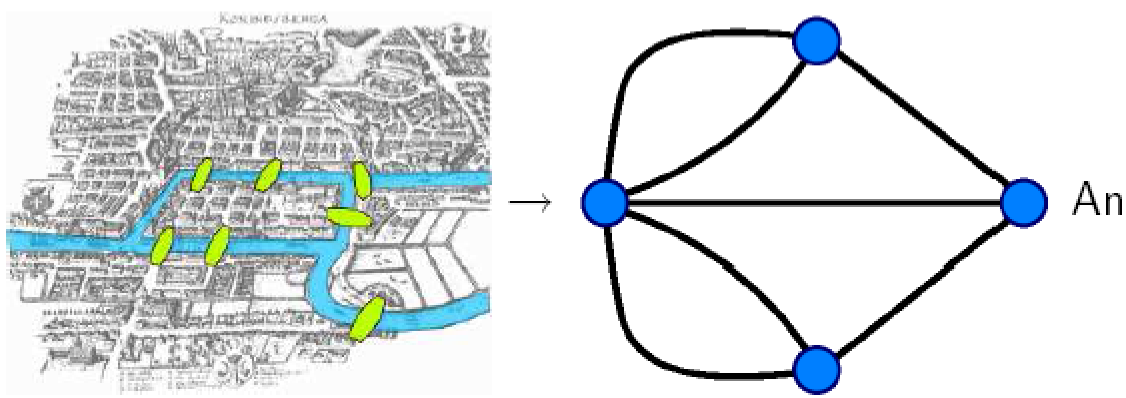
- 应用

Graph	Nodes	Edges
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires

3.1 基本定义与应用

- 图的起源

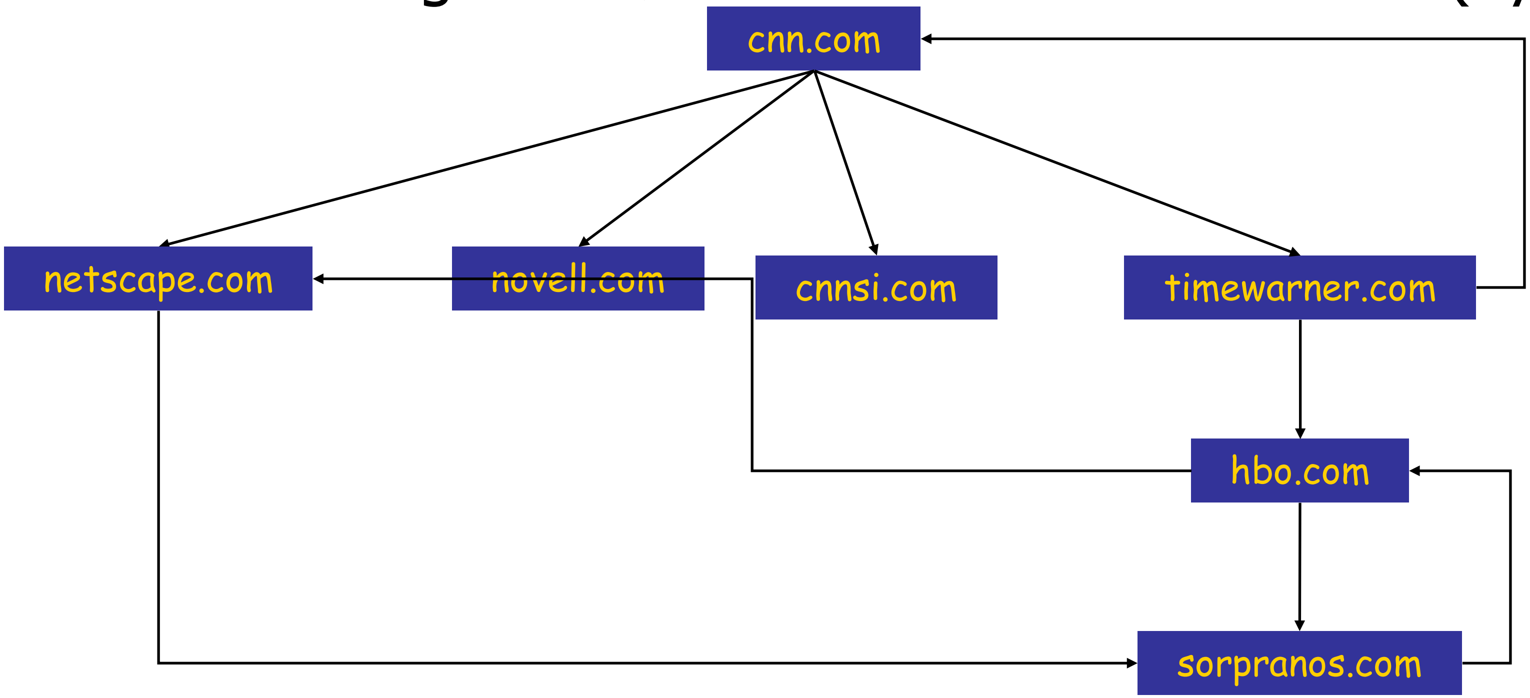
In 1736, Leonhard Euler solved the Seven bridges of Königsberg



Euler path exists if and only if the graph is connected and has 0 or 2 vertices with odd degrees.

