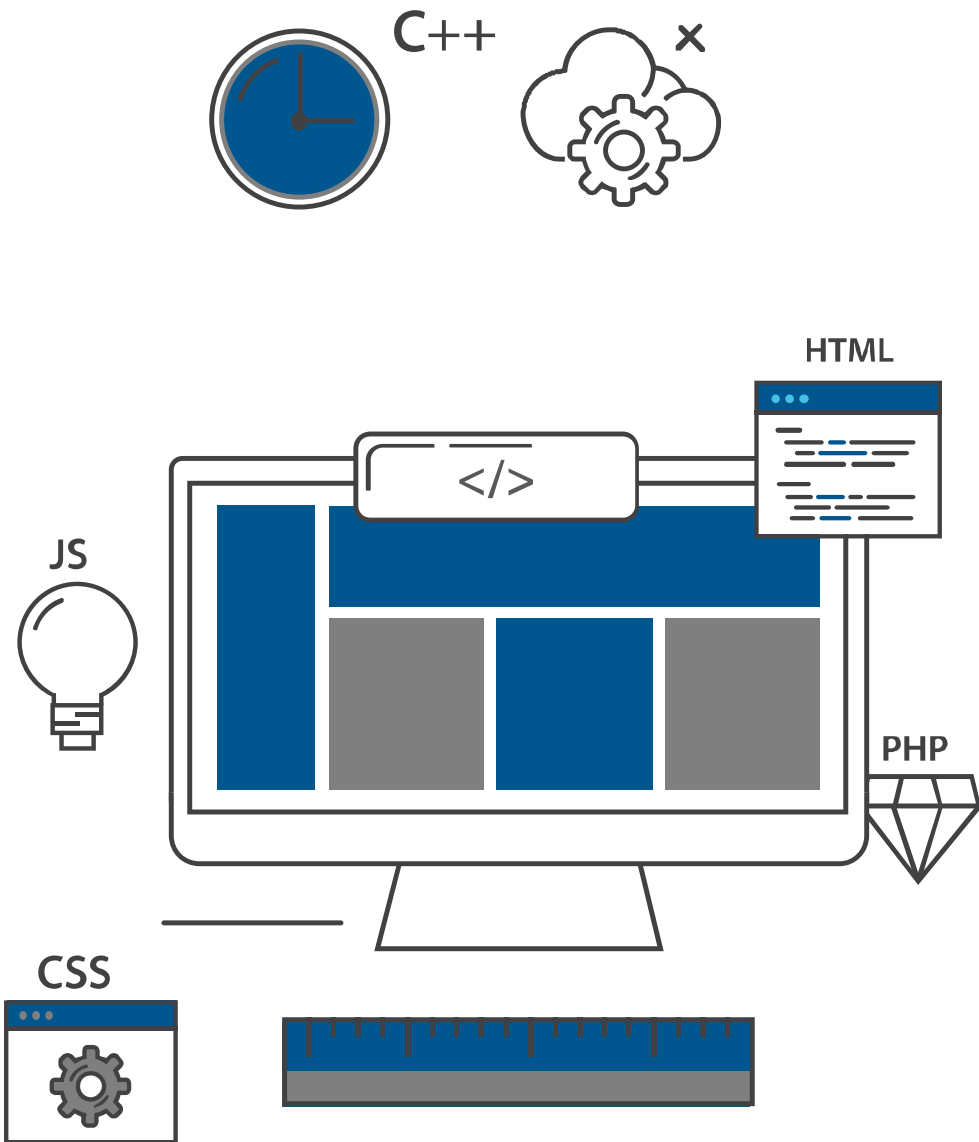


第三章 图

李翔

<https://implus.github.io/>





课堂作业二（回顾）

- 给定一个不降序整数数组A（长度为 $N>0$ ），例如 $-1, 0, 1, 1, 4, 6, 6, 6, 10$ ，请用 $\log(N)$ 的复杂度完成如下C++语言算法，数组下标从0开始。给定一个查询整数q，返回该查询q在数组A中**第一次出现**的下标位置，若未出现，则返回-1；（例如 $q=6$ ，返回5）

```
int A[N+1]; # 假设A已经赋值完毕
A[N] = inf;
int l = _____, r = _____, m;
while(r - l > _____) {
    m = (l + r) / 2;
    if (A[_____] _____ q) r = m; else l = m;
}
if (A[_____] _____ q) cout<<r; else cout<<-1;
```

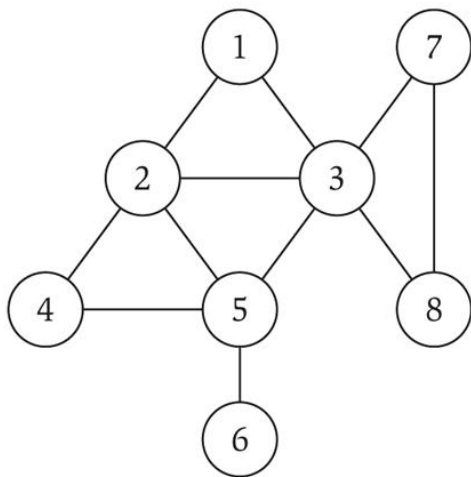


3.1 基本定义与应用

- 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

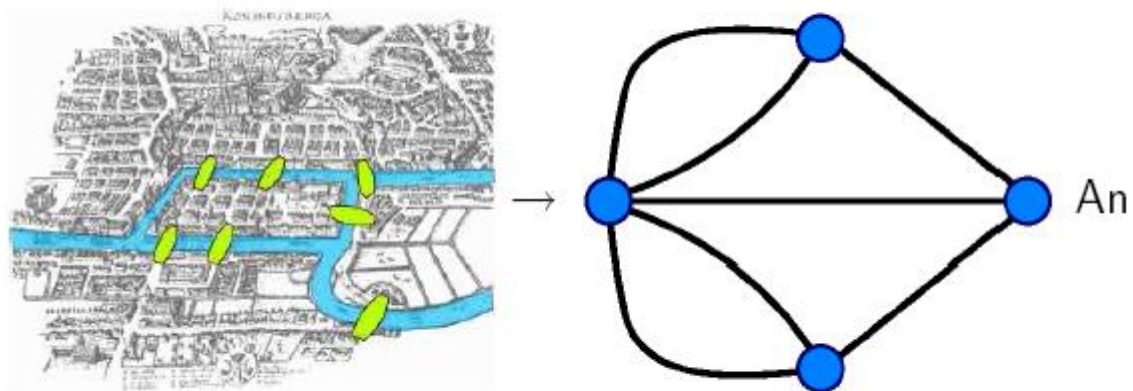
■ 应用

<i>Graph</i>	<i>Nodes</i>	<i>Edges</i>
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

