在邻接表类中实现带负权值的单源最短路径函数。

【解】带负权值的单源最短路径可以采用穷举方法，从源点开始检查结点。对每个被检查的结点 x，检查它的所有后继。如果从源点经过 x 到达 x 的某个后继结点的路径比原来的路径好，则再检查 x 的这个后继结点。因为它的后继结点也可能会有一条更好的路径。重复这个过程，直到所有结点都找不到更好的路径为止。

这个算法实现时必须保存需要检查它的后继结点的结点，这可以利用一个队列。开始时，将源点 start 放入队列，然后重复以下过程，直到队列为空：出队一个结点 v，对 v 的所有邻接点 w，如果 start 经过 v 到 w 的距离比已知的 start 到 w 的距离短，则更新 w 的距离，并将 w 入队。算法的完整实现见代码清单 14-3。

代码清单 14-3 带负权值的最短路径函数

```
1.  template <class TypeOfVer, class TypeOfEdge>
2.  void adjListGraph<TypeOfVer, TypeOfEdge>::minusShortDistance
3.      (TypeOfVer start, TypeOfEdge noEdge) const
4.  { TypeOfEdge *distance = new TypeOfEdge[Vers];  // 保存源点到每个结点的距离
5.    int *prev = new int [Vers];                   // 保存源点到每个结点的路径
6.    int u, sNo, i;
7.    edgeNode *p;
8.    linkQueue<int> q;                           // 保存待检查结点
9.
10.   for (sNo = 0; sNo < Vers; ++sNo)             // 检察源点是否存在
11.       if (verList[sNo].ver == start) break;
12.   if (sNo == Vers){
13.       cout << "起始结点不存在" << endl;
14.       return;
15.   }
16.
17.   // 初始化
18.   for (i = 0; i < Vers; ++i) distance[i] = noEdge;
19.   distance[sNo] = 0;
20.   prev[sNo] = sNo;
21.   q.enQueue(sNo);
22.
23.   while (!q.isEmpty()) {                      // 检查每一个可能的结点
24.       u = q.deQueue();
25.       for (p = verList[u].head; p != NULL; p = p->next)  // 检查 u 的每个后继
26.           if (distance[p->end] > distance[u] + p->weight) {//找到更好的路径
27.           distance[p->end] = distance[u] + p->weight;
28.           prev[p->end] = u;
29.                   q.enQueue(p->end);
30.       }
```

```
31.     }
32.
33.    for (i = 0; i < Vers; ++i) {                        // 输出路径
34.            cout << "从" << start << "到" << verList[i].ver << "的路径为:" << endl;
35.            printPath(sNo, i, prev);
36.            cout << "\t 长度为 :  " << distance[i] << endl;
37.    }
38.   }
```