计算机领域本科教育教学改革试点
工作计划（“101计划”）研究成果

数据结构

俞勇、张铭、陈越、韩文弢

上海交通大学、北京大学、浙江大学、清华大学

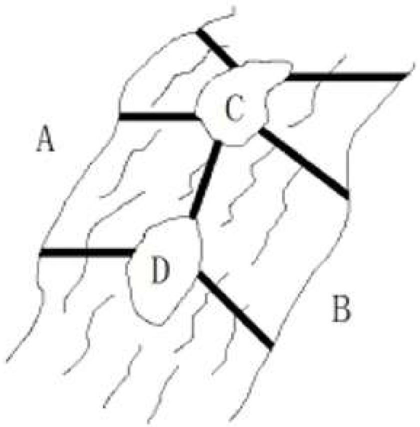

第 7 章
图
张同珍
上海交通大学

提纲

问题引入：哥尼斯堡七桥问题

问题：18世纪的哥尼斯堡，一条河流穿城而过，城市除被一分为二外，还包含了河中的两个小岛，河上有七座桥把这些陆地和岛屿联系了起来，可否从一个陆地或岛屿出发，一次经过全部的七座桥且每座桥只走一遍，最后还能回到出发点？

问题分析：2个问题，如何判断是否有解？如果有解，如何找到解？

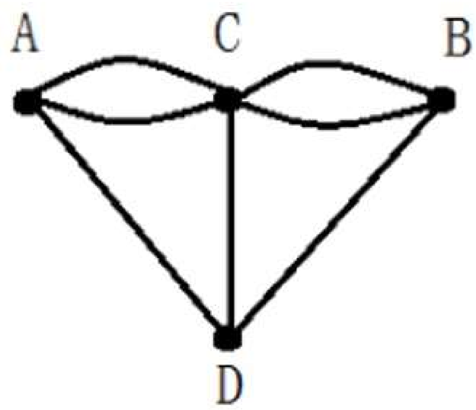


问题求解：哥尼斯堡七桥问题

问题抽象：抽象地表达和描述-陆地或者岛屿为元素（顶点），桥为元素间关系（边）。涉及到的数学工具-图

问题转化为：是否存在从任意一个顶点出发，经过每条边一次且仅一次，最后回到该顶点的路径（一笔画问题）。

分析：
(1)要经过所有的边，必须经过图中每个顶点至少一次！
(2)起点是“先出后入”，其余顶点都是”先入后出”，而最终要回到起点，说明每个顶点出和入的次数相同
(3)因为每条边只能经过一次，则每个顶点连接有偶数条边，即一半用于走出，一半用于走入

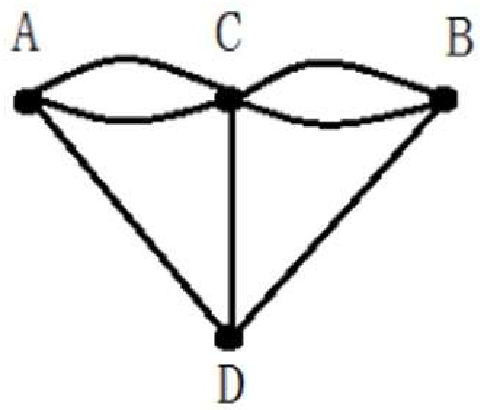


思考：使上图可一笔画，至少需要添加几条边?

高等教育出版社

单选题 1分

要使右图可一笔画，至少需要添加几条边?



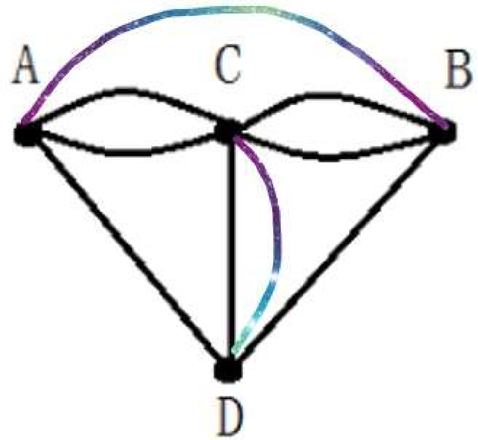
- A 1
- B 2
- C 3
- D 4

问题求解：哥尼斯堡七桥问题

问题抽象：抽象地表达和描述-陆地或者岛屿为元素（顶点），桥为元素间关系（边）。涉及到的数学工具-图

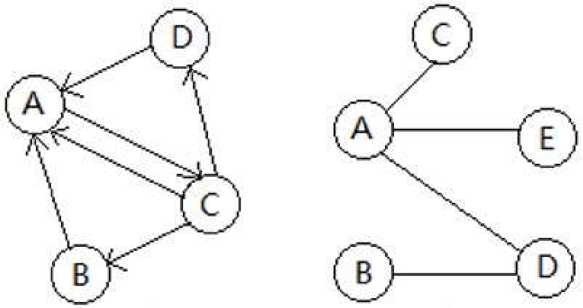
问题转化为：是否存在从任意一个顶点出发，经过每条边一次且仅一次，最后回到该顶点的路径（一笔画问题）。