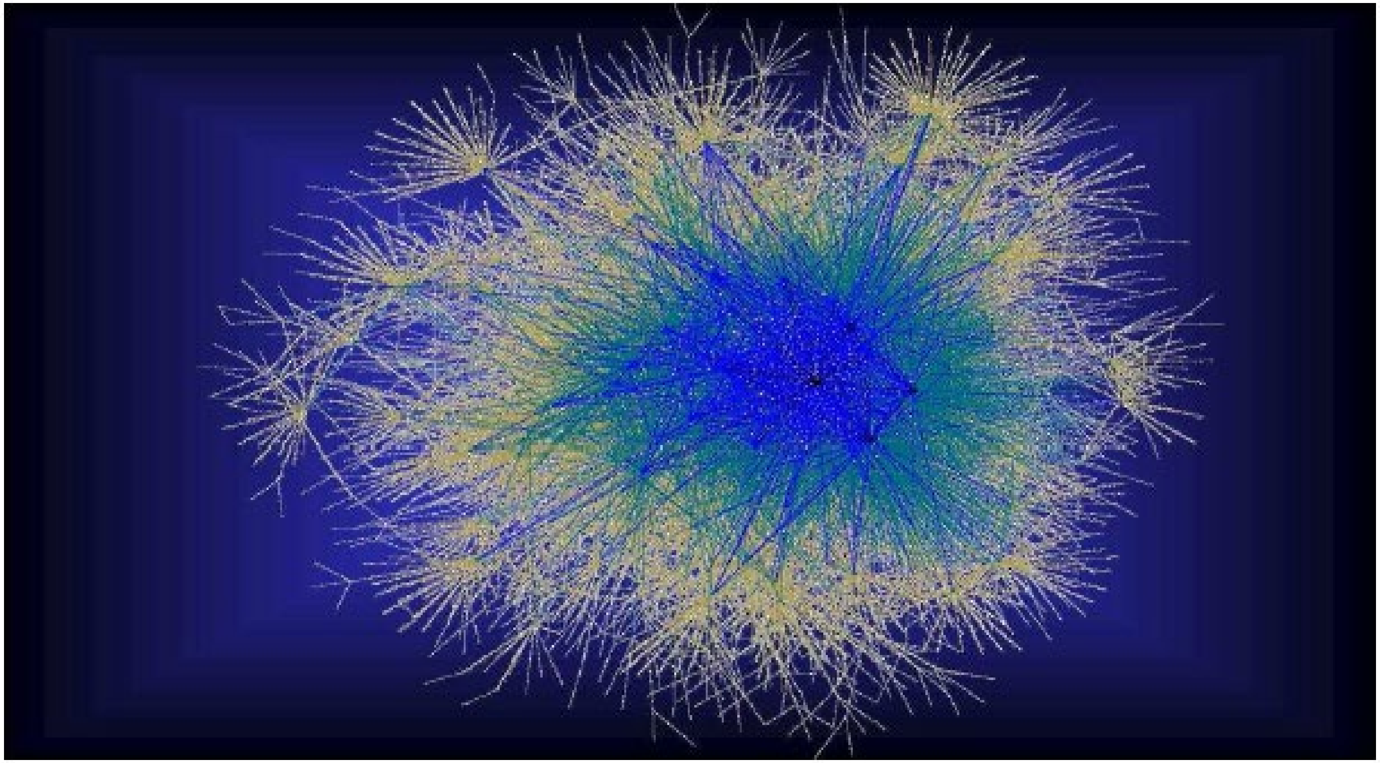
3.1 基本定义与应用

- Web graph.
  - Node: 网页
  - Edge: 一个网页到另外网页的超级链接(hyperlink).



