在代码清单 14-3 实现的 Dijkstra 算法中，寻找具有最短路径的结点是采用顺序搜索。因此，每次获得一个具有最短路径的结点的时间复杂度是 O（|V|）。如果能将所有结点的距离放在一个优先级队列中，则获得具有最短路径的结点所需要的时间是 O（log|V|）。使用优先及队列类实现 Dijkstra 算法，并分析它的时间复杂度。（提示：每当结点距离更新时，将此结点入队。也就是说，某个结点在优先及队列中可能出现多次，但每次出现的距离是不同的，后入队的元素有更短的距离。每次执行处对操作后，要检查出队的元素是否在 S 集合中。如果已在 S 集合中，则放弃此结点的处理。

【解】修改后的 Dijkstra 函数见代码清单 14-7。函数定义了一个优先级队列（第 9 行）保存源点到各结点的已知路径长度。队列的元素包含两个部分：源点到某结点的距离及该结点的序号。为此，在邻接表类中定义了一个私有内嵌类 queueNode，它的定义见代码清单 14-8。首先将源点入队（第 28 行），然后不断地从队列中取出一个结点的距离信息，处理该结点，直到队列为空。此时所有结点都找到了最短路径。

如果取出的结点 min 已经找到最短路径，则不作任何处理。否则将结点 min 标记为已找到最短路径。然后检查它的每一个尚未找到最短路径的后继。如果经过 min 到这个后继是一条更好的路径，则修改后继的路径，将后继结点的信息入队。

**代码清单 14-7 采用优先级队列的 Dijkstra 算法**

```
1.  template <class TypeOfVer, class TypeOfEdge>
2.  void adjListGraph<TypeOfVer, TypeOfEdge>::dijkstra(TypeOfVer start,
    TypeOfEdge noEdge) const
3.  { TypeOfEdge *distance = new TypeOfEdge[Vers];
4.    int *prev = new int [Vers];
5.    bool *known = new bool[Vers];
6.    int sNo, i;
7.    edgeNode *p;
8.    priorityQueue<queueNode> q;
9.    queueNode min, succ;
10.
11.   for (i = 0; i< Vers; ++i) {                    // 初始化
12.         known[i] = false;
13.      distance[i] = noEdge;
14.   }
15.
16.   for (sNo = 0; sNo < Vers; ++sNo)               // 寻找源点的序号
17.         if (verList[sNo].ver == start) break;
18.   if (sNo == Vers){
19.         cout << "起始结点不存在" << endl;
20.         return;
21.   }
22.
23.   distance[sNo] = 0;
24.   prev[sNo] = sNo;
```

```
25.   min.dist = 0;
26.   min.node = sNo;
27.   q.enQueue(min);
28.
29.   while (!q.isEmpty()) {                          // 寻找最短路径
30.         min = q.deQueue();
31.         if (known[min.node]) continue;
32.       known[min.node] = true;
33.          for (p = verList[min.node].head; p != NULL; p = p->next)
34.                if (!known[p->end] && distance[p->end] > min.dist + p->weight) {
35.               succ.dist = distance[p->end] = min.dist + p->weight;
36.               prev[p->end] = min.node;
37.                           succ.node = p->end;
38.                           q.enQueue(succ);
39.          }
40.    }
41.
42.   for (i = 0; i < Vers; ++i) {                    //  输出路径
43.         cout << "从" << start << "到" << verList[i].ver << "的路径为:" << endl;
44.         printPath(sNo, i, prev);
45.         cout << "\t 长度为：  " << distance[i] << endl;
46.   }
47.  }
```

代码清单 14-8 是私有内嵌类 queueNode 的定义。队列中的每个元素保存两个信息：结点的序号 node 和源点到 node 的已知路径长度 dist。queueNode 的对象被作为优先级队列中的元素，必须能被比较，因此类中有一个小于运算符重载函数。

**代码清单 14-8 queueNode 的定义**

```
48.  struct queueNode{
49.          TypeOfEdge dist;
50.          int node;
51.
52.          bool operator<(const queueNode &rp) const {return dist < rp.dist;}
53.  };
```