分析：
(1)要经过所有的边，必须经过图中每个顶点至少一次!
(2)起点是“先出后入”，其余顶点都是”先入后出”，而最终要回到起点，说明每个顶点出和入的次数相同
(3)因为每条边只能经过一次，则每个顶点连接有偶数条边，即一半用于走出，一半用于走入



图的定义

图：可以用一个二元组G = (V,E)表示，其中V是顶点的非空集合，E是两个顶点间边（弧）的集合。



G1　　G2

G1:
- 顶点集合V = {A,B,C,D}
- 边集合E={<B,A>，<A,C>，<C,A>，<C,D>，<D,A>，<C,B>}

G2:
- 顶点集合V ={A,B,C,D,E}
- 边集合E ={ (A,C)，(A,E)，(D,B)，(D,A)}构成

图的术语

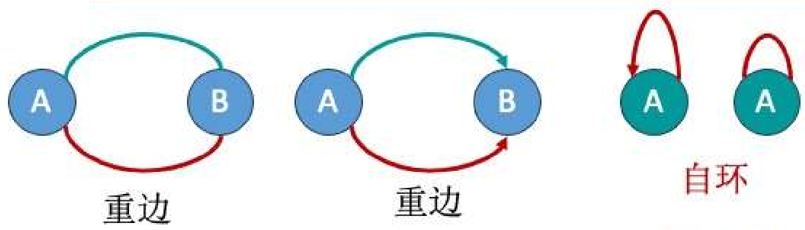
有向边：边带有方向，用带尖括号的顶点对来表示，如<D,A>，表示由D射向A的边，A为**弧头**，D为**弧尾**。

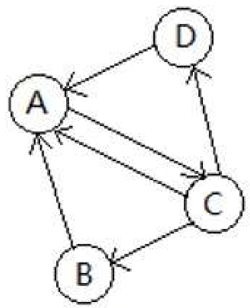
有向图：由顶点集和有向边集合组成的图。G 1 就是一个有向图。

无向边：边不带有方向，用带圆括号的顶点对来表示，如 (C,A)，表示C和A之间有条边。

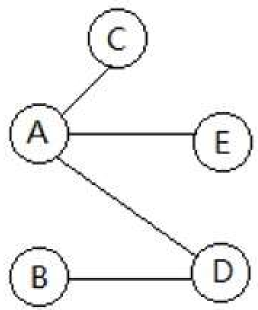
无向图：由顶点集和无向边集合组成的图。G 2 就是一个无向图。

简单图: 图中没有“自环”和“重边”





G1



G2

图的术语

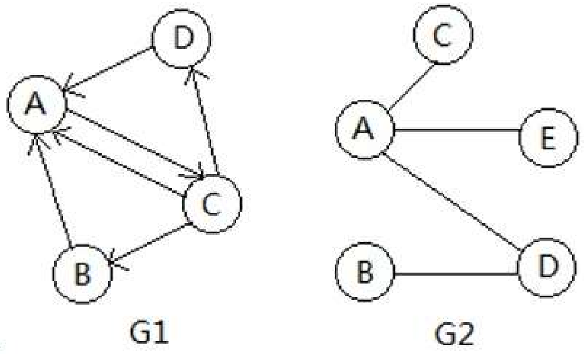
邻接：图的顶点间有边相连，称顶点间有邻接关系。

- (vi,vj)是无向边，称vi和vj邻接、vj和vi邻接、边(vi,vj)邻接于顶点vi和vj
- <vi,vj>是有向边，称vi**邻接到**vj 、或vj和vi邻接、边<vi,vj>邻接于顶点vi和vj

