

# 《算法设计与分析》

## 6-图的遍历 (Graph Traversal)

杨启哲

上海师范大学信机学院计算机系

2024 年 10 月 3 日

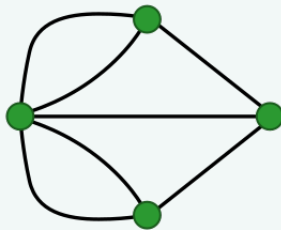
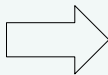
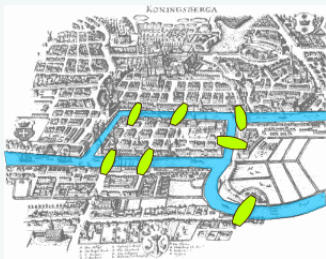
- › 图的介绍
- › 深度优先搜索
- › 广度优先搜索

 图的介绍

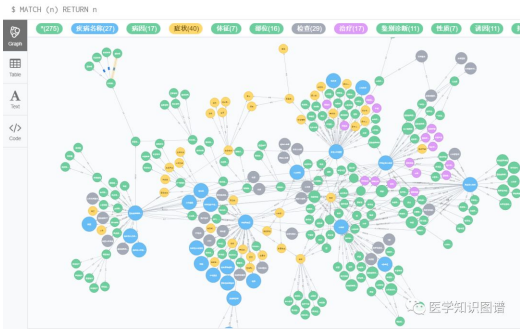
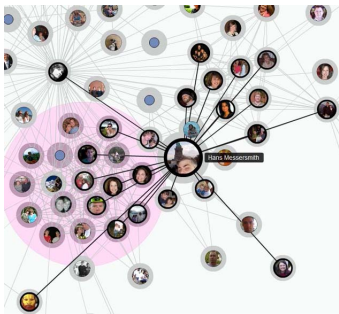
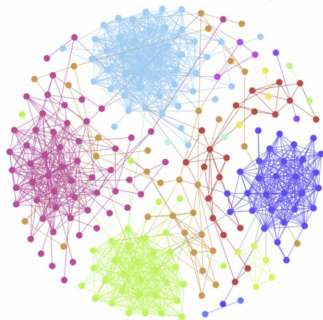
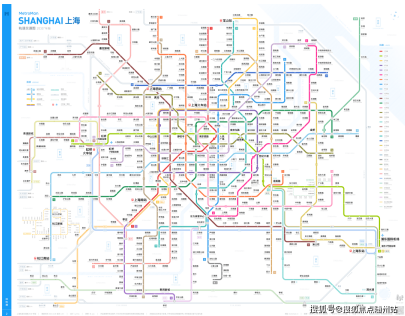
一般认为，欧拉于 1736 年出版的关于哥尼斯堡七桥问题的论文是图论领域的第一篇文章。

## 哥尼斯堡七桥问题 (Königsberg Bridge Problem)

在 18 世纪，东普鲁士柯尼斯堡（今日俄罗斯加里宁格勒）市区跨普列戈利亚河两岸，河中心有两个小岛。小岛与河的两岸有七条桥连接。在所有桥都只能走一遍的前提下，如何才能把这个地方所有的桥都走遍？



# 图无处不在!



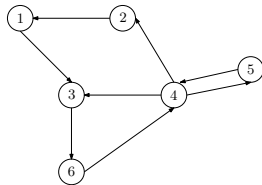
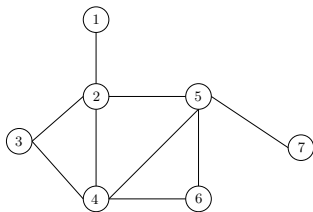
## 定义 1

[图].

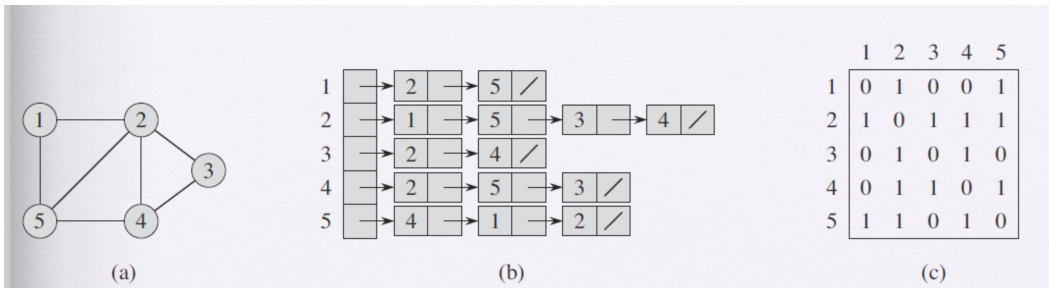
一个图由一个顶点集合  $V$  和边集合  $E$  组成, 记作  $G = (V, E)$ , 其中  $E \subseteq V \times V$ .

