数据结构

深圳技术大学
大数据与互联网学院

第七章 图

7.1 图的定义和术语
7.2 图的存储结构
7.3 图的遍历
7.4 图的连通
7.5 有向无环图及其应用
7.6 最短路径

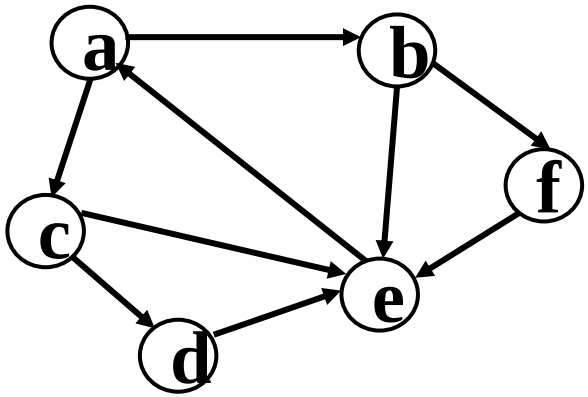
上节复习

- 图的其他存储结构
  - 十字链表、多重表
- 图的遍历
  - 深度优先搜索DFS
  - 广度优先搜索BFS
- 图的连通和生成树
  - DFS生成树和BFS生成树
- 最小生成树
  - 用于带权的连通图
  - 用n-1条边连接n个顶点，不能有回路，权值总和最小
  - 普里姆(Prim)算法，从已选择顶点中选择权最小的边
  - 克鲁斯卡尔(Kruskal)算法，从未选择边中选择权最小的边

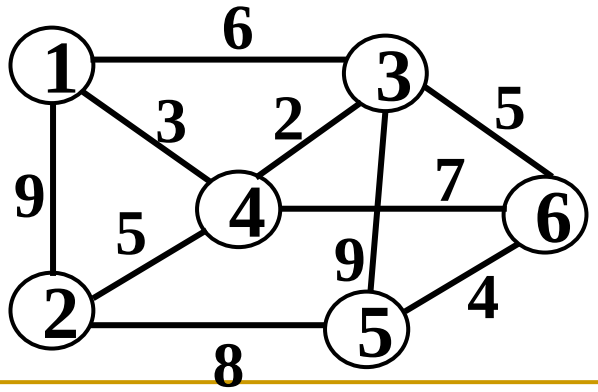
练习

一. 已知下图所示，从字母最小的顶点出发，求解

- 写出该图的深度优先搜索和广度优先搜索结果
- 画出以下图的DFS生成树或生成森林，和BFS生成树或生成森林



- 已知下图所示，采用普里姆(Prim)算法和克鲁斯卡尔(Kruskal)算法求最小生成树，要求写出详细求解过程



7.6 最短路径

一. 最短路径

- 最短路径是求从图（或网）中某一顶点，到其余各顶点的最短路径
- 最短路径与最小生成树主要有三点不同：
  - 最短路径的操作对象主要是有向图(网)，而最小生成树的操作对象是无向图
  - 最短路径有一个始点，最小生成树没有
  - 最短路径关心的是始点到每个顶点的路径最短，而最小生成树关心的是整个树的代价最小

7.6 最短路径

一. 最短路径

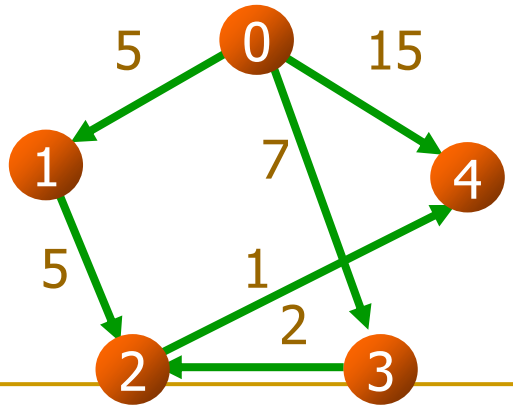
- 最短路径可以采用迪杰斯特拉(Dijkstra)算法求解，该算法采用按路径长度递增的次序产生最短路径
  - 在Dijkstra算法中，引进了一个辅助向量D
  - 每个分量D[i]表示当前所找到的从始点到每个终点v_i的最短路径长度
  - D[i]初值为始点v_0到各终点v_i的直接距离，即若从始点到某终点有(出)弧，则为弧上的权值，否则为∞