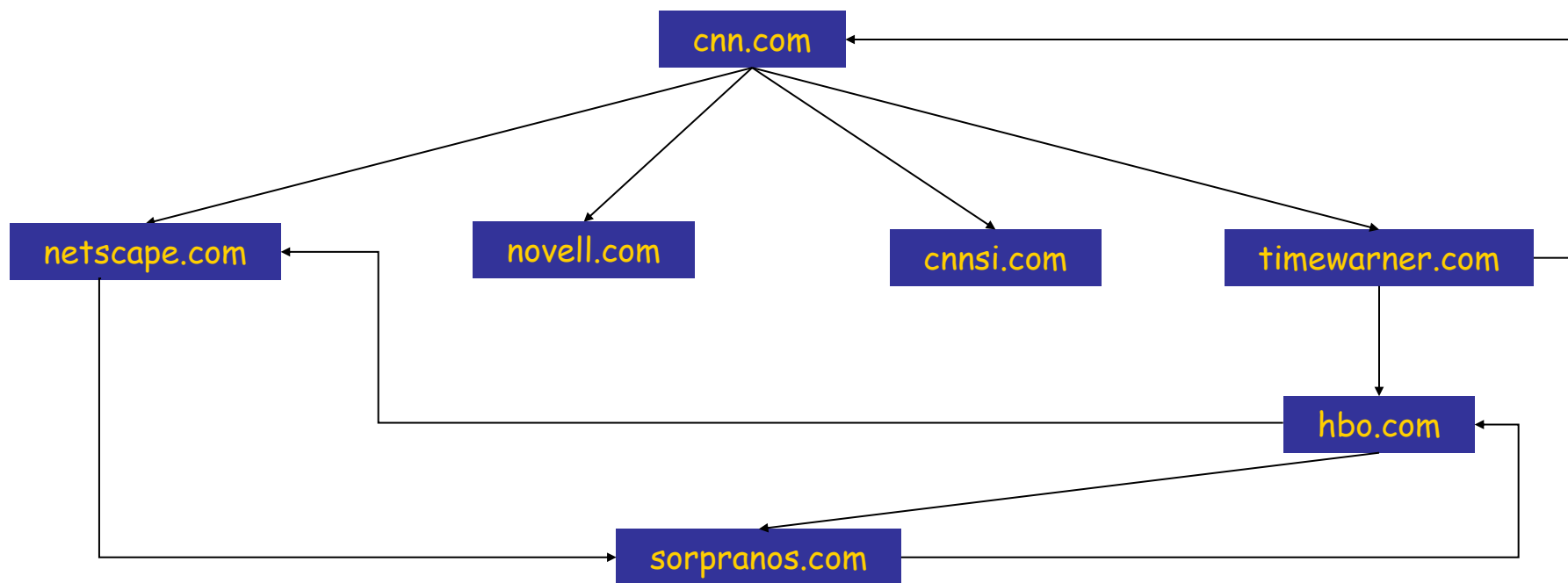


Euler path exists if and only if the graph is connected and has 0 or 2 vertices with odd degrees.

3.1 基本定义与应用

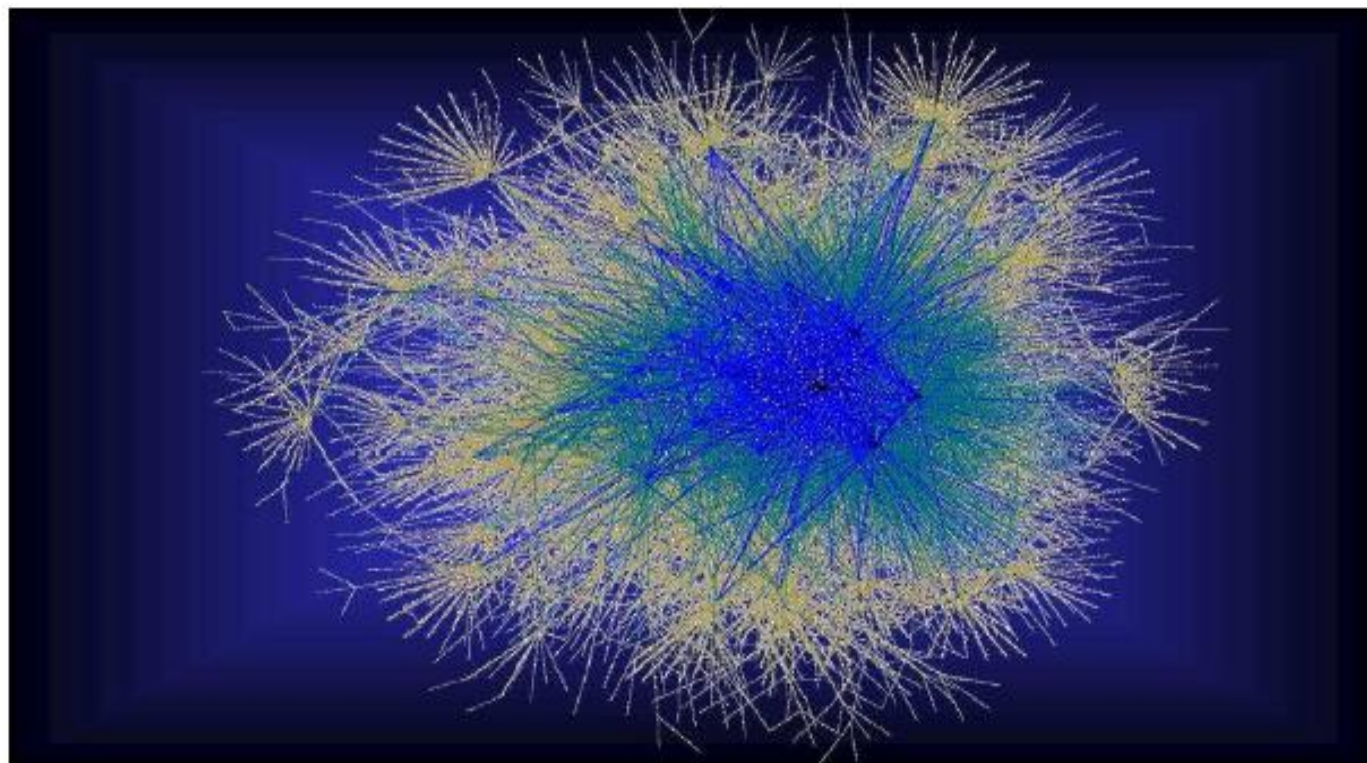
■ Web graph.

- 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

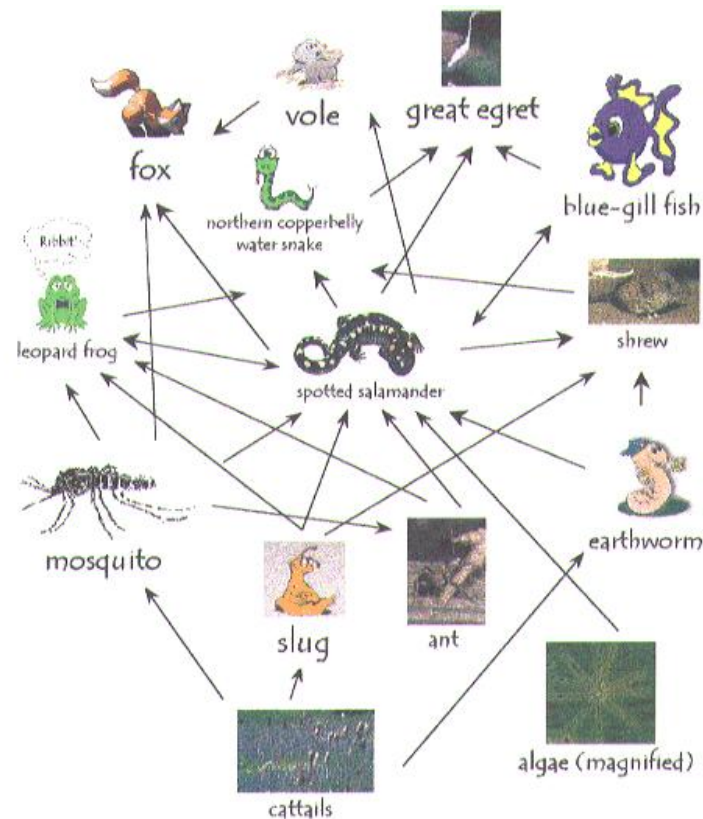


3.1 基本定义与应用

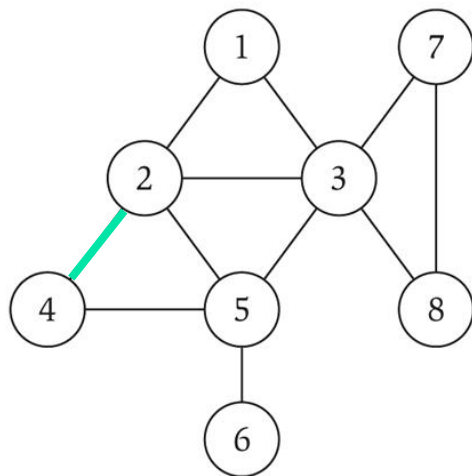
- 食物生态链图

- ✓ Node = 生物种类.