## 图的种类

- **无向图:** 任何一条边都是无向的, 即  $(u, v) \in E$  和  $(v, u) \in E$  等价。
- **有向图:** 边存在方向, 即  $(u, v) \in E$  表示一条从  $u$  到  $v$  的边。



# 图的表示：邻接矩阵 or 邻接列表？



图的两种表示：

- **邻接表**： $i$  所对应的列表包含了所有  $i$  指向的边。
  - 无向图： $n + 2m$
  - 有向图： $n + m$
- **邻接矩阵**： $G[i][j]$  表示顶点  $i$  和顶点  $j$  之间是否有边。
  - 无向图： $n^2$
  - 有向图： $n^2$

- **遍历问题。**  
从一个顶点出发，访问所有的顶点。
- **路径问题。**  
给定途中的两个点  $s, t$ ，问是否存在  $s$  到  $t$  的一条边？
- **连通性问题。**  
一个图可以被分成几个连通块？
- **拓扑序问题。**  
给定一个有向图，给出一个顶点的排列，使得对于任意一条边  $(u, v)$ ， $u$  在排列中都在  $v$  的前面。
- ...



输入一个图，如何去访问图中所有的顶点？

---

我们接下来将给出两种系统化的访问方法：

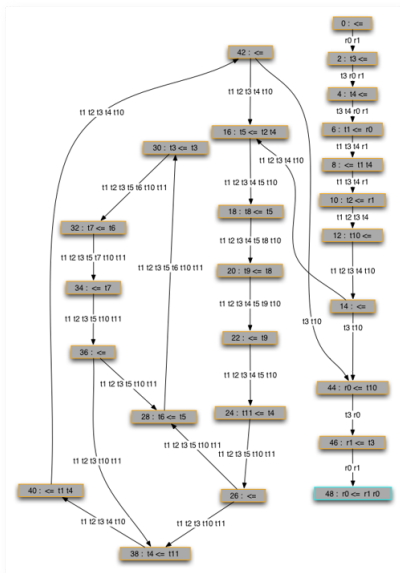
- 深度优先搜索 (Depth First Search)
- 广度优先搜索 (Breadth First Search)

一个程序实际上可以看成一张有向图:

- 顶点: 程序中的语句。
- 边: 程序中的跳转。

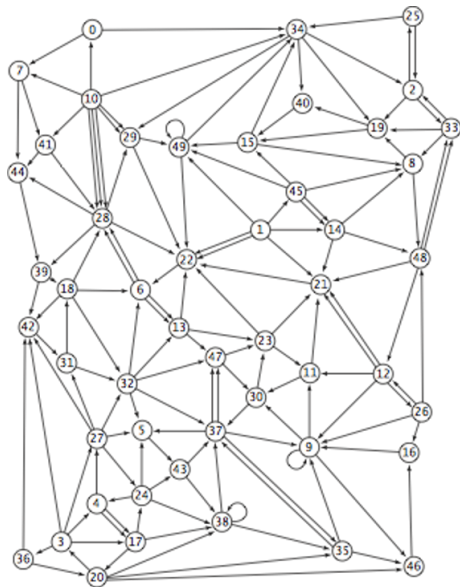
## 程序分析

- 死码删除 (Dead-code elimination)  
寻找到无法遍历的点即可。
- 死循环检测 (Loop detection)
- 判断程序是否会终止?