7.6 最短路径

一. 最短路径

- Dijkstra算法
  - 对于下图，如果0是始点v_0，则D[i]的初值为：D[i]={5, ∞, 7, 15}
  - 显然D[j]=Min{D[i] | $v_i \in V$} 是从始点出发的长度最短的一条最短路径
  - 设S为已求得的最短路径的终点的集合
  - 下一条长度次短的最短路径(设其终点为v_j)为以下之一：

1. 中间只经过S中的顶点v_i而后到达顶点v_j的路径
2. 弧$<v_0, v_j>$，D[j]=Min{$<v_0, v_j>$ or D[i]+$<v_i, v_j>$} $i \in V-S$



7.6 最短路径

一. 最短路径

- Dijkstra算法

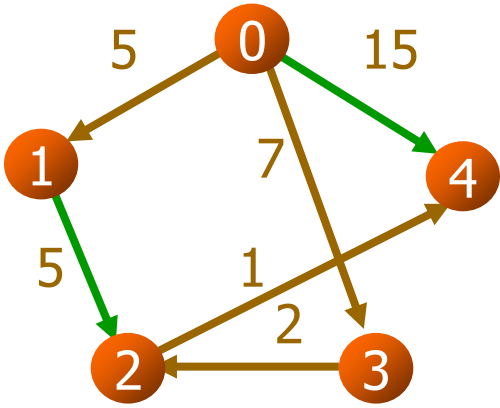
1. 初始化： S ← {v_0};
   D[i] ← arc[0][i], i = 1, 2, ..., n-1;
   P[i] = {0, i} // 从v_0到v_i的路径
2. 求出最短路径的长度：
   D[j] ← min { D[i] }, i ∈ V-S ; S←S U {j};
3. 修改：
   if (D[i] > D[j] + arc[j][i]) {D[i] = D[j]+arc[j][i];
   P[i] = P[j] U {i} ; } i ∈ V-S // 更新从v_0到v_i的路径
4. 判断：若 S = V, 则算法结束，否则转 2。

7.6 最短路径

一. 最短路径

- Dijkstra算法举例

顶点	D[i]			
1	5 {0,1}			
2	∞	10 {0,1,2}	9 {0,3,2}	
3	7 {0,3}	7 {0,3}		
4	15 {0,4}	15 {0,4}	15 {0,4}	10 {0,3,2,4}
终点j	1	3	2	4
S	{0,1}	{0,1,3}	{0,1,3,2}	{0,1,3,2,4}



7.6 最短路径

一. 最短路径

- Dijkstra算法

```
void ShortestPath_DIJ(MGraph G,int v0,PathMatrix
&P,ShortPathTable &D)
{ // 若P[v][w]为TRUE，则w是从v0到v当前求得最短路径上的顶点。
  // final[v]为TRUE当且仅当v∈S,即已经求得从v0到v的最短路径。
  int i=0,j, v,w,min;
  bool final[MAX_VERTEX_NUM];
  for (v=0; v<G.vexnum; ++v) {
    final[v] = FALSE;
    D[v] = G.arcs[v0][v].adj;
    for (w=0; w<G.vexnum; ++w)
        P[v][w] = FALSE;  // 设空路径
    if (D[v] < INFINITY) {
      P[v][v0] = TRUE;
      P[v][v] = TRUE; }
  }
  // ……
```