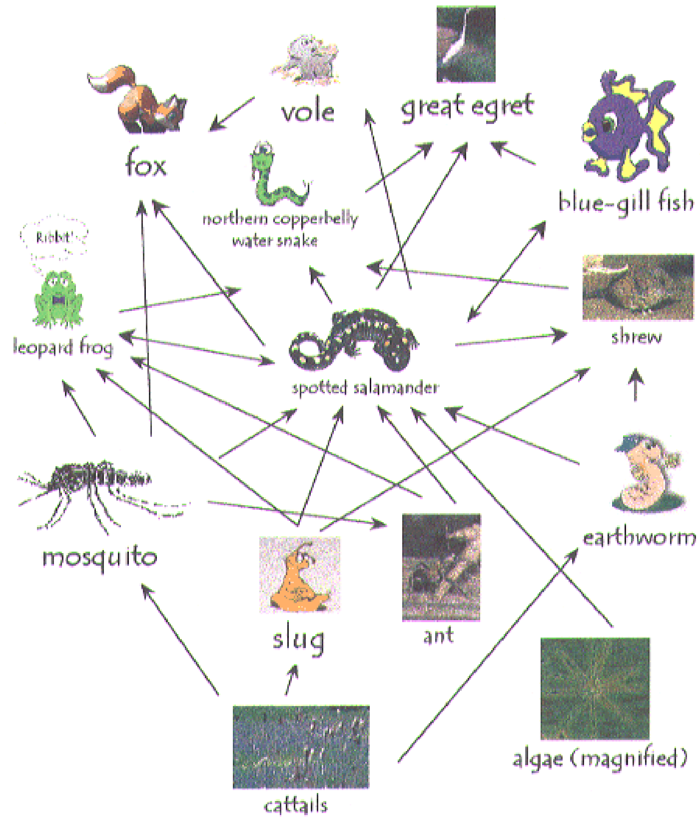
3.1 基本定义与应用

- 复杂图的例子：如下是6400个结点，13000条边的图的一部分

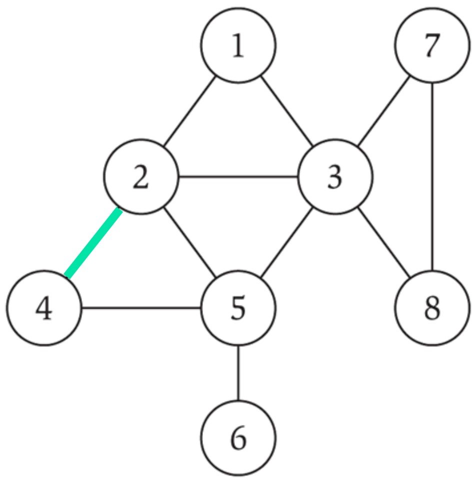


3.1 基本定义与应用

- 食物生态链图
  - ✓ Node = 生物种类.
  - ✓ Edge = 从猎物到捕食者



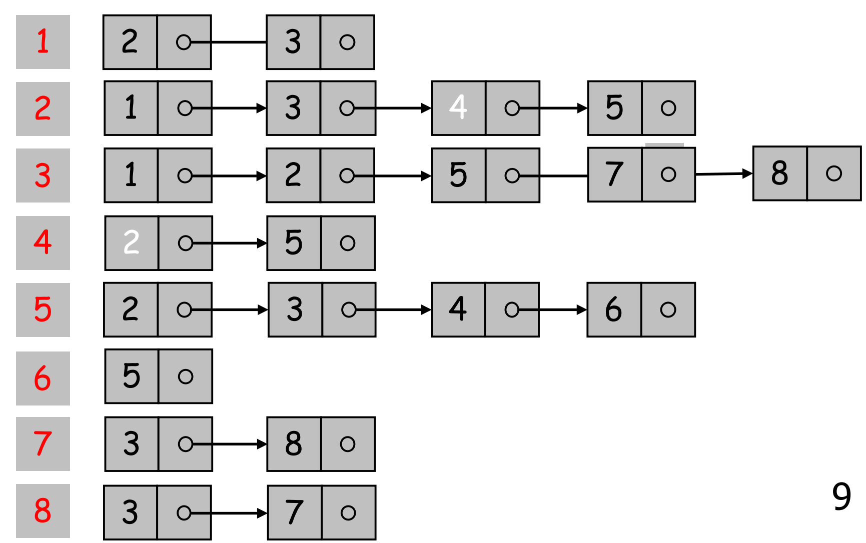
图的表示



邻接矩阵

	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

邻接链表



路径与连通性

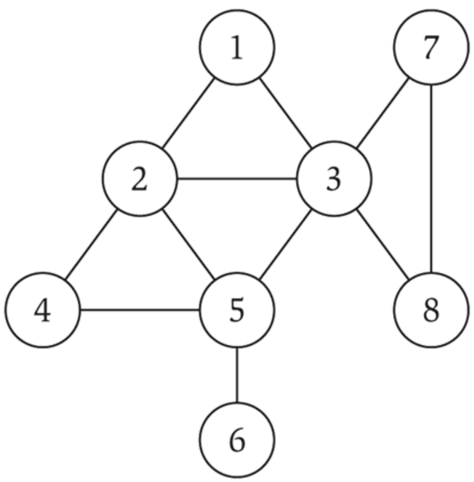
- Def. 无向图 G = (V, E) 的一条路径P定义为如下的结点序列 $v_1, v_2, \ldots, v_{k-1}, v_k$, 其中 v_i, v_{i+1} 是 E中的一条边. P被称为从v_1到v_k的路径。

- Def. 如果一条路径所有的结点都是相互不同的，就称为简单的。

- Def. 如果对于无向图中任意两个顶点u,v, 都存在一条路径，那么无向图被称为是连通的.

圈的定义

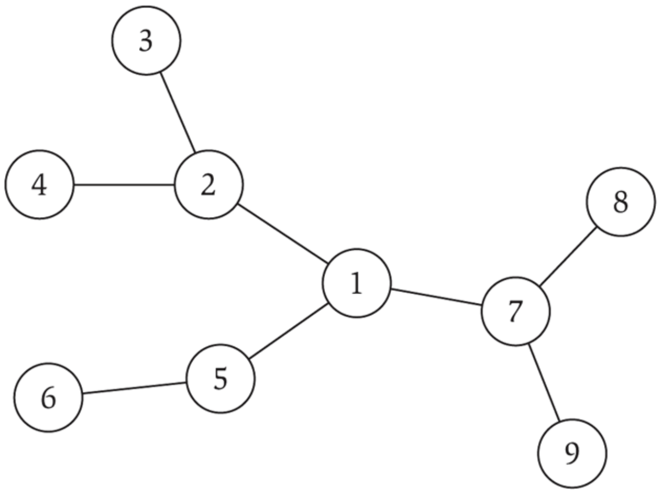
- Def. 路径 $v_1, v_2, \ldots, v_{k-1}, v_k$ 被称为一个圈(Cycle)，如果$v_1 = v_k$, $k > 2$，而且前k-1个顶点两两不同。



cycle C = 1-2-4-5-3-1

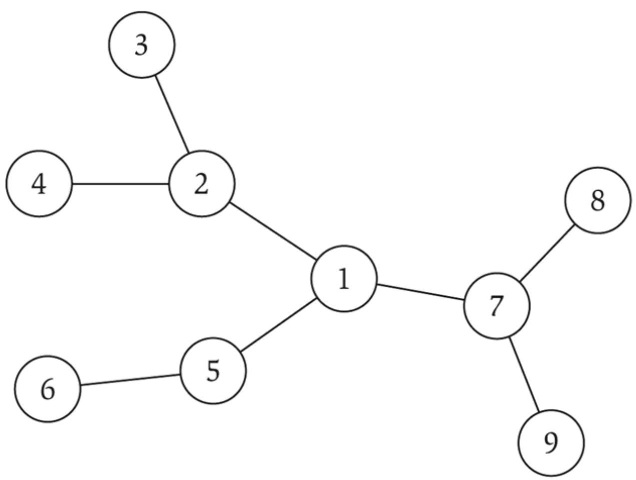
树的定义

- Def. 一个无向图被称为是树，如果它是连通的而且没有圈。
- 定理. 设 G 是一个具有n个结点的无向图。下面任意两条都可以推出第三条。
  - G 是连通的.
  - G 不包含一个圈.
  - G 有n-1条边.

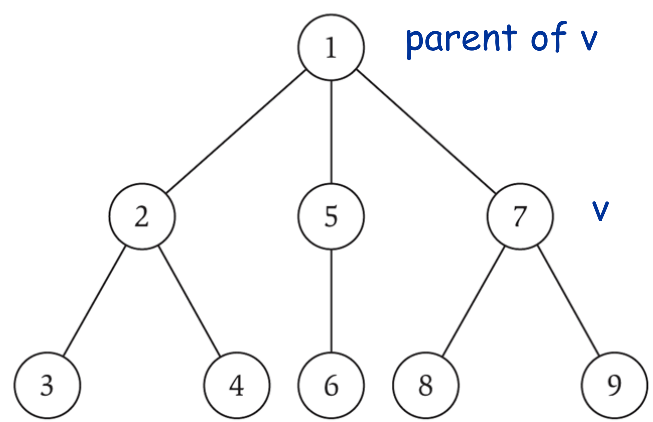


有根树

- 给一个树 T, 选择一个根节点 r， 安排所有的边从r出发.
- 目的是为了建立分层结构。



a tree



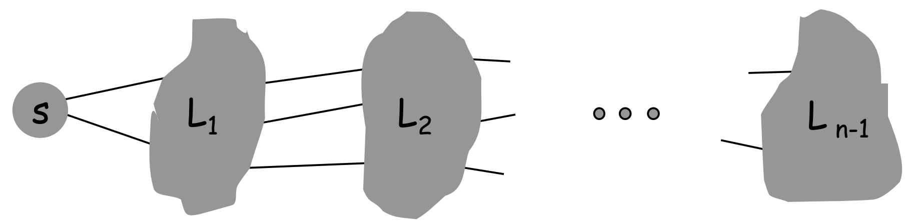
the same tree, rooted at 1

3.2 图的连通性与图的遍历

- s-t **连通性**问题. 给定图中两个顶点s, t, 是否存在一条s到t的路径?
- s-t **最短路径**问题. 给定图中两个顶点s, t, s-t之间的*最短路径*是多长?
- 一些应用场景.
  - 穿越迷宫
  - 通信网络中最少的多跳连接路径

