第六章 图（二）

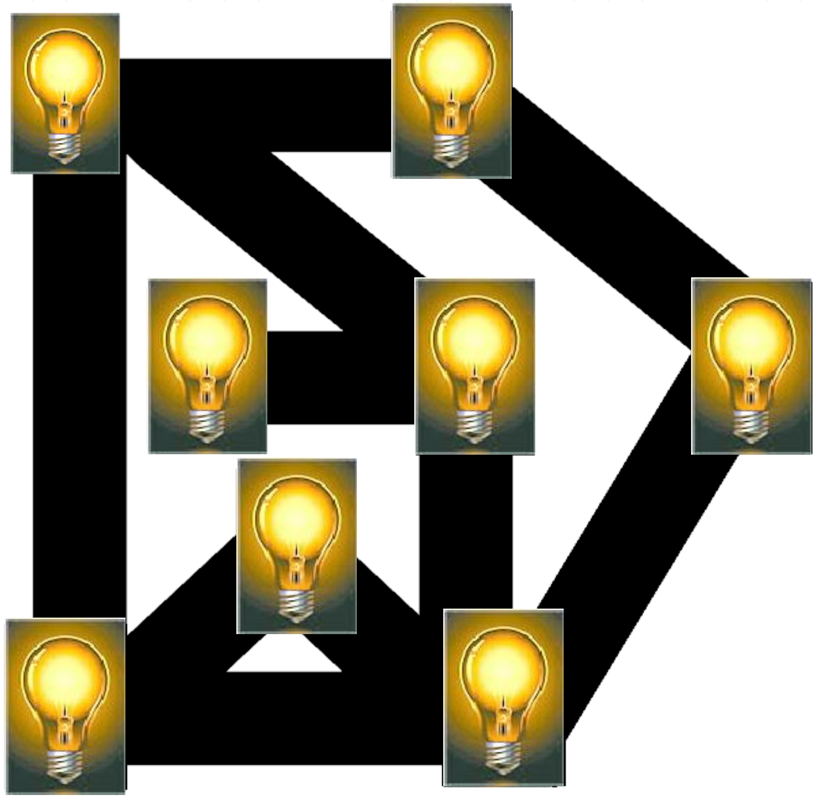
图的遍历

- 从图中某一顶点出发，对图中所有顶点访问一次且仅访问一次

> 图的遍历要解决的关键问题
>
> - 从某个顶点起始可能到达不了其它所有顶点，怎么办？
>
>   ☑ 解决办法：多次调用从某顶点出发遍历图的算法。
>
> - 图中可能存在回路，且图的任一顶点都可能与其它顶点相邻接，在访问完某个顶点之后可能会又回到了曾经访问过的顶点。如何避免某些顶点被重复访问？
>
>   ☑ 解决办法：附设访问标志数组 **visited[n]**
>
> - 在图中，一个顶点可以和其它多个顶点相连，当该顶点访问过后，如何选取下一个要访问的顶点？
>
>   ☑ 解决办法：深度优先搜索（**Depth-First Search**）和广度优先搜索（**Broadth-First Search**）