7.6 最短路径

一. 最短路径

- Dijkstra算法

```
D[v0] = 0;  final[v0] = TRUE;  // 初始化，v0顶点属于S集
//--- 开始主循环，每次求得v0到某个v顶点的最短路径，并加v到S集 ---
for (i=1; i<G.vexnum; ++i) { // 其余G.vexnum-1个顶点
  min = INFINITY;  // 当前所知离v0顶点的最近距离
  for (w=0; w<G.vexnum; ++w)
    if (!final[w])    // w顶点在V-S中
      if (D[w]<min) {
        v = w;  min = D[w]; }  // w顶点离v0顶点更近
  final[v] = TRUE;     // 离v0顶点最近的v加入S集
  for (w=0; w<G.vexnum; ++w)  // 更新当前最短路径及距离
    if (!final[w] && (min+G.arcs[v][w].adj<D[w])) {
      // 修改D[w]和P[w], w∈V-S
      D[w] = min + G.arcs[v][w].adj;
      for(j=0;j<G.vexnum;j++)
        P[w][j] = P[v][j]; //第v行赋值于第w行
      P[w][w] = TRUE;    // P[w] = P[v]+[w]
    }//if  }//for  } // ShortestPath_DIJ
```

7.6 最短路径

一. 最短路径

- 上述程序的不足
  - 能够找出V0到其他顶点的最短距离
  - 能够找出V0到其他顶点的最短路径的顶点
  - 但是不能找到最短路径的顶点排列顺序
- 求每条最短路径的顶点序列的算法
  - 设置一个数组，初始化清空，然后根据算法不断在里面加入路径的顶点
  - 如果求n个顶点的最短路径，就需要设置二维数组，数组每一行表示一个顶点的最短路径

7.6 最短路径

一. 最短路径

- 迪杰斯特拉(Dijkstra)算法特点
  - 找出当前离v0最近的顶点v，加入顶点集合S
  - 以当前顶点集合为基础，更新v0到其他顶点的路径和距离

- 采用的是贪心算法的逻辑
  - 每次都选择当前最短的路径

- 要求有向网中的权重不能为负
  - 加入顶点集合S的最短路径不再变更
  - 负权重会导致选择迪杰斯特拉(Dijkstra)算法最短路径不再成为全局最优解

7.6 最短路径

一. 最短路径

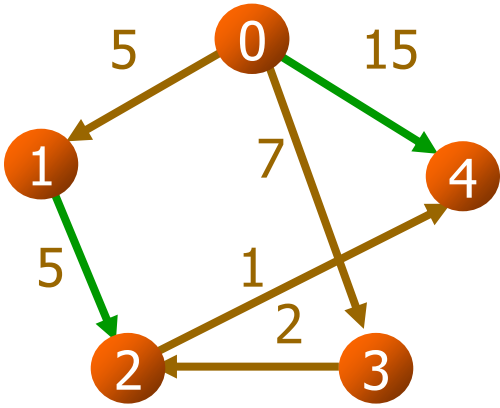
- 最短路径也可以采用弗洛伊德(Floyd)算法求解，该算法采用按路径长度递增的次序来逐步产生最短路径
  - 在Floyd算法中，引进了一个辅助矩阵D， D[i][j]对应存储任意两个顶点i和j顶点的最短路径
  - 辅助矩阵D初始值中，对角线为D[i][i]=0，D[i][j]表示弧<i，j>的权重，不存在该弧时则记为∞
  - 逐步在矩阵D中所记录路径中加入中间顶点，若加入中间顶点之后路径标段，D则记录新的路径。

7.6 最短路径

一. 最短路径

- 弗洛伊德(Floyd)算法举例

顶点	D[0][i]			
1	5 {0,1}	5 {0,1}	5 {0,1}	5 {0,1}
2	∞	9 {0,3,2}	9 {0,3,2}	9 {0,3,2}
3	7 {0,3}	7 {0,3}	7 {0,3}	7 {0,3}
4	15 {0,4}	15 {0,4}	10 {0,3,2,4}	10 {0,3,2,4}
路径最长长度	1	2	3	4



7.6 最短路径

一. 最短路径

- 弗洛伊德(Floyd)算法特点
  - 以路径长度为线索探索最短路径的所有可能性
- 采用的是动态规划的逻辑
  - 对全局最优解进行探索
- 时间复杂度会高于迪杰斯特拉(Dijkstra)算法
  - 可以出现负权重，但不能出现负权重回路

练习

- 已知带权有向图如下，请求出顶点0到其他顶点的最短路径和长度，要求使用迪杰斯特拉算法写出求解过程