数据结构

第九讲 图

孙猛

http://www.math.pku.edu.cn/teachers/sunm

2017年10月30日


Bern
Bolzano
Belluno
Treviso
Venice
Ferrara
Ravenna
Bosnia and Hercegovina
Pisa
Elba
Vatican City
Latina
Benevento
Naples
Salerno
Capri
Sea
Trapani
Strait of Sicily
Agrigen
Ragusa
Siracuse
Sea
Algeria
Tunis
Tunisia
Pantelleria (IT.)
Malta
Isole
0 100 km
0 100 mi
44°
40°
36°
16°

课程内容

- 图的基本概念及抽象数据类型
- 存储表示
- 图的周游

基本概念及抽象数据类型

- 比树更一般、更复杂的结构——图。
- 图中的结点(图中通常称为顶点) 之间的关系是任意的，即不再限制结点的前驱和后继个数。

- 图由顶点的非空有穷集合V和边的集合E组成，记为$G=(V, E)$。
- 每条边就是一个顶点的偶对，所以E也就是V上的关系 $E \subseteq V \times V$ 。

有向图和无向图

- 有向图：有向图中的边有方向，边是顶点的有序对。
  - 有序对用尖括号表示。$<v_i, v_j>$ 表示从 v_i 到 v_j 的有向边，v_i 是边的始点，v_j是边的终点。称顶点v_i邻接到顶点v_j。也称这条边与顶点 v_i 、v_j关联。
- 无向图：无向图中的边没有方向，是顶点的无序对。
  - 无序对用圆括号表示，(v_i, v_j)和(v_j, v_i)表示同一条无向边。称顶点v_i和顶点v_j邻接。也称这条边与顶点v_i、v_j关联。
- 完全图：任意两个顶点之间都有边的有向图(或无向图)。
  - n 个顶点的无向完全图有 $n*(n-1)/2$ 条边;
  - n 个顶点的有向完全图有$n*(n-1)$ 条边。

数据结构对简单图的两个限制

(1)不考虑顶点到自身的边，若 (v_i, v_j) 或$<v_i, v_j>$ 是 G 的边，则$v_i \neq v_j$；

(2)顶点间没有重复出现的边，若(v_i, v_j) 或 $<v_i, v_j>$ 是 G 的边，则它是唯一的。

顶点的度

- 顶点的度$D(v_i)$：与一个顶点邻接的边的条数。
- 对于有向图，顶点的度分为入度和出度，分别表示以该顶点为终点和始点的边的条数。

$$D(v_i) = ID(v_i) + OD(v_i)$$

- 无论对有向图还是无向图，顶点数n、边数e和顶点度数满足下面关系：

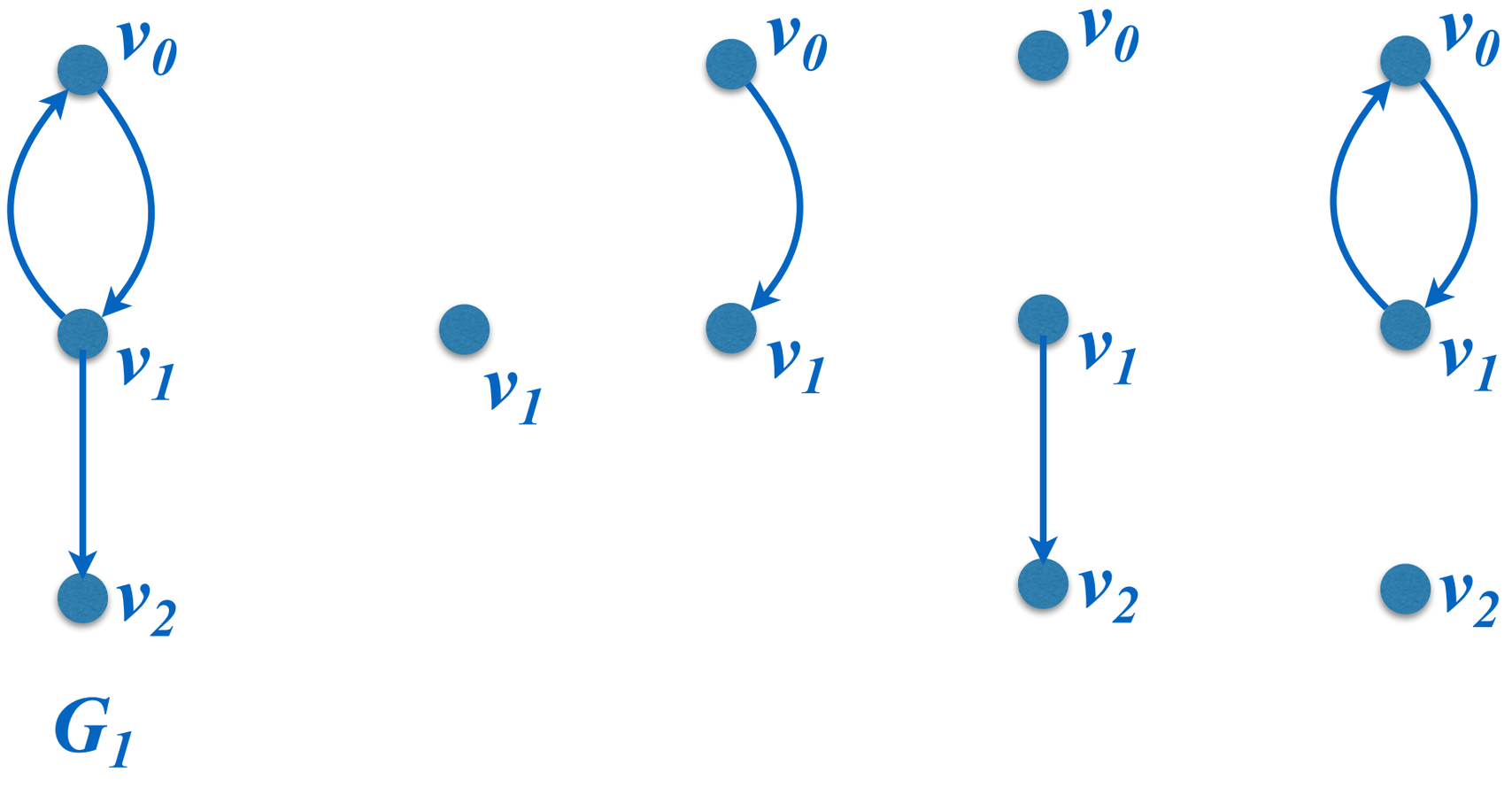
$$e = \frac{1}{2}\sum_{i=1}^{n} D(v_i)$$