图的遍历（cont.）

1. 深度优先搜索（**Depth First Search, DFS**）



---类似于树的先序遍历

DFS是一个递归过程（需要用到栈）

图的遍历（cont.）

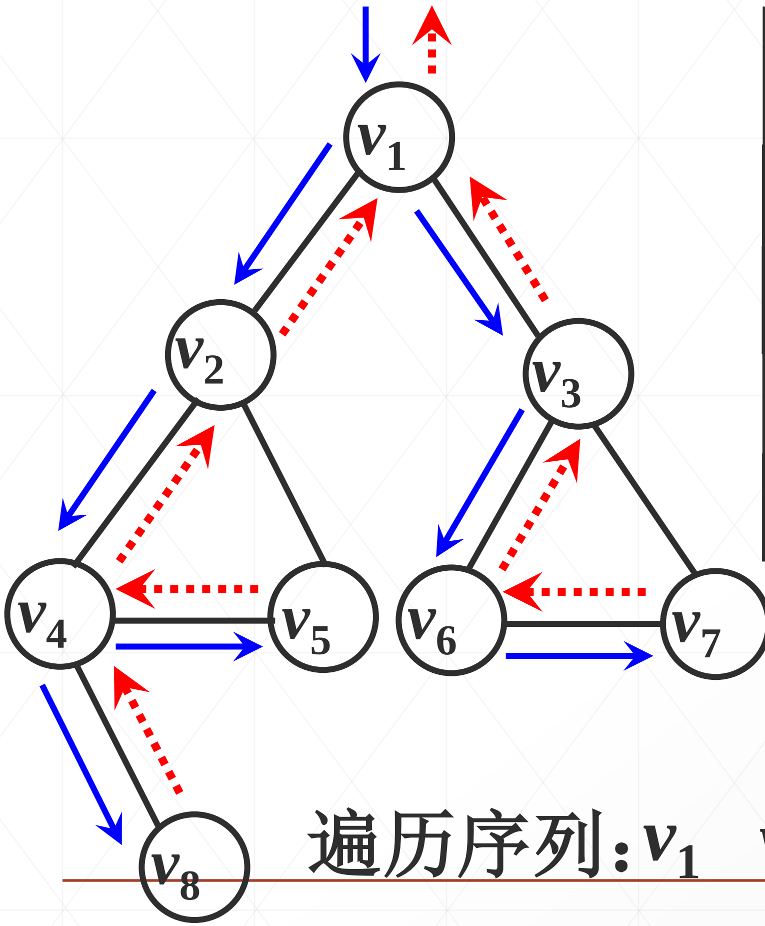
1. 深度优先搜索

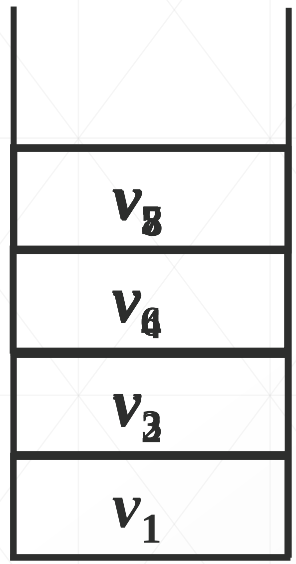
- 设图G的初态是所有顶点都未访问过, 即visited []设置为False
- 从G中任选一个顶点 *v*为初始出发点，
  ① 首先访问出发点*v*，且设visited [*v*]=TRUE;
  ② 然后从 *v*出发，依次考察与 *v*相邻的顶点 *w*；若visited [*w*]= FALSE, 则以 *w* 为新的出发点递归地进行深度优先搜索，直到图中所有与源点 *v* 有路径相通的顶点均被访问为止;
  ③ 若此时图中仍有未被访问过的顶点，则另选一个“未访问过”的顶点作为新的搜索起点，重复上述过程，直到图中所有顶点都被访问过为止。

图的遍历（cont.）

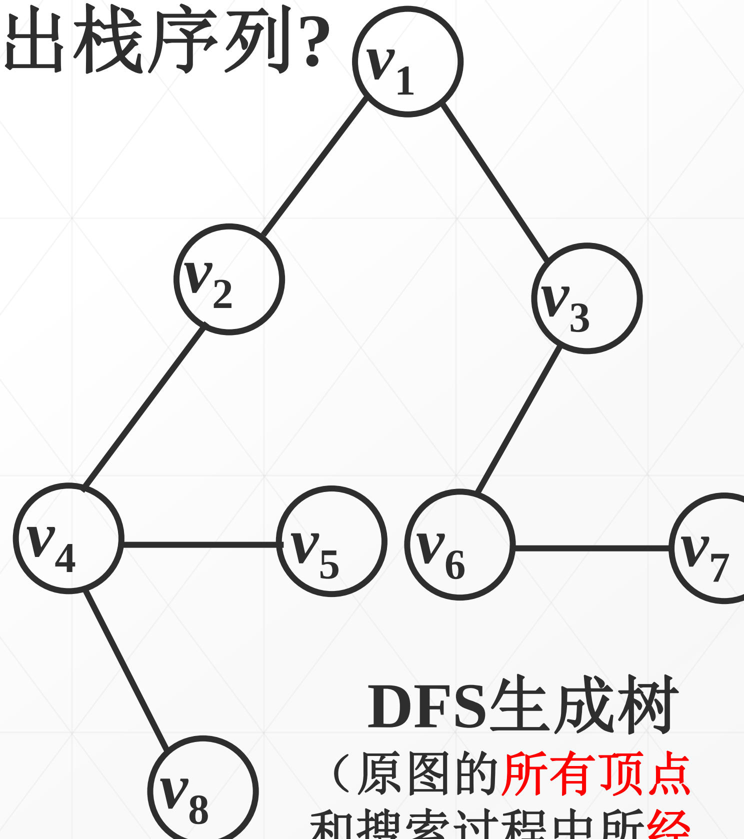
1. 深度优先搜索示例

- 深度优先搜索序列?入栈序列?出栈序列?





