8.17

	1	2	3	4	5	6
Ve	0	19	15	29	38	43
Vl	0	19	15	37	38	43

	<1,2>	<1,3>	<3,2>	<2,4>	<2,5>	<3,5>	<4,6>	<5,6>
Ae	0	0	15	19	19	15	29	38
Al	17	0	15	27	19	27	37	38
Al-Ae	17	0	0	8	0	12	8	0

这个工程最早结束时间为43。关键活动为<1,3><3,2><2,5><5,6>，关键活动加速可使整个工程提前完成。

8.23

```
struct MSTEdgeNode {
    int head, tail;
    int weight;
    MSTEdgeNode() :head(-1), tail(-1), cost(0) { }
};
void Dijkstra(Graph<int, int>& G, MinSpanTree& Tree) {
    int n = G.NumberOfVertices();
    UFSets<int> D;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            int w = G.getWeight(i, j);
            if (w < MaxValue) {
                if (D.Find(i) == D.Find(j)) {
                    D.Union(i, j);
                    MSTEdgeNode temp;
                    temp.head = i; temp.tail = j; temp.weight = w;
                    Tree.InsertEdge(temp);
                }
            }
            else {
                int k = D.CommonAncestors(i, j);
                int p, q, max1, max2;
                int s1, s2, t1, t2;
                max1 = -MaxValue;
                p = i; q = D.Father(p);
                while (q <= k) {
                    if (G.getWeight(p, q) > max1) {
                        max1 = G.getWeight(p, q);
                        s1 = p; s2 = q;
                    }
                    p = q; q = D.father(p);
```

```
                }
                max2 = -MaxValue;
                p = j; q = D.Father(p);
                while (q <= k) {
                    if (G.getWeight(p, q) > max2) {
                        max2 = G.getWeight(p, q);
                        t1 = p; t2 = q;
                    }
                    p = q; q = D.father(p);
                }
                if (max1 <= max2) {
                    s1 = t1; s2 = t2; max1 = max2;
                }
                D.Remove(s1);
                D.Union(s1, s2);
                MSTEdgeNode temp;
                temp.head = s1; temp.tail = s2; temp.weight = max1;
                Tree.InsertEdge(temp);
            }
        }
    }
}
```

8.25

```
void Dijkstra(Graph<int, int>& G, int dist[], int pre[]) {
    SortedChain<int, int> T;
    for (int i = 1; i < G.NumberOfVertices(); i++) {
        dist[i] = MaxValue; pre[i] = 0;
        T.Insert(i);
    }
    EdgeNode* p = G.NodeTable[0].adj;
    while (p != NULL) {
        dist[p->vertex] = p->length;
        p = p->link;
    }
    while (1) {
        ChainNode<int, int> q = T.Begin();
        if (q->link == NULL) break;
        int min = MaxValue, u = 0;
        while (q != NULL) {
            if (dist[q->data] < min) {
                min = dist[q->data];
                u = q->data;
            }
            q = q->link;
        }
        p = G.NodeTable[u].adj;
        while (p != NULL) {
            int k = p->vertex;
            if (T.Search(k) != NULL && dist[u] + p->length < dist[k]) {
                dist[k] = dist[u] + p->length;
                pre[k] = u;
            }
            p = p->link;
```

```
        }
        T.Remove(u);
    }
}
```

8.29

```
void topoSort_dfs(GraphInk& G, int visited[], int indegree[], int v, int& count)
{
    visited[v] = ++count;
    cout << G.NodeTable[v].data << ",";
    EdgeNode* p = G.NodeTable[v].adj;
    while (p != NULL) {
        int w = p->vertex;
        indegree[w]--;
        if (!visited[w] && !indegree[w]) topoSort_dfs(G, visited, indegree, w,
count);
        p = p->link;
    }
}
```