数据结构-17周

范围：图的DFS、最小生成树、Dijkstra算法、Floyd算法以及相关编程题

张天戈

本周习题

1. 请给出对 n 个顶点组成的无向图进行深度优先搜索遍历（DFS）的算法思想、伪代码，必要时给出注释，并分析算法的时空复杂度。

本周习题

与广度优先搜索不同，深度优先搜索（Depth-First-Search，DFS）类似于树的先序遍历。如其名称中所暗含的意思一样，这种搜索算法所遵循的搜索策略是尽可能“深”地搜索一个图。

它的基本思想如下：首先访问图中某一起始顶点 v，然后由 v 出发，访问与 v 邻接且未被访问的任一顶点 w_1，再访问与 w_1 邻接且未被访问的任一顶点 w_2……重复上述过程。当不能再继续向下访问时，依次退回到最近被访问的顶点，若它还有邻接顶点未被访问过，则从该点开始继续上述搜索过程，直至图中所有顶点均被访问过为止。

基本思想

一般情况下，其递归形式的算法十分简洁，算法过程如下：

算法过程

```
bool visited[MAX_VERTEX_NUM];   //访问标记数组
void DFSTraverse(Graph G){      //对图 G 进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE;       //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v)     //本代码中是从 v=0 开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){        //从顶点 v 出发，深度优先遍历图 G

    visit(v);                   //访问顶点 v
    visited[v]=TRUE;            //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighor(G,v,w))
        if(!visited[w]){        //w 为 u 的尚未访问的邻接顶点
            DFS(G,w);
        }  //if
}
```

本周习题

性能分析

DFS 算法是一个递归算法，需要借助一个递归工作栈，故其空间复杂度为 $O(|V|)$。

遍历图的过程实质上是对每个顶点查找其邻接点的过程，其耗费的时间取决于所用的存储结构。以邻接矩阵表示时，查找每个顶点的邻接点所需的时间为 $O(|V|)$，故总的时间复杂度为 $O(|V|^2)$。以邻接表表示时，查找所有顶点的邻接点所需的时间为 $O(|E|)$，访问顶点所需的时间为 $O(|V|)$，此时，总的时间复杂度为 $O(|V|+|E|)$。

本周习题

2. 设计一个算法，判断一个无向图 G 是否为一棵树，若是一棵树，则算法返回 true，否则返回 false。给出算法思想和可执行代码，必要时给出注释，并分析算法的时空复杂度。

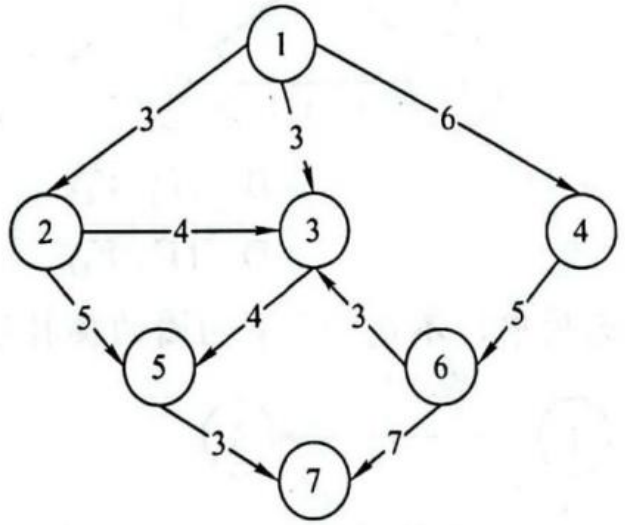
本周习题

一个无向图 G 是一棵树的条件是，G 必须是无回路的连通图或有 $n-1$ 条边的连通图。这里采用后者作为判断条件。对连通的判定，可用能否遍历全部顶点来实现。可以采用深度优先搜索算法在遍历图的过程中统计可能访问到的顶点个数和边的条数，若一次遍历就能访问到 n 个顶点和 $n-1$ 条边，则可断定此图是一棵树。算法实现如下：

```
bool isTree(Graph& G){
    for(i=1;i<=G.vexnum;i++)
        visited[i]=FALSE;              //访问标记 visited[]初始化
    int Vnum=0,Enum=0;                 //记录顶点数和边数
    DFS(G,1,Vnum,Enum,visited);
    if(Vnum==G.vexnum&&Enum==2*(G.vexnum-1))
        return  true;                  //符合树的条件
    else
        return  false;                 //不符合树的条件
}
void DFS(Graph& G,int v,int& Vnum,int& Enum,int visited[]){
//深度优先遍历图 G，统计访问过的顶点数和边数，通过 Vnum 和 Enum 返回
    visited[v]=TRUE;Vnum++;            //作访问标记，顶点计数
    int w=FirstNeighbor(G,v);          //取 v 的第一个邻接顶点
    while(w!=-1){                      //当邻接顶点存在
        Enum++;                        //边存在，边计数
        if(!visited[w])                //当该邻接顶点未访问过
            DFS(G,w,Vnum,Enum,visited);
        w=NextNeighbor(G,v,w);
    }
}
```