- ✓ Edge = 从猎物到捕食者.



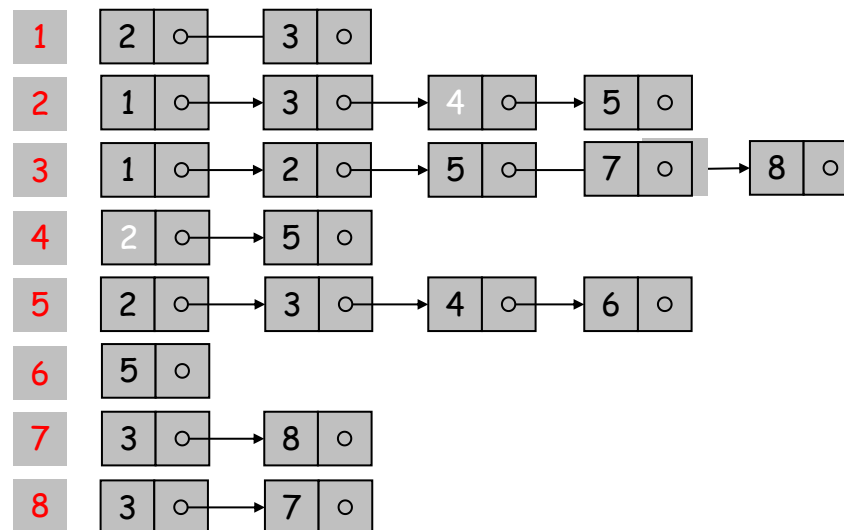
图的表示



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



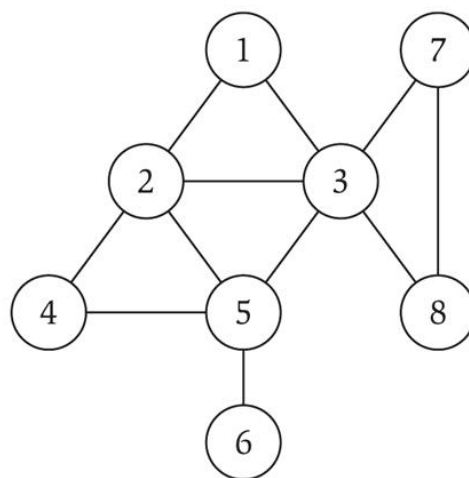


路径与连通性

- Def. 无向图 $G = (V, E)$ 的一条路径 P 定义为如下的结点序列 $v_1, v_2, \dots, v_{k-1}, v_k$, 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

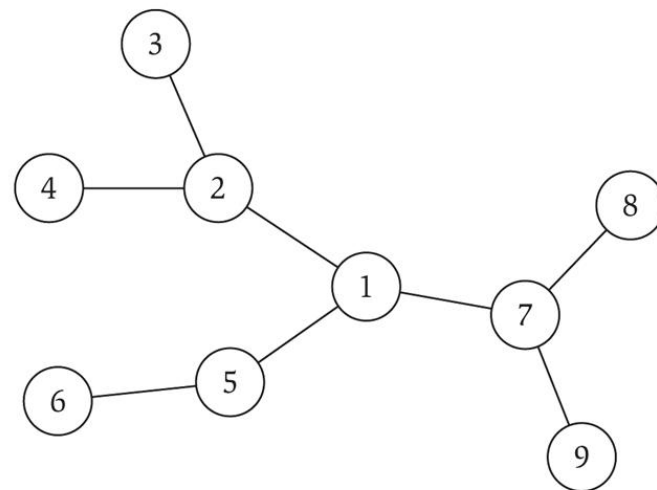
- Def. 路径 $v_1, v_2, \dots, v_{k-1}, v_k$ 被称为一个圈(Cycle), 如果 $v_1 = v_k$, $k > 2$, 而且前 $k-1$ 个顶点两两不同。



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

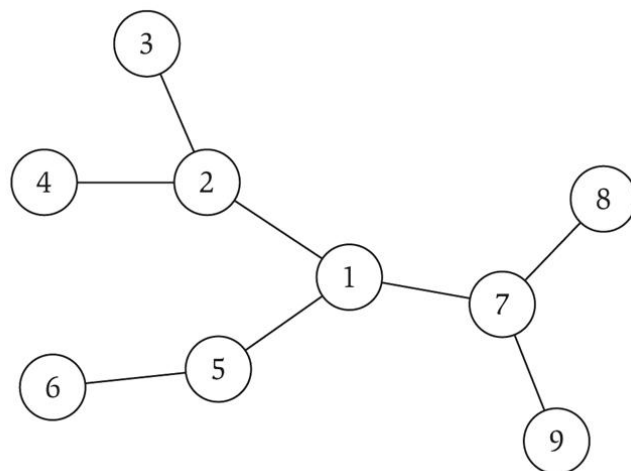
树的定义

- **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.