出度：有向图中一个顶点的**出度**是指由该顶点射出的有向边的条数。

入度：有向图中一个顶点的**入度**是指射入该顶点的有向边的条数。

度：无向图中一个顶点的**度**是指邻接于该顶点的边的总数。



单选题 1分

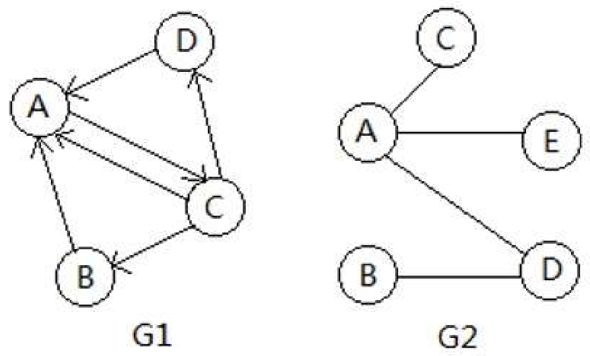
G1和G2中，不是A邻居的结点是：

A C 和 C、E、D

B B、C、D 和 B

C B、D 和 B

D B、C、D 和 B、C、D、E



图的术语

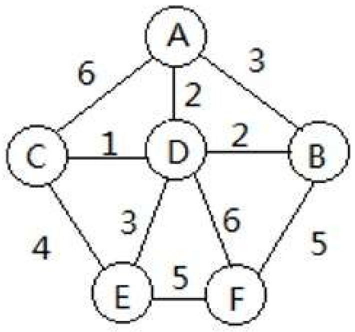
无向完全图：当无向图中边的条数达到最大，为n（n - 1） / 2时的图。

有向完全图：当有向图中边的条数达到最大，为n（n - 1）时的图。

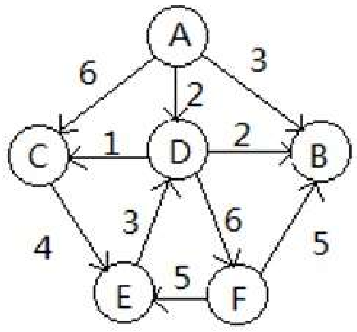
加权有向图：边上带有权重的有向图。

加权无向图：边上带有权重的无向图。

网络：加权有向图和加权无向图，统称为网络。



加权无向图G3



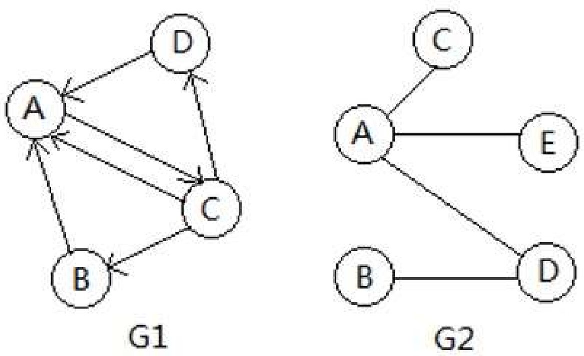
加权有向图G4

高等教育出版社

图的术语

路径： 如果可以从顶点v_i出发经过若干条无向边或者有向边到达顶点v_j，称顶点v_i到顶点v_j之间存在着一条路径。

路径的长度： 是顶点v_i到顶点v_j之间的这条路径上无向边或有向边的条数；如果边上有权重，路径长度也可以用路径上所有边的权重之和来表示。



A到B的路径

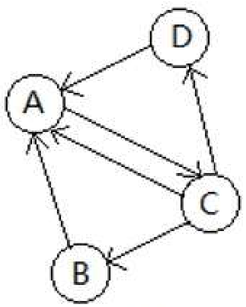
G1: <A, C, B>, <A, C, A, C, B>, <A, C, D, A, C, B>...

G2: <A, D, B>, <A, C, A, D, B>, <A, C, A, E, A, D, B>...

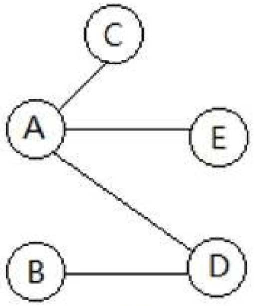
图的术语

简单路径： 一条路径上除了第一个顶点和最后一个顶点可能相同之外，其余各顶点都不相同。

简单回路或**简单环：** 简单路径上第一个顶点和最后一个顶点相同。



G1



G2

A到B的简单路径

G1: <A, C, B>

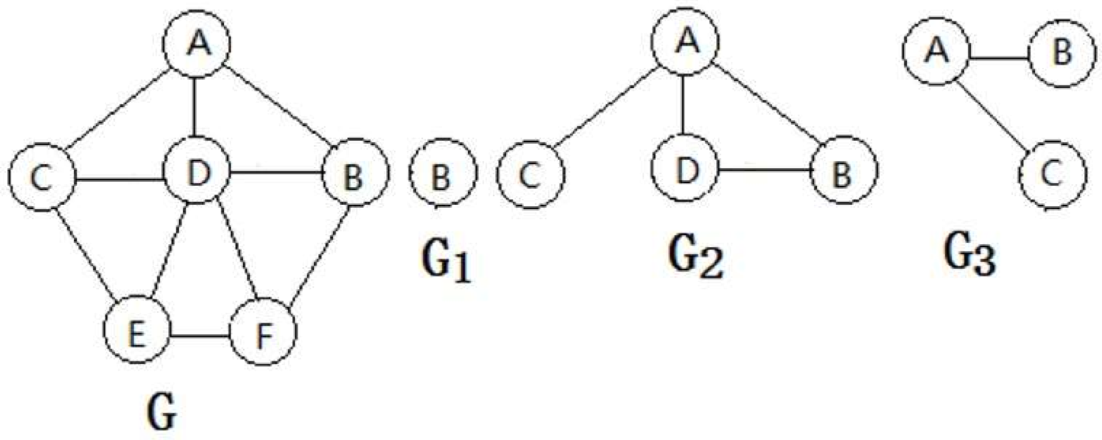
G2: <A, D, B>

简单回路

G1: <A, C, A>, <A, C, B, A>, <A, C, D, A>

图的术语

子图：假设有两个图G = (V,E)，G′ = (V′，E′)，且 V′是V的子集，E′是E的子集，称G′是G的子图。



图G1、图G2、图G3均是图G的子图。

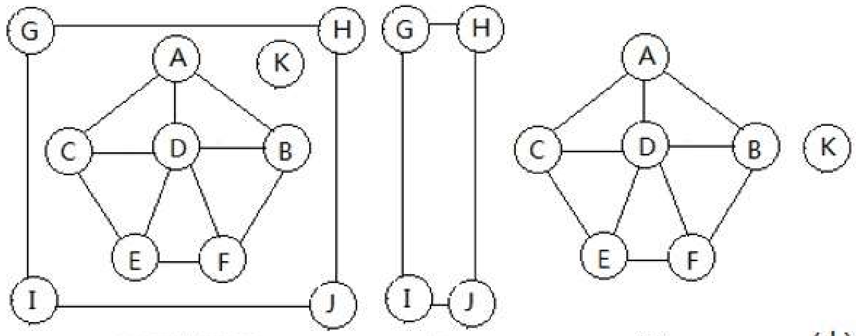
图的术语

连通：在一个图中，如果顶点v_i到v_j之间有路径存在，称顶点v_i到v_j是连通的。

连通图：在一个无向图G中，如果任意两个顶点对之间都是连通的，称G是连通图。

极大连通子图：将该子图外的任意一个顶点增加进子图都会造成子图不连通，且该子图包含了其中顶点间所有的边，该子图称极大连通子图。

连通分量：无向图的极大连通子图称连通分量。



(a) 无向图G5　(b)　(c)　(d)