## 利用 BFS 爬取

- 用某个网页作为根节点
- 维护一个需要访问的队列
- 维护一个已经访问过网页的集合
- 弹出下一个待访问的网页并将其里面未访问的链接全部扔进队列里。



## 深度优先搜索

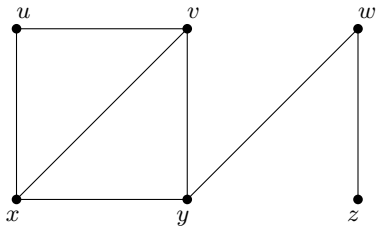


## 深度优先搜索 DFS

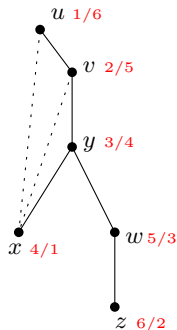
- 从一个顶点出发。
- 标记经过的所有顶点。
- 当没有未标记的顶点时，回溯到上一个顶点。

## 无向图的例子

考察下面的无向图，我们从顶点  $u$  出发，进行深度优先搜索。我们用  $pre/post$  来记录第一次访问和最后一次访问的顺序。

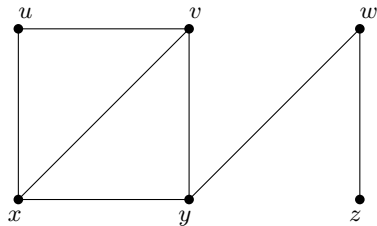


DFS 树

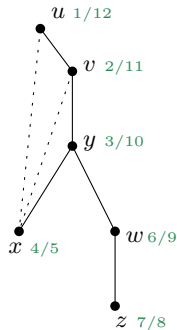


## 无向图的例子

当然我们也可以把 pre 和 post 合并起来用一个序列来记录其访问时间。



DFS 树



该序列描述了顶点进栈出栈的情况。

在无向图的深度优先搜索中，一共存在两类边：

- **树边**，即访问到未访问点的边。
- **回边**，所有其他的边。

怎么去找到回边？

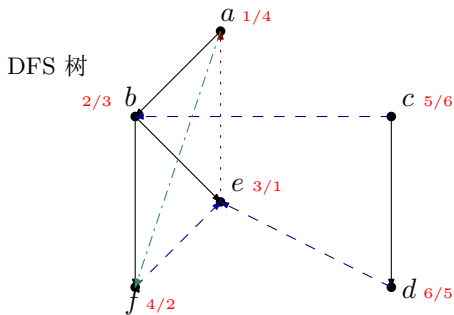
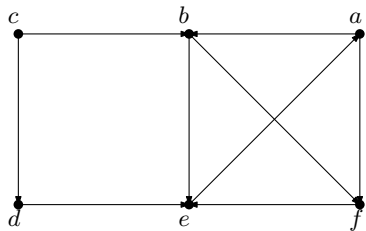
---

- 从一个顶点出发，如果遇到了已经访问过的顶点，那么就是回边。



## 有向图的例子

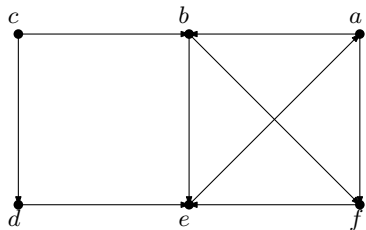
有向图的例子与无向图基本是类似的，但是我们可以看到似乎多了几种边。



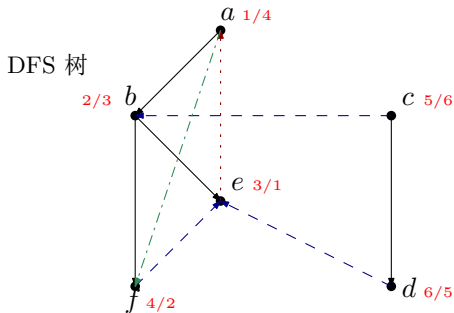
在有向图的深度优先搜索中，一共存在四类边：

- **树边**，即访问到未访问点的边。
- **回边**，在构建的树中， $w$  是  $v$  的祖先，当访问  $(v, w)$  时若  $w$  已经被访问过，则  $(v, w)$  是回边。
- **前向边**，在构建的树中， $w$  是  $v$  的后裔，当访问  $(v, w)$  时若  $w$  已经被访问过，则  $(v, w)$  是回边。
- **横跨边**，所有其他的边。

# 有向图的例子



- 树边:  $(a, b)$ ,  $(b, e)$ ,  $(b, f)$ ,  $(c, d)$ .
- 回边:  $(e, a)$
- 前向边:  $(a, f)$
- 横跨边:  $(c, b)$ ,  $(d, e)$ ,  $(f, e)$ .



