$$\max_{w \text{ in } V} \{ \text{minimum length of a path from } w \text{ to } v \}$$

The *center* of G is a vertex of minimum eccentricity. Thus, the center of a digraph is a vertex that is closest to the vertex most distant from it.

Example 6.11. Consider the weighted digraph shown in Fig. 6.22.

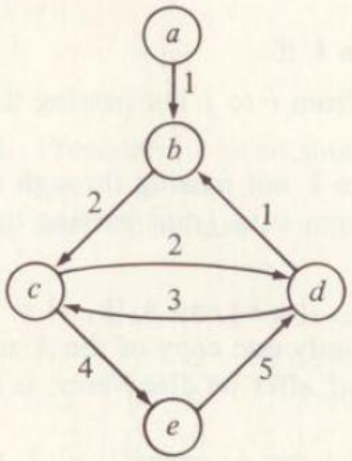

Fig. 6.22. Weighted digraph.

The eccentricities of the vertices are

vertex	eccentricity
a	∞
b	6
c	8
d	5
e	7

Thus the center is vertex d. □

Finding the center of a digraph G is easy. Suppose C is the cost matrix for G.

1. First apply the procedure *Floyd* of Fig. 6.16 to C to compute the all-pairs shortest paths matrix A.
2. Find the maximum cost in each column i. This gives us the eccentricity of vertex i.
3. Find a vertex with minimum eccentricity. This is the center of G.

This running time of this process is dominated by the first step, which takes

$O(n^3)$ time. Step (2) takes $O(n^2)$ time and step (3) $O(n)$ time.

Example 6.12. The APSP cost matrix for Fig. 6.22 is shown in Fig. 6.23. The maximum value in each column is shown below.

	a	b	c	d	e
a	0	1	3	5	7
b	∞	0	2	4	6
c	∞	3	0	2	4
d	∞	1	3	0	7
e	∞	6	8	5	0
max	∞	6	8	5	7

Fig. 6.23. APSP cost matrix.

6.5 Traversals of Directed Graphs

To solve many problems dealing with directed graphs efficiently we need to visit the vertices and arcs of a directed graph in a systematic fashion. Depth-first search, a generalization of the preorder traversal of a tree, is one important technique for doing so. Depth-first search can serve as a skeleton around which many other efficient graph algorithms can be built. The last two sections of this chapter contain several algorithms that use depth-first search as a foundation.

Suppose we have a directed graph G in which all vertices are initially marked *unvisited*. Depth-first search works by selecting one vertex v of G as a start vertex; v is marked *visited*. Then each unvisited vertex adjacent to v is searched in turn, using depth-first search recursively. Once all vertices that can be reached from v have been visited, the search of v is complete. If some vertices remain unvisited, we select an unvisited vertex as a new start vertex. We repeat this process until all vertices of G have been visited.

This technique is called depth-first search because it continues searching in the forward (deeper) direction as long as possible. For example, suppose x is the most recently visited vertex. Depth-first search selects some unexplored arc $x \rightarrow y$ emanating from x. If y has been visited, the procedure looks for another unexplored arc emanating from x. If y has not been visited, then the procedure marks y visited and initiates a new search at y. After completing the search through all paths beginning at y, the search returns to x, the vertex from which y was first visited. The process of selecting unexplored arcs emanating from x is then continued until all arcs from x have been explored.

An adjacency list $L[v]$ can be used to represent the vertices adjacent to vertex v, and an array *mark*, whose elements are chosen from (*visited*, *unvisited*), can be used to determine whether a vertex has been previously visited. The recursive procedure *dfs* is outlined in Fig. 6.24. To use it