路径与路径长度

- 路径：对$G=(V,E)$，若存在顶点序列$v_{i0},v_{i1},\cdots,v_{i(m)}$，使$(v_{i0},v_{i1}),(v_{i1},v_{i2}),\ldots,(v_{i(m-1)},v_{i(m)})$都在$E$中(对有向图是$<v_{i0},v_{i1}>$，$<v_{i1},v_{i2}>$，…，$<v_{i(m-1)},v_{i(m)}>$都在$E$中)，则称从顶点$v_{i0}$到$v_{i(m)}$存在一条路径$<v_{i0},v_{i1},\ldots,v_{i(m)}>$，也称为从顶点$v_{i0}$到$v_{i(m)}$是可达的；
- 路径长度：路径上的边数；
- 回路(环)：起点和终点相同的路径。如果其中的其他顶点均不相同，则称为简单回路；
- 简单路径：内部不包含环的路径，即，路径上的顶点除起点和终点可能相同外，其它顶点均不相同。

子图

- 对两个图$G=(V,E)$和$G^*=(V^*,E^*)$，如果V^*是V的子集，而且E^*是E的子集，那么就称G^*是G的子图。



有根（有向）图

- 有向图G中若存在一顶点v，从v有路径可以到达图G中其它所有顶点，则称G为有根图，v称为图G的根。

- 有根图中的根可能不唯一。

- 树是有唯一根的有根图（且所有顶点的入度至多为1）。

连通（无向）图

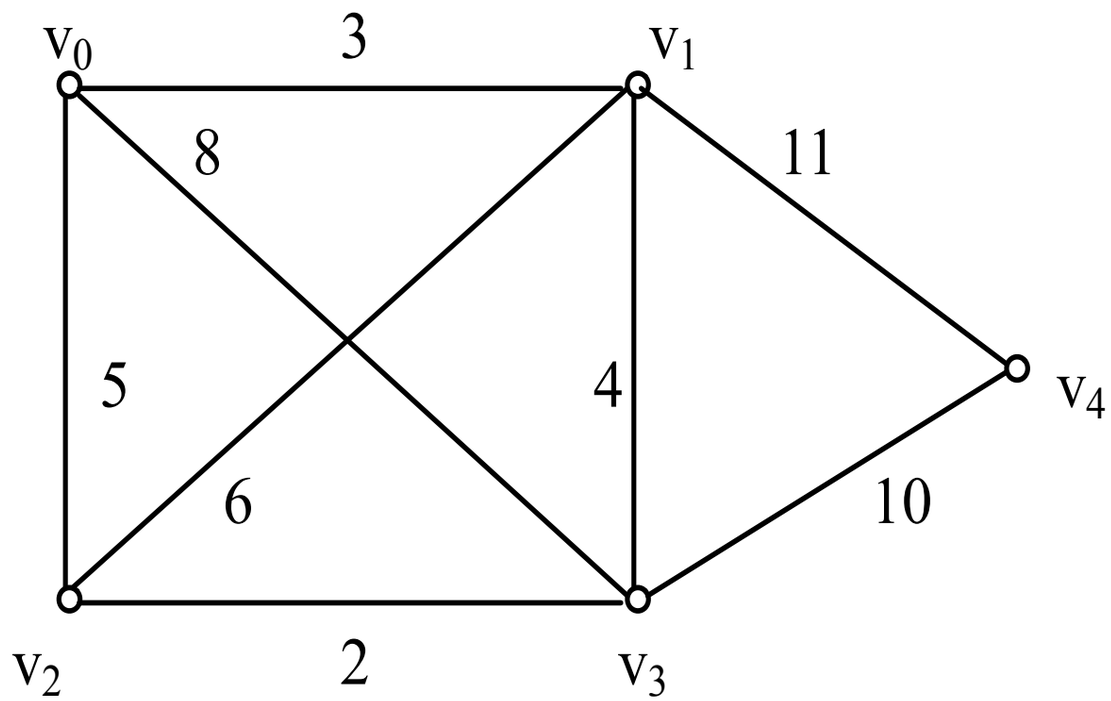
- **连通**：若无向图G中存在从v_i到v_j的路径（显然它也是从v_j到v_i的路径)，则称v_i与v_j连通。
- **连通图**：如果无向图G中的任意两个不同顶点v_i和v_j都连通（即存在路径），则称G为连通图。
- **连通分量**：无向图G中的一个极大连通子图（不存在包含它的更大的连通子图）称为G的一个连通分量。若G不连通，它的连通分量将多于一个。

有向图的连通性

- **强连通图**：如果有向图G中任意两个不同顶点v_i和v_j之间都存在从v_i到v_j以及从v_j和v_i的路径，则称G是强连通图；
- **强连通分量**：有向图G的极大强连通子图称为其强连通分量。

带权图和网络

- 若图的每条边都被赋以一个权值，这种图称为**带权图**；
- 带权的连通无向图称为**网络**。权值可用于表示实际应用中的某种与边有关的量。



抽象数据类型

- 有些特殊的图，例如二叉树、树和表等前面已经给了详细的讨论，在那里并没有讨论下面定义的许多运算，因为如果统一用这些运算来规范各种图上的操作，结果可能把原来简单的问题复杂化；
- 由于不同的应用要求，逻辑结构相同的图需要实现的操作可能会有很大的差别，所以它们的抽象数据类型也有很大差别。
- 图是一种十分广泛的结构，图的抽象数据类型难以给出统一的定义。

图的基本操作