栈



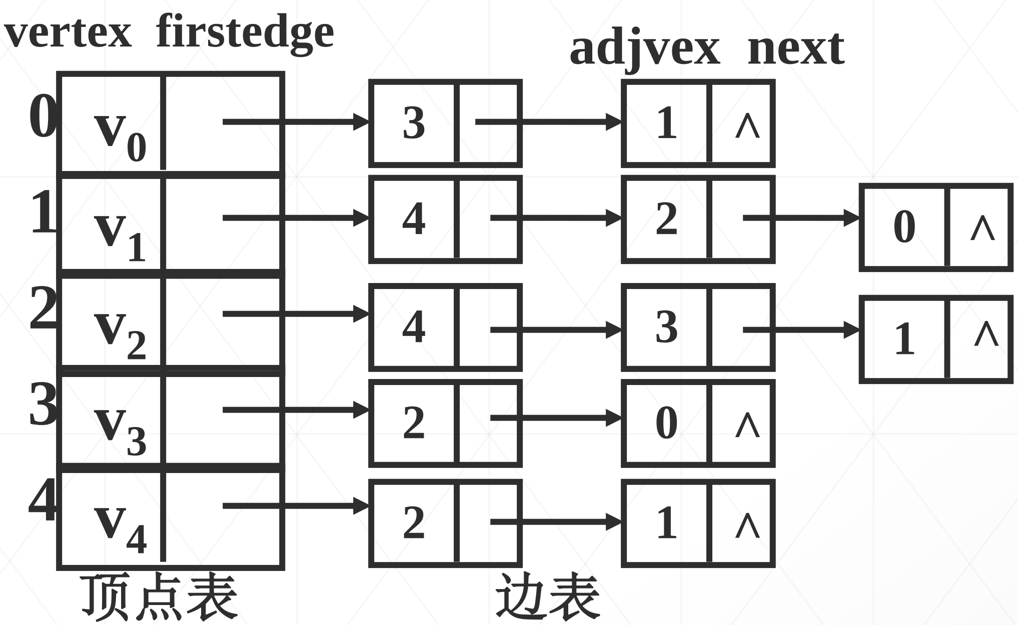
DFS生成树

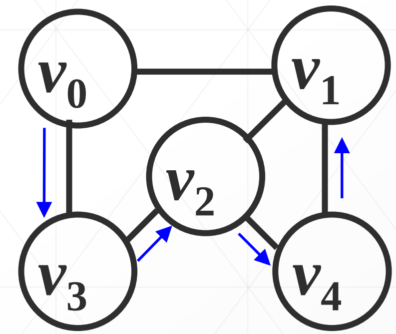
（原图的所有顶点和搜索过程中所经过的边构成的子图）

遍历序列：v_1 v_2 v_4 v_5 v_8 v_3 v_6 v_7

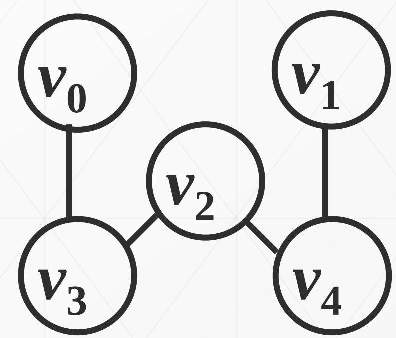
图的遍历（cont.）

1. 深度优先搜索举例（邻接表）





➢ **DFS序列为** $\underline{v_0，\ v_3，\ v_2，\ v_4，\ v_1}$



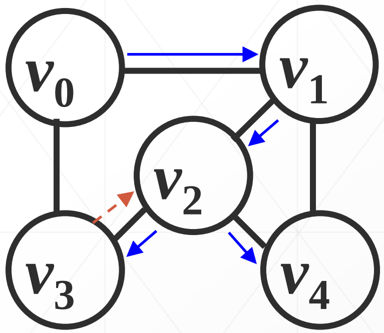
DFS生成树

图的遍历（cont.）

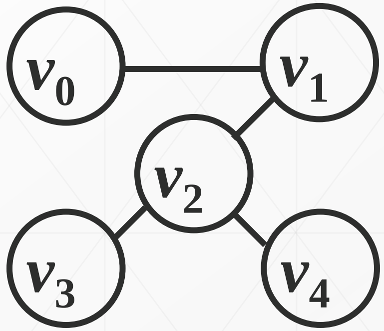
1. 深度优先搜索举例（邻接矩阵）

$$
\begin{array}{c|c}
 & \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\
\begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} &
\begin{bmatrix}
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0
\end{bmatrix}
\end{array}
$$

边表



- ➢ **DFS序列为**$\underline{v_0，v_1，v_2，v_3，v_4}$



DFS生成树

图的遍历（cont.）

1. 深度优先搜索的特点

- 是一个递归的过程，是尽可能对纵深方向上进行搜索，故称**深度优先搜索**。
- 深度优先搜索过程中，根据访问顺序得到的顶点序列，称为**DFS序列。**
- 深度优先搜索结果不唯一，是由该图的存储结构决定的。
- 若有n个顶点、e条边，时间复杂度是