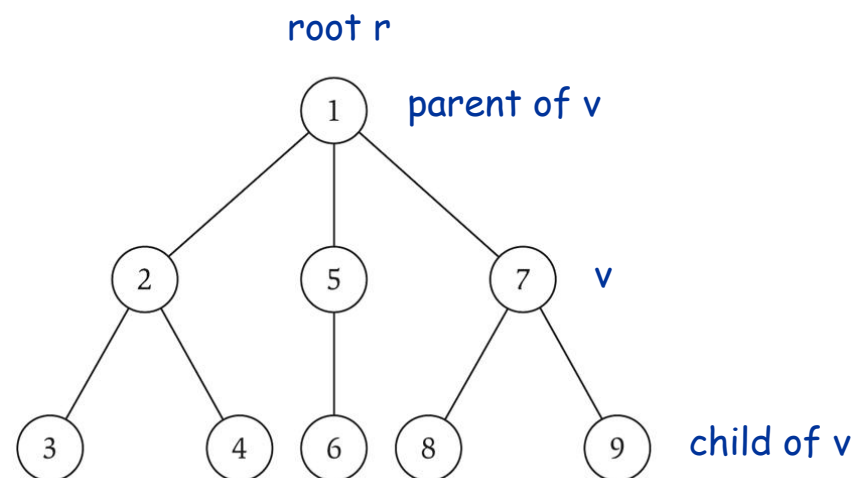


有根树

- 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

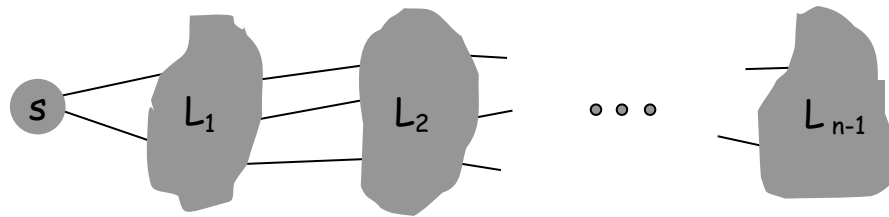


3.2 图的连通性与图的遍历

- **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

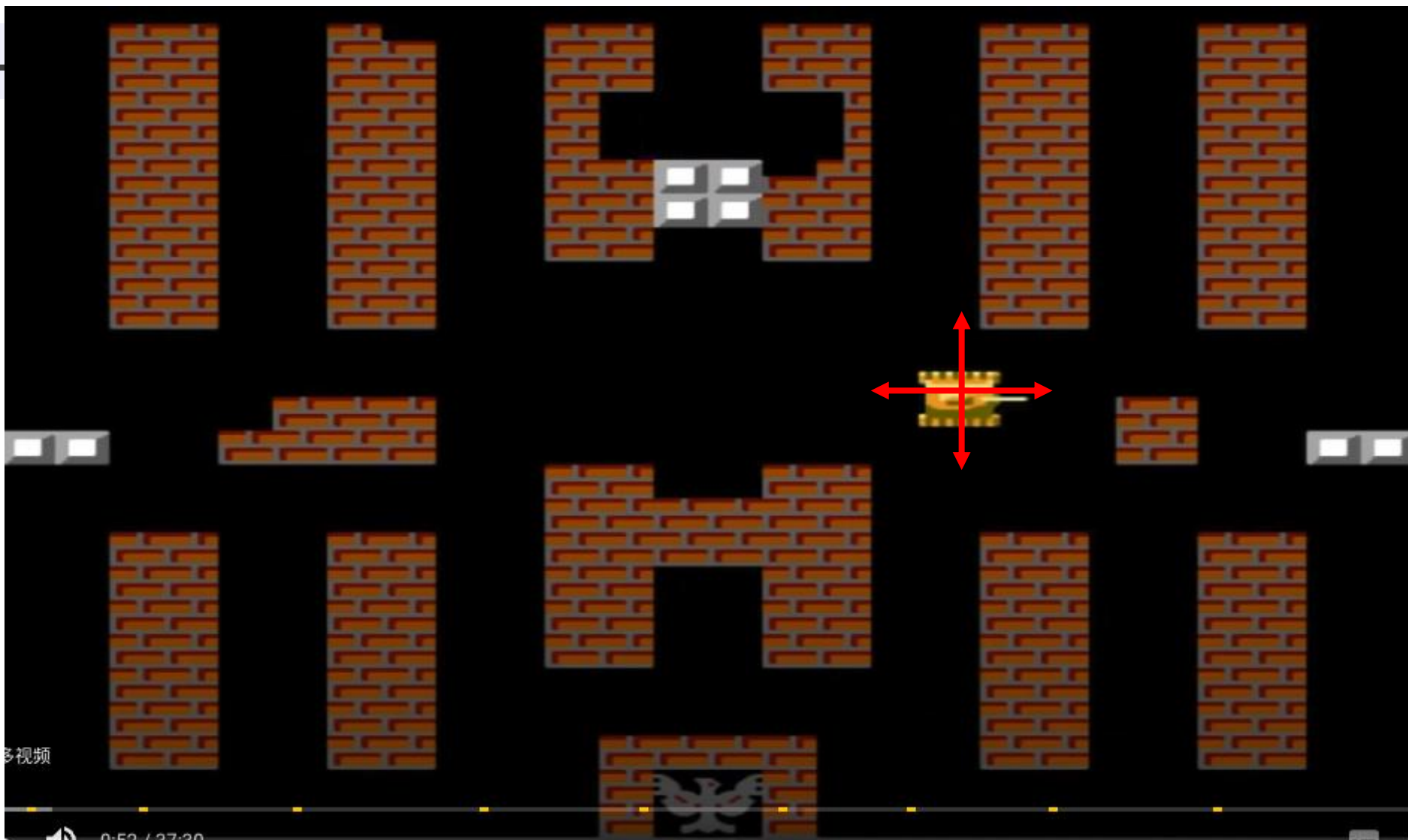
广度优先(Breadth-First Search)

- **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 .

一个典型的BFS应用场景



ZTE中兴图灵之坦克大战



选择左边AI

KeyBoard
AIDebug.dll
aurora.dll
K-2.dll
suixin.dll
TTMQL2.dll
TTMQLZ1.dll
Yumir.dll

选择右边AI

KeyBoard
AIDebug.dll
aurora.dll
K-2.dll
suixin.dll
TTMQL2.dll
TTMQLZ1.dll
Yumir.dll

VS

Start WebSite Exit

ZTE中兴图灵之坦克大战



选择左边AI

KeyBoard

AIDebug.dll

aurora.dll

K-2.dll

suixin.dll

TTMQLZ.dll

TTMQLZ1.dll

Yumir.dll

VS

选择右边AI

KeyBoard

AIDebug.dll

aurora.dll

K-2.dll

suixin.dll

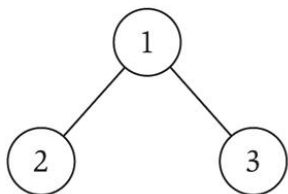
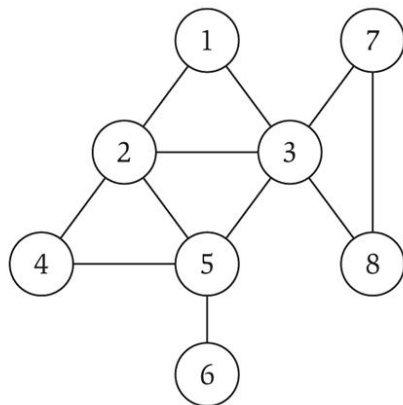
TTMQLZ.dll

TTMQLZ1.dll

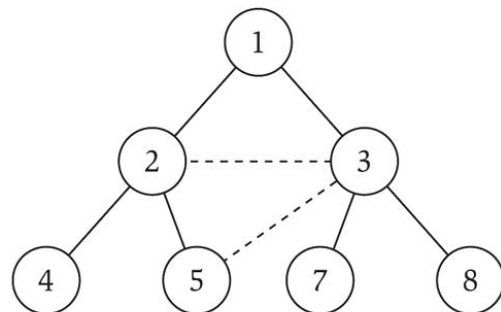
Yumir.dll

Start WebSite Exit

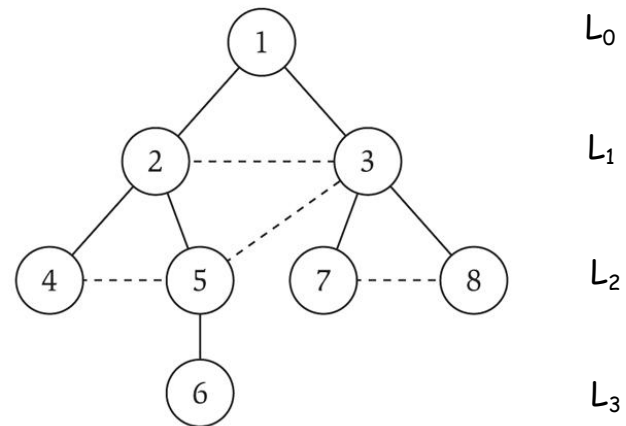
BFS性质



(a)



(b)



(c)

L_0
 L_1
 L_2
 L_3



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 Discovered[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 $\sum_{u \in V} \deg(u) = 2m$; 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;

