数据结构与算法 (七)

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6（“十一五”国家级规划教材)

http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

第7章　图

- 7.1　图的定义和术语
- 7.2　图的抽象数据类型
- 7.3　图的存储结构
- 7.4　图的遍历
- 7.5　最短路径
- 7.6　最小生成树

图的遍历 (graph traversal)

- 给出一个图G和其中任意一个顶点V_0，从V_0出发系统地访问G中所有的顶点，每个顶点访问而且只访问一次

- **深度优先遍历**
- **广度优先遍历**
- **拓扑排序**

图遍历的考虑

- 从一个顶点出发，试探性访问其余顶点，同时必须考虑到下列情况
  - 从一顶点出发，可能不能到达所有其它的顶点
    - 如 **非连通图**；
  - 也有可能会陷入死循环
    - 如 **存在回路的图**

解决办法

- 为每个顶点保留一个 **标志位** (mark bit)
- 算法开始时，所有顶点的标志位置零
- 在遍历的过程中，当某个顶点被访问时，其标志位就被标记为已访问

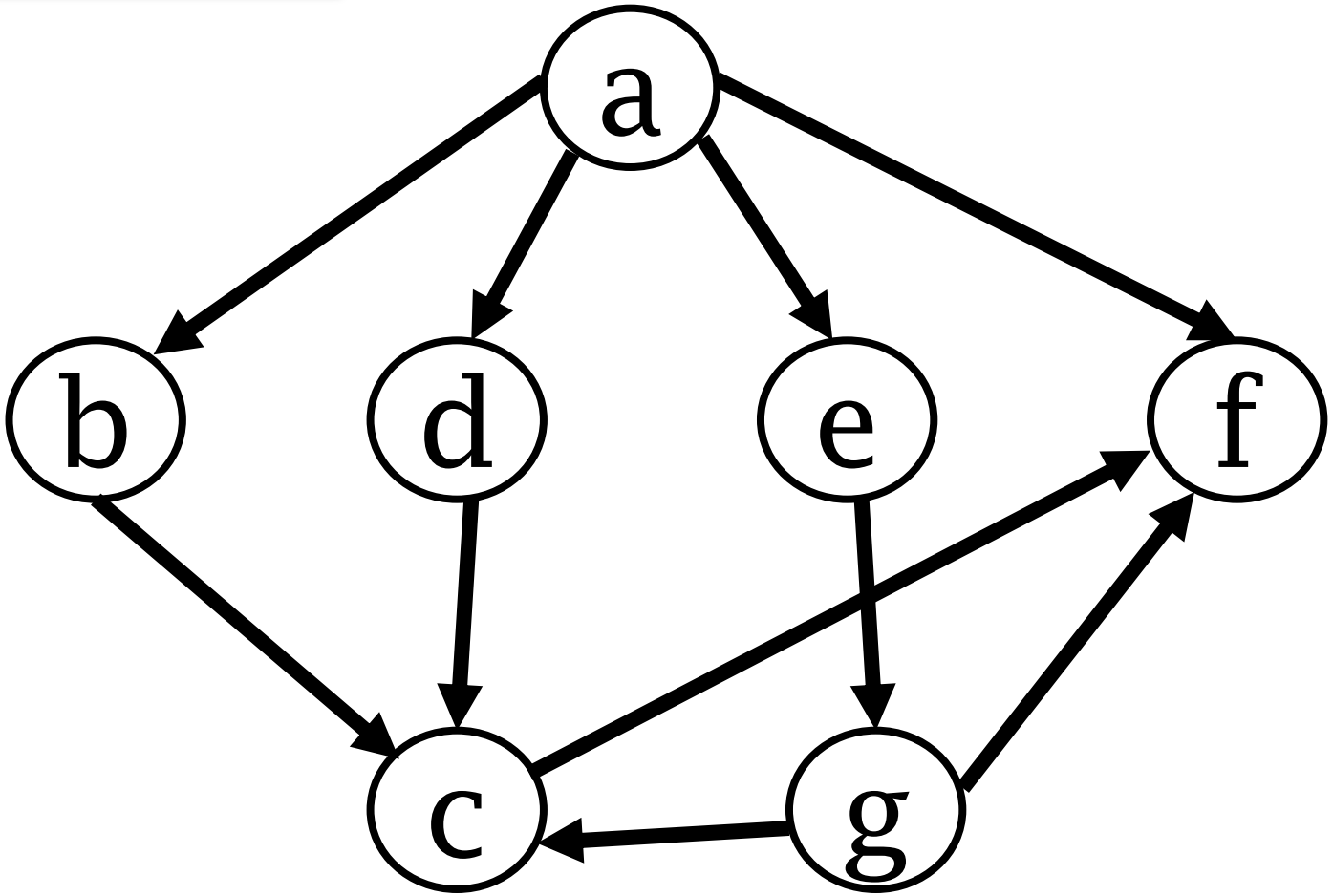
图的遍历算法框架

```
void graph_traverse(Graph& G) {
    // 对图所有顶点的标志位进行初始化
    for(int i=0; i<G.VerticesNum(); i++)
        G.Mark[i] = UNVISITED;
    // 检查图的所有顶点是否被标记过，如果未被标记，
    // 则从该未被标记的顶点开始继续遍历
    // do_traverse函数用深度优先或者广度优先
    for(int i=0; i<G.VerticesNum(); i++)
        if(G.Mark[i] == UNVISITED)
            do_traverse(G, i);
}
```