当我们将 pre 和 post 合并成一个序列观察的时候，有以下结论：

## 定理 2

## [括号化定理].

给定图  $G = (V, E)$ ，则对于其一个深度优先搜索和任意两个顶点  $u$  和  $v$ ，有三种情况：

- 区间  $[u.pre, u.post]$  和  $[v.pre, v.post]$  完全不相交。这种情况下， $u$  不是  $v$  的后代， $v$  也不是  $u$  的后代。
- 区间  $[u.pre, u.post]$  完全包含在  $[v.pre, v.post]$  内。这种情况下， $u$  是  $v$  的后代。
- 区间  $[v.pre, v.post]$  完全包含在  $[u.pre, u.post]$  内。这种情况下， $v$  是  $u$  的后代。

## 推论 3.

给定有向图  $G = (V, E)$  和其一颗深度优先搜索树，对于任意一条边  $e = (u, v) \in E$ ：

- $e$  是回边当且仅当  $v.pre < u.pre < u.post < v.post$ .
- $e$  是前向边或者树边当且仅当  $u.pre < v.pre < v.post < u.post$ .
- $e$  是横跨边当且仅当  $v.pre < v.post < u.pre < v.post$ .



## 算法: DFS

**输入:** 无向图或者有向图  $G = (V, E)$

**输出:** 相应深度优先搜索树中关于顶点的前序和后序

- 1:  $\text{predfn} \leftarrow 0, \text{postdfn} \leftarrow 0$
- 2: **for** each vertex  $v \in V$  **do**
- 3:     Mark  $v$  unvisited
- 4: **for** each vertex  $v \in V$  **do**
- 5:     **if**  $v$  is unvisited **then**
- 6:          $\text{dfs}(v)$

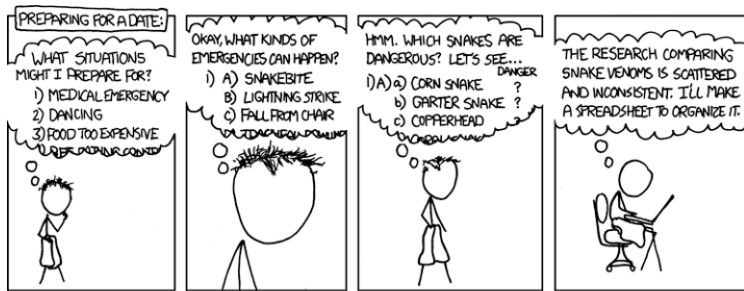
**过程:**  $\text{dfs}(v)$

- 7: Mark  $v$  visited
- 8:  $\text{predfn} \leftarrow \text{predfn} + 1$
- 9: **for** each edge  $(v, w) \in E$  **do**
- 10:     **if**  $w$  is unvisited **then**
- 11:          $\text{DFS}(w)$
- 12:  $\text{postdfn} \leftarrow \text{postdfn} + 1$

给定图  $G = (V, E)$ ,  $n = |V|$ ,  $m = |E|$ , 则有:

图的表示方法	时间	空间
邻接表	$O(n + m)$	$O(n)$
邻接矩阵	$O(n^2)$	$O(n)$

表: 深度优先搜索的时间复杂度



I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

如何判断图没有回路，即判断图中是否有一个圈？

---

其深度优先搜索树中没有回边！

### 关于圈

图中的圈是个非常有意思的对象，从圈的种类来说我们有所谓的欧拉回路、哈密顿回路等；我们也可以利用圈的性质去判断图的种类，比如是否存在奇圈可以帮助我们判断一个图是否是二分图。我们后续会有更进一步的讨论。

一个略微有点挑战的问题，**给定一张图，能如何数出所有的圈？**

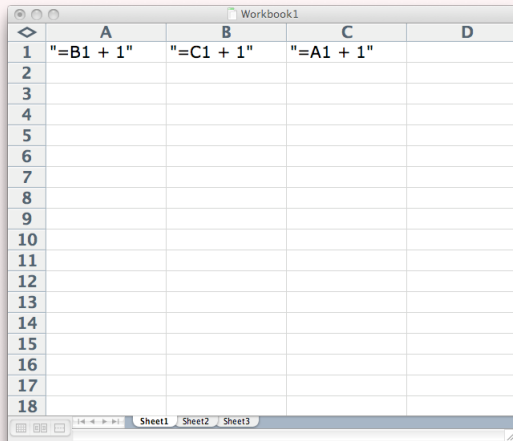
### 没有圈的的图-有向无环图 (Directed Acyclic Graph,DAG)