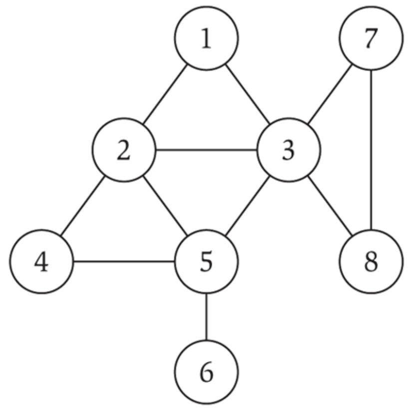
广度优先(Breadth-First Search)

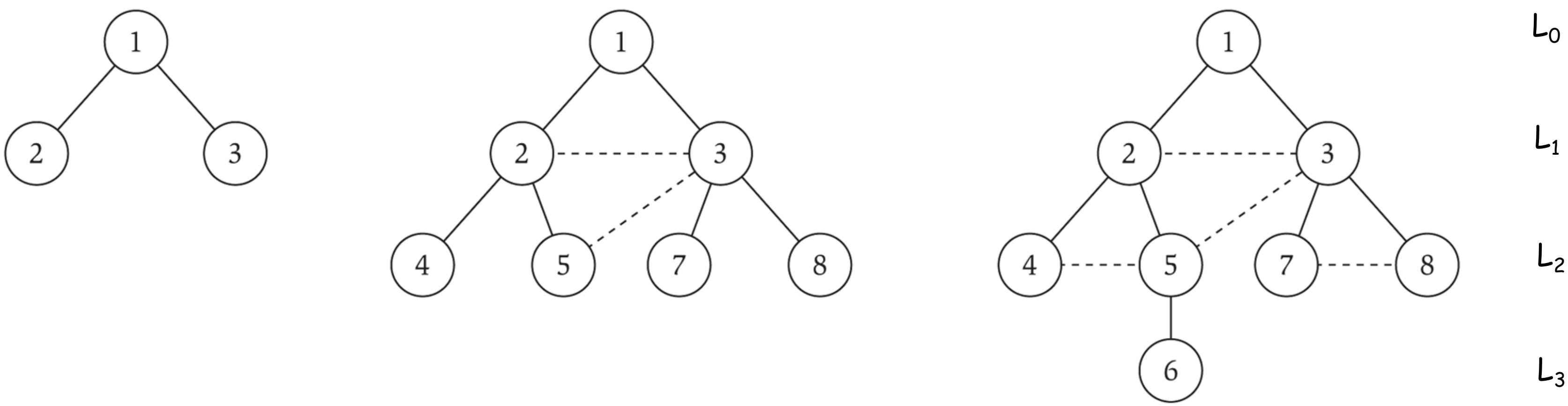
- BFS算法简介．从顶点 s开始向各个方向“探索”，把探索到的顶点依次加在各层上。



- BFS algorithm.
  - $L_0 = \{ s \}$.
  - L_1 = all neighbors of L_0.
  - L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
  - L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

BFS性质





(a) (b) (c)

BFS性质

```
BFS(s):
  置Discovered[s]=true, 其他v, 置Discovered[v]=false
  初始化L[0], 单个元素s构成
  置层计数器i=0
  置BFS树T=空集
  While L[i]非空
    初始化一个空表L[i+1]
    For 每个结点L[i]中结点u
      考虑每条关联到u的边(u,v)
      If Discovered[v]=false then
          置 Discoverd[v]=true
          把边(u,v)加到树T上
          把v加到表L[i+1]
      Endif
    Endfor
   层计数器加1
  Endwhile
```

BFS性质

- 定理3. 11 如果图是邻接表给出，BFS算法的上述实现将以O(m+n)时间运行。
- Pf. 考虑结点 u, 存在 deg(u) 条与u相连的边 (u, v)，所以处理所有边的时间是 $\Sigma_{u \in V}$ deg(u) = 2m；此外对于顶点, 还要O(n)的额外时间来管理数组Discovered.

BFS 性质

- 定理　BFS算法产生的层 L_i 就是到源点s距离为i的顶点集合．存在一条s到t的路径当且仅当 t 出现在某一层中.

- 定理 设T是图G = (V, E)的一棵宽度优先搜索树，(x, y)是G中的一条边.那么 x，y 所属的层数至多相差 1.

BFS & DFS

- BFS: 从 s开始出发探查，按照距离分层；
- DFS: 从s开始出发探查，方式更“冲动”；

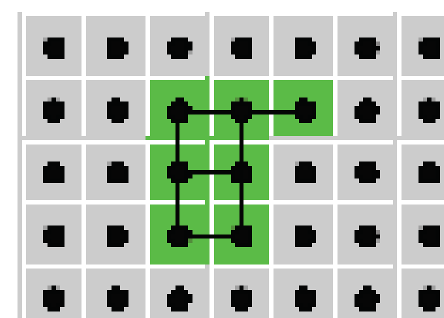
- BFS: 一般用**队列**来维护即将被处理的结点；
- DFS: 用**栈**来维护即将被处理的结点；

DFS

- DFS(s)
-   初始化S为具有一个元素s的栈
-   While S 非空
-     从S中取出一个节点u
-     If Explored[u]=false then
-       置Explored[u]=true
-       For每条与u关联的边(u,v)
-         把v加到栈S
-       Endfor
-     EndIf
-   EndWhile