  1）用邻接表存储图，有$O(n+e)$

  2）用邻接矩阵存储图，有$O(n^2)$

图的遍历（cont.）

2. 广度优先搜索

- 设图G的初态是所有顶点都未访问过, 即visited []设置为False
- 从G中任选一个顶点 v为源点，

① 首先访问出发点v，且设visited [v]=TRUE;

② 接着依次访问所有与v相邻的顶点w_1，w_2…w_t;

③ 然后依次访问与w_1，w_2… w_t相邻的所有未访问的顶点;

④ 依次类推，直至图中所有与源点v有路相通的顶点都已访问过为止;

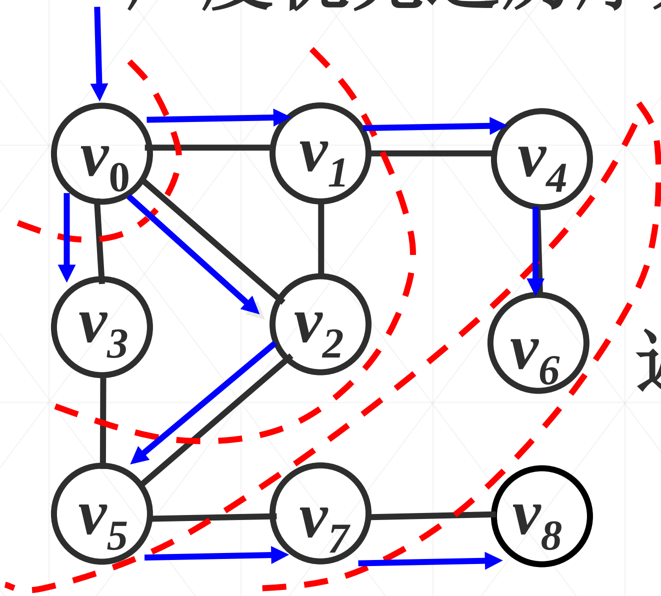
⑤此时，从 v开始的搜索结束，若G是连通的，则遍历完成;若G是非连通的，则在图中另选一个尚未访问的顶点作为新源点，继续上述搜索过程，直到G中的所有顶点均已访问为止。

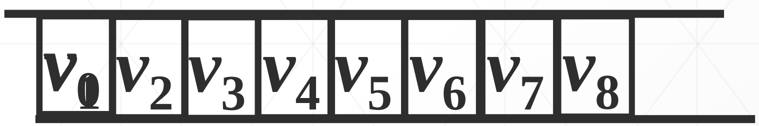
图的遍历（cont.）

--BFS是一种分层搜索方法
--每走一步可能访问一批顶点，
不存在往回退的情况
--BFS不是一个递归的过程

2. 广度优先搜索示例

- 广度优先遍历序列?入队序列?出队序列?