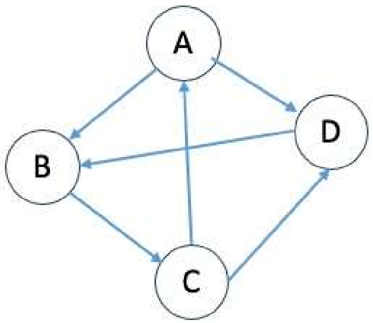
连通分量

图的术语

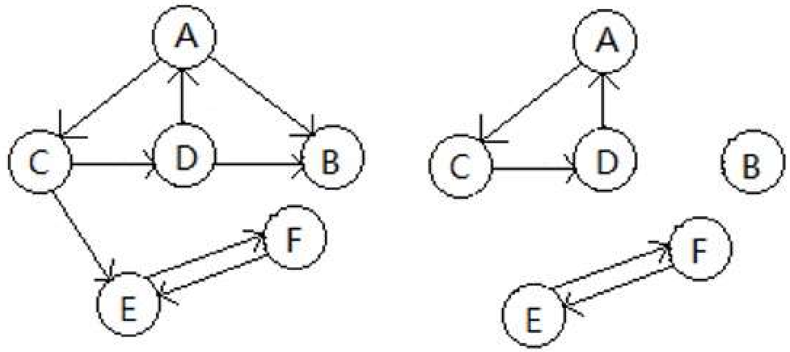
强连通图： 在一个有向图G中，如果任意两个顶点对之间都是连通的，称G是强连通图

弱连通图： 如果将有向图的有向边换成无向边得到的图连通，则此有向图是弱连通图

强连通分量： 有向图的极大连通子图，称强连通分量。



强连通



(a) 有向图G6 (b) G6的三个强连通分量

弱连通?

单选题 1分

含n个结点的强连通图，其中边的数目的最大值和最小值是：

A. n^2 和 n

B. n(n-1) 和 n-1

C. n^2 和 n+1

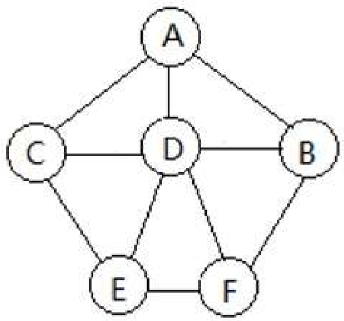
D. n(n-1) 和 n

图的术语

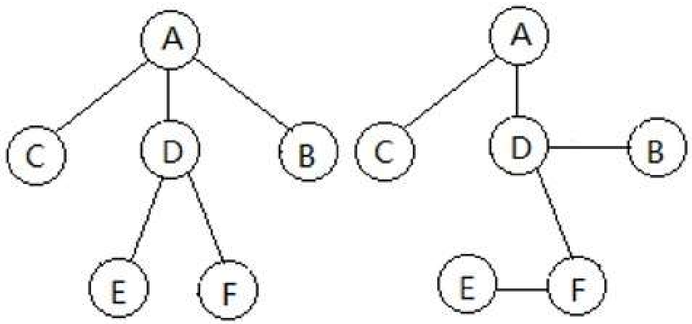
树：无回路的连通图

生成树：连通图的**极小连通子图**，该子图包含连通图的所有*n*个顶点，但只含它的*n*-1条边。如果去掉一条边，这个子图将不连通；如果增加一条边，必存在回路。

不唯一性：一个连通图的生成树并不保证唯一。



(a) G7



(b)G7的两个生成树

线性、树、图结构的比较

- **线性结构**：每个元素只有一个直接前驱和一个直接后继。
- **树形结构**：每个数据元素有一个直接前驱，但可以有多个直接后继。
- **图形结构**：数据元素之间的关系是任意的。每个数据元素可以和任意多个数据元素相关，有任意多个直接前驱和直接后继。在无向图中，甚至是互为前驱后继。

图结构的应用例

- **地图导航**：用结点表示各地点位置，边表示地点之间有公路、铁路等交通工具连接，边有权重，表示距离、时间或交通费等；由此可计算两地之间的最短路径。
- **社交网络**：人与人之间的关系可以用图来表示，结点代表用户，边代表互相关注或其他连接。Facebook、抖音等社交媒体平台利用图算法来推荐朋友、内容和广告。
- **知识图谱**：图结构化的知识表示方法，在自然语言处理、搜索引擎、生成式AI等领域有广泛的应用。例如，谷歌的知识图谱连接了数百亿个实体，使得搜索引擎能够更好地理解用户的查询意图。
- **图神经网络GNN**：专门处理图数据的深度学习模型，能够通过学习结点之间的关系，完成结点分类、链接预测等任务。例如，推荐系统使用GNN来捕捉用户与物品之间的复杂关系，从而提高推荐精度。