没有圈的有向图被称作**有向无环图**，简称 DAG。



有向无环图是图中一类非常简单的图，但是其却有着非常重要的应用。

## 例 4.



The screenshot shows a spreadsheet window titled "Workbook1" with columns A, B, C, and D, and rows 1 through 18. The following table represents the data in the spreadsheet:

	A	B	C	D
1	"=B1 + 1"	"=C1 + 1"	"=A1 + 1"	
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

The spreadsheet interface includes a status bar at the bottom with sheet tabs for "Sheet1", "Sheet2", and "Sheet3".

有向无环图可以反映事件的**优先次序**，其一个重要应用就是**拓扑排序**。

## 定义 5

[拓扑排序].

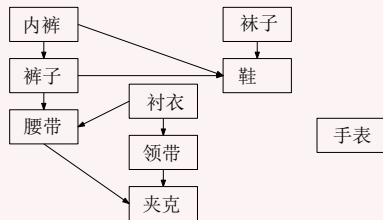
对于一个 DAG  $G = (V, E)$ ，其拓扑序是一个顶点的排列  $(v_1, v_2, \dots, v_n)$ ，使得对于任意一条边  $(v_i, v_j) \in E$ ，都有  $i < j$ 。

## 例 6.

下图描述了 Bumstead 教授每天早上穿衣服的次序图。

其穿着可能有如下顺序：

- 1 袜子 → 内裤 → 裤子 → 腰带 → 衬衣 → 领带 → 夹克 → 手表 → 鞋子。
- 2 内裤 → 裤子 → 袜子 → 衬衣 → 腰带 → 领带 → 夹克 → 手表 → 鞋子。



在上述顺序中，显然只有满足图上拓扑序的序列 2 是合理的。

## 引理 7.

对于有向无环图  $G = (V, E)$  中的任何一条边  $(u, v)$  和它的 DFS 树, 都有  $v.post < u.post$ 。

用 DFS 可以很快的给出一个有关拓扑序的算法:

- `postdfn` 的顺序实际上定义了图中的一个拓扑序的逆序。

## 算法: `TopoSort(G)`

**输入:** 有向无环图  $G = (V, E)$

**输出:** 关于顶点的拓扑排序

- 1: 调用 `DFS(G)` 计算得到  $G$  深度优先搜索的 `post` 序列
- 2: 根据 `post` 序列逆序输出顶点序列

显然这是一个  $O(|V| + |E|)$  的算法。

在无向图中如果存在一条  $u$  到  $v$  的边，则也存在  $v$  到  $u$  的一条边。有向图中则显得麻烦许多，我们引入**强连通**的概念。

## 定义 8 [强连通分量 (Strong Connected Component, SCC)].

对于有向图  $G = (V, E)$ ，如果对于任意一对顶点  $u$  和  $v$ ，都存在一条从  $u$  到  $v$  的路径，那么称  $G$  是强连通的。 $G$  的一个强连通分量是  $G$  的一个极大强连通子图。

## 强连通分量的作用

强连通分量可以视作一个巨大的点，因为一个强连通分量内部的所有点都是相互可达的。从而任何一个有向图可以看成其强连通分量作为点构成的有向无环图。

强连通分量指的是对于其中任两个点  $u$  和  $v$  中，如果  $u$  能有一条路径到  $v$ ， $v$  也能有一条路径到  $u$ ，则  $u$  和  $v$  就是互相连通的。

- $u$  到  $v$  是好判断的，在 DFS 过程中如果  $u$  能到  $v$ ，则  $u$  一定是  $v$  的祖先。
- 怎么同时保证  $v$  到  $u$ ?

---

我们可以将图  $G$  的边全部**翻转**过去!