深度优先遍历（depth-first search）

- 深搜（简称DFS）类似于树的先根次序遍历，尽可能先对纵深方向进行搜索
- 选取一个未访问的点 v_0 作为源点
  - 访问顶点 v_0
  - 递归地深搜遍历 v_0 邻接到的其他顶点
  - 重复上述过程直至从 v_0 有路径可达的顶点都已被访问过
- 再选取其他未访问顶点作为源点做深搜，直到图的所有顶点都被访问过

7.4 图的遍历



深度优先搜索的顺序是： a→b→c→f→d→e→g

图的深度优先遍历 (DFS) 算法

```
void DFS(Graph& G, int v) {    // 深度优先搜索的递归实现
  G.Mark[v] = VISITED;         // 把标记位设置为 VISITED
  Visit(G,v);                  // 访问顶点v
  for (Edge e = G.FirstEdge(v); G.IsEdge(e);
            e = G.NextEdge(e))
    if (G.Mark[G.ToVertex(e)] == UNVISITED)
      DFS(G, G.ToVertex(e));
  PostVisit(G,v);              // 对顶点v的后访问
}
```

广度优先遍历

- **广度优先搜索 (breadth-first search，简称 BFS)**。其遍历的过程是：
  - 从图中的某个顶点 v_0 出发
    - 访问并标记了顶点 v_0 之后
    - 一层层横向搜索 v_0 的所有邻接点
    - 对这些邻接点一层层横向搜索，直至所有由 v_0 有路径可达的顶点都已被访问过
  - 再选取其他未访问顶点作为源点做广搜，直到所有点都被访问过

图的广度优先遍历(BFS)算法

```
void BFS(Graph& G, int v) {
    using std::queue;   queue<int> Q;    // 使用STL中的队列
    Visit(G,v);                          // 访问顶点v
    G.Mark[v] = VISITED;  Q.push(v);     // 标记,并入队列
    while (!Q.empty()) {                 // 如果队列非空
        int u = Q.front ();              // 获得队列顶部元素
        Q.pop();                         // 队列顶部元素出队
        for (Edge e = G.FirstEdge(u); G.IsEdge(e);
              e = G.NextEdge(e))         // 所有未访问邻接点入队
           if (G.Mark[G.ToVertex(e)] == UNVISITED){
              Visit(G, G.ToVertex(e));
              G.Mark[G.ToVertex(e)] = VISITED;
              Q.push(G.ToVertex(e));
           }
}}
```

图搜索的时间复杂度

- DFS 和 BFS 每个顶点访问一次，对每一条边处理一次 (无向图的每条边从两个方向处理)
  - 采用邻接表表示时，有向图总代价为 $\Theta(n+e)$，无向图为 $\Theta(n+2e)$
  - 采用相邻矩阵表示时，处理所有的边需要 $\Theta(n^2)$ 的时间 ，所以总代价为

