

---

## Tree-Search Algorithms

### Contents

---

<b>6.1 Breadth-First and Depth-First Search . . . . .</b>	<b>135</b>
BREADTH-FIRST SEARCH AND SHORTEST PATHS . . . . .	137
DEPTH-FIRST SEARCH . . . . .	139
FINDING THE CUT VERTICES AND BLOCKS OF A GRAPH . . . . .	142
<b>6.2 Minimum-Weight Spanning Trees . . . . .</b>	<b>145</b>
THE JARNÍK-PRIM ALGORITHM . . . . .	146
<b>6.3 Branching-Search . . . . .</b>	<b>149</b>
FINDING SHORTEST PATHS IN WEIGHTED DIGRAPHS . . . . .	149
DIRECTED DEPTH-FIRST SEARCH . . . . .	151
FINDING THE STRONG COMPONENTS OF A DIGRAPH . . . . .	152
<b>6.4 Related Reading . . . . .</b>	<b>156</b>
DATA STRUCTURES . . . . .	156

---

### 6.1 Breadth-First and Depth-First Search

We have seen that connectedness is a basic property of graphs. But how does one determine whether a graph is connected? In the case of small graphs, it is a routine matter to do so by inspection, searching for paths between all pairs of vertices. However, in large graphs, such an approach could be time-consuming because the number of paths to examine might be prohibitive. It is therefore desirable to have a systematic procedure, or *algorithm*, which is both efficient and applicable to all graphs. The following property of the trees of a graph provides the basis for such a procedure. For a subgraph  $F$  of a graph  $G$ , we simply write  $\partial(F)$  for  $\partial(V(F))$ , and refer to this set as the *edge cut* associated with  $F$ .

Let  $T$  be a tree in a graph  $G$ . If  $V(T) = V(G)$ , then  $T$  is a spanning tree of  $G$  and we may conclude, by Theorem 4.6, that  $G$  is connected. But if  $V(T) \subset V(G)$ , two possibilities arise: either  $\partial(T) = \emptyset$ , in which case  $G$  is disconnected, or  $\partial(T) \neq \emptyset$ . In the latter case, for any edge  $xy \in \partial(T)$ , where  $x \in V(T)$  and  $y \in V(G) \setminus V(T)$ ,