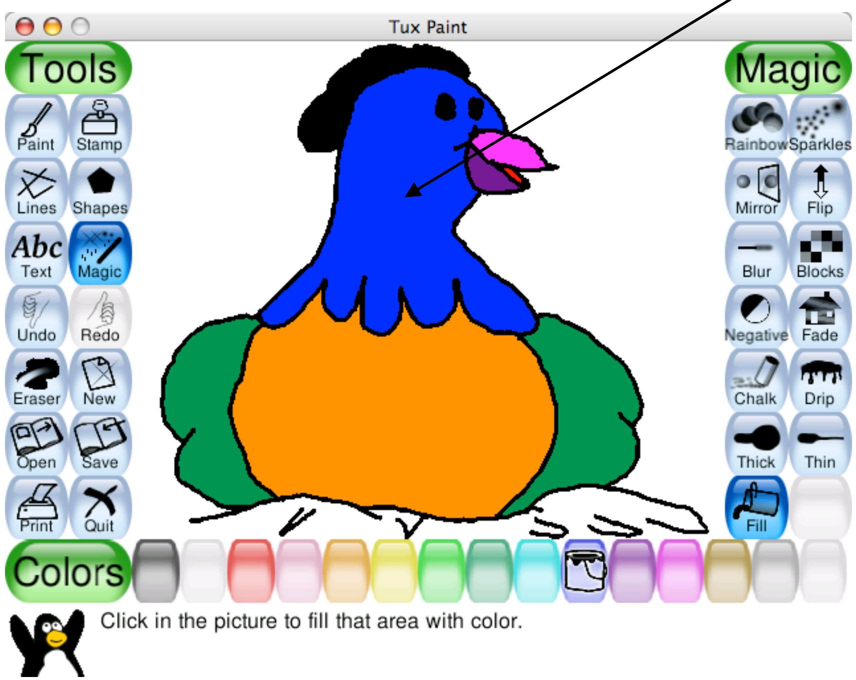
应用实例

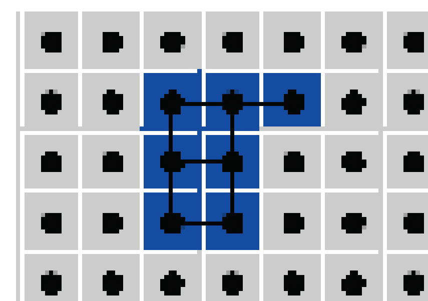
- 图形填充问题：亮绿色-→蓝色
- 顶点：像素点；边：是否都是亮绿色



应用实例

需要找到同种亮绿色的结构，然后改变颜色

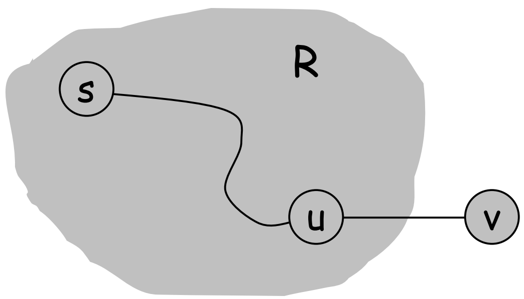




连通分支

- BFS算法发现的是从始点s可达的结点。把这个集合R看作G的包含s的连通分支。

```
R will consist of nodes to which s has a path
Initially R = {s}
While there is an edge (u, v) where u ∈ R and v ∉ R
  Add v to R
Endwhile
```



定理： 算法结束产生的集合R恰好是G的包含s的连通分支。

连通性质

- 定理3.8： 对**无向图**中任两个结点s与t,它们的连通分支或者相等，或者不相交 。

- 定理3.17 对**有向图**中的任何两个结点s与t,它们的***强连通***分支或者相等，或者不相交 。

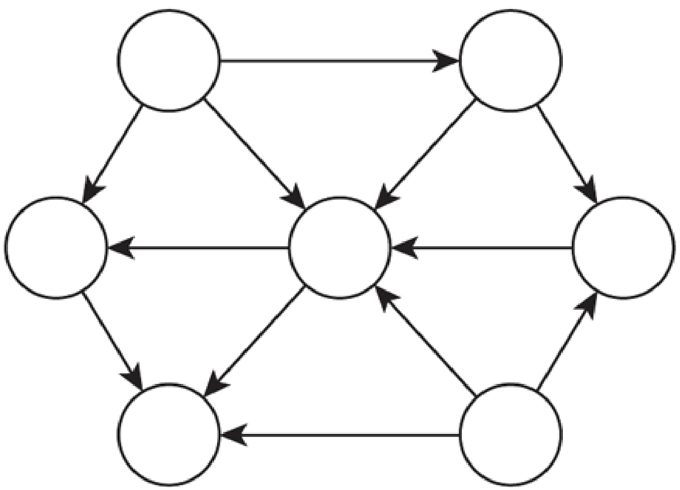
连通性质

- 如何找出所有连通分支的集合?

- 时间复杂度?

有向图中的连通性

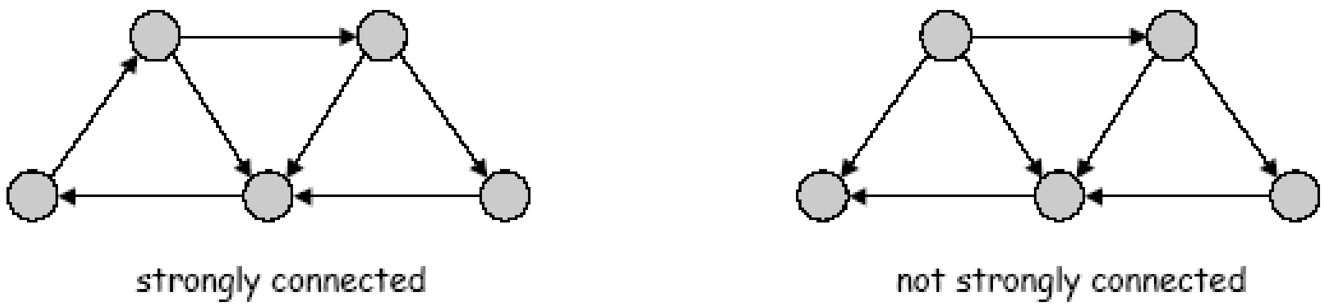
- 有向图(Directed graph) G = (V, E)
  - Edge (u, v)：从u到 v