本周习题

3. 给定如下有向图。



（1）写出该图的邻接矩阵；

（2）给出该图任意两个拓扑序列；

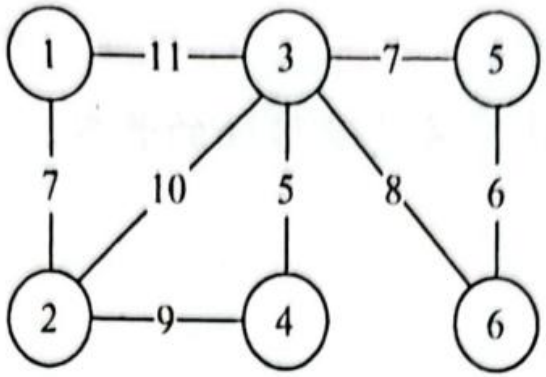
（3）若将该图视为无向图，分别用 Prim 算法和 Kruskal 算法求最小生成树，给出详细步骤。

本周习题

山东大学 SHANDONG UNIVERSITY

本周习题

4. 对下图所示的无向图，按照 Dijkstra 算法，写出从顶点 1 到其他各个顶点的最短路径和最短路径长度。



本周习题

山东大学 SHANDONG UNIVERSITY

本周习题

算法基本思想与实现

求所有顶点之间的最短路径问题描述如下：已知一个各边权值均大于 0 的带权有向图，对任意两个顶点 $v_i \neq v_j$，要求求出 v_i 与 v_j 之间的最短路径和最短路径长度。

Floyd 算法的基本思想是：递推产生一个 n 阶方阵序列 $A^{(-1)}, A^{(0)}, \cdots, A^{(k)}, \cdots, A^{(n-1)}$，其中 $A^{(k)}[i][j]$ 表示从顶点 v_i 到顶点 v_j 的路径长度，k 表示绕行第 k 个顶点的运算步骤。初始时，对于任意两个顶点 v_i 和 v_j，若它们之间存在边，则以此边上的权值作为它们之间的最短路径长度；若它们之间不存在有向边，则以∞作为它们之间的最短路径长度。以后逐步尝试在原路径中加入顶点 k（$k = 0, 1, \cdots, n-1$）作为中间顶点。若增加中间顶点后，得到的路径比原来的路径长度减少了，则以此新路径代替原路径。算法描述如下：

定义一个 n 阶方阵序列 $A^{(-1)}, A^{(0)}, \cdots, A^{(n-1)}$，其中，

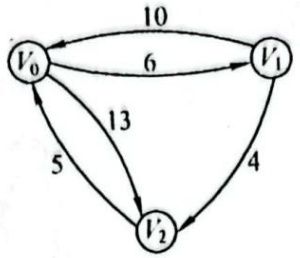
$$A^{(-1)}[i][j] = \text{arcs}[i][j]$$

$$A^{(k)}[i][j] = \text{Min}\{A^{(k-1)}[i][j],\ A^{(k-1)}[i][k] + A^{(k-1)}[k][j]\},\quad k = 0, 1, \cdots, n-1$$

式中，$A^{(0)}[i][j]$是从顶点 v_i 到 v_j、中间顶点是 v_0 的最短路径的长度，$A^{(k)}[i][j]$是从顶点 v_i 到 v_j、中间顶点的序号不大于 k 的最短路径的长度。Floyd 算法是一个迭代的过程，每迭代一次，在从 v_i 到 v_j 的最短路径上就多考虑了一个顶点；经过 n 次迭代后，所得到的 $A^{(n-1)}[i][j]$就是 v_i 到 v_j 的最短路径长度，即方阵 $A^{(n-1)}$中就保存了任意一对顶点之间的最短路径长度。