- 作为复杂的数据结构，图上可能定义许多操作。下面列举一些可能的操作：
  - 创建图，创建空图，或基于顶点和边的数据创建图
  - 判断一个图是否为空图
  - 顶点的个数（vertex_num）和边的条数
  - 所有顶点的集合 vertices ，所有边的集合 edges
  - 增加一条边 <v1,v2> 或 (v1,v2)，add_edge (v1 ，v2)；是否存在边 <v1,v2> 或 (v1,v2)，get_edge(g , v1 , v2)
  - 顶点 v 的入度和出度（结果用二元序列表示），vdegree (v)
  - 找出顶点 v 相邻边（集合或表），如出边 out_edges(v)
- 遍历操作。与树遍历的最重要差异是：
  - 需要防止再次进入已经遍历过的部分
  - 需要考虑图的连通性问题（如果图不连通，遍历完一个连通分支并不是整个图遍历的结束，还需要继续遍历其他连通分支）

ADT Graph **is**　/*重点关心图中结点和边的处理 */

operations

Graph createGraph (void)

int isNullGraph (Graph g)

Vertex firstVertex (Graph g)

Vertex nextVertex (Graph g , Vertex vi)

Vertex searchVertex (Graph g , Vertex vi)

Graph addVertex (Graph g , Vertex vi)

Graph deleteVertex (Graph g , Vertex v)

Graph deleteEdge (Graph g , Vertex vi , Vertex vj)

Graph addEdge (Graph g , Vertex vi , Vertex vj)

int findEdge (Graph g , Vertex vi , Vertex vj)

Vertex firstAdjacent (Graph g , Vertex v)

Vertex nextAdjacent (Graph g , Vertex vi , Vertex vj)

……

end ADT Graph

图的存储表示

- 图的结构比较复杂，任意两个顶点间都可能存在边
  - 需要表示顶点及顶点间的边，存在很多很多可能的方法
  - 应该根据具体应用和需要做的操作等选择图的表示方法
- 图的最基本表示方法是邻接矩阵表示法
- 表示图中邻接关系的矩阵通常非常稀疏，空间浪费可能很大。很多图中边数与顶点数成线性关系，如中国铁路线路图。为降低空间代价，人们提出了许多其他方法，它们都可看作邻接矩阵的“压缩”版，如：
  - 邻接表表示法
  - 邻接多重表表示法
  - 图的十字链表表示，等