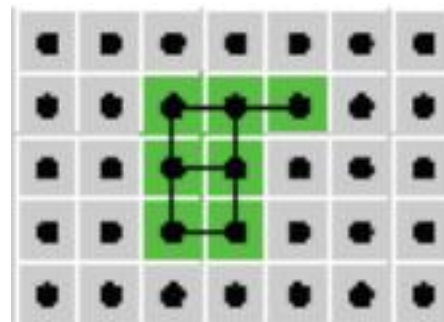


DFS

- 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

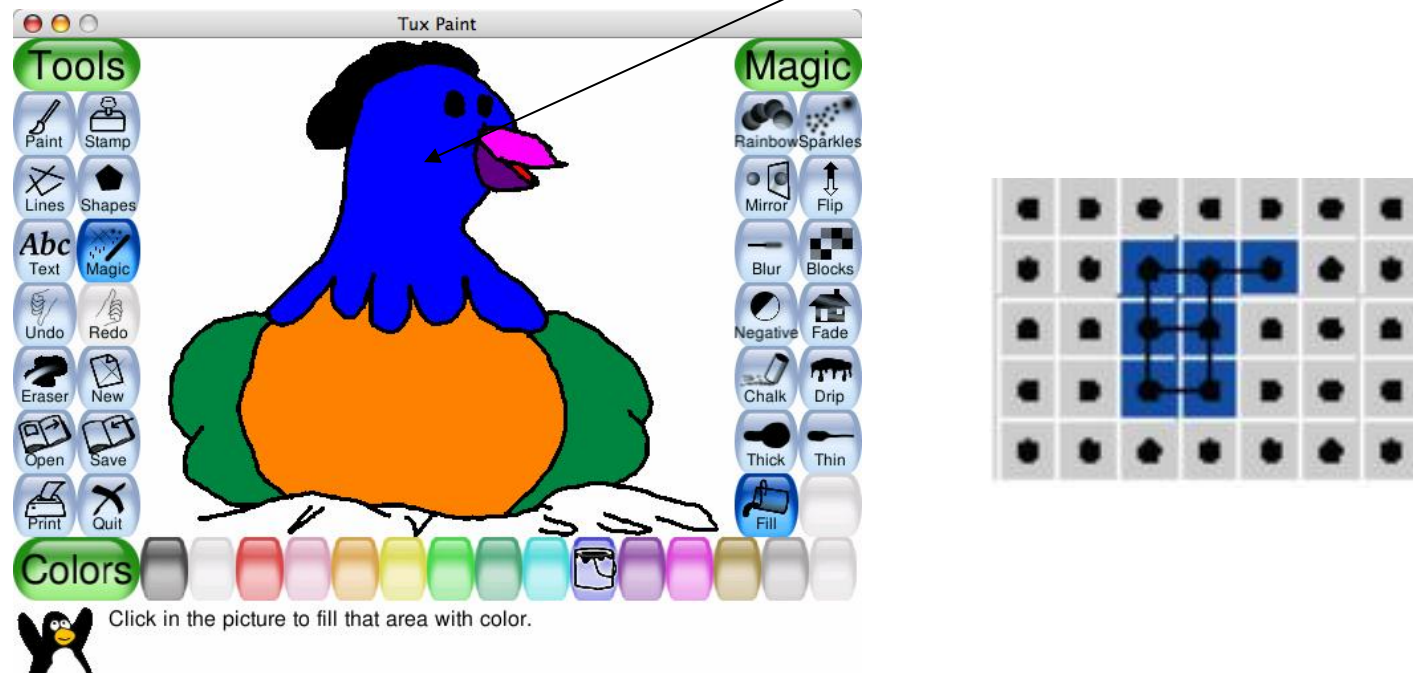
应用实例

- 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

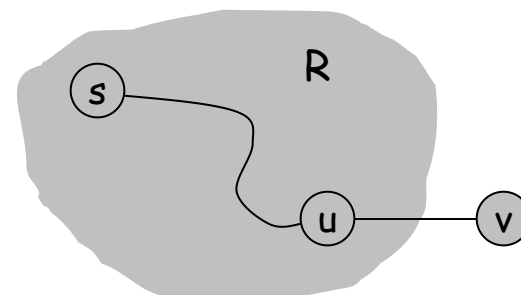
需要找到同种亮绿色的结构，然后改变颜色



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 $u \in R$ and $v \notin R$
 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。

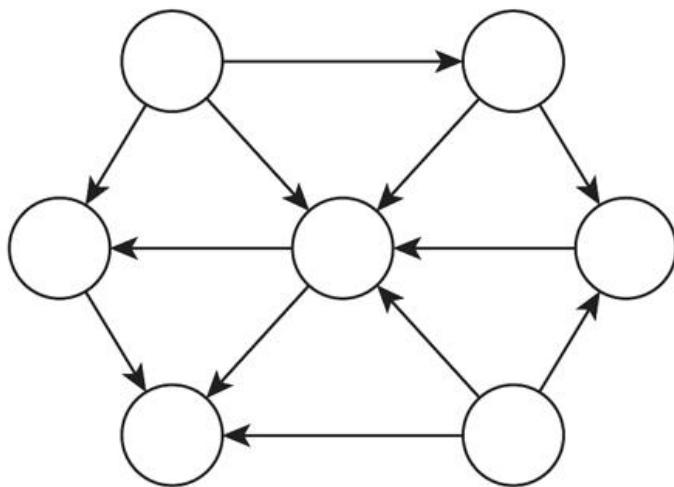


连通性质

- 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

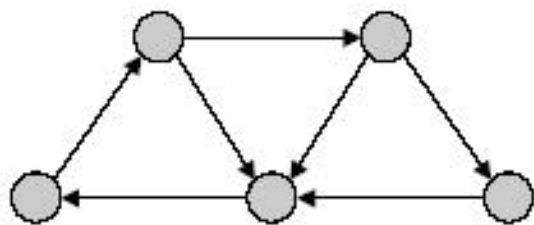
- 有向图(Directed graph) $G = (V, E)$
 - Edge (u, v) : 从 u 到 v



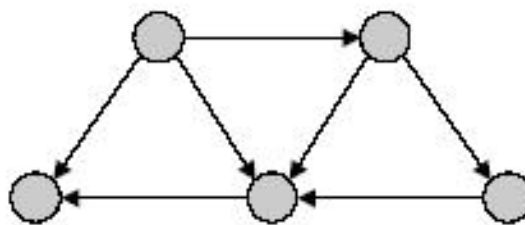
类似的，无向图中的一些概念，问题在这里可以推广

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。



strongly connected



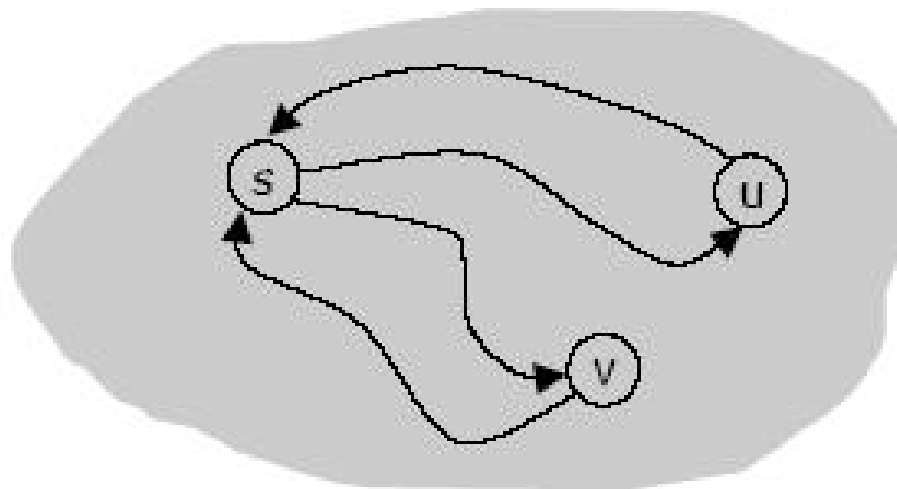
not strongly connected

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.





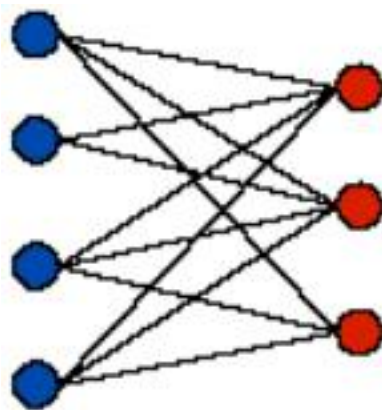
强连通性

- 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

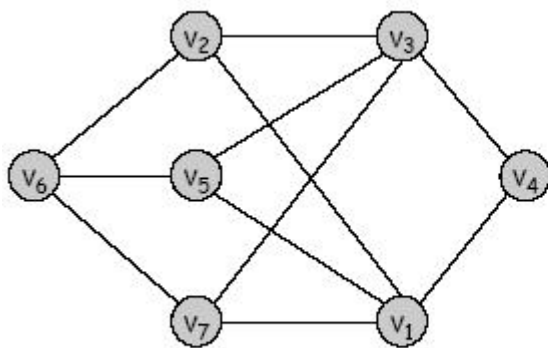
- 二部图是一个图，其中结点集可以划分为 X 与 Y ,每条边一端在 X 中，另一端在 Y 中。
- 直观的，一个图是二部图，如果能对结点着红色和蓝色，使得每条边有一个红端点和一个蓝端点。



