bool Kruskal(edge<T> \*spanningTreeEdges)

{//求最小生成树，若该图连通则返回true，且sTE[0:n-2]中存有最小生成树的边

//n是图中的顶点数，e是边数

int n=this->n;

int e=this->e;

edge<T> \*Edge=new edge<T> [n+1];

int k=1; //数组Edge的索引

//下面对edge数组初始化：把所有边加入edge数组

for(int i=1;i<=n;i++)

{//取所有关联顶点i的边

myIterator \*ii=iterator(i);

int j; T w;

while( (j=ii->next(w)) !=0)

{

if(i<j) //避免重复加边

Edge[k++]=edge<int> (i,j,w);

}

}

//用小根堆表示边集的效率较好

minHeap< edge<T> > heap(1);

heap.initialize(Edge,e); cout<<heap<<endl;

fastUnionFind uf(n);

k=0; //索引

while( e>0 && k<n-1 )

{//生成树没有完成并且还有边存在

edge<T> x=heap.top();

heap.pop();

e--;

int a=uf.find(x.vertex1());

int b=uf.find(x.vertex2());

if(a!=b)

{//保证在没有环路的情况下选取边x

spanningTreeEdges[k++]=x;

uf.unite(a,b);

}

}

cout<<"k="<<k<<",n-1="<<n-1<<endl;

if(k==n-1) return true;

else return false;

}

bool kruskal(edge<T> \*spanningTreeEdges)

{

int n =this->n;

int e =this->e;

edge<T> \*Edge = new edge<T> [e + 1];

int k = 0;

for (int i = 1; i <= n; i++)

{

myIterator \*ii=iterator(i);

int j; T w;

while ((j = ii->next(w)) != 0)

if (i < j) // add to edge array

Edge[++k] = edge<int> (i, j, w);

}

// put edges in min heap

minHeap<edge<T> > heap(1);

heap.initialize(Edge, e);

fastUnionFind uf(n); // union/find structure

// extract edges in cost order and select/reject

k = 0; // use as cursor for t now

while (e > 0 && k < n - 1)

{// spanning tree not complete & edges remain

edge<T> x = heap.top();

heap.pop();

e--;

int a = uf.find(x.vertex1());

int b = uf.find(x.vertex2());

if (a != b)

{// select edge x

spanningTreeEdges[k++] = x;

uf.unite(a,b);

}

}

return (k == n - 1);

}