邻接矩阵表示法

图的邻接矩阵表示包括：

- **一个顺序存储顶点信息的顶点表**
- **一个存储顶点间相互关系的关系矩阵。**

具体定义

- **设$G=(V,E)$为具有 n 个顶点的图，关系矩阵是一个 $n \times n$ 的方阵$arcs$，具有以下性质：**

$$arcs[i,j]=\begin{cases}1, \text{若}(v_i,v_j)\text{或}<v_i,v_j>\text{是图}G\text{的边}\\0, \text{若}(v_i,v_j)\text{或}<v_i,v_j>\text{不是图}G\text{的边}\end{cases}$$

- **邻接矩阵只表示图中顶点个数和顶点间联系（边）**
  - **每个顶点对应一个矩阵下标**
  - **通过一对下标可以确定图中一条边的有无**
- **如果顶点本身还有其他信息，需要用另外的方式表示**
  - **例如可以考虑另外用一个顶点表vexs**
  - **可以用与矩阵同样下标引用同一顶点的表元素**

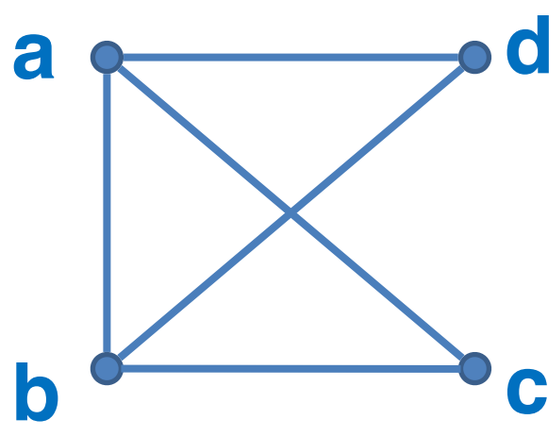
C语言描述

```
typedef char VexType;
typedef float AdjType;
typedef struct {
   VexType vexs[VN];                    /* 顶点信息 */
  AdjType arcs[VN][VN];                 /*关系矩阵*/
}GraphMatrix;
```

例子：无向图G

- **vexs1=[‘a’, ‘b’, ‘c’, ‘d’]**

$$arcs_1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$



例子：有向图G

- **vexs2=[‘a’, ‘b’, ‘c’, ‘d’, ‘e’]**

$$arcs_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

firstAdjacent(G,a)
nextAdjacent(G, a ,b)

图的邻接矩阵表示法的特点

- 无向图的关系矩阵一定是一对称矩阵。
- 无向图的关系矩阵的第 i 行(或第 i 列)非零元素个数为第 i 个顶点的度 $D(v_i)$。
- 有向图的关系矩阵的第 i 行非零元素个数为第 i 个顶点的出度 $OD(v_i)$，第 i 列非零元素个数就是第 i 个顶点的入度$ID(v_i)$。
- 从图的邻接矩阵表示，很容易确定图中任意两个顶点之间是否有边相连。

带权图

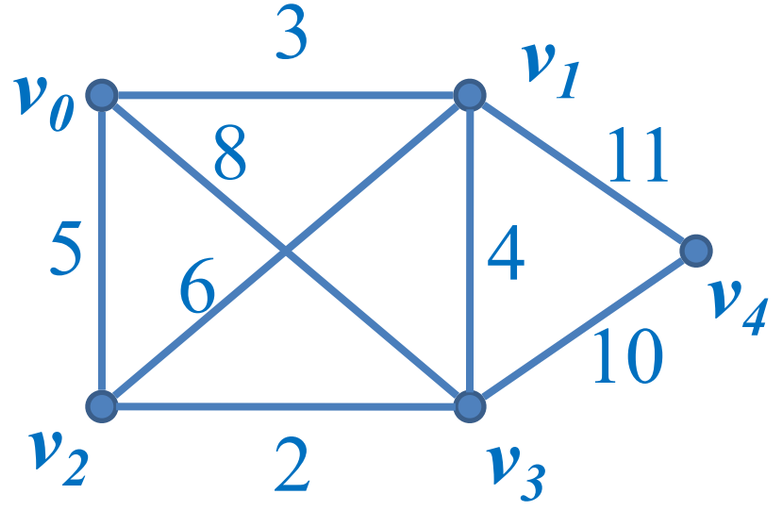
- 如果G是带权图，w_{ij}是边(v_i, v_j)或$<v_i, v_j>$的权，则其关系矩阵的所有对角线的值为0，其它元素定义为：

$$arcs[i,j]=\begin{cases} w_{ij}, 若(v_i,v_j)或<v_i,v_j>是图G的边(i\neq j) \\ \infty, 若(v_i,v_j)或<v_i,v_j>不是图G的边(i\neq j) \end{cases}$$

