修改代码清单 13-3 中的 Prim 函数，使之能指定起始结点。

【解】代码清单 13-1 中的 prim 函数总是将 0 号结点作为起始结点。为了能指定起始结点，我们在邻接表类中又增加了一个 prim 函数。该函数有两个参数。其中第一个参数是指定的起始结点，第二个参数是代表无穷大的值。这个函数在正式寻找最小生成树前，先查找指定结点的序号，将这个结点作为起始结点。后面的过程与 prim 函数完全相同。具体见代码清单 13-4。

代码清单 13-4 可以指定起始结点的 prim 函数

```
1.  template <class TypeOfVer, class TypeOfEdge>
2.  void adjListGraph<TypeOfVer, TypeOfEdge>::prim
3.          (TypeOfVer startVer, TypeOfEdge noEdge) const
4.  { bool *flag = new bool[Vers];
5.    TypeOfEdge *lowCost = new TypeOfEdge[Vers];
6.    int *startNode = new int[Vers];
7.
8.    edgeNode *p;
9.    TypeOfEdge min;
10.   int start, i, j;                  // start 为起始结点的序号
11.
12.   // 寻找起始结点的序号
13.   for (start = 0; start < Vers; ++start)
14.         if (verList[start].ver == startVer) break;
15.   if (start == Vers){
16.         cout << "起始结点不存在" << endl;
17.         return;
18.   }
19.
20.   for (i = 0; i < Vers; ++i) {        // 初始化
21.         flag[i] = false;
22.         lowCost[i] = noEdge;
23.   }
24.
25.   for ( i= 1; i < Vers; ++i) {        // 将 Vers-1 个结点一一添加到生成树
26.         for (p = verList[start].head; p != NULL; p = p->next)  // 更新工作数组
27.               if (!flag[p->end] && lowCost[p->end] > p->weight) {
28.                     lowCost[p->end] = p->weight;
29.               startNode[p->end] = start;
30.               }
31.         flag[start] = true;
32.         min = noEdge;
33.         for (j = 0; j < Vers; ++j)          //寻找连接两个集合的权值最小的边
34.               if (lowCost[j] < min) {min = lowCost[j]; start = j;}
35.         cout << '(' << verList[startNode[start]].ver << ','
```

```
36.             << verList[start].ver << ")\t";
37.             lowCost[start] = noEdge;
38.     }
39.     delete [] flag;
40.     delete [] startNode;
41.     delete [] lowCost;
42.  }
```