类似的，无向图中的一些概念，问题在这里可以推广

强连通性

- Def. u和v是相互可达的，如果彼此之间存在到达对方的路径。
- Def. 如果图中每对结点是相互可达的，那么此图是强连通的。

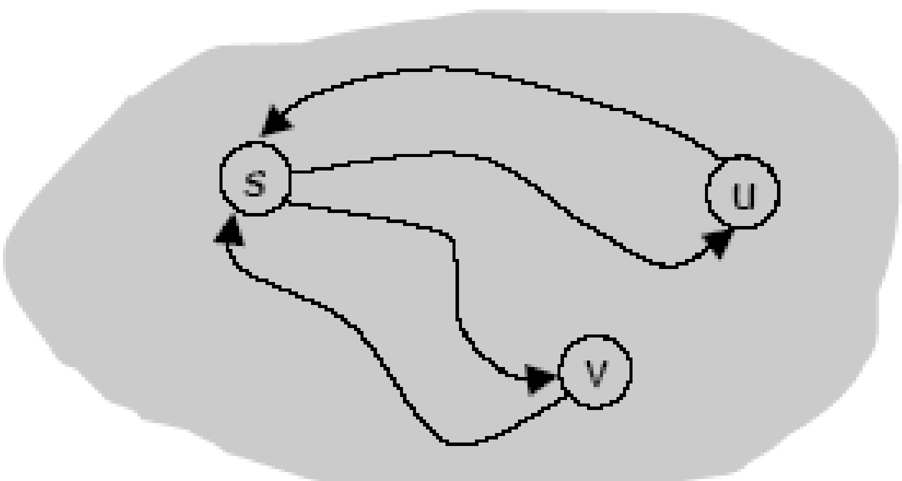


如何判断一个图是否强连通？所需时间代价？

强连通性

- 引理. 设s是图G中的任意一个结点。G是强连通的，当且仅当图中的每个结点能够与s相互可达。

Pf.



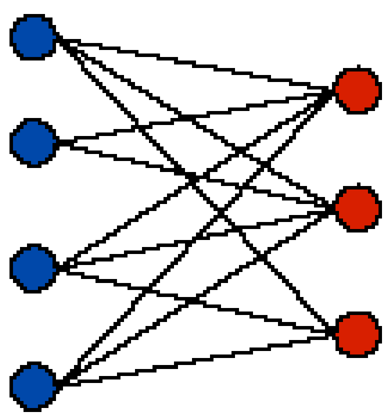
强连通性

- Pick any node s.
- Run BFS from s in G.
- Run BFS from s in G^{rev}.
- Return true iff all nodes reached in both BFS executions.

存在$O(m + n)$的有效算法判别图G是否强连通。

3.4 二分性测试

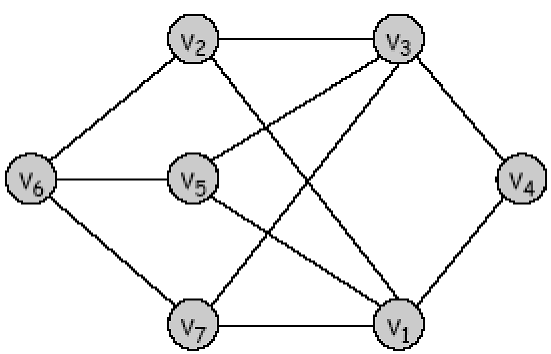
- 二部图是一个图，其中结点集可以划分为X与Y,每条边一端在X中，另一端在Y中。
- 直观的，一个图是二部图，如果能对结点着红色和蓝色，使得每条边有一个红端点和一个蓝端点。



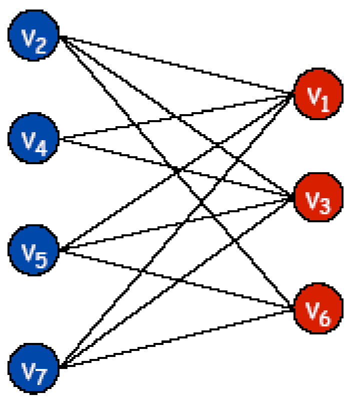
a bipartite graph

3.4 二分性测试

- 给一个图G,它是二部图吗?
- 很多有关图的问题，如果是二部图，那么问题会变得容易一些。



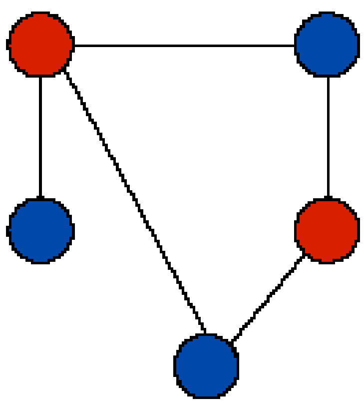
a bipartite graph G



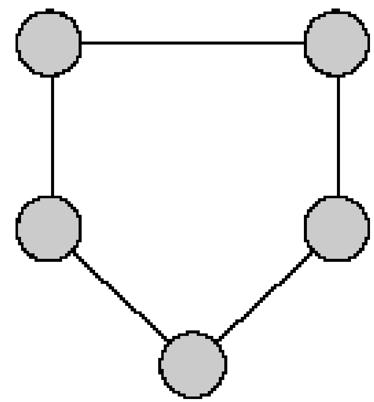
another drawing of G

二部图性质

- 定理 如果一个图是二部图，那么它不可能包含一个奇圈。



bipartite
(2-colorable)



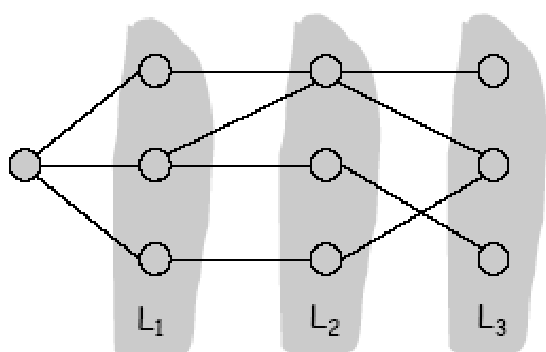
not bipartite
(not 2-colorable)

二部图性质

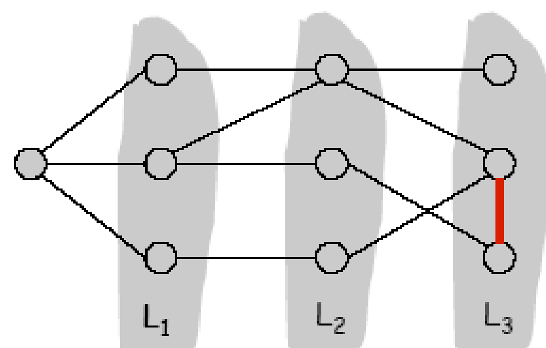
- 定理 设G是一个连通图， $L_0, \ldots, L_k$ 是从顶点s由BFS算法生成的层。那么下面两件事一定恰好成立其一：

