

DISCRETE MATHEMATICS

W W L CHEN

© W W L Chen, 1992, 2003.

This chapter is available free to all individuals, on the understanding that it is not to be used for financial gains, and may be downloaded and/or photocopied, with or without permission from the author.

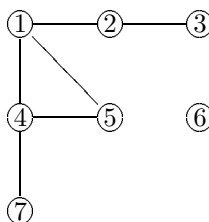
However, this document may not be kept on any information storage and retrieval system without permission from the author, unless such system is not accessible to any individuals other than its owners.

Chapter 19

SEARCH ALGORITHMS

19.1. Depth-First Search

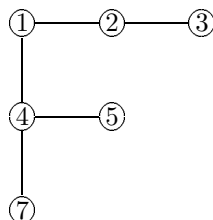
EXAMPLE 19.1.1. Consider the graph represented by the following picture.



Suppose that we wish to find out all the vertices v such that there is a path from vertex 1 to vertex v . We may proceed as follows:

- (1) We start from vertex 1, move on to an adjacent vertex 2 and immediately move on to the adjacent vertex 3. At this point, we conclude that there is a path from vertex 1 to vertex 2 or 3.
- (2) Since there is no new adjacent vertex from vertex 3, we back-track to vertex 2. Since there is no new adjacent vertex from vertex 2, we back-track to vertex 1.
- (3) We start from vertex 1 again, move on to an adjacent vertex 4 and immediately move on to an adjacent vertex 7. At this point, we conclude that there is a path from vertex 1 to vertex 2, 3, 4 or 7.
- (4) Since there is no new adjacent vertex from vertex 7, we back-track to vertex 4.
- (5) We start from vertex 4 again and move on to the adjacent vertex 5. At this point, we conclude that there is a path from vertex 1 to vertex 2, 3, 4, 5 or 7.
- (6) Since there is no new adjacent vertex from vertex 5, we back-track to vertex 4. Since there is no new adjacent vertex from vertex 4, we back-track to vertex 1. Since there is no new adjacent vertex from vertex 1, the process ends. We conclude that there is a path from vertex 1 to vertex 2, 3, 4, 5 or 7, but no path from vertex 1 to vertex 6.

Note that a by-product of this process is a spanning tree of the component of the graph containing the vertices 1, 2, 3, 4, 5, 7, given by the picture below.

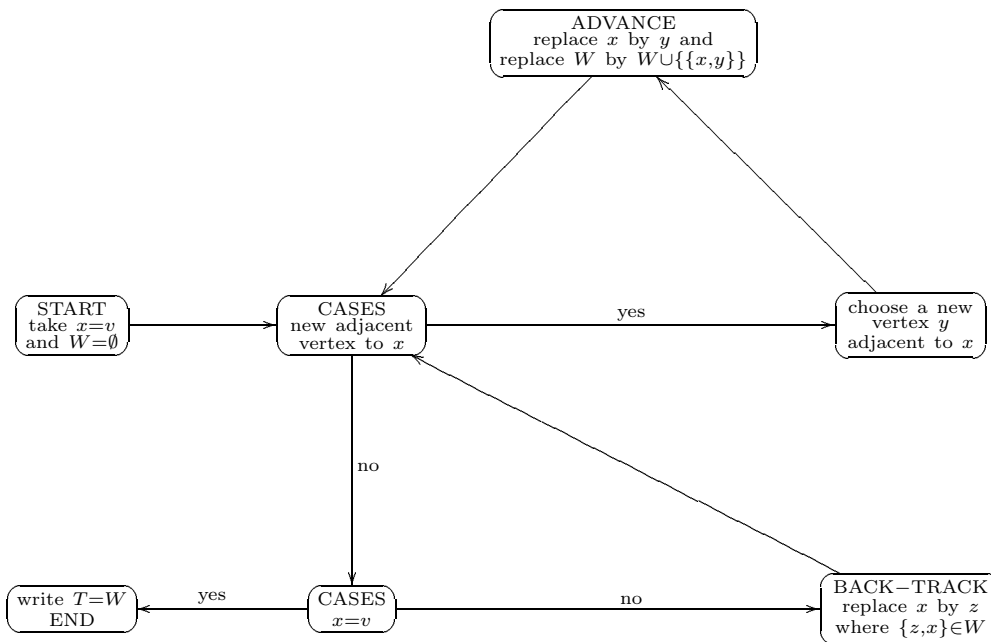


The above is an example of a strategy known as depth-first search, which can be regarded as a special tree-growing method. Applied from a vertex v of a graph G , it will give a spanning tree of the component of G which contains the vertex v .

DEPTH-FIRST SEARCH ALGORITHM. Consider a graph $G = (V, E)$.