$$\Theta(n+n^2)=\Theta(n^2)$$

拓扑排序

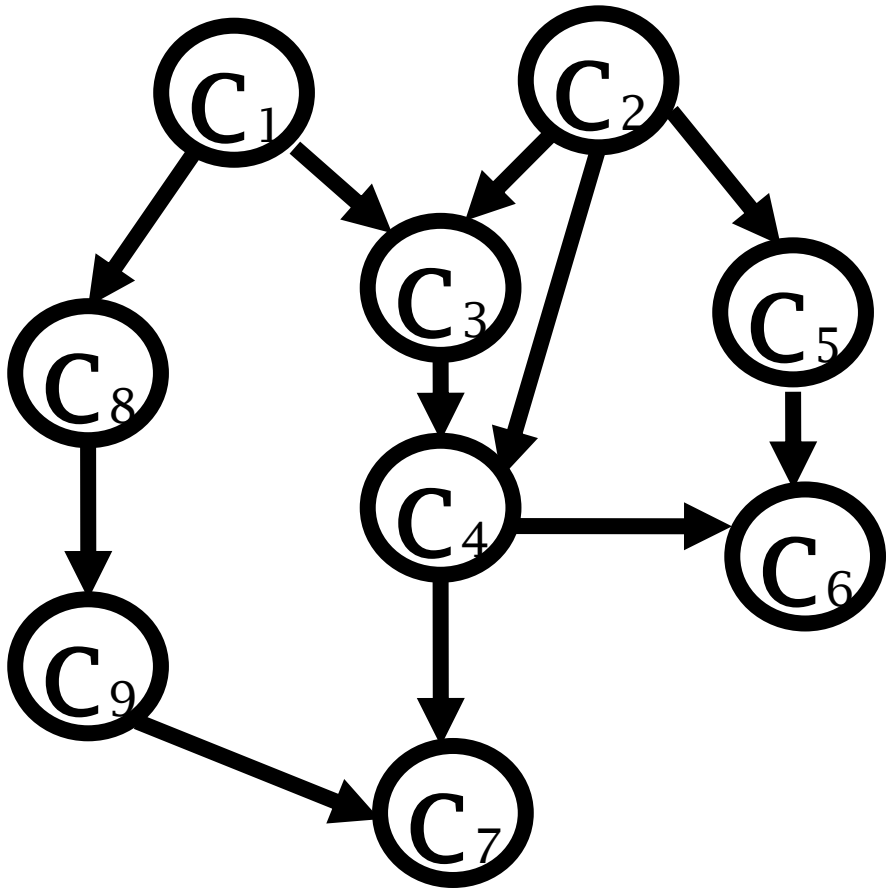
- 对于 **有向无环图** G= (V，E) ，V 里顶点的线性序列称作一个 **拓扑序列**，该顶点序列满足：
  - 若在有向无环图 G 中从顶点 v_i 到 v_j 有一条路径，则在序列中顶点 v_i 必在顶点 v_j 之前
- 拓扑排序 (topological sort)
  - 将一个 **有向无环图** 中所有顶点在不违反 **先决条件关系** 的前提下排成线性序列的过程称为 **拓扑排序**

7.4 图的遍历

课程代号	课程名称	先修课程
C1	高等数学	
C2	程序设计	
C3	离散数学	C1，C2
C4	数据结构	C2，C3
C5	算法分析	C2
C6	编译技术	C4，C5
C7	操作系统	C4，C9
C8	普通物理	C1
C9	计算机原理	C8

