数据结构作业(第十次)

PB20111686 黄瑞轩

7.22

图的邻接表储存结构如下所示：

```
#define MAX_VERTEX_NUM 20
typedef int InfoType;
typedef int VertexType;
typedef struct ArcNode {
    int adjVex;              //边指向的顶点序号
    struct ArcNode* nextArc; //下一条边
    InfoType* info;          //边上信息
} ArcNode;
typedef struct VNode {
    VertexType data;         //顶点信息
    ArcNode* firstArc;       //第一条边指针
} VNode, AdjList[MAX_VERTEX_NUM];
typedef struct {
    AdjList vertices;
    int vexNum, arcNum;      //当前顶点数和弧数
    int kind;                //图的种类标志
} ALGraph;
```

算法如下所示：

```
std::string isVisited;
void initSTring(std::string* s, int k) {
    while (k--)s += '0';
}
bool helper(ALGraph* G, int i, int j) {
    if (isVisited[i] == '1') {
        //如果当前已被访问，说明从i到j没有找到路径
        return false;
    }
    isVisited[i] = '1';
    auto trav = G->vertices[i]->firstArc; //trav指向v_i的第一条依附的边
    while (trav != nullptr) {
        if (isVisited[trav->adjVex] == '0') {
            //未被访问
            isVisited[trav->adjVex] = '1';
            if (trav->adjVex != j) {
                if (helper(G, trav->adjVex, j)) {
                    return true;
                }
                else {
                    trav = trav->nextArc;
                }
            }
        }
        else {
            trav = trav->nextArc;
        }
    }
    return false;
}
bool ExistRoad(ALGraph* G, int i, int j) {
    initString(&isVisited, G->vexNum);
    return helper(G, i, j);
}
```

7.27

需要给定顶点序号i,j和轨道长度k。算法如下所示：

```
void initSTring(std::string* s, int k) {
    while (k--)s += '0';
}
bool helper(ALGraph* G, int i, int j, int k, int dep) {
    std::string path;//当前路径
```

```
    if (dep == 0)initString(&path, k);
    if (i == j && dep == k)return true;
    if (dep == k)return false;
    if (path[i] == '1')return false;//成环了
    path[i] = '1';
    for (auto trav = G->vertices[i]->firstArc; trav != nullptr; trav = trav->nextArc) {
        if (helper(G, trav->adjVex, j, k - 1, dep + 1)) {
            return true;
        }
    }
    if (dep == 0) {
        path.clear();
    }
    return false;
}
bool ExistRoad(ALGraph* G, int i, int j, int k) {
    return helper(G, i, j, 0);
}
```

图的非递归DFS遍历

算法如下：

```
bool isVisited(std::vector<int>* Visited, int t) {
    int size = visited->size();
    for (int i = 0; i < size; i++) {
        if ((*Visited)[i] == t)return true;
    }
    return false;
}
void Graph_DFS_Traverse(Graph* G, int S) {
    // S是起始顶点序号
    std::vector<int> Visited;
    std::stack<int> stk;
    stk.push(S);
    while (stk.size()) {
        int cur = stk.top();
        stk.pop();
        if (!isVisited(&Visited, cur)) {
            Visited.push(cur);
            for (auto trav = G->vertices[i]->firstArc; trav != nullptr; trav = trav->nextArc) {
                if (!isVisited(&Visited, trav->adjVex)) {
                    stk.push(trav->adjVex);
                }
            }
        }
    }
}
```

7.7最小生成树

（1）这个图的邻接矩阵为

$$\begin{bmatrix} 0 & 4 & 3 & \infty & \infty & \infty & \infty & \infty \\ 4 & 0 & 5 & 5 & 9 & \infty & \infty & \infty \\ 3 & 5 & 0 & 5 & \infty & \infty & \infty & 5 \\ \infty & 5 & 5 & 0 & 7 & 6 & 5 & 4 \\ \infty & 9 & \infty & 7 & 0 & 3 & \infty & \infty \\ \infty & \infty & \infty & 6 & 3 & 0 & 2 & \infty \\ \infty & \infty & \infty & 5 & \infty & 2 & 0 & 6 \\ \infty & \infty & 5 & 4 & \infty & \infty & 6 & 0 \end{bmatrix}$$

从$U=\{a\},TE=\varnothing$开始按Prim算法求其最小生成树：

- $U=\{a,c\},TE=\{ac\}$;
- $U=\{a,c,b\},TE=\{ac,ab\}$;
- $U=\{a,c,b,d\},TE=\{ac,ab,bd\}$;
- $U=\{a,c,b,d,h\},TE=\{ac,ab,bd,dh\}$;
- $U=\{a,c,b,d,h,g\},TE=\{ac,ab,bd,dh,dg\}$;
- $U=\{a,c,b,d,h,g,f\},TE=\{ac,ab,bd,dh,dg,gf\}$;
- $U=\{a,c,b,d,h,g,f,e\},TE=\{ac,ab,bd,dh,dg,gf,fe\}$;

$\bar{U}=\varnothing$，算法退出，此时的TE即为最小生成树边集。

（2）这个图的邻接表为：

$$[a], N = b, c$$
$$[b], N = c, d, e$$
$$[c], N = a, b, d, h$$
$$[d], N = b, c, e, f, g, h$$
$$[e], N = b, d, f$$
$$[f], N = d, e, g$$
$$[g], N = d, f, h$$
$$[h], N = c, d, g$$

从$U = \varnothing, TE = \varnothing$按Kruscal算法求其最小生成树：

- $U = \{f, g\}, TE = \{fg\}$;
- $U = \{f, g, e\}, TE = \{fg, ef\}$;
- $U = \{f, g, e, a, c\}, TE = \{fg, ef, ac\}$;
- $U = \{f, g, e, a, c, d, h\}, TE = \{fg, ef, ac, dh\}$;
- $U = \{f, g, e, a, c, d, h, b\}, TE = \{fg, ef, ac, dh, ab\}$;
- $U = \{f, g, e, a, c, d, h, b\}, TE = \{fg, ef, ac, dh, ab, cd, dg\}$;

此时$|TE| = |V(G)| - 1$，算法退出，此时的TE即为最小生成树边集。

7.34

默认这是连通图，算法如下：

```
int in_degree[MAX_VERTEX_NUM] = { 0 };
void stat_in_degree(ALGraph* G) {
    int size = G->vexNum;
    auto vex = G->vertices;
    for (int i = 0; i < size; i++) {
        for (auto trav = vex[i]->firstArc; trav != nullptr; trav = trav->nextArc) {
            in_degree[trav->adjVex]++;
        }
    }
}
void encode(ALGraph* G) {
    stat_in_degree(G);
    int size = G->vexNum;
    auto vex = G->vertices;
    int cur_no = 0;
    while (1) {
        bool isEnd = true;
        for (int i = 0; i < size; i++) {
            if (in_degree[i] == 0) {
                isEnd = false;
                in_degree[i]--;
                give_number(i, cur_no);//预先定义的赋值函数
                for (auto trav = vex[i]->firstArc; trav != nullptr; trav = trav->nextArc) {
                    in_degree[trav->adjVex]--;//所有外邻顶点入度减1
                }
            }
        }
        //删除所有入度为0的顶点
        if (isEnd) {
            break;
        }
        cur_no++;
    }
}
```