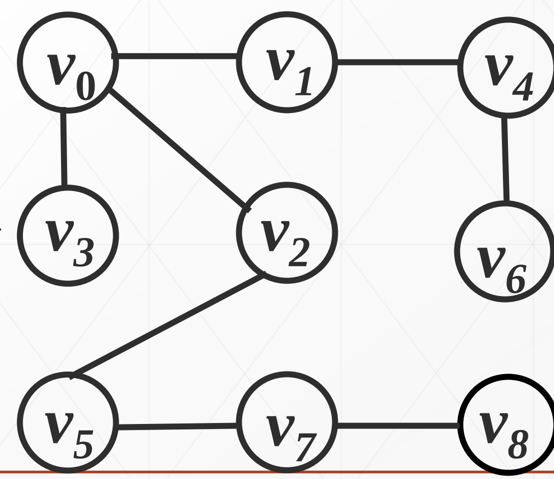




队列

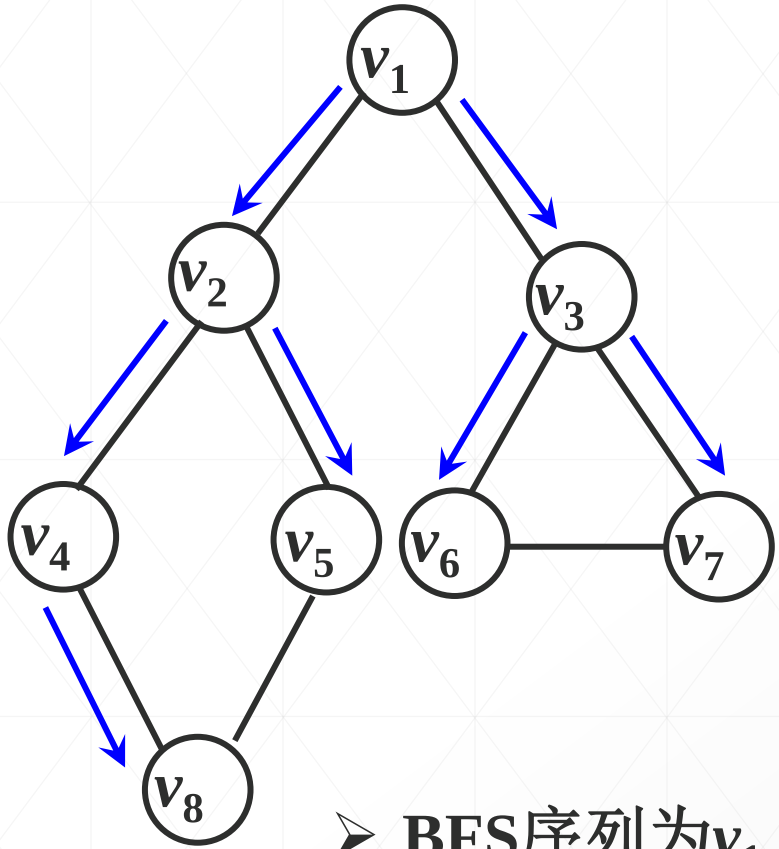
遍历序列: v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