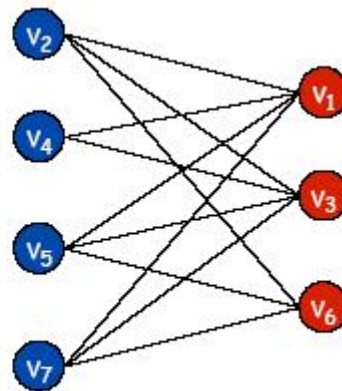
a bipartite graph

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。



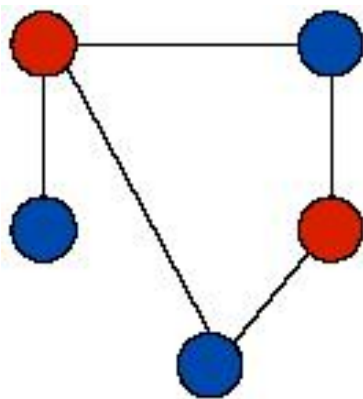
a bipartite graph G



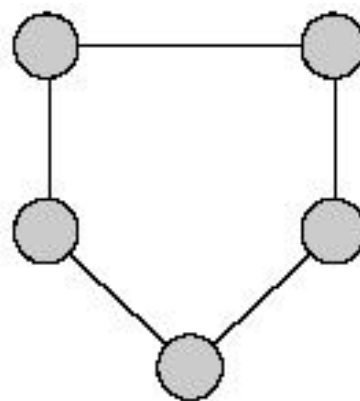
another drawing of G

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。



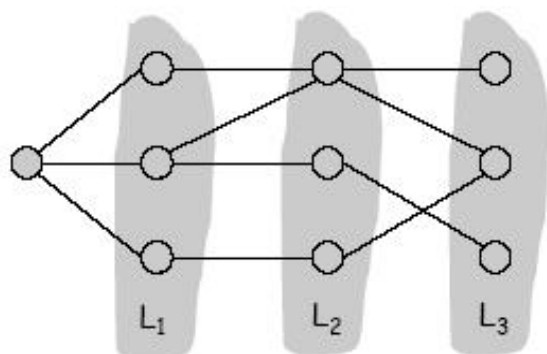
bipartite
(2-colorable)



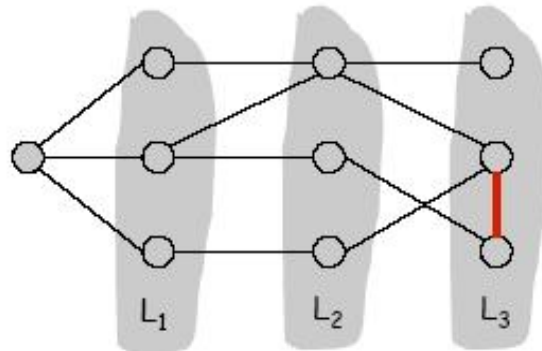
not bipartite
(not 2-colorable)

二部图性质

- 定理 设 G 是一个连通图， L_0, \dots, 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。



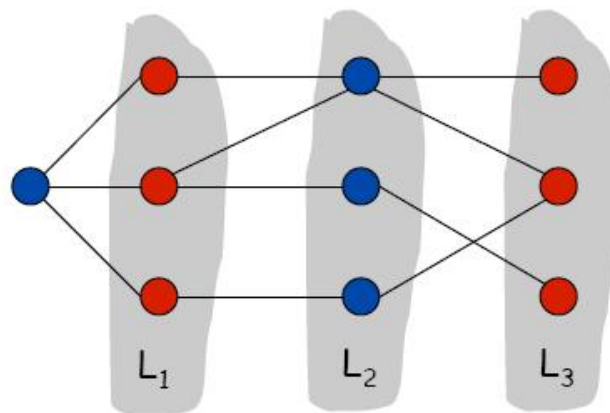
Case (i)



Case (ii)

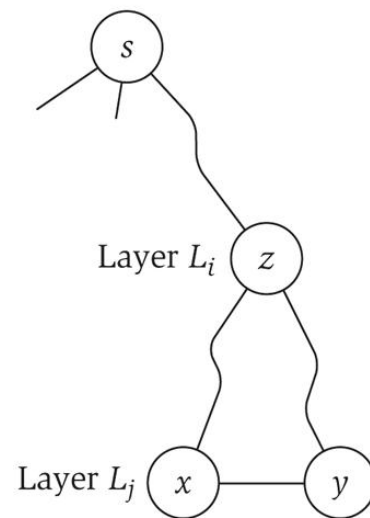
二部图性质

- Pf. (i) 红蓝交替着色



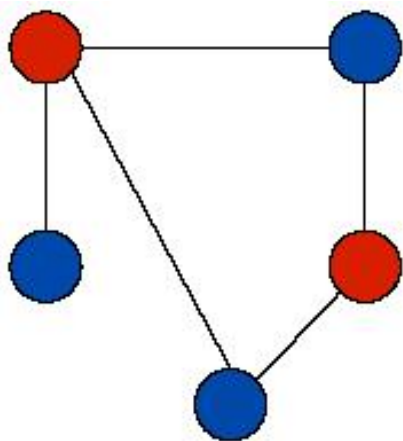
(ii) 证明产生一个奇圈

Case (i)

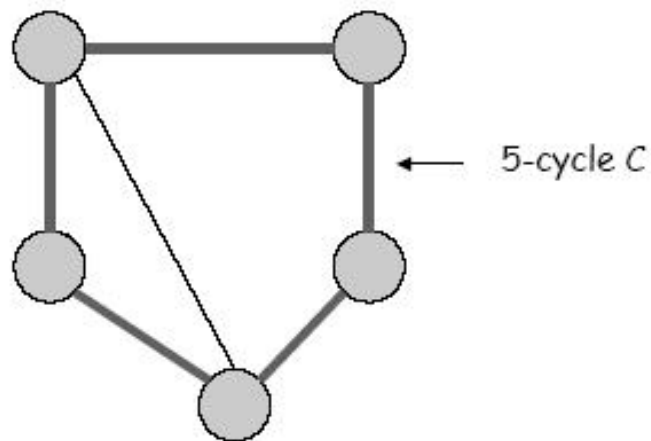


二部图性质

- 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)

DeepSeek大法

二部图与奇圈无矛盾的证明

图 G 是二部图当且仅当图中没有奇圈。如何简单易懂的证明？



已深度思考（用时 59 秒） ^

嗯，我现在要证明图 G 是二部图当且仅当图中没有奇圈。这个定理好像挺重要的，对吧？不过对于怎么证明，我还得仔细想想。

首先，二部图的定义是顶点集可以分成两个不相交的子集，使得图中的每一条边都连接这两个子集中的顶点。换句话说，二部图里不存在边连接同一个子集内的顶点。那么，二部图的一个重要特性就是它的所有环的长度都是偶数，也就是没有奇圈。这应该就是定理的一个方向。

不过现在需要证明的是双向的，即当且仅当。也就是说，要证明两个方向：如果 G 是二部图，那么它没有奇圈；反过来，如果 G 中没有奇圈，那么 G 是二部图。这两个方向都得证明对吧？



证明：图 G 是二部图当且仅当图中没有奇圈

(\Rightarrow) 若 G 是二部图，则 G 中无奇圈

1. 二部图的结构：设 G 的顶点集分为两个不相交的子集 U 和 V ，所有边均连接 U 和 V 中的顶点。
2. 环的路径分析：任取一个环 $C = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow v_1$ 。由于边交替连接 U 和 V ，顶点在环中交替属于 U 和 V 。
3. 环长的奇偶性：若环长为奇数 k ，则从 $v_1 \in U$ 出发，经过 k 步后回到 v_1 ，此时 v_1 需同时属于 U 和 V ，矛盾。因此，环长必为偶数，即 G 中无奇圈。



(\Leftarrow) 若 G 中无奇圈, 则 G 是二部图

1. 构造二色染色: 假设 G 连通 (否则对每个连通分量分别处理)。任选一顶点 u , 定义染色规则:

- 将 u 染为黑色;
- 对任意顶点 v , 若存在从 u 到 v 的路径长度为偶数, 则 v 染黑色; 若长度为奇数, 则染白色。

2. 染色一致性证明:

- 假设存在矛盾: 若存在边 (v, w) 且 v 与 w 同色, 则从 u 到 v 和 u 到 w 的两条路径奇偶性相同。连接这两条路径与边 (v, w) , 形成一个环, 其长度为偶数 (路径长度之和) 加1 (边 (v, w)), 结果为奇数, 与“ G 中无奇圈”矛盾。
- 结论: 所有相邻顶点颜色不同, 染色有效。

3. 二部划分: 按颜色将顶点分为 U (黑色) 和 V (白色), 边仅存在于 U 和 V 之间, 故 G 为二部图。

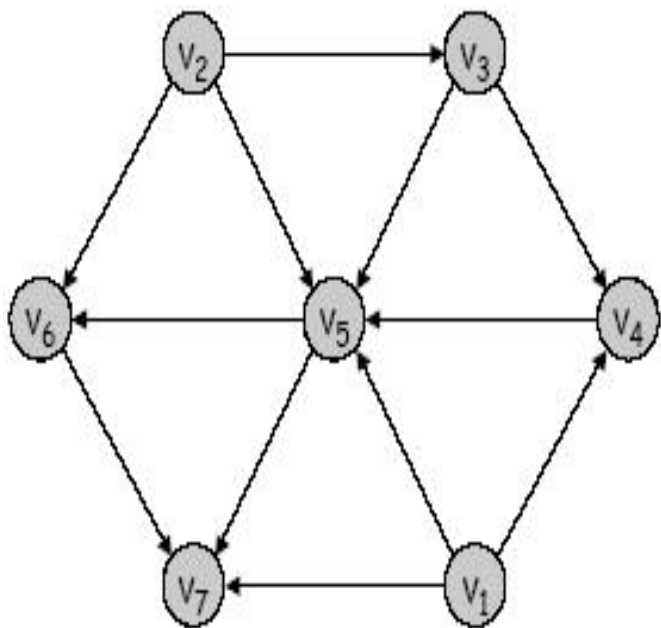
综上, 图 G 是二部图当且仅当 G 中无奇圈。



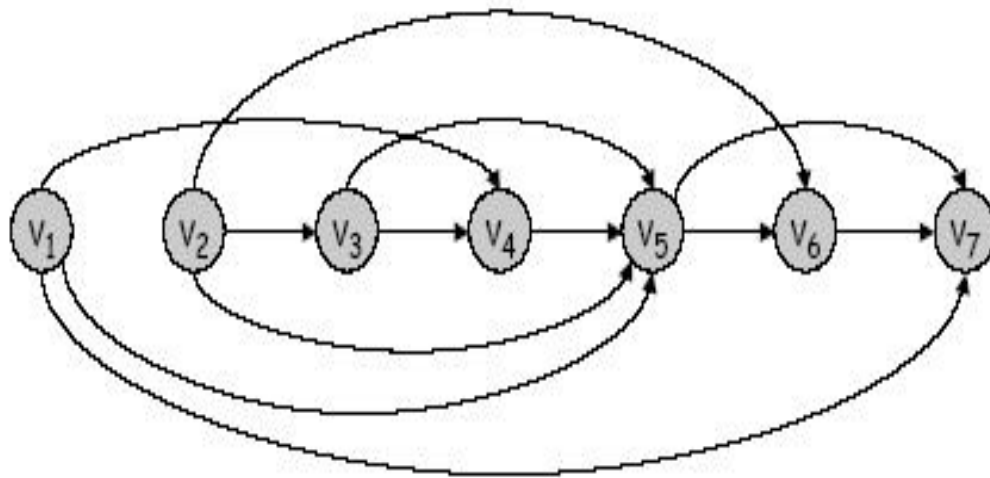
3.6 有向无圈图与拓扑排序

- 问题的背景：存在标记为 $\{1, 2, \dots, n\}$ 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, \dots, 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

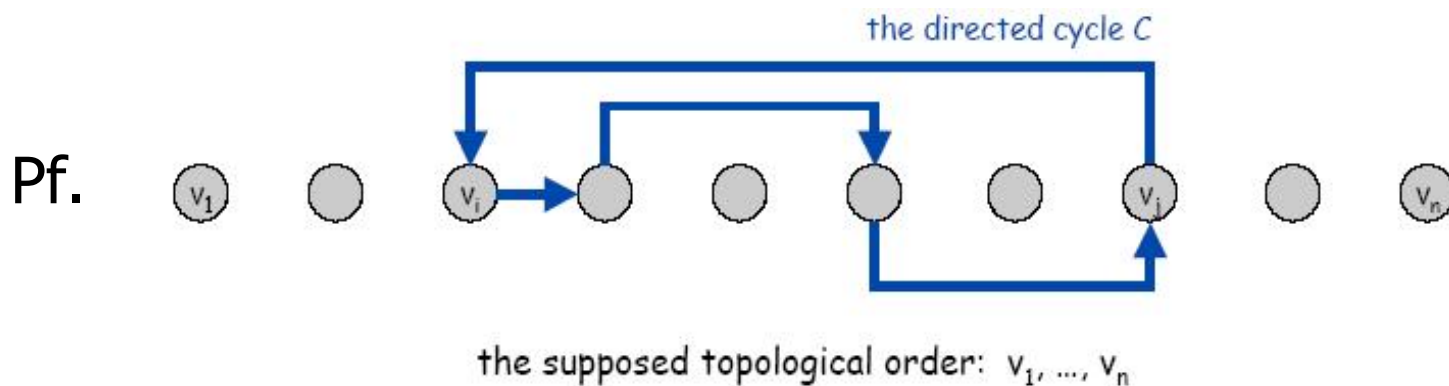


a topological ordering

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.





DAG

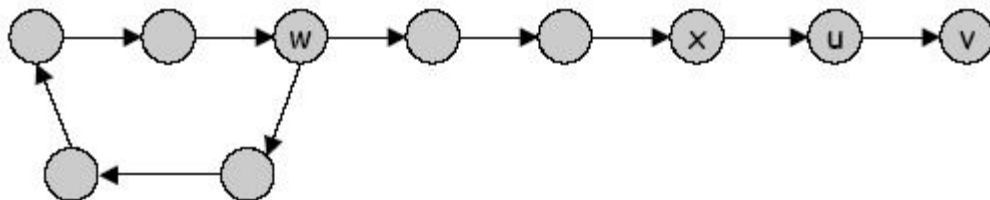
- Q1 是否每一个DAG都存在拓扑排序?
- Q2 如果存在, 如何计算?

DAG

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 (利用命题)

DeepSeek大法

二部图与奇圈无矛盾的证明

如何证明：在每一个DAG中，存在一个没有输入边的结点。



✖ 已深度思考 (用时 55 秒) ^

嗯，我现在得仔细想想怎么证明在每一个有向无环图（DAG）中，存在一个没有输入边的结点。也就是说，每个DAG里至少有一个结点的入度是零。这个问题看起来应该和DAG的性质有关，比如没有环，以及有向边的关系。不过具体怎么证明呢？让我慢慢理清楚思路。

首先，DAG是有向无环图，所以里面没有循环，也就是说从任何一个结点出发沿着有向边走，不可能回到起点。这点很重要，因为如果有环的话，可能环中的每个结点都有输入边，但DAG里不存在这种情况。那如何利用这个特性来证明存在入度为零的结点呢？

也许可以用反证法。假设DAG中所有结点的入度都至少为1，那么会发生什么呢？比如说，每个结点都至少有一个入边。这样的话，能不能构造出一个环？比如，从一个结点开始，沿着入边往回走，因为每个结点都有前驱，这样会不会无限走下去导致出现环？