例子：带权图G

- 下面带权图采用邻接矩阵表示时的关系矩阵为$arcs_3$。

$$arcs_3 = \begin{bmatrix} 0 & 3 & 5 & 8 & \infty \\ 3 & 0 & 6 & 4 & 11 \\ 5 & 6 & 0 & 2 & \infty \\ 8 & 4 & 2 & 0 & 10 \\ \infty & 11 & \infty & 10 & 0 \end{bmatrix}$$



邻接表表示法

邻接表表示法包括两大部分：

- 顺序存储的顶点表
- 与每个顶点相关联的（n个）链式存储的边表

具体表示

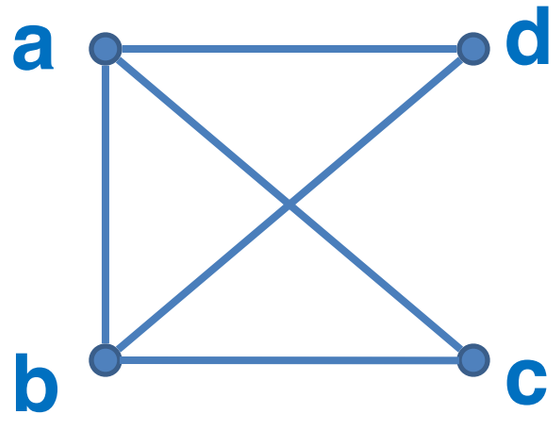
- 顶点表中每个表目对应于图中一个顶点，包括两个字段：
  - 顶点字段(vertex)——存放顶点v_i的信息，
  - 边表指针(edgelist)——存放与vi相关联的边表中的第一个边结点的位置；
- 边表中每个边结点表示的都是与v_i关联的边，包括：
  - 终点字段(endvex)——边的另外一端在顶点表中的位置，
  - 权字段(weight)——存放边的权值（如果不是带权图，则应该省略权字段），
  - 链字段(nextedge)——指向边表的下一个边结点。

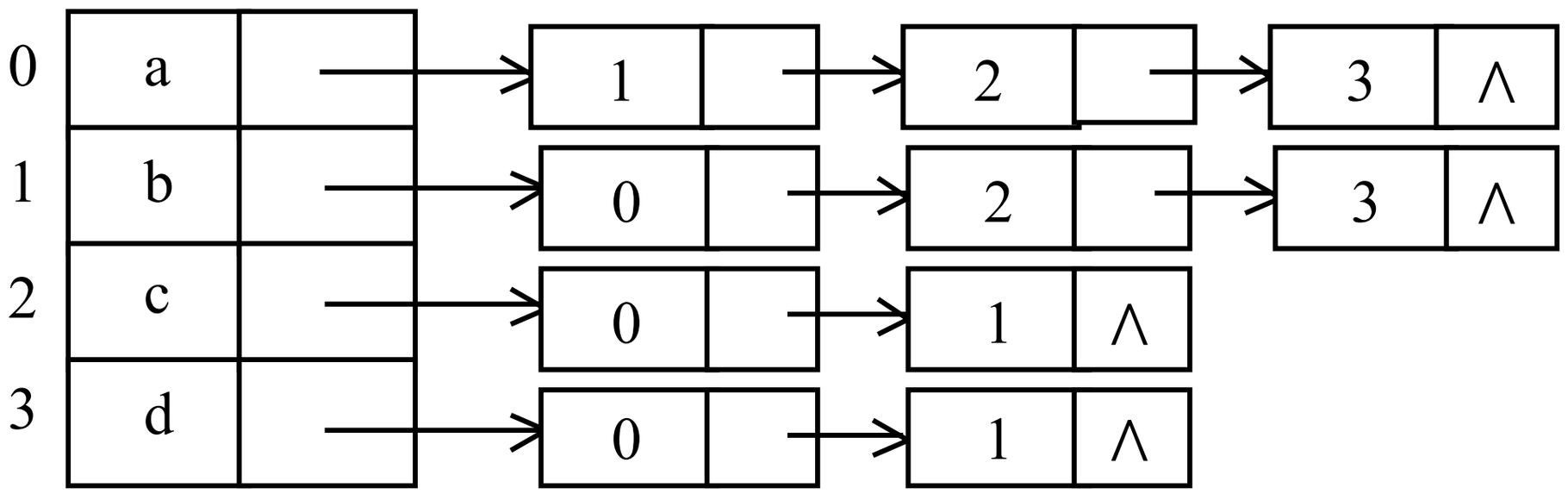
具体表示

```
struct EdgeNode;
typedef struct EdgeNode * PEdgeNode;
typedef struct EdgeNode * EdgeList;
struct EdgeNode{
        int endvex;                 /* 相邻顶点在顶点表中下标 */
        AdjType weight;             /* 边的权，非带权图应该省略 */
        PEdgeNode nextedge;         /* 链字段 */
};                                  /* 边表中的结点 */
typedef struct{
        VexType vertex;             /* 顶点信息 */
        EdgeList edgelist;          /* 边表头指针 */
} VexNode;                          /* 顶点表中的结点 */
typedef struct{
        int n;                      /* 图的顶点个数 */
        VexNode *vexs;              /*顶点表 */
}GraphList;                         /* 图的邻接表表示 */
```