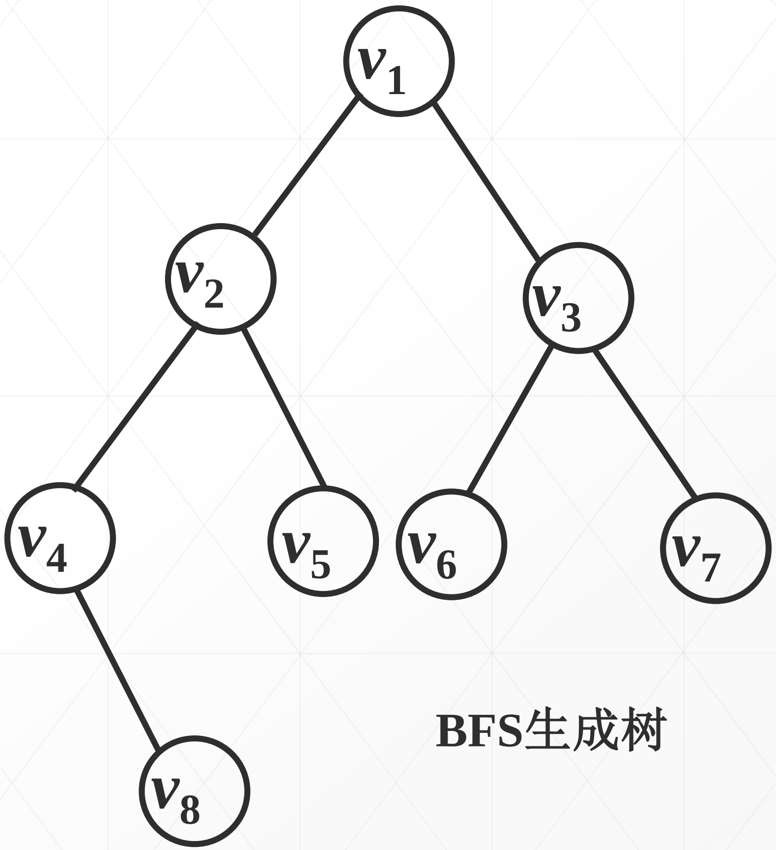
BFS生成树



图的遍历（cont.）

2. 广度优先搜索举例



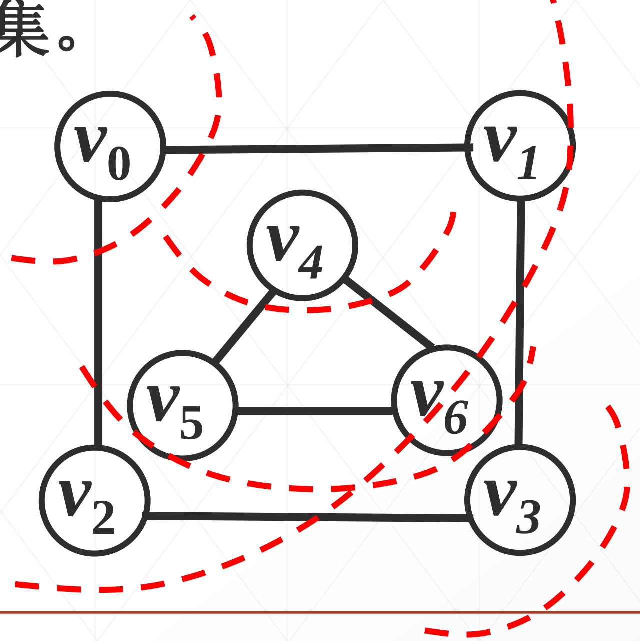


BFS生成树

- **BFS序列为**v_1，v_2，v_3，v_4，v_5，v_6，v_7，v_8

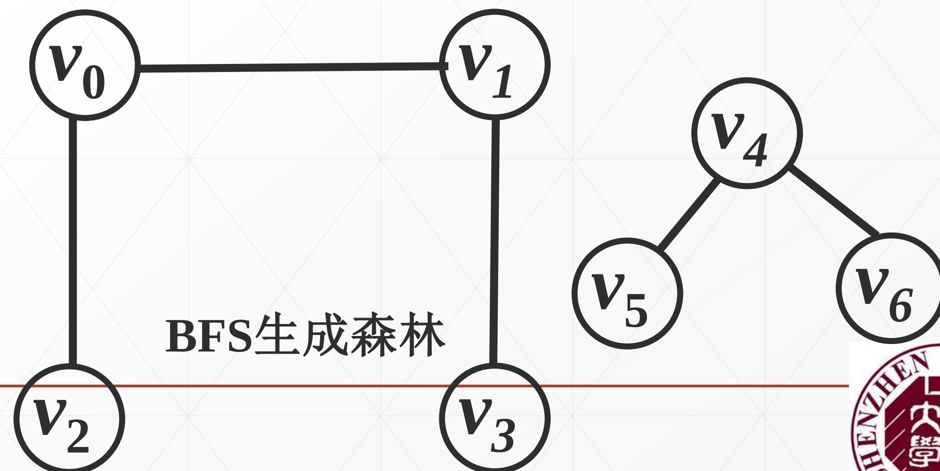
图的连通性问题

- 在对无向图进行遍历时：
  - 若是连通图，仅需从图中任一顶点出发，就能访问图中所有顶点；
  - 若是非连通图，需从图中多个顶点出发，每次从一个新顶点出发进行搜索得到的顶点序列恰为其各个连通分量的顶点集。



- BFS序列为v_0，v_1，v_2，v_3，v_4，v_5，v_6

v_4为另一个起始点



图的连通性问题(cont.)

- 无向图的连通分量
  - 若从无向图的每一个连通分量中的一个顶点出发进行**DFS**或**BFS**遍历，可求得无向图的所有连通分量的生成树（**DFS**或**BFS**生成树）；
  - 所有连通分量的生成树组成了非连通图的生成森林

最小生成树 (Minimum Spanning Tree)

- 什么是最小生成树?
  - 是一棵树
    - 无回路
    - n个顶点一定有$n-1$条边
  - 是生成树
    - 包含全部顶点
    - $n-1$条边都在图里
  - 边的权重和最小

最小生成树存在 $\leftrightarrow$ 图连通

最小生成树(Minimum-Cost Spanning Tree，MST)是代价最小的无向连通网的生成树

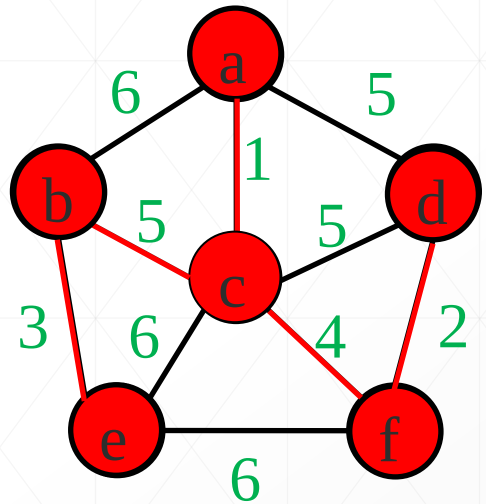
最小生成树 (cont.)

- 贪心算法
  - 什么是“贪”：每一步都要最好的
  - 什么是“好”：权重最小的边
  - 需要约束（构造最小生成树的准则）：
    - 只能用图里有的边；
    - 只能正好用掉$n-1$条边连接结图中的n个顶点；
    - 不能使用产生回路的边；
    - 各边上的权值的总和达到最小

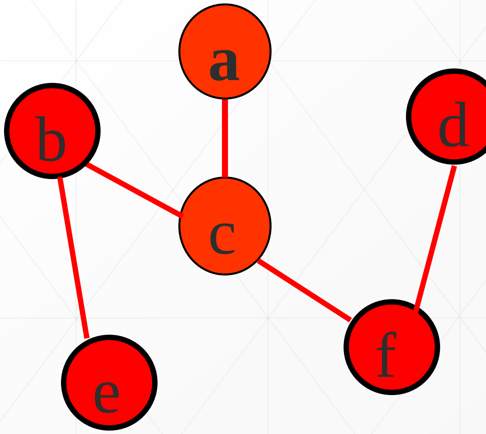
最小生成树 (cont.)