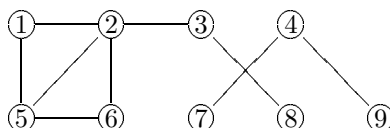
- (1) Start from a chosen vertex $v \in V$.
- (2) Advance to a new adjacent new vertex, and further on to a new adjacent vertex, and so on, until no new vertex is adjacent. Then back-track to a vertex with a new adjacent vertex.
- (3) Repeat (2) until we are back to the vertex v with no new adjacent vertex.

REMARK. The algorithm can be summarized by the following flowchart:



PROPOSITION 19A. Suppose that v is a vertex of a graph $G = (V, E)$, and that $T \subseteq E$ is obtained by the flowchart of the Depth-first search algorithm. Then T is a spanning tree of the component of G which contains the vertex v .

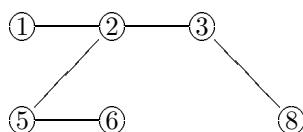
EXAMPLE 19.1.2. Consider the graph described by the picture below.



We shall use the Depth-first search algorithm to determine the number of components of the graph. We shall start with vertex 1, and use the convention that when we have a choice of vertices, then we take the one with lower numbering. Then we have the following:

$$\begin{array}{cccccccc} 1 & \xrightarrow{\text{advance}} & 2 & \xrightarrow{\text{advance}} & 3 & \xrightarrow{\text{advance}} & 8 & \xrightarrow{\text{back-track}} & 3 & \xrightarrow{\text{back-track}} & 2 \\ & \xrightarrow{\text{advance}} & 5 & \xrightarrow{\text{advance}} & 6 & \xrightarrow{\text{back-track}} & 5 & \xrightarrow{\text{back-track}} & 2 & \xrightarrow{\text{back-track}} & 1 \end{array}$$

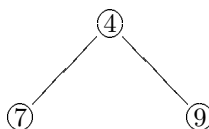
This gives the component of the graph with vertices 1, 2, 3, 5, 6, 8 and the following spanning tree.



Note that the vertex 4 has not been reached. So let us start again with vertex 4. Then we have the following:

$$4 \xrightarrow{\text{advance}} 7 \xrightarrow{\text{back-track}} 4 \xrightarrow{\text{advance}} 9 \xrightarrow{\text{back-track}} 4$$

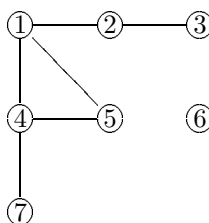
This gives a second component of the graph with vertices 4, 7, 9 and the following spanning tree.



There are no more vertices outstanding. We conclude that the original graph has two components.

19.2. Breadth-First Search

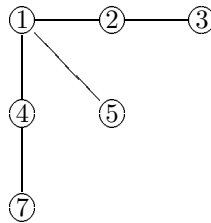
EXAMPLE 19.2.1. Consider again the graph represented by the following picture.



Suppose again that we wish to find out all the vertices v such that there is a path from vertex 1 to vertex v . We may proceed as follows:

- (1) We start from vertex 1, and work out all the new vertices adjacent to it, namely vertices 2, 4 and 5.
- (2) We next start from vertex 2, and work out all the new vertices adjacent to it, namely vertex 3.
- (3) We next start from vertex 4, and work out all the new vertices adjacent to it, namely vertex 7.
- (4) We next start from vertex 5, and work out all the new vertices adjacent to it. There is no such vertex.
- (5) We next start from vertex 3, and work out all the new vertices adjacent to it. There is no such vertex.
- (6) We next start from vertex 7, and work out all the new vertices adjacent to it. There is no such vertex.
- (7) Since vertex 7 is the last vertex on our list 1, 2, 4, 5, 3, 7 (in the order of being reached), the process ends.

Note that a by-product of this process is a spanning tree of the component of the graph containing the vertices 1, 2, 3, 4, 5, 7, given by the picture below.