图结构的ＡＤＴ

ADT Graph {

数据对象：

$\{v_i | v_i \in \text{ElemSet}, i=1,2,3,\ldots\ldots n, n>0\}$ 或 Φ; ElemSet为顶点集合。

数据关系：

$\{<v_i, v_j>$或$(v_i, v_j) | v_i$、$v_j \in$ElemSet, 且$P(v_i, v_j)$, $i, j=1,2,3,\ldots\ldots n\}$,

其中：$<v_i, v_j>$表示从顶点v_i到顶点v_j的一条边，

(v_i, v_j)表示顶点v_i与顶点v_j互连，

$P(v_i, v_j)$定义了$<v_i, v_j>$或(v_i, v_j)的意义或信息。

基本操作:

InitGraph(*graph*，*kMaxVertex*，*no_edge_value*，*directed*)：初始化一个空的图*graph*。其中*kMaxVertex*是最多可能的顶点数；*no_edge_value*是当顶点间不存在边时，在图中给顶点关系赋予的权值；*directed* 为**true**时图是有向的，为**false**时图是无向的。

CreateGraph(*graph*)：构造一个图*graph*。

DestroyGraph(*graph*)：释放图*graph*占用的所有空间。

NumberOfVex(*graph*)：返回图*graph*中顶点的个数。

NumberOfEdge(*graph*)：返回图*graph*中边的条数。

ExistEdge(*graph, u, v*)：判断图*graph*中顶点*u*到*v*之间是否存在边，有返回**true**，无返回**false**。

高等教育出版社

GetPosition(*graph,v*)：返回顶点*v*在图*graph*中的位置，无则返回NIL。

GetValue(*graph,v*)：返回图*graph*中顶点*v*的值

PutValue(*graph,v,value*)：为图*graph*中顶点*v*赋值*value*。

FirstAdjVex(*graph,v*)：返回图*graph*中顶点*v*的第一个邻接顶点，若*v*无邻接顶点返回NIL。

NextAdjVex(*graph,u,v*)：返回图*graph*中*u*顶点相对*v*顶点的下一个邻接顶点，无则返回NIL。

InsertVex(*graph,v*)：在图*graph*中插入顶点*v*。

InsertEdge(*graph,u,v,weight*)：在图*graph*中顶点*u*和*v*之间插入一条边，权值为*weight*。

RemoveVex(*graph,v*)：在图*graph*中删除顶点*v*及所有邻接于顶点*v*的边。

RmoveEdge(*graph,u,v*)：在图*graph*中删除顶点*u*和*v*之间的边。

DFS(*graph*)：按深度优先遍历图*graph*中顶点。

DFS(*graph,v,visited*)：从顶点v开始深度优先遍历，visited记录顶点访问标记

BFS(*graph*)：按广度优先遍历图*graph*中顶点。

BFS(*graph, v, visited*)：从顶点v开始广度优先遍历，visited记录顶点访问标记

}

图的存储

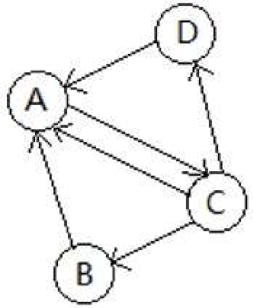
- 图的存储既要考虑到顶点的存储又要考虑到边的存储。
- 如果按照线性结构和树结构的存储思路，找到一个类似的、既能同时存储顶点又能存储表示顶点间关系的边的结构就非常困难。不妨换个思路，将顶点和边的存储独立开来，顶点归顶点存、边归边存。
- 顶点用一维数组存，边用二维矩阵存 ---**邻接矩阵存储法。**

  顶点用一维数组存，边用单链表存 ---**邻接表存储法。**

邻接矩阵

- 在一维数组中存储顶点信息，在二维矩阵中存储边的信息。
- 如果非加权图中，存在一条自顶点vi到vj 的有向边或无向边，那么在二维矩阵（如a）中，a[i][j] = 1，否则 a[i][j] = 0。
- 按照简单图的定义，主对角线上元素a[i][i] = 0，即顶点到自身没有边相连。
- 无向图的邻接矩阵是以主对角线为轴对称的。

邻接矩阵

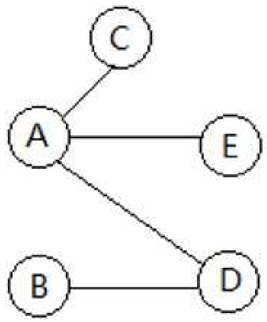


G8

0	1	2	3
A	B	C	D

$$\begin{array}{c c} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

G8的邻接矩阵



G9

0	1	2	3	4
A	B	C	D	E

$$\begin{array}{c c} & \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

G9的邻接矩阵

多选题 1分

用邻接矩阵存储图，下面哪些操作的时间复杂度是O(1)，用n表示结点的数目

- A 判断两个结点间有无边连接
- B 计算结点的度，即相邻结点的个数
- C 查找度最大的结点
- D 添加或删除一条边
- E 添加或删除一个结点

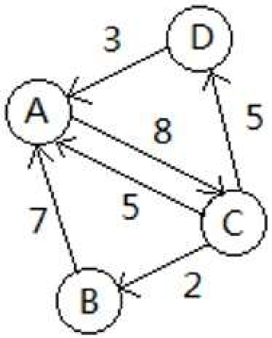
邻接矩阵

用邻接矩阵存储图，下面哪些操作的时间复杂度是O(1)，用n表示结点的数目

A	判断两个结点间有无边连接	O(1)
B	计算结点的度，即相邻结点的个数	O(n)
C	查找度最大的结点	$O(n^2)$
D	添加或删除一条边	O(1)
E	添加或删除一个结点	O(?) ______