比如，假设有一个结点 v_1 ，它有一个入边来自 v_2 ，而 v_2 又有一个入边来自 v_3 ，依此类推。由于图是有限的，结点数量是有限的，所以如果一直这样走下去，最终必须重复访问某个结点，这就形成了环。但DAG是不能有环的，所以这就和假设矛盾了。因此，原来的假设错误，即至少存在一个入度为零的结点。

不过这里可能需要注意细节。比如，如何确保在每一步都能找到前驱结点？因为每个结点都有入度至少为1，所以每个结点都有至少一个前驱。于是，可以构造一个无限长的路径，但因为图中结点有限，所以路径中必然出现重复结点，这就形成了环，而DAG不允许有环，矛盾。

或者另一种思路，用拓扑排序。DAG可以进行拓扑排序，拓扑排序中的第一个结点是不是入度为零？因为在拓扑排序中，每次选择一个入度为零的结点加入序列，然后删除该结点及其出边。如果整个图是DAG，那么拓扑排序可以完成，即存在这样的结点。所以这说明在DAG中存在至少一个入度为零的结点，否则无法开始拓扑排序。

这两种方法似乎都可行。第一种是反证法，第二种是构造性证明，通过拓扑排序的存在性来证明。不过可能用户需要更直接的证明，不需要依赖拓扑排序的概念，因为可能有些读者还不了解拓扑排序。所以用反证法可能更基础一些。

再仔细想想反证法的步骤。假设DAG中所有结点的入度 ≥ 1 ，那么可以构造一个无限长的路径，但由于图是有限的，路径中必存在重复结点，导致环，矛盾。所以原命题成立。

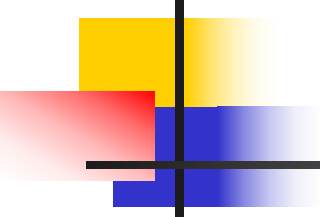
不过这里可能需要注意，是否真的能构造这样的路径。比如，从某个结点出发，每次选择它的一个前驱结点，这样不断前进，因为每个结点都有前驱，所以理论上可以无限进行下去。但由于结点数量有限，根据鸽巢原理，路径中至少有一个结点会被重复访问，从而形成环，这与DAG无环矛盾。因此，原假设不成立，即DAG中存在至少一个入度为零的结点。

这样的证明是否严谨？可能需要更详细地描述步骤。例如：

1. 假设DAG中每个结点的入度 ≥ 1 。
2. 任选一个结点 v_0 ，由于它的入度 ≥ 1 ，存在 $v_1 \rightarrow v_0$ 。
3. 同样， v_1 的入度 ≥ 1 ，存在 $v_2 \rightarrow v_1$ 。
4. 继续这个过程，得到序列 $v_0 \leftarrow v_1 \leftarrow v_2 \leftarrow \dots \leftarrow v_k$ 。
5. 因为图中结点数有限，设为 n 个，所以在 $n+1$ 步时，至少有一个结点被访问两次，比如 $v_i = v_j$ ，其中 $i < j$ 。
6. 于是，从 v_i 到 v_j 的路径形成一个环，这与DAG无环矛盾。
7. 因此假设错误，DAG中存在至少一个入度为零的结点。

这样步骤更清晰，应该是对的。所以总结下来，用反证法构造路径导致环的出现，从而证明原命题成立。





另外，也可以考虑数学归纳法。比如，对于结点数为 n 的DAG，当 $n=1$ 时，显然该结点入度为0。假设对于所有结点数小于 k 的DAG命题成立，那么当考虑 k 个结点的DAG时，是否存在入度为0的结点？如果存在，则命题成立；如果不存在，则每个结点都有入度 ≥ 1 ，根据之前的反证法，形成环，矛盾。所以归纳法也适用。

不过可能归纳法不如反证法直接。因为反证法更直观地展示了矛盾的产生。

综上所述，证明的关键在于假设所有结点入度 ≥ 1 ，导致存在环，与DAG定义矛盾，从而证明原命题成立。

方法：反证法

1. 假设相反：假设DAG中所有结点的入度均至少为1（即每个结点都有至少一条输入边）。

2. 构造无限路径：

- 任选一个结点 v_0 ，由于 v_0 的入度 ≥ 1 ，存在结点 v_1 使得 $v_1 \rightarrow v_0$ 。
- 同理， v_1 的入度 ≥ 1 ，存在结点 v_2 使得 $v_2 \rightarrow v_1$ 。
- 重复此过程，得到一条无限长的反向路径：

$$v_0 \leftarrow v_1 \leftarrow v_2 \leftarrow v_3 \leftarrow \cdots$$

3. 有限结点导致矛盾：

- 由于图中只有有限个结点（设为 n 个），在构造路径时，最多经过 $n + 1$ 步后，必然出现重复访问的结点（由鸽巢原理）。
- 设 $v_i = v_j$ ($i < j$)，则路径段 $v_i \leftarrow v_{i+1} \leftarrow \cdots \leftarrow v_j$ 形成一个环，即：

$$v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_j \rightarrow v_i$$

- 这与DAG的无环性矛盾。

4. 结论：原假设不成立，因此DAG中至少存在一个入度为0的结点。

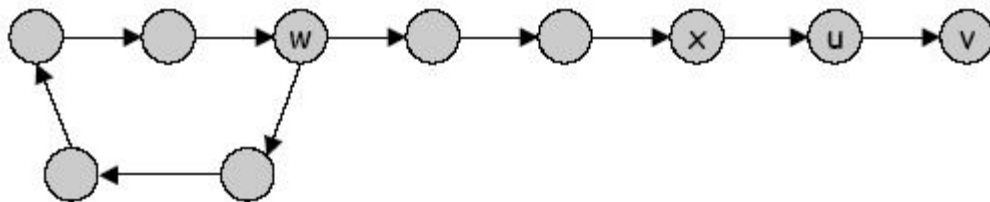


DAG

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 (利用命题)

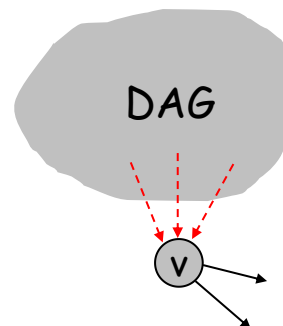


证明思路

1. 归纳基础：单个顶点的图显然存在拓扑排序。
2. 归纳假设：假设所有节点数小于 n 的DAG均存在拓扑排序。
3. 关键引理：DAG中必存在入度为0的顶点。
 - 若所有顶点入度 ≥ 1 ，则可通过反复回溯前驱顶点构造无限路径，导致环，矛盾。
4. 归纳步骤：
 - 选取入度为0的顶点 v ，将其作为拓扑排序的首元素。
 - 移除 v 及其出边，剩余子图仍为DAG且节点数为 $n - 1$ 。
 - 根据归纳假设，子图存在拓扑排序，将其接在 v 后即得原图的拓扑排序。

■ 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 $O(n)$.

以下是使用深度优先搜索（DFS）实现拓扑排序的 Python 代码，并附带详细注释：

代码实现

python

复制

```
def topological_sort_dfs(graph):
    visited = {}          # 记录节点状态: 未访问 / 正在访问 / 已访问
    result = []           # 保存拓扑排序结果 (逆序)

    # 定义DFS递归函数
    def dfs(node):
        if node in visited:
            # 若遇到正在访问的节点, 说明存在环
            if visited[node] == "visiting":
                return False
            # 若已访问过, 直接跳过
            return True

        visited[node] = "visiting" # 标记为正在访问
        for neighbor in graph.get(node, []):
            if not dfs(neighbor): # 递归访问邻接节点
                return False

        visited[node] = "visited" # 标记为已访问
        result.append(node)       # 后序位置加入结果
        return True

    # 遍历所有未访问的节点
    for node in graph:
        if node not in visited:
            if not dfs(node):
                raise ValueError("图中有环, 无法拓扑排序!")

    return result[::-1] # 反转后序结果, 得到拓扑排序
```



AI Coding IDE



南開大學
Nankai University

Coding with AI

以“Trae AI IDE快速搭建arXiv推荐”为例