数据结构 hw10

刘良宇 PB20000180

7.22

```
bool is_path(Graph& g, int i, int j, bool reset = true) {
    static bool is_visit[MAX_VERTEX_NUM];
    if (reset) {
        for (int i = 0; i < g.vexnum; i++) {
            is_visit[i] = false;
        }
    }
    if (i == j)
        return true;
    if (is_visit[i])
        return false;
    is_visit[i] = true;
    for (ArcNode* p = g.vertices[i]->firstArc; p; p = p->nextArc) {
        if (is_path(g, p->adjvex, j, false))
            return true;
    }
    return false;
}
```

7.27

```
bool is_path(Graph& g, int i, int j, int k, int depth = 0) {
    static int* path;       // 记录当前路径
    if (depth == 0) {
        path = new int[k]();
    }
    if (i == j && depth == k)
        return true;
    if (depth == k)
        return false;
    // 如果这个点在当前路径上曾经经过，那么就不要
    for (int _i = 0; _i < depth; _i++) {
        if (i == path[_i])
            return false;
    }
    path[depth] = i;
    for (ArcNode* p = g.vertices[i]->firstArc; p; p = p->nextArc) {
        bool res = is_path(g, p->adjvex, j, k - 1, false);
        if (res && depth == 0) {
            delete path;
            return true;
        }
        if (res) {
            return true;
        }
    }
    if (depth == 0)
```

```
            delete path;
        return false;
}
```

图的非递归 DFS 遍历

```
bool is_visited(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) {
    // 从 s 开始遍历
    std::vector<int> Visited;
    std::stack<int> stk;
    stk.push(S);
    while (!stk.empty()) {
        int cur = stk.top();
        stk.pop();
        if (!is_visited(&Visited, cur)) {
            Visited.push(cur);
            for (ArcNode* p = G->vertices[i]->firstArc; p;
                 p = p->nextArc) {
                if (!is_visited(&Visited, trav->adjVex)) {
                    stk.push(trav->adjVex);
                }
            }
        }
    }
}
```

7.7

邻接矩阵 & Prim

0	**4**	**3**	∞	∞	∞	∞	∞
4	0	5	5	9	∞	∞	∞
3	5	0	5	∞	∞	∞	5
∞	5	5	0	7	6	5	4
∞	9	∞	7	0	3	∞	∞
∞	∞	∞	6	3	0	2	∞
∞	∞	∞	5	∞	2	0	6
∞	∞	5	4	∞	∞	6	0

假设初始选择 a

则执行步骤：

ac

ac, ab

ac, ab, bd

ac, ab, bd, dh

ac, ab, bd, dh, dg

ac, ab, bd, dh, dg, gf

ac, ab, bd, dh, dg, gf, fe

邻接表 & Kruskal

a: b, c

b: a, c, d, e

c: a, b, d, h

d: b, c, e, f, g, h

e: b, d, f

f: d, e, g

g: d, f, h

h: c, d, g

依次选择权最小，不成环的边：

fg, ef, ac, ab, dh, cd, dg

7.34

每次找到入度为 0 的顶点即可。

```
void get_current_in(Graph& g, int* in) {
    for (int i = 0; i < g.vexnum; i++) {
        in[i] = 0;
    }
    for (int i = 0; i < g.vexnum; i++) {
        for (Arcnode* p = g.vertices[i]->firstArc; p; p = p->nextArc) {
            in[p->adjvex]++;
        }
    }
}

void sort(Graph& g, int* out) {
    // 输出到 out 数组，从 0 开始，是结点的编号
    int now = 0;  // 现在到了 out 的第几位
    // 首先申请一个数组统计各个结点的入度
    int in = new int[g.vexnum]();
    while (true) {
        get_current_in(g, in);
```

```
        if (g.vexnum == 0)
            break;
        for (int i = 0; i < g.vexnum; i++) {
            if (in[i] == 0) {
                DeleteVex(g, i);
                out[now++] = i;
            }
        }
    }
}
```