加权邻接矩阵

- 当图中边带有权值时，可以用加权邻接矩阵表示加权有向图或无向图。
- 如果加权图中，存在一条自顶点vi到vj 的有向边或无向边，那么在二维矩阵（如a）中，a[i][j] = 权值，否则 a[i][j] = ∞。
- 按照简单图的定义，主对角线上元素a[i][i] = 0或者∞，即顶点到自身没有边相连。



G10

0	1	2	3
A	B	C	D

$$
\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3
\end{array}
\begin{array}{c}
\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\
\left(\begin{array}{cccc}
0 & \infty & 8 & \infty \\
7 & 0 & \infty & \infty \\
5 & 2 & 0 & 5 \\
3 & \infty & \infty & 0
\end{array}\right)
\end{array}
$$

G10的邻接矩阵

高等教育出版社

邻接矩阵和加权邻接矩阵的优缺点

- 判断任意二个顶点v_i和v_j之间是否存在一条边非常容易，直接看a[i][j]，O(1)的时间复杂度。
- 在任意两个顶点间添加新的边或者删除现有的边非常容易！
- 在用邻接矩阵表示无向图和有向图时，可以很容易地得到顶点的度或者出度、入度。
- 当边的总数远远小于n^2，也需n^2个内存单元来存储边的信息，空间消耗太大。

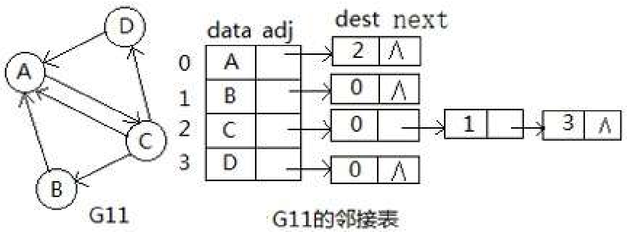
一般情况: |E| = O(|V|)

邻接矩阵的适应情况和特殊图的存储处理

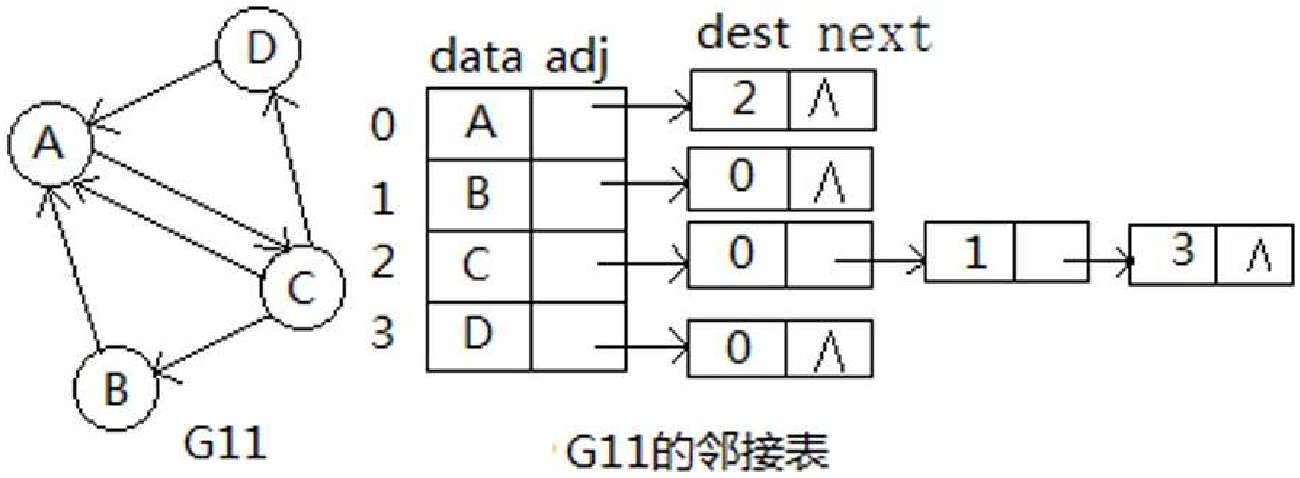
- 如果图是稠密图（边数非常多）：
  对有向图，采用邻接矩阵是合适的。
  对无向图，因关于主对角线对称，可只存储其上三角矩阵或下三角矩阵。
- 如果图是稀疏图（边数很少)，且非零元素的分布没有规律:
  - ✓ 通常的做法是只存储其中的非零元素和非零元素所在的位置，每个非零元素a[i][j]用一个三元组来表示：（i, j, a[i][j]）.
  - ✓ 将此三元组按照一定的次序排列，如先按照行序再按照列序排列。
  - ✓ 三元组可以放在顺序表或者链表中。

邻接表

- 顶点依然用一个一维数组来存储，而边的存储是将由同一个顶点出发的所有边组成一条单链表。
- 存储顶点的一维数组称**顶点表**，存储边信息的单链表称**边表**。一个图由顶点表和边表共同表示。
- 顶点表不仅保存各个顶点的信息，还保存由该顶点射出的边形成的单链表中首结点的地址（首指针），这种方法称**邻接表**表示法。



邻接表



与树的孩子表示法相似！

多选题 1分

用邻接表存储n个结点的有向图，哪些操作的时间是O(n)?