本周习题

实例分析



(a) 有向图G

$$\begin{bmatrix} 0 & 6 & 13 \\ 10 & 0 & 4 \\ 5 & \infty & 0 \end{bmatrix}$$

(b) G的邻接矩阵

图 6.19　带权有向图 G 及其邻接矩阵

表 6.2　Floyd 算法的执行过程

A	$A^{(-1)}$			$A^{(0)}$			$A^{(1)}$			$A^{(2)}$		
	V_0	V_1	V_2	V_0	V_1	V_2	V_0	V_1	V_2	V_0	V_1	V_2
V_0	0	6	13	0	6	13	0	6	**10**	**0**	6	10
V_1	10	0	4	10	0	4	10	0	4	**9**	0	4
V_2	5	∞	0	5	**11**	0	5	11	0	5	11	0

图 6.19 所示为带权有向图 G 及其邻接矩阵。应用 Floyd 算法求所有顶点之间的最短路径长度的过程如表 6.2 所示。算法执行过程的说明如下。

初始化：方阵 $A^{(-1)}[i][j] = \text{arcs}[i][j]$。

第一轮：将 v_0 作为中间顶点，对于所有顶点对 $\{i, j\}$，如果有 $A^{-1}[i][j] > A^{-1}[i][0] + A^{-1}[0][j]$，则将 $A^{-1}[i][j]$ 更新为 $A^{-1}[i][0] + A^{-1}[0][j]$。有 $A^{-1}[2][1] > A^{-1}[2][0] + A^{-1}[0][1] = 11$，更新 $A^{-1}[2][1] = 11$，更新后的方阵标记为 A^0。

第二轮：将 v_1 作为中间顶点，继续检测全部顶点对 $\{i, j\}$。有 $A^0[0][2] > A^0[0][1] + A^0[1][2] = 10$，更新 $A^0[0][2] = 10$，更新后的方阵标记为 A^1。

第三轮：将 v_2 作为中间顶点，继续检测全部顶点对 $\{i, j\}$。有 $A^1[1][0] > A^1[1][2] + A^1[2][0] = 9$，更新 $A^1[1][0] = 9$，更新后的方阵标记为 A^2，此时 A^2 中保存的就是任意顶点对的最短路径长度。

本周习题

复杂度分析

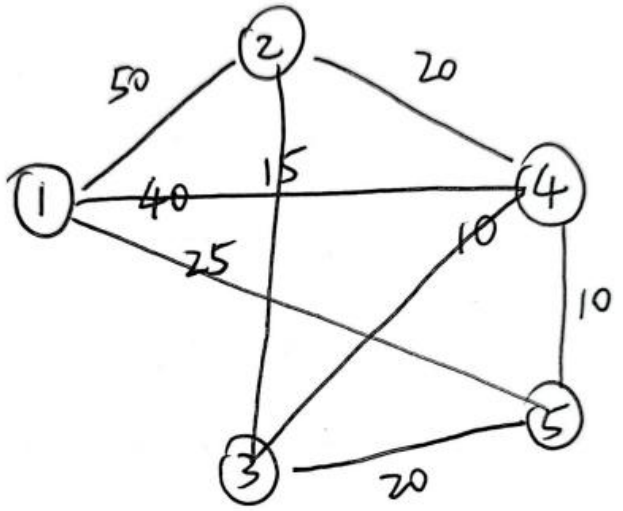
Floyd 算法的时间复杂度为 $O(|V|^3)$。不过由于其代码很紧凑，且并不包含其他复杂的数据结构，因此隐含的常数系数是很小的，即使对于中等规模的输入来说，它仍然是相当有效的。

Floyd 算法允许图中有带负权值的边，但不允许有包含带负权值的边组成的回路。Floyd 算法同样适用于带权无向图，因为带权无向图可视为权值相同往返二重边的有向图。

也可以用单源最短路径算法来解决每对顶点之间的最短路径问题。轮流将每个顶点作为源点，并且在所有边权值均非负时，运行一次 Dijkstra 算法，其时间复杂度为 $O(|V|^2)\cdot|V| = O(|V|^3)$。

本周习题

5. 给定如下无向图，按照基于动态规划思想的 Floyd 算法，求出各顶点对之间的最短路径长度。



本周习题

山东大学 SHANDONG UNIVERSITY

本周习题

山东大学 SHANDONG UNIVERSITY

Q&A