i. **G中没有边与同一层的两个结点相交。**这种情况下G是二部图，其中偶数层的结点可以着红色，奇数层结点可以着蓝色。

ii. **G中有一条边与同一层的两个结点相交。**此种情形下，存在一个奇圈，不可能是二部图。



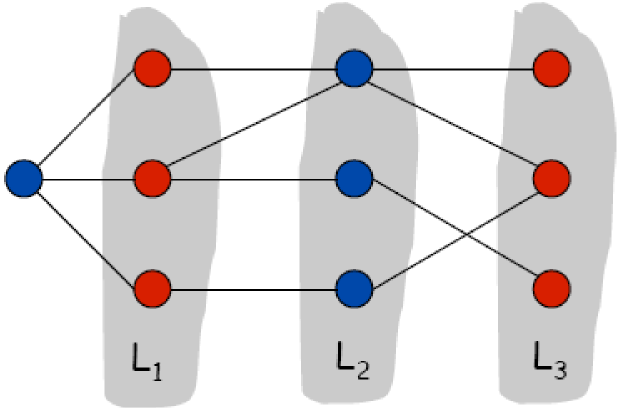
Case (i)



Case (ii)

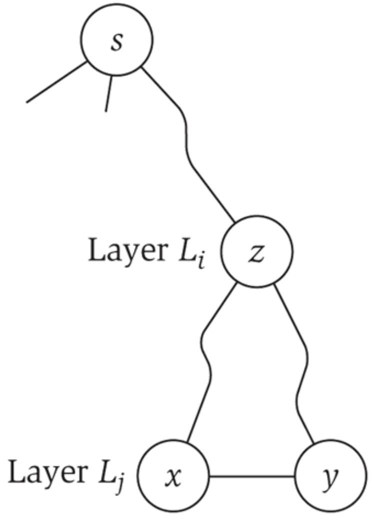
二部图性质

- Pf. (i)红蓝交替着色



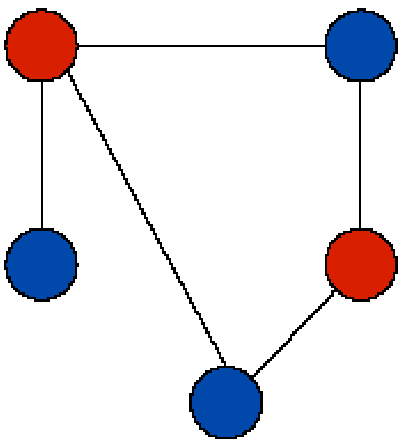
Case (i)

(ii)证明产生一个奇圈

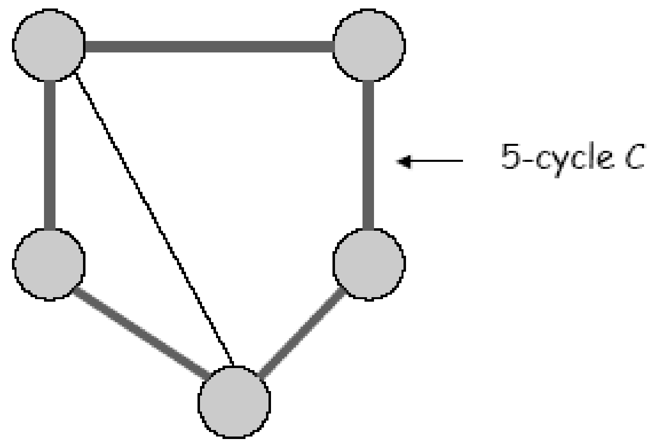


二部图性质

- 引理 图G是二部图**当且仅当**图中没有奇圈。



bipartite
(2-colorable)



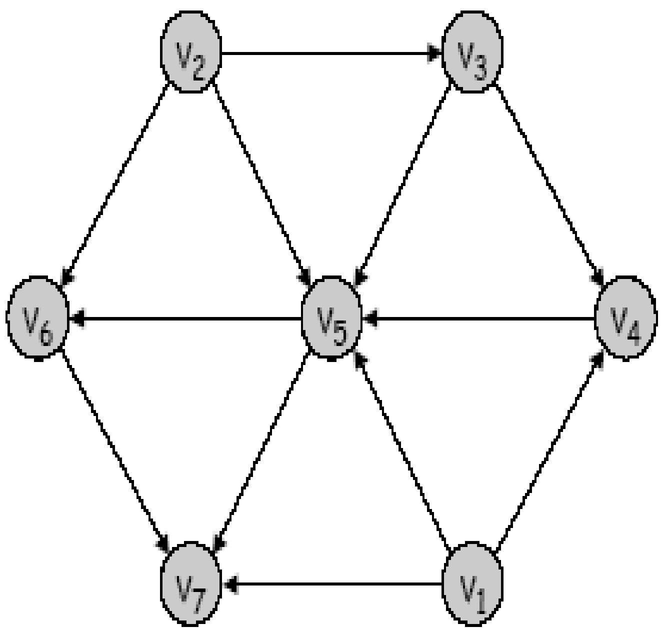
not bipartite
(not 2-colorable)

3.6 有向无圈图与拓扑排序

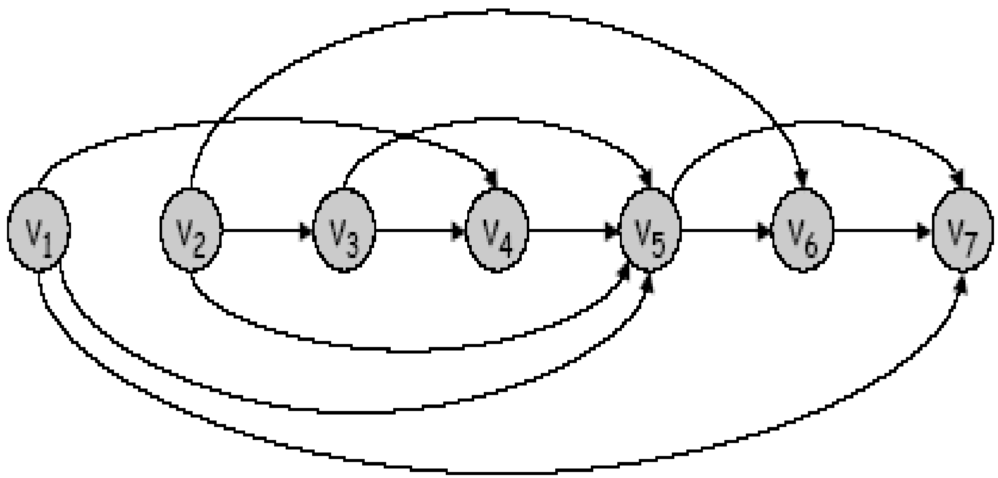
- 问题的背景：存在标记为{1,2,…,n}任务集，之间存在先后的依赖性，是否存在一个有效的执行任务的**次序**？