A 判断两个结点间是否有边连接

B 在两个结点间添加边或删除已有的边

C 计算结点的出度，即从结点出发指向其他结点的边数

D 计算结点的入度，即从其他结点指向该结点的边数

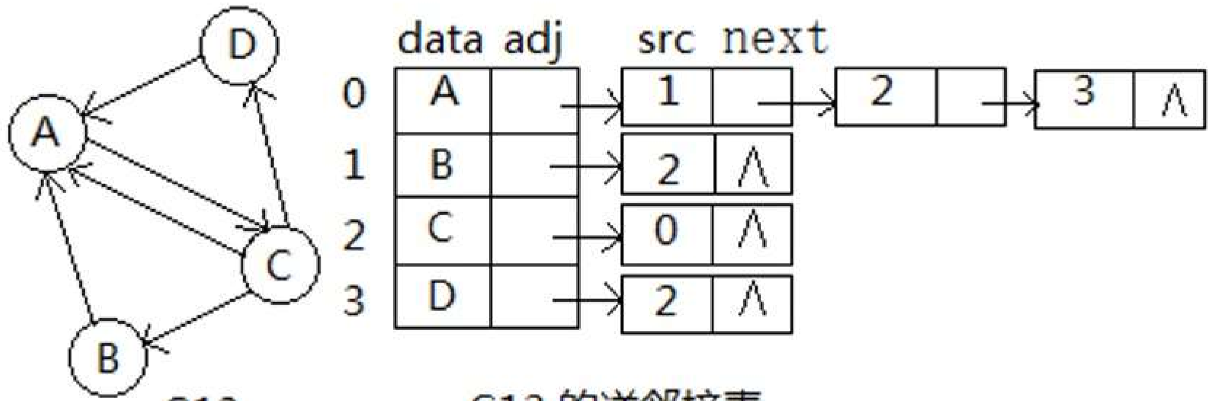
邻接表存储特点

- 仅存储有边的信息，不存储无边信息，在图比较稀疏的情况下，空间的利用率大大提高。
- 无向图，同一条边存储了两次。
- 计算某个顶点v的出度（有向图）或者度（无向图），只需遍历该顶点v指向的边表，即利于计算出度。
- 计算某个顶点v的入度（有向图），需要遍历所有顶点v指向的边表，即不利于计算入度。

时间复杂度: O(|V|+|E|) 或 O(|E|) （如果|E|>|V|）

邻接表存储

逆邻接表：有向图的逆邻接表中，顶点表保存该顶点的射入边形成的单链表的首结点地址，有利于计算顶点的入度。



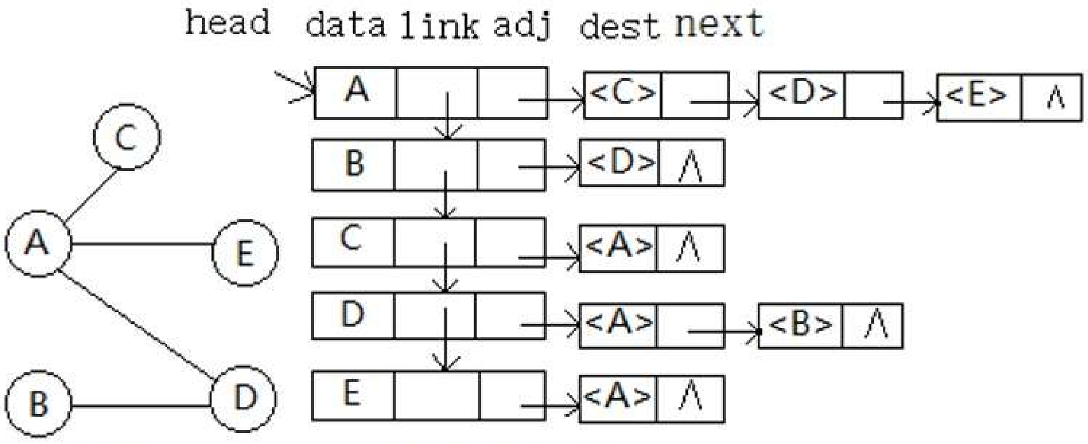
G13　　G13 的逆邻接表

另外一种邻接表存储

邻接表中，顶点表用了一维数组，图初始化时需要预估数组规模。

一种改进：

顶点表也用一个单链表表示。此时，dest用顶点结点地址而不用下标。



G14　　G14的邻接表

思考：如何添加或删除结点?