### 逆图

给定有向图  $G = (V, E)$ ，图  $G$  的逆图定义为  $G^R = (V, E^R)$ ，其中  $E^R = \{(u, v) | (v, u) \in E\}$ 。

### 深度优先搜索的性质

给定一个有向图  $G$  和其一颗深度优先的搜索树

1. 如果  $(u, v) \in E$  但不存在  $v$  到  $u$  的一条路径，则  $u.post > v.post$ .
2. 令  $C, C'$  是  $G$  的两个强连通分量，如果存在一条由  $C$  中的点到  $C'$  中的点的边，则所有  $C$  中的点在 DFS 树中的最大  $post$  值要大于  $C'$  中的点在 DFS 树中的最大  $post$  值。

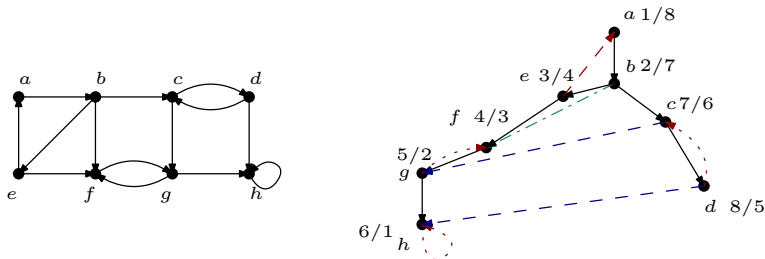
---

上述性质帮助我们得出一个求强连通分量的算法：

- 运用 DFS 获取按  $post$  值大小排列的顶点序列。
- 按照这个序列在其逆图上调用 DFS，每个点能访问到的点即其原图上的强连通分量。

## 一个例子 (I)

让我们来看如下的例子。下图是一个有向图和其一个深度优先搜索树。



其按 post 降序的顶点序列为:  $a, b, c, d, e, f, g, h$

我们现在按照这个序列对其进行逆图的 DFS。



- 依据上面的顺序，我们从 a 开始进行 DFS，寻找到  $abe$  后，我们发现  $e$  已经被访问过了，所以  $abe$  是一个强连通分量。
- 下一个是  $c$ ，以此类推，我们最终得到了所有的强连通分量  $\{a, b, e\}$ ,  $\{c, d\}$ ,  $\{f, g\}$ ,  $\{h\}$ .



## 算法: $SCC(G)$

输入: 有向图  $G = (V, E)$

输出:  $G$  中的强连通分量

- 1: 在  $G$  中调用  $DFS(G)$ , 计算得到  $post$  序列
- 2: 计算  $G^R$
- 3: 在  $G^R$  中按照  $post$  序列逆序调用  $DFS(G^R)$ , 每个进行  $DFS$  搜索的顶点以及其访问到的顶点构成  $G$  的一个强连通分量。

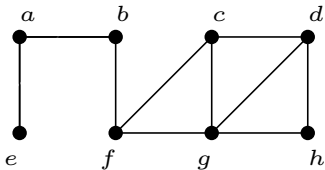
## 额外说明

- 这是一个  $O(|V| + |E|)$  时间的算法。
- 该算法也被称作 **Kosaraju-Sharir 算法**。

## ► 广度优先搜索

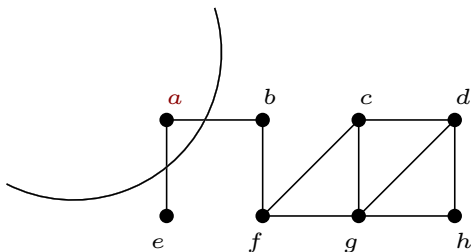
## 广度优先搜索 (Breadth First Search)