- 有向图G = (V, E)的一个拓扑排序是它的结点作为v_1, v_2, …, v_n的一个排序，**对每条边(v_i, v_j)，都有 $i < j$.**

拓扑排序



a DAG



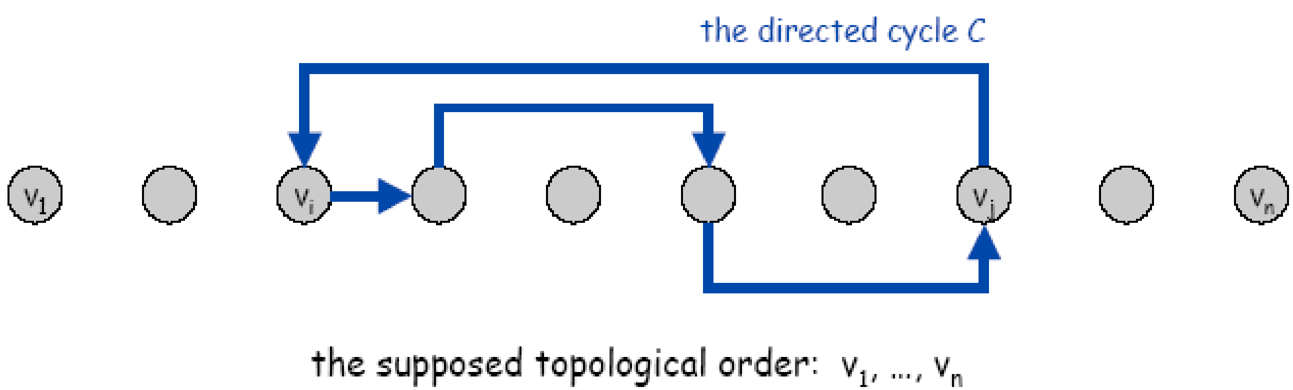
a topological ordering

有向无圈图与拓扑排序

- Def. DAG(Directed Acyclic Graphs):有向图没有圈

定理： 如果G有一个拓扑排序，那么G是一个DAG.

Pf.



DAG

- Q1 是否每一个DAG都存在拓扑排序?

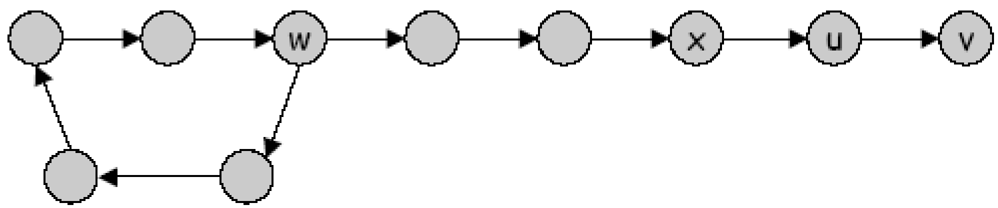
- Q2 如果存在，如何计算?

DAG

万事开头难：第一个结点

- 命题： 在每一个DAG中，存在一个没有输入边的结点。

Pf.



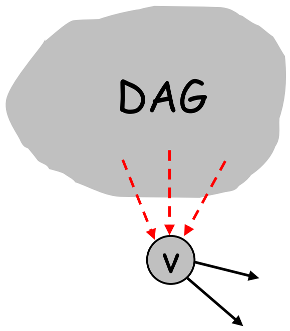
定理 如果G是一个DAG, 那么G有一个拓扑排序。

Pf. 归纳证明

DAG

- 算法

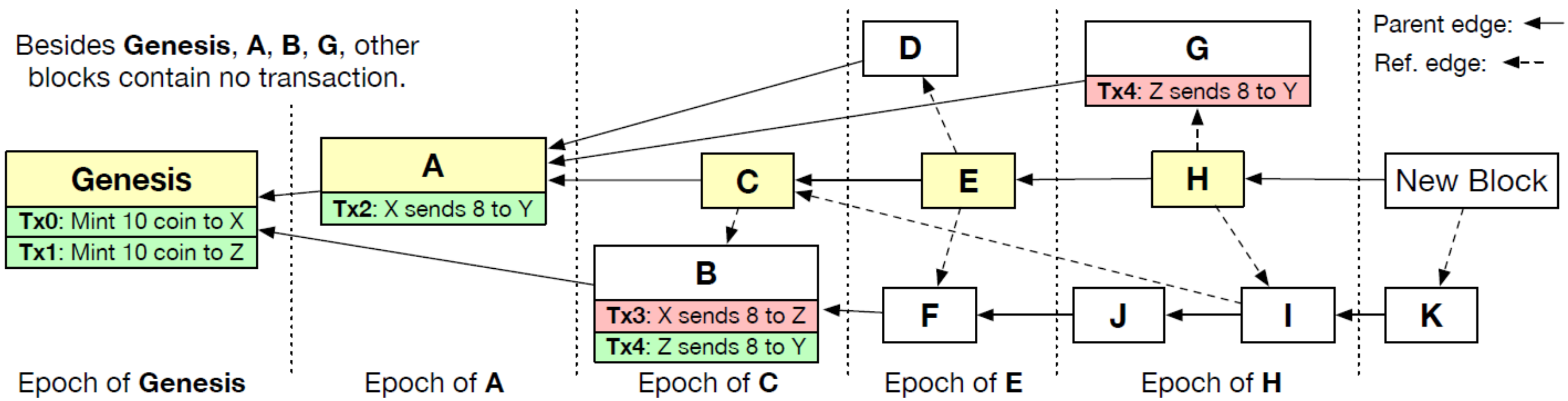
To compute a topological ordering of G:
Find a node v with no incoming edges and order it first
Delete v from G
Recursively compute a topological ordering of $G-\{v\}$
  and append this order after v



定理 上面算法在 O(m + n) 时间内找到一个拓扑排序.

Pf. 考虑边逐次递减的代价O(m); 追踪被删除的结点代价O(n).

拓扑排序 & DAG



the consensus algorithm of Conflux