例子：无向图G

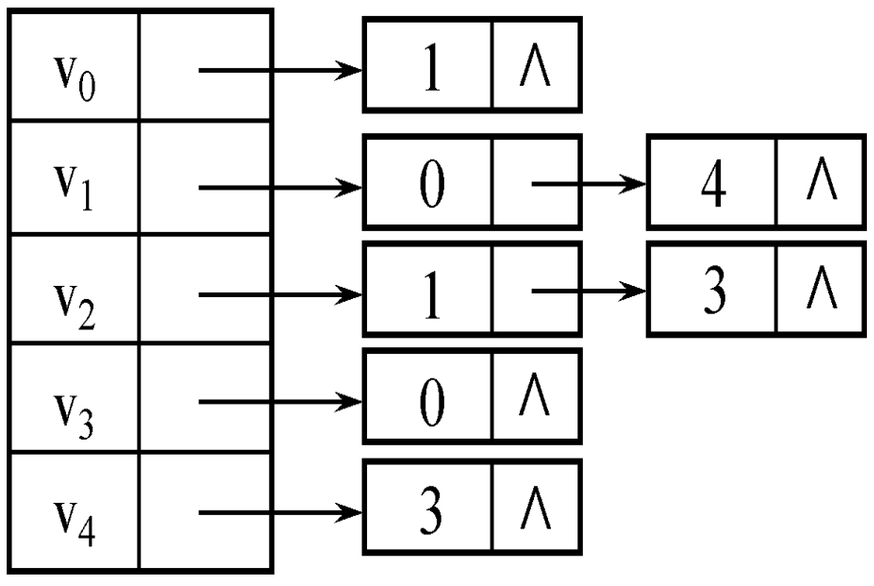
- 右边无向图的邻接表表示为：



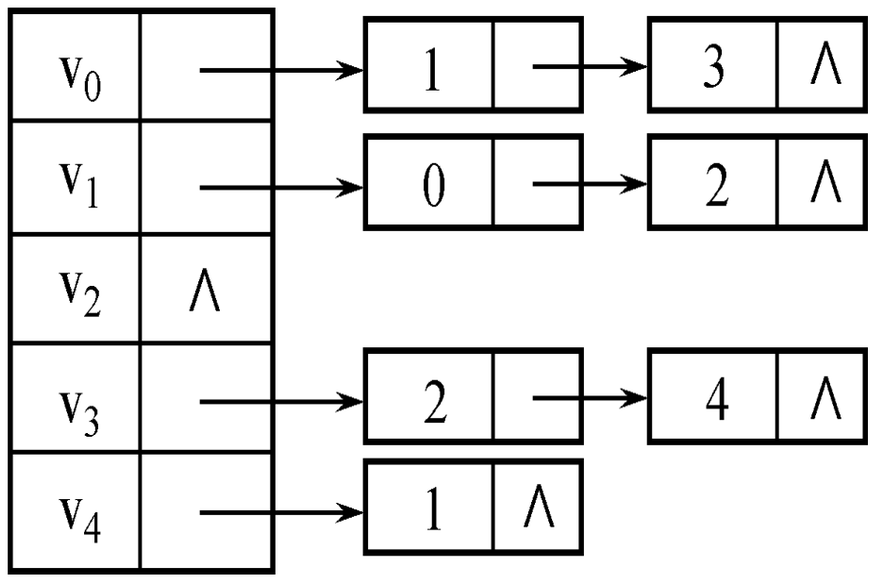


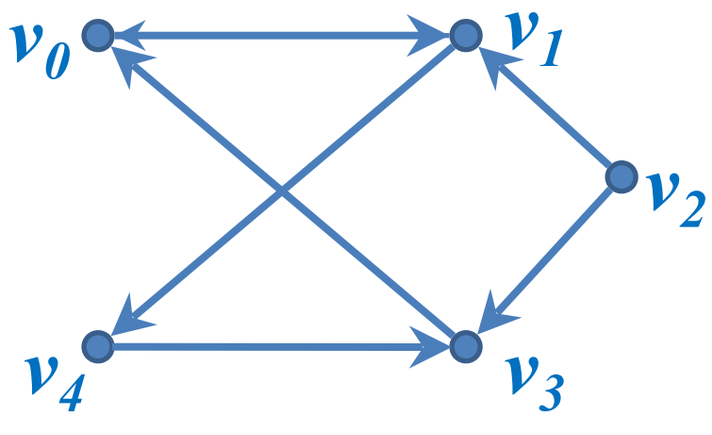
例子：有向图的两种邻接表表示

出边表



入边表





firstAdjacent(G,v0)
nextAdjacent(G, v0 ,v1)