邻接多重表★

邻接表中，无向图时每条边都用了两个边结点，即同一条边被存储了两次。

1）空间浪费，

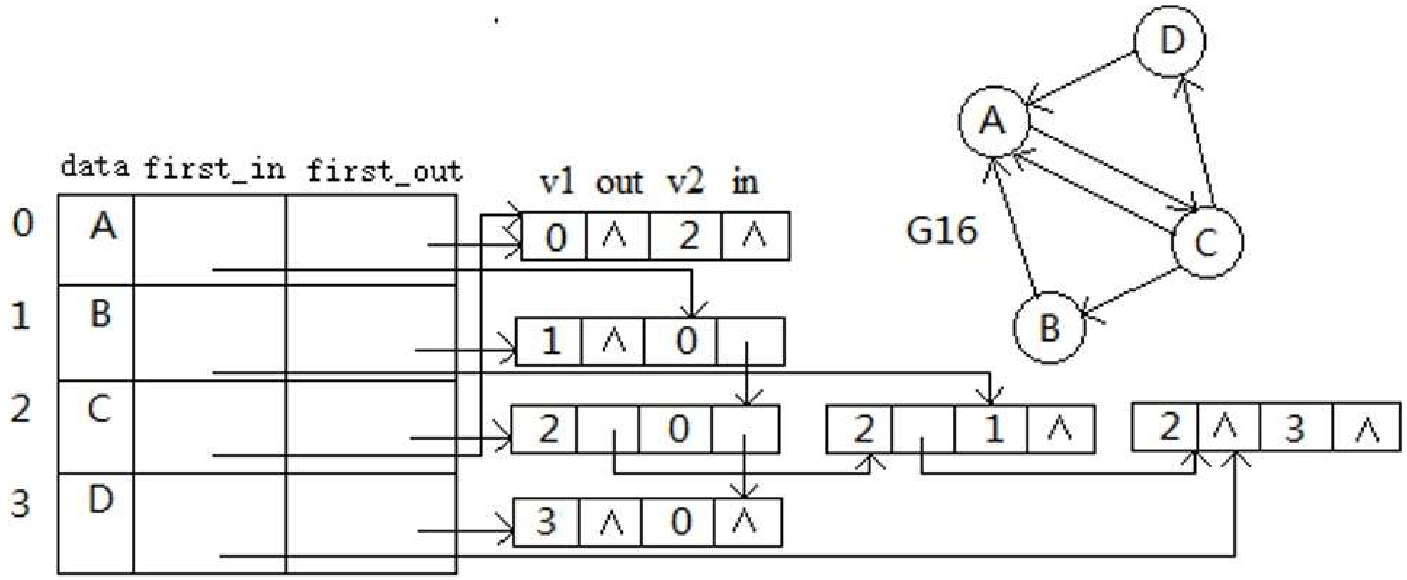
2）在某些应用中，如遍历所有边时因重复而不方便，

邻接多重表：

1. 每条边仅使用一个结点来表示，即只存储一次，但这个边结点同时要在它邻接的两个顶点的边表中被链接。
2. 为了方便两个边表同时链接，每个边结点不再像邻接表中那样只存储边的一个顶点，而是存储两个顶点。

十字链表«

十字链表将有向图的邻接表和逆邻接表结合在了一起，既利于求出度又利于求入度。



G16的十字链表

图的基本操作实现

图的操作所需要花费的时间通常既和顶点的个数n有关，又和边的条数m有关，因此时间复杂度会包含n和m两个变量。

- **邻接矩阵表示的图**

假设已知条件为：

图中实际顶点个数 *n_verts*、图中实际边的条数*m_edges*、

图中顶点可能的最大数量*kMaxVertex*、保存顶点数据的一维数组*ver_list*、

保存邻接矩阵内容的二维数组*edge_matrix*、

无边时权重的赋值*no_edge_value* (一般图为0，网为无穷大kMaxNum）、

有向或无向图标志*directed*(有向图为真，无向图为假)。

图用邻接矩阵表示时部分基本操作算法描述

算法7-1: 获取图的顶点个数 NumberOfVex(*graph*)
输入：图*graph*
输出：图的顶点个数
1.**return** *graph.n_verts*

时间复杂度 O（1）

算法7-2: 判断边是否存在 ExistEdge(graph, u, v)
输入：图graph、两个顶点u和v
输出：u到v有边返回 true，否则返回 false
1. **if** u <graph. *n_verts* 且 v <graph. *n_verts* **then**
2. | **if** u≠v 且 graph. edge_matrix[u][v]≠graph. *no_edge*_value **then**
3. | | **return true**
4. | end
5. end
6. **return false**

时间复杂度 O（1）

图用邻接矩阵表示时部分基本操作算法描述

算法7-4: 向图中插入边 InsertEdge(*graph, u,v,weight*)

输入：图*graph*，边的两个端点*u*和*v*，边的权重*weight*

输出：插入了边(*u,v*)或<*u,v*>的图

1. **if** $u \neq v$ 且 ExistEdge(*graph, u, v*) = **false then**
2. | *graph.edge_matrix*[*u*][*v*]←*weight*
3. | *graph.m_edges*←*m_edges*+*1*
4. | **if** *graph.directed* = **false then** //如果是无向图，对主对角线对称的元素赋值
5. | | *graph.edge_matrix*[*v*][*u*] ←*weight*
6. | **end**
7. **end**

时间复杂度 O（1）