- Prim算法 --让一棵小树长大

距离此最小生成树最近的顶点是树中所有顶点都已加入到最小生成树中



当前最小生成树为：



最小生成树 (cont.)

➢ **Prim算法的基本思想**

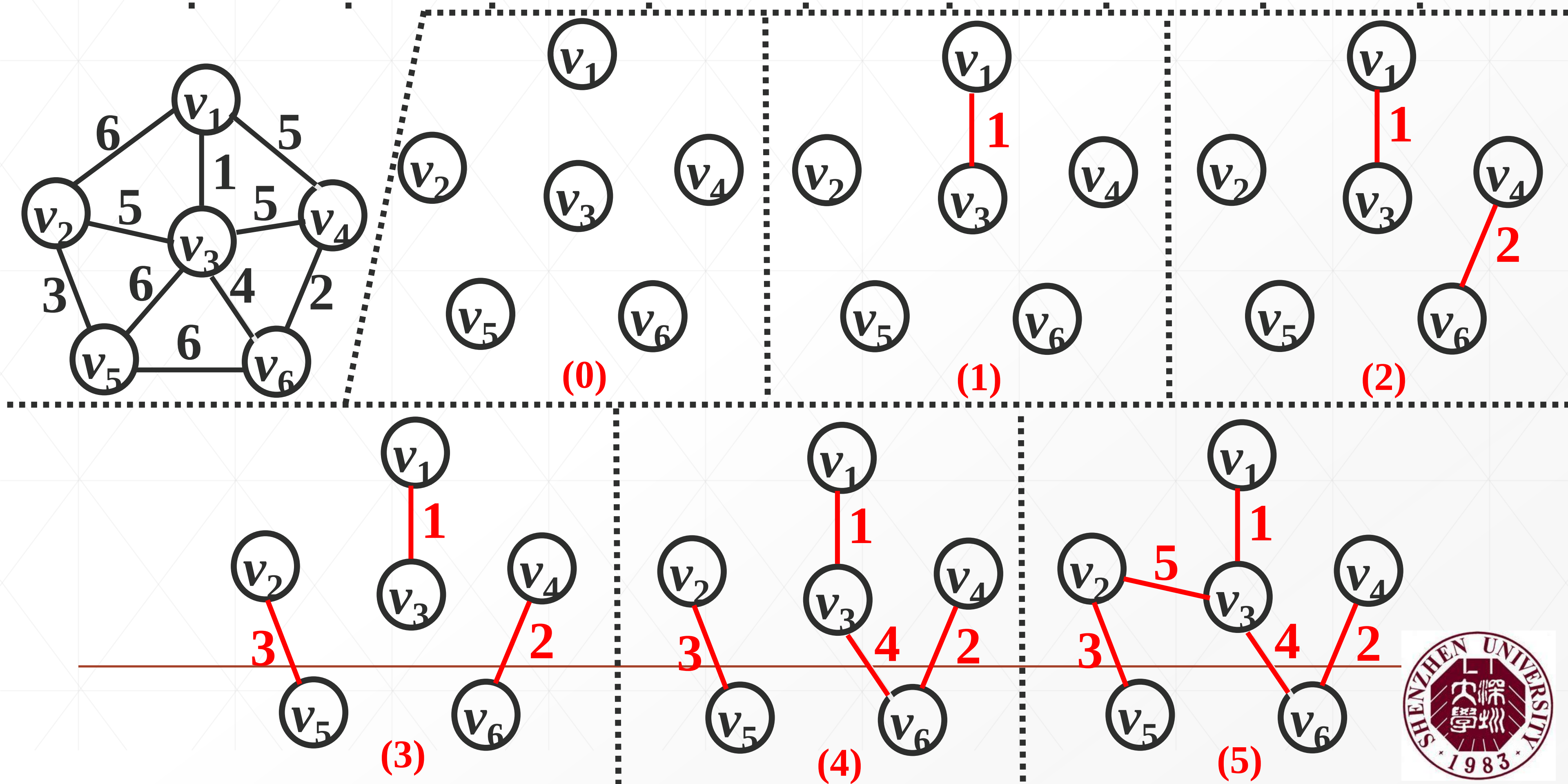
- 假设N=(V, E)是连通网，TE是N上最小生成树中边的集合

1. 首先从集合V中任取一顶点(如顶点v_0)放入集合U中。这时U={v_0}，边集TE={ }；
2. 在所有u∈U，v∈V-U的边(u，v) ∈E中找出**权值最小的边**，将该边加入TE，并将顶点v加入集合U；
3. 重复2直到U=V为止。

最小生成树 (cont.)