广度优先搜索 (BFS) 则可以理解成是一层层展开广撒网的搜索过程。



# 广度优先搜索 (Breadth First Search)

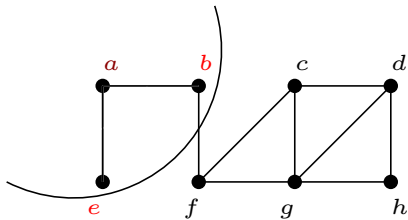
广度优先搜索 (BFS) 则可以理解成是一层层展开广撒网的搜索过程。



# 广度优先搜索 (Breadth First Search)

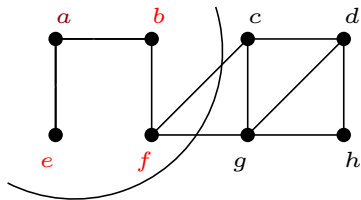


广度优先搜索 (BFS) 则可以理解成是一层层展开广撒网的搜索过程。



## 广度优先搜索 (Breadth First Search)

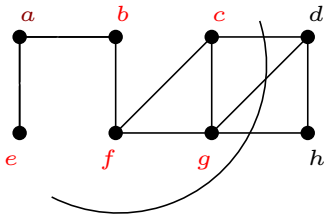
广度优先搜索 (BFS) 则可以理解成是一层层展开广撒网的搜索过程。



# 广度优先搜索 (Breadth First Search)

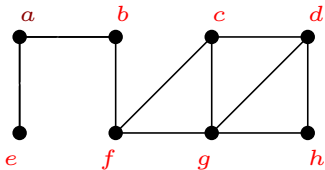


广度优先搜索 (BFS) 则可以理解成是一层层展开广撒网的搜索过程。



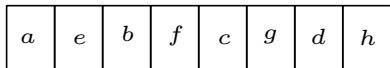
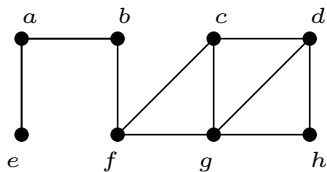
## 广度优先搜索 (Breadth First Search)

广度优先搜索 (BFS) 则可以理解成是一层层展开广撒网的搜索过程。

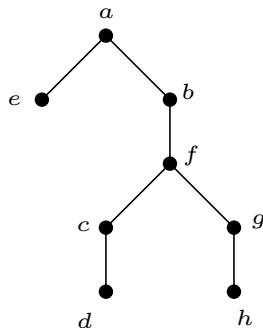




BFS 的树就如同对根节点距离的排列，第  $i$  层的就是距离为  $i$  的点。



BFS 树



在 BFS 中，顶点的顺序就是个先进先出的队列顺序。



## 算法: BFS

输入: 无向图或者有向图  $G = (V, E)$

输出: 相应广度优先搜索树中顶点的编号

- 1:  $bf_n \leftarrow 0$
- 2: **for** each vertex  $v \in V$  **do**
- 3:     Mark  $v$  unvisited
- 4: **for** each vertex  $v \in V$  **do**
- 5:     **if**  $v$  is unvisited **then**  $bf_s(v)$

过程:  $bf_s(v)$

- 6:  $Q \leftarrow \{v\}$
- 7: Mark  $v$  visited
- 8: **while**  $Q$  is not empty **do**
- 9:      $u \leftarrow pop(Q)$ ,  $bf_n \leftarrow bf_n + 1$
- 10:    **for** each edge  $(u, w) \in E$  **do**
- 11:     **if**  $w$  is unvisited **then**
- 12:        $push(Q, w)$
- 13:       Mark  $w$  visited

给定图  $G = (V, E)$ ,  $n = |V|$ ,  $m = |E|$ , 则有:

图的表示方法	时间	空间
邻接表	$O(n + m)$	$O(n)$
邻接矩阵	$O(n^2)$	$O(n)$

表: 广度优先搜索的时间复杂度

BFS 的一个最大应用便是寻找最短路径。

如何寻找一个图中  $s$  到  $t$  的最短路径？

---

- DFS 是不行的，因为其可能正好绕了远路找到  $t$ 。
- BFS, **Yes!**

## 定理 9.

给定图  $G = (V, E)$ ， $s$  到  $t$  的最短路径长度等于以  $s$  为起点对  $G$  进行 BFS 过程中  $t$  在树中的深度。

如果边上有了**权重**，我们又该如何寻找到最短路径呢？

### 深度优先搜索

对于深度优先搜索来说, 其向着图尽可能深的方向进行搜索, 仅当没有新的顶点的时候再回退。

- 它存在很好的特性, 比如对于 DAG 的搜索能保证拓扑序。
- 但是其对于两个顶点之间的距离, 可能会通过一条很遥远的路径才能到达。
- 表现行为像栈。

### 广度优先搜索

对于广度优先搜索来说, 其像水波的传播过程, 一层一层的向外扩展, 是一种更宽广、更扁平的搜索过程。

- 当边的权重一致时, 其严格按照到起始点的距离的升序进行搜索。
- 但是对于一些纵向的搜索, 其可能会有很多的冗余。
- 表现行为像队列。



## 本节内容

- 图的基本概念
- 遍历问题
  - 深度优先搜索
  - 广度优先搜索
  - BFS, DFS 的应用