```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//   最小生成树及其结点的类定义
//   Author：Melissa M.CAO
//   Belong：Section of software theory，School of Computer Engineering & Science，
Shanghai University
//   Version： 1.0
////////////////////////////////////////////////////////////////////////////////
//////////////////////////

#pragma once

#include "MyHeap.h"

template<class vertexType, class arcType> class MinSpanTree;
template<class vertexType, class arcType> class Graph;
template<class MSTArcNodeForHeap> class MinSpanTreeHeap;

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//   最小生成树结点的类定义
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class vertexType, class arcType> class MSTArcNode
{
    friend class MinSpanTree<vertexType, arcType>;
    friend class Graph<vertexType, arcType>;
    friend class MinSpanTreeHeap<MSTArcNode>;

    private:
        vertexType adjvex1, adjvex2;
        arcType weight;
    public:
        vertexType GetFirst() { return adjvex1;}
        vertexType GetSecond() { return adjvex2;}
        arcType GetWeight() { return weight; }
        void SetWeight(arcType w) { weight = w; }
        void SetAdiverFirst(vertexType d) { adjvex1 = d;}
        void SetAdiverSecond(vertexType d) { adjvex2 = d;}
};

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//   最小生成树的类定义
```

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class vertexType, class arcType> class MinSpanTree
{
    friend class Graph<vertexType, arcType>;

    private:
        static const int MaxNumArc = 20;    //最大边数，与图定义中最大顶点数一致
        MSTArcNode<vertexType, arcType> *arctable;
        int CurrentNumArcs;

    public:
        MinSpanTree() : CurrentNumArcs(0) { arctable = new MSTArcNode<vertexType,
arcType>[MaxNumArc]; }; //构造函数
        MinSpanTree(int  size)  :  CurrentNumArcs(0)  {  arctable  =  new
MSTArcNode<vertexType, arcType>[size]; };  //构造函数
        void Insert(MSTArcNode<vertexType, arcType> &e) { arctable[CurrentNumArcs]
= e; CurrentNumArcs++; }    //插入边函数
        void Display();     //显示内容函数
};


////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//   最小生成树结点的类定义，用于堆，即可以与堆派生，又与 MSTArcNode 直接对应
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template<class arcType> class MSTArcNodeForHeap
{
    //friend class MinSpanTreeHeap<MSTArcNodeForHeap>;

    private:
        arcType weight;
        int id;
    public:
        void SetWeight(arcType w) { weight = w; }
        void SetID(int d) { id = d;}
        int GetID() {return id;}
        arcType GetWeight() { return weight;}
};

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   最小生成树中用到的“堆的”的类定义--主要是元素的比较，牵扯向上和向下调整
```

```
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
template<class Type> class MinSpanTreeHeap : public MyHeap<Type>
{
    private:
        void FilterDown(int p);
        void FilterUp(int p);

    public:
    //  MinSpanTreeHeap(Type *a, int n, int heaptpye) : MyHeap(a, n, heaptpye)  {};
// 调用父类的构造函数
        //用上述构造函数，调用的父类的 FilterDown 函数，
        //调用插入和删除时，确使用本类的 FilterDown 和 Filterp 函数。
        //关于构造函数与父类构造函数间的关系，有待细看语法
    //构造函数不能继承。其它的主要是重载和函数 FilterDown 和 Filterp
        MinSpanTreeHeap(int n, int heaptpye) : MyHeap(n, heaptpye)  {};        // 调
用父类的构造函数
        void out();
};


////////////////////////////////////////////////////////////////////////////////
/////////////////////////
//      最小生成树类的实现，显示函数，用于验证结果
////////////////////////////////////////////////////////////////////////////////
/////////////////////////
template<class vertexType, class arcType> void MinSpanTree<vertexType, arcType> ::
Display()
{
    int i;

    cout << "最小生成树按序生成的边依次为：" << endl;
    for (i = 0; i < CurrentNumArcs; i++)
    {
        cout << "第" << i+1 << "条边：（" << arctable[i].adjvex1 << "，" <<
arctable[i].adjvex2 << "）    ";
        cout << arctable[i].weight << ";         ";
        if (i+1 % 3 == 0)
            cout << endl;
    }
    cout<<endl;
}

/*-----------为调试程序-----------*/
```

```
template<class Type> void MinSpanTreeHeap<Type> :: out()
{
    int i;
    for (i = 0; i < heapCurrentSize; i++)
    {
        cout  <<  ″权 值″  <<  heapArr[i]->GetWeight()  <<  ″， 输 入 序 号″  <<
heapArr[i]->GetID() << ″; ″;
    }
    cout << endl;
}

////////////////////////////////////////////////////-----------------------------/
///////////////////////////
//   最小生成树中用到的“堆的”的类实现--主要是元素的比较，如果不重载，直接比较的
是指针地址之类的值
//  本质上就是 Type 引起的“元素比较[<，>，=，!=]”操作的重载问题
//  第二种解决方案：对所定义的堆元素类型/抽象类型的比较操作[<，>，=，!=]进行重载
--有兴趣者可以尝试
////////////////////////////////////////////////////-----------------------------/
///////////////////////////
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   向下调整函数
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
template<class Type> void MinSpanTreeHeap<Type>::FilterDown(const int start)
{
    int i = start, j;
    Type temp = heapArr[i];
    j = 2*i+1;
    while(j <= heapCurrentSize-1)
    {
        if ( myType == 1)
        {
            if   (j   <   heapCurrentSize-1   &&   heapArr[j]->GetWeight()   >
heapArr[j+1]->GetWeight())
                j++;
            if (temp->GetWeight() <= heapArr[j]->GetWeight())
                break;
            else
            {
                heapArr[i] = heapArr[j];
                i = j;
                j = 2*j+1;
            }
        }
        else
        {
            if ( myType == 2)
            {
                if  (j  <  heapCurrentSize-1  &&  heapArr[j]->GetWeight()  <
heapArr[j+1]->GetWeight())
                    j++;
                if (temp->GetWeight() >= heapArr[j]->GetWeight())
                    break;
                else
                {
                    heapArr[i] = heapArr[j];
                    i = j;
                    j = 2*j+1;
                }
            }
            else
            {
                cout << "既不是小顶堆又不是大顶堆，程序出错了！请退出运行过程仔细
检查！" << endl;
                exit(1);
            }
        }
    }
    heapArr[i] = temp;
}

////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//   向上调整函数
////////////////////////////////////////////////////////////////////////////////
///////////////////////////
template<class Type> void MinSpanTreeHeap<Type>::FilterUp(int p)
{
    int j = p,i;
    Type temp = heapArr[j];
    i = (j-1)/2;
    while (j > 0)
    {
        if ( myType == 1)
        {
            if (heapArr[i]->GetWeight() <= temp->GetWeight())
```

```
                break;
            else
            {
                heapArr[j] = heapArr[i];
                j = i;
                i = (j-1)/2;
            }
        }
        else
        {
            if ( myType == 2)
            {
                if (heapArr[i]->GetWeight() >= temp->GetWeight())
                    break;
                else
                {
                    heapArr[j]->SetWeight(heapArr[i]->GetWeight());
                    j = i;
                    i = (j-1)/2;
                }
            }
            else
            {
                cout << "既不是小顶堆又不是大顶堆，程序出错了！请退出运行过程仔细
检查！" << endl;
                exit(1);
            }
        }
    }
    heapArr[j] = temp;
}
```