拓扑排序图例

学生课程的安排图



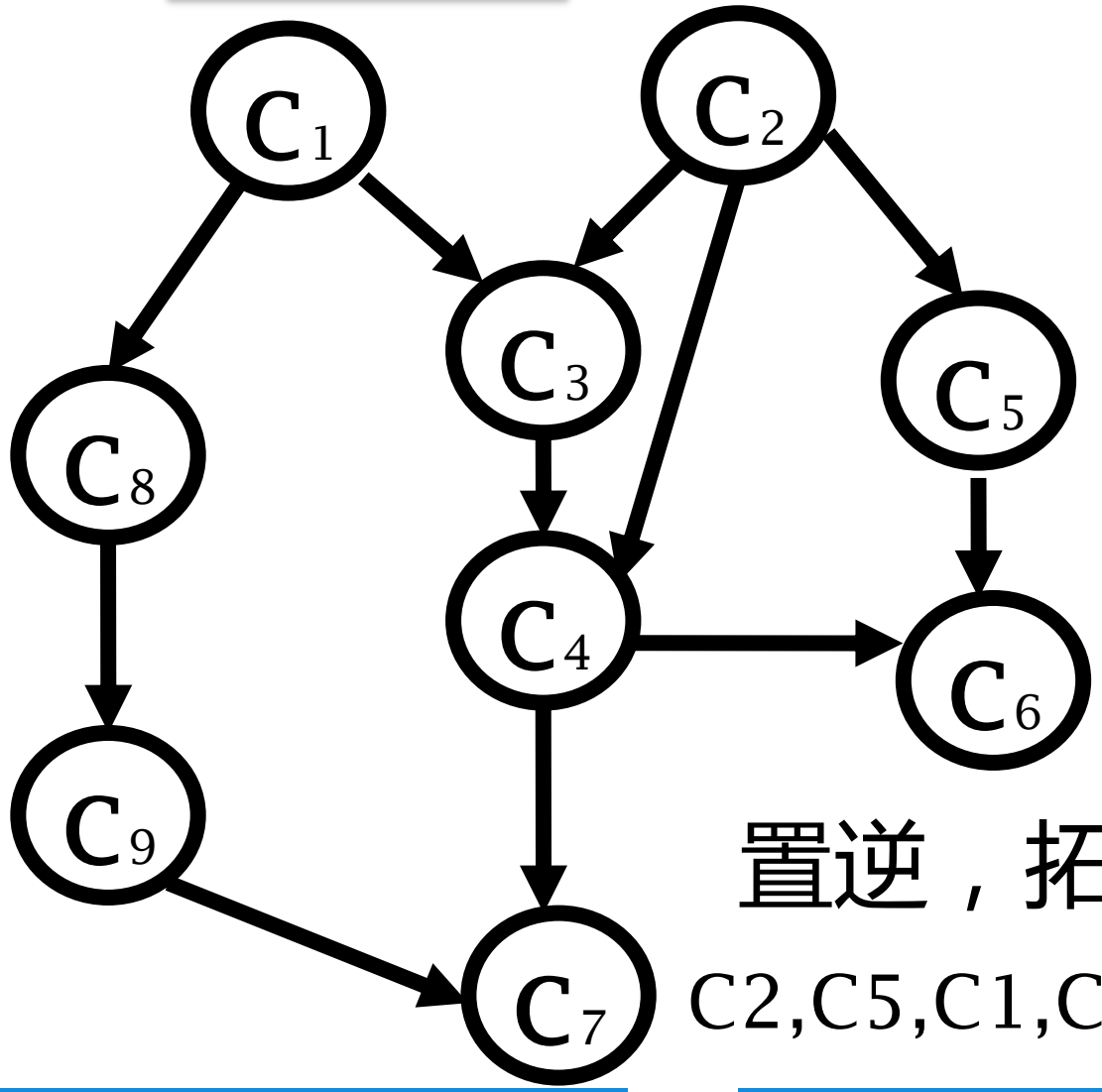
拓扑排序方法

- 任何 **有向无环图 (DAG)** ，其顶点都可以排在一个拓扑序列里，其拓扑排序的方法是：
  - (1) 从图中选择**任意**一个入度为0的顶点且输出之
  - (2) 从图中删掉此顶点及其所有的出边，将其入度减少1
  - (3) 回到第 (1) 步继续执行

用队列实现的图拓扑排序

```
void TopsortbyQueue(Graph& G) {
    for (int i = 0; i < G.VerticesNum(); i++) G.Mark[i] = UNVISITED; // 初始化
    using std::queue; queue<int> Q;                  // 使用STL中的队列
    for (i = 0; i < G.VerticesNum(); i++)            // 入度为0的顶点入队
        if (G.Indegree[i] == 0)  Q.push(i);
    while (!Q.empty()) {                             // 如果队列非空
        int v = Q.front(); Q.pop();                  // 获得队列顶部元素， 出队
        Visit(G,v);      G.Mark[v] = VISITED;        // 将标记位设置为VISITED
        for (Edge e = G.FirstEdge(v); G.IsEdge(e); e = G.NextEdge(e)) {
            G.Indegree[G.ToVertex(e)]--;             // 相邻的顶点入度减1
            if (G.Indegree[G.ToVertex(e)] == 0)      // 顶点入度减为0则入队
                Q.push(G.ToVertex(e));
    } }
    for (i = 0; i < G.VerticesNum(); i++)            // 判断图中是否有环
        if (G.Mark[i] == UNVISITED) {
            cout<<" 此图有环！";       break;
} }
```

