作业八

1 假设以邻接表表示法作为图的存储结构，设计图的深度优先遍历递归算法。

```
template<class T>
inline void ALGraph<T>::DFS(int v, vector<bool>& visited)
{
    cout << adjlist[v].data << " ";
    visited[v] = true;
    for (EdgeNode* p = adjlist[v].firstedge; p; p = p->nextedge)
        if (!visited[p->adjvex])
            DFS(p->adjvex, visited);
}
template<class T>
inline void ALGraph<T>::DFS()
{
    vector<bool> visited(vexnum, false);
    for (int i = 0; i < vexnum; i++)
        if (!visited[i])
            DFS(i, visited);
    }
```

2. 试基于图的广度优先搜索策略编写一种算法，判别以邻接表方式存储的有向图中是否存在由顶点 v_i 到顶点 v_j 的路径（$i \neq j$）。

```
template<class T>
inline bool ALGraph<T>::ifExistPath_BFS(T a, T b)
{
    int i = GetSubscriptOfVex(a), j = GetSubscriptOfVex(b);
    int e, k;
    vector<bool> visited(vexnum, false);
    queue<int> q;
    q.push(i);
    while (!q.empty())
    {
        e = q.front();
            q.pop();
        visited[e] = true;
        for (EdgeNode* p = adjlist[e].firstedge; p; p = p->nextedge)
        {
            k = p->adjvex;//当前指向顶点的位置
            if (k == j)
                return true;
            else if (!visited[k])
                q.push(k);
        }
```

```
    }
    return false;
}
```

3. 试修改 Prim 算法，使之能在邻接表存储结构上实现求有向图的最小生成森林（森林的存储结构为孩子兄弟链表）。

```
template<class T>
inline void ALGraph<T>::Prim(int v, vector<bool>& visited, vector<Couple<T>>& mintree)
{
    vector<Edge<T>> miniedges(vexnum);//建立该点邻接点的数组，记录到达所需权值
    for (int i = 0; i < vexnum; i++)
    {
        miniedges[i].adjvex = adjlist[v].data;
        miniedges[i].lowcost = GetWeight(v, i);
    }
    miniedges[v].lowcost = 0;
    for (int i = 1; i < vexnum; i++)
    {
        int k = 0;
        int min = INT_MAX;
        for (int j = 0; j < vexnum; j++)
            if (miniedges[j].lowcost < min && miniedges[j].lowcost != 0)//注意!=0
            {
                min = miniedges[j].lowcost;
                k = j;
            }
        Couple<T> tmp;
        tmp.parent = miniedges[k].adjvex;
        tmp.child = adjlist[k].data;
        visited[GetSubscriptOfVex(tmp.parent)] = true;
        visited[GetSubscriptOfVex(tmp.child)] = true;
        mintree.push_back(tmp);
        miniedges[k].lowcost = 0;//此处排除已进入的点
        for (int j = 0; j < vexnum; j++)
        {
            int costkj = GetWeight(k, j);
            if (costkj < miniedges[j].lowcost && miniedges[j].lowcost != 0 && costkj !=
0)//注意!=0
            {
                miniedges[j].adjvex = adjlist[k].data;
                miniedges[j].lowcost = costkj;
            }
        }
    }
}
```

```
template<class T>
inline void ALGraph<T>::MinSpanForest_Prim()
{
    vector<bool> visited(vexnum, false);
    vector<vector<Couple<T>>> mintrees(vexnum);
    vector<Couple<T>> minforest;
    for (int i = 0; i < vexnum; i++)
        if (!visited[i])
            Prim(i, visited, mintrees[i]);
    for(int i=0;i<vexnum;i++)
        if (!mintrees[i].empty())
        {
            Couple<T> tmp;
            tmp.parent = 'R';
            tmp.child = adjlist[i].data;
            minforest.push_back(tmp);
            int len = mintrees[i].size();
            for (int j = 0; j < len; j++)
                minforest.push_back(mintrees[i][j]);
        }
    for (int i = 0; i < minforest.size(); i++)
        cout << minforest[i].parent << "," << minforest[i].child << endl;
    //此处应使用森林生成算法（二元组），待补
}
```

4. 采用邻接表存储结构，编写一个判别无向图中任意两个给定的顶点之间是否存在一条长度为 *k* 的简单路径的算法。

```
template<class T>
inline bool ALGraph<T>::ExistPath_DFS_Lenth(int i, int j, int k, int &n, vector<bool>
&visited)
{
    EdgeNode* p;
    visited[i] = true;
    if (i == j && n == k)
        return true;
    n++;
    p = adjlist[i].firstedge;
    while (p)
    {
        if (!visited[p->adjvex])
            if (ExistPath_DFS_Lenth(p->adjvex, j, k, n, visited))
                return true;
        p = p->nextedge;
    }
    visited[i] = false;
```

```
    n--;
    return false;
}
template<class T>
inline bool ALGraph<T>::ExistPath_Lenth(int i, int j, int k)
{
    vector<bool> visited(vexnum, false);
    int n = 0;
    return ExistPath_DFS_Lenth(i, j, k, n, visited);
}
```

5. 找出邻接矩阵表示的图中的一个回路

```
template<class T>
inline void MGraph<T>::FindLoop_DFS(int vi, int v, vector<bool>& visited, vector<int>&
path)
{
    if (v != vi)
        visited[v] = true;
    path.push_back(v);
    for (int i = 0; i < vexnum; i++)
    {
        if (edges[v][i] ==1 && i == vi)
        {
            path.push_back(i);
            for (int i = 0; i < path.size(); i++)
                cout << path[i] << " ";
            cout << endl;
        }
        if (edges[v][i] ==1 && !visited[i])
            FindLoop_DFS(vi, i, visited, path);
    }
    path.pop_back();
}
template<class T>
inline void MGraph<T>::FindLoop(int vi)
{
    vector<int> path;
    bool exist = false;
    vector<bool> visited(vexnum, false);
    FindLoop_DFS(vi, vi, visited, path);
    if (path.empty())
        cout << vi << " 顶点无回路" << endl;
}
```