➢ **Kruskal算法 --将森林合并成树**

(v_1,v_3)	(v_4,v_6)	(v_2,v_5)	$(v3,v_6)$	(v_1,v_4)	(v_3,v_4)	(v_2,v_3)	(v_1,v_2)	(v_3,v_5)	(v_5,v_6)
1	2	3	4	5	5	5	6	6	6



最小生成树 (cont.)

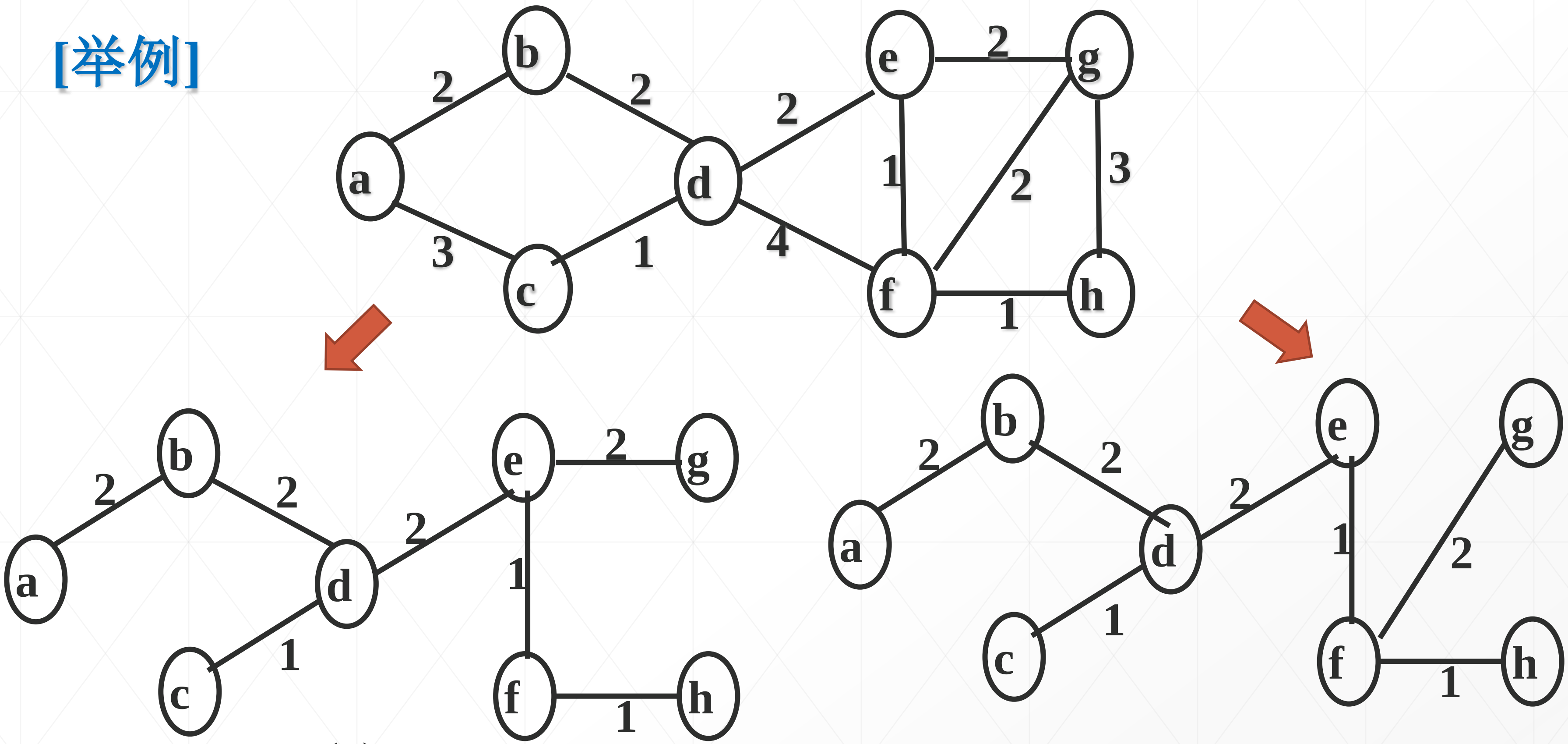
➢ **Kruskal算法的基本思想**

- 假设N=(V, E)是连通网，

1. 初始化：非连通图T={V, {}}, 图中的每个顶点自成一个连通分量；
2. 在E中选择一条权值最小的，且其两个顶点分别依附不同的连通分量的边，将其并入T中；
3. 重复2直到T中所有顶点都在同一个连通分量上

最小生成树 (cont.)

[举例]



(a) (b)

- 当各边有相同权值时，由于选择的任意性，产生的生成树可能不唯一；
- 当各边的权值不相同时，产生的生成树是唯一的。