7.4 图的遍历



按结点编号深度优先：

C6C7C4C3C9C8C1C5C2

置逆，拓扑序列为：

C2,C5,C1,C8,C9,C3,C4,C7,C6

深度优先搜索实现的拓扑排序

```
int  *TopsortbyDFS(Graph& G)  {                  // 结果是颠倒的
    for(int i=0; i<G.VerticesNum(); i++)         // 初始化
        G.Mark[i] = UNVISITED;
    int *result=new int[G.VerticesNum()];
    int index=0;
    for(i=0; i<G.VerticesNum(); i++)             // 对所有顶点
        if(G.Mark[i] == UNVISITED)
            Do_topsort(G, i, result, index);     // 递归函数
    for(i=G.VerticesNum()-1; i>=0; i--)          // 逆序输出
        Visit(G, result[i]);
    return result;
}
```

拓扑排序递归函数

```
void Do_topsort(Graph& G, int V, int *result, int&
index) {
   G.Mark[V] = VISITED;
   for (Edge e = G.FirstEdge(V);
         G.IsEdge(e); e=G.NextEdge(e)) {
        if (G.Mark[G.ToVertex(e)] == UNVISITED)
            Do_topsort(G, G.ToVertex(e),
                                result, index);
   }
   result[index++]=V;  // 相当于后处理
}
```

拓扑排序的时间复杂度

- 与图的深度优先搜索方式遍历相同
  - 图的每条边处理一次
  - 图的每个顶点访问一次
- 采用邻接表表示时，为 $\Theta(n+e)$
- 采用相邻矩阵表示时，为 $\Theta(n^2)$

图算法需要考虑的问题

- 是否支持
  - 有向图、无向图
  - 有回路的图
  - 非连通图
  - 权值为负
- 如果不支持
  - 则修改方案？

递归与非递归的拓扑排序

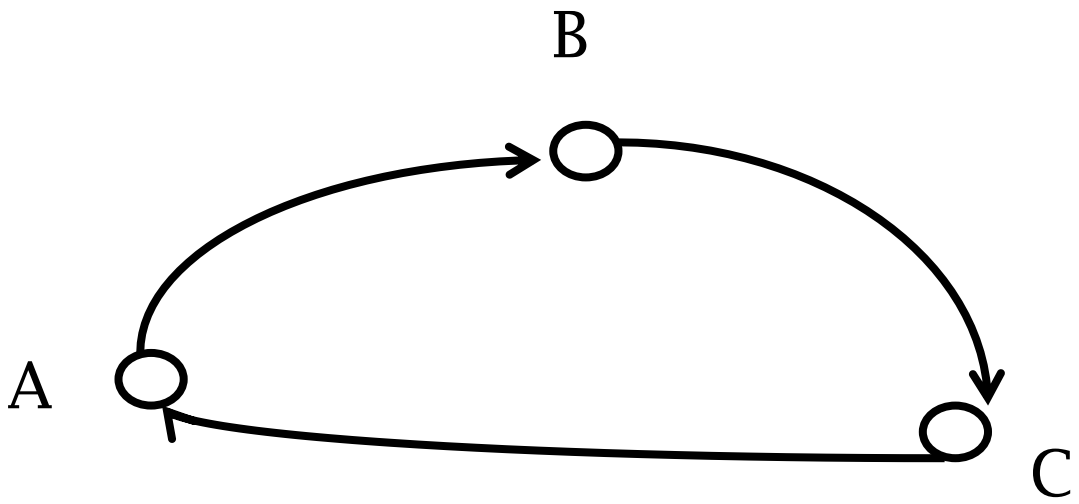
- 必须是有向图
- 必须是无环图
- 支持非连通图
- 不用考虑权值
- 回路
  - 非递归的算法，最后判断 (若还有顶点没有输出，肯定有回路)
  - 递归的算法要求判断有无回路

队列实现的拓扑排序讨论

- 怎么知道图中所有顶点的入度？
- 是否可以用栈来取代队列？

深度优先搜索拓扑排序讨论

- 对于起始点是否有要求？
- 是否可以处理有环的情况？



数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材