两种表示的空间开销

- 设图G有n个顶点，e条边。
- 图的邻接矩阵表示的空间代价为$O(n^2)$；
- 特别对于无权图而言，关系矩阵的每个元素实际上只要一个二进制位就可以表示。
- 图的邻接表表示的空间代价：
  - 若图G是无向图，则为O(n+2e)；
  - 若图G是有向图，则为O(n+e)。

图算法

- 很多实际问题可以归结为图和图上的算法，例如：
  - 网络路由（我们每天都在用）
  - 集成电路（和印刷电路板）的设计和布线
  - 运输和物流中的各种规划安排问题
  - 工程项目的计划安排
  - 许多社会问题计算，如金融监管（例如关联交易检查）
- 一旦从应用中抽象出“图”的模型，应用问题的解决就可能变成图算法问题
- 下面将讨论一些图上的算法
  - 这些算法都有很清晰的应用背景
  - 很多算法里蕴涵着巧妙的想法和理论根据，其正确性需要（也可以）严格证明，复杂性非常重要（需要处理的问题规模可能很大）
  - 实现中常常用到前面讨论的一些数据结构

Where is Baba's key?

图的周游

- 图的周游：从图中某个顶点出发，按照某种方式系统访问图中所有顶点，且每个顶点仅访问一次。也称图的遍历。
- 这种周游的基本部分是访问一个顶点所在的（强）连通分量里的全部结点。如果不是（强）连通图，还需要去访问其他（强）连通分量。
- 常见的图周游方法有：
  - 深度优先周游（通过深度优先搜索的方式周游）
  - 广度优先周游（通过广度优先搜索的方式周游）
- 两种方式对有向图和无向图都适用。

深度优先周游

- **深度优先周游 (Depth-First Traversal) 的策略是深度优先搜索(Depth-First Search)，具体思想是：**

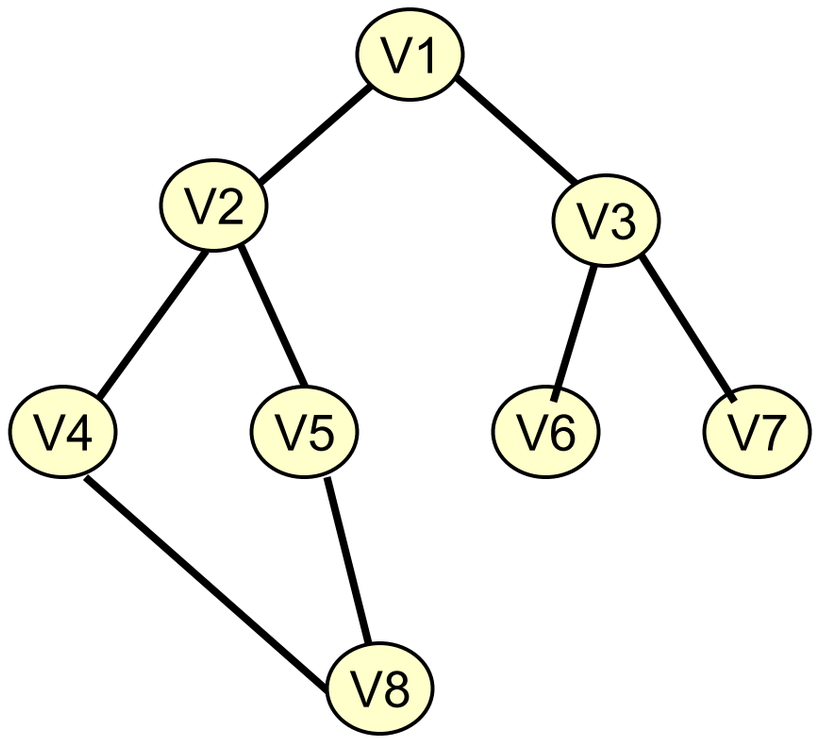
  1. 从指定顶点 v 出发，先访问 v 并将其标记为已访问过；

  2. 依次从 v 的未被访问过的各邻接顶点 w 出发（递归）进行深度优先搜索(算法需要为 v 的邻接结点假定一种顺序)，直到图中与 v 连通的所有顶点都访问过；

  3. 如果图中还有未访问顶点，则选一个未访问顶点，由它出发重复上述过程，直到图中所有顶点都被访问为止。

- **对图进行深度优先周游时，按访问顶点的先后次序所得到的顶点序列，称为该图的深度优先周游序列，简称 DFS 序列。**

例子



如果假定各个顶点的邻接顶点从左到右排序，得到的DFS 序列:

v1→v2→ v4 → v8→v5→v3→v6→v7

深度优先周游 (DFT) 和深度优先搜索 (DFS) 算法

```
void dft (Graph g){
    Vertex v;
    for (v=firstVertex(g); v != NULL; v=nextVertex(g,v))
        if (unvisited(v)) dfs(g,v) ;                    /* 调用 dfs */
}
```

```
void dfs(Graph g, Vertex v) {
    Vertex v1;
    mark_visited(v);
    for(v1=firstAdjacent (g,v);v1!= NULL; v1=nextAdjacent(g,v,v1))
        if (unvisited(v1)) dfs (g ,v1);                 /* 递归调用 dfs */
}
```

广度优先周游

- **广度优先周游 (Breadth-First Traversal) 的策略是广度优先搜索(Breadth-First Search)，具体过程是：**

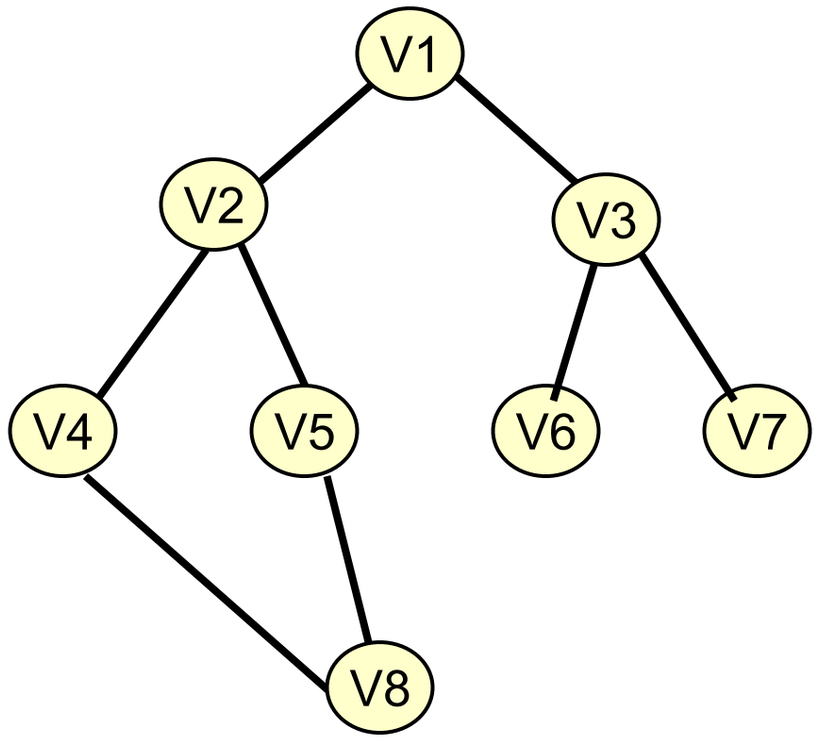
  1. 从指定顶点 v 出发，先访问 v 并将其标记为已访问过；

  2. 依次访问 v 的所有未访问过的相邻顶点 w_1, w_2, …, w_x （算法需要为 v 的邻接顶点假定一种顺序），然后依次访问与 w_1, w_2, …, w_x 邻接的所有未访问顶点；依次类推，直到所有已访问顶点的相邻顶点都已访问为止；

  3. 如果图中还有未访问顶点，则选择一个未访问顶点，由它出发进行广度优先搜索，直到所有顶点都已访问为止。

- **对图进行广度优先周游时，按访问顶点的先后次序所得到的顶点序列，称为该图的广度优先周游序列，简称BFS 序列。**

例子



按广度优先周游得到的BFS 序列:

v1→v2→ v3 → v4→v5→v6→v7→v8

广度优先周游（BFT）和广度优先搜索（BFS）算法

```
void bft (Graph g){
    Vertex v;
    for(v =firstVertex (g); v!= NULL;v = nextVertex(g,v))
        if (unvisited(v)) bfs (g,v);
}

void bfs(Graph g, Vertex v) {
    Vertex v1, v2;
    Queue q = createEmptyQueue () ;                    /* 元素类型为Vertex */
    enQueue(q,v);
    while(!isEmptyQueue(q)){
       v1=frontQueue(q);  deQueue(q);
       mark_visited(v1); v2=firstAdjacent(g,v1);
       while(v2!=NULL){
           if(unvisited(v2)) enQueue(q,v2);
           v2=nextAdjacent(g,v1,v2);
        }
     }
}
```

本讲重点

- 基本概念
- 存储表示
  - 邻接矩阵表示法
  - 邻接表表示法
  - 其他方法：邻接多重表*、十字链表*等
- 图的周游
  - 深度优先周游
  - 广度优先周游