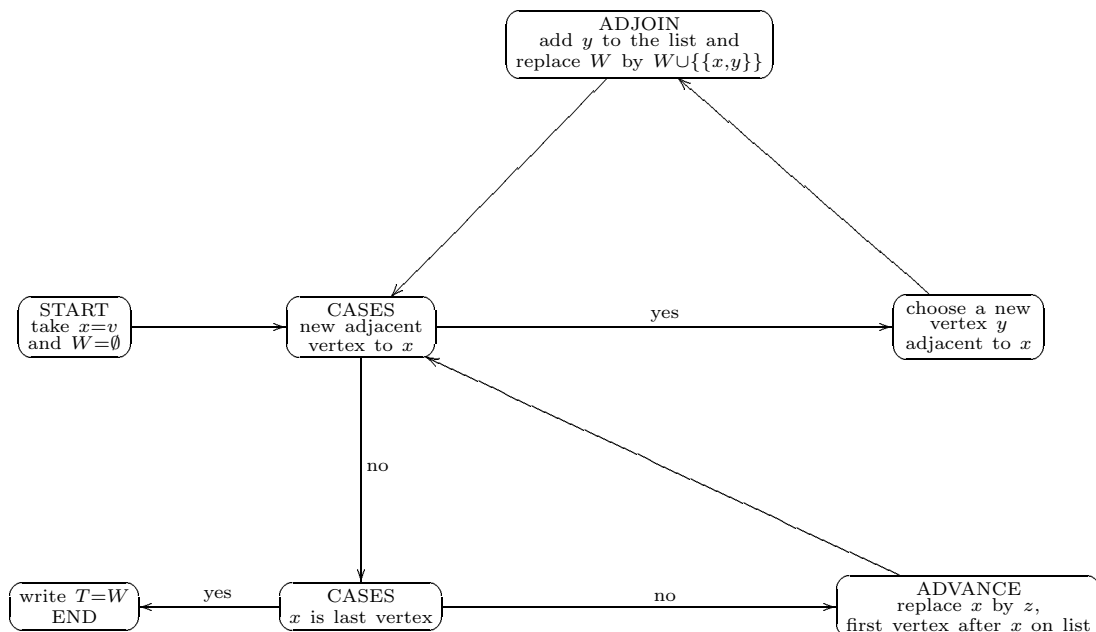


The above is an example of a strategy known as breadth-first search, which can be regarded as a special tree-growing method. Applied from a vertex v of a graph G , it will also give a spanning tree of the component of G which contains the vertex v .

BREADTH-FIRST SEARCH ALGORITHM. Consider a graph $G = (V, E)$.

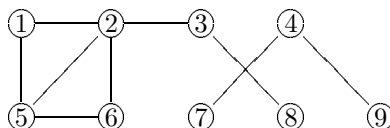
- (1) Start a list by choosing a vertex $v \in V$.
- (2) Write down all the new vertices adjacent to v , and add these to the list.
- (3) Consider the next vertex after v on the list. Write down all the new vertices adjacent to this vertex, and add these to the list.
- (4) Consider the second vertex after v on the list. Write down all the new vertices adjacent to this vertex, and add these to the list.
- (5) Repeat the process with all the vertices on the list until we reach the last vertex with no new adjacent vertex.

REMARK. The algorithm can be summarized by the following flowchart:



PROPOSITION 19B. Suppose that v is a vertex of a graph $G = (V, E)$, and that $T \subseteq E$ is obtained by the flowchart of the Breadth-first search algorithm. Then T is a spanning tree of the component of G which contains the vertex v .

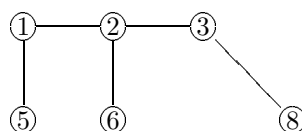
EXAMPLE 19.2.2. Consider again the graph described by the picture below.



We shall use the Breadth-first search algorithm to determine the number of components of the graph. We shall again start with vertex 1, and use the convention that when we have a choice of vertices, then we take the one with lower numbering. Then we have the list

1, 2, 5, 3, 6, 8.

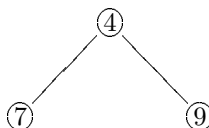
This gives a component of the graph with vertices 1, 2, 3, 5, 6, 8 and the following spanning tree.



Again the vertex 4 has not been reached. So let us start again with vertex 4. Then we have the list

4, 7, 9.

This gives a second component of the graph with vertices 4, 7, 9 and the following spanning tree.



There are no more vertices outstanding. We therefore draw the same conclusion as in the last section concerning the number of components of the graph. Note that the spanning trees are not identical, although we have used the same convention concerning choosing first the vertices with lower numbering in both cases.

19.3. The Shortest Path Problem

Consider a connected graph $G = (V, E)$, with weight function $w : E \rightarrow \mathbb{N}$. For any pair of distinct vertices $x, y \in V$ and for any path

$$v_0(= x), v_1, \dots, v_r(= y)$$

from x to y , we consider the value of

$$(1) \quad \sum_{i=1}^r w(\{v_{i-1}, v_i\}),$$

the sum of the weights of the edges forming the path. We are interested in minimizing the value of (1) over all paths from x to y .

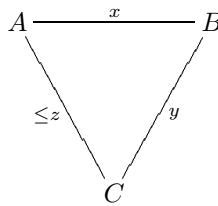
REMARK. If we think of the weight of an edge as a length instead, then we are trying to find a shortest path from x to y .

The algorithm we use is a variation of the Breadth-first search algorithm. To understand the central idea of this algorithm, let us consider the following analogy.

EXAMPLE 19.3.1. Consider three cities A, B, C . Suppose that the following information concerning travelling time between these cities is available:

$$\begin{array}{ccc} AB & BC & AC \\ \hline x & y & \leq z \end{array}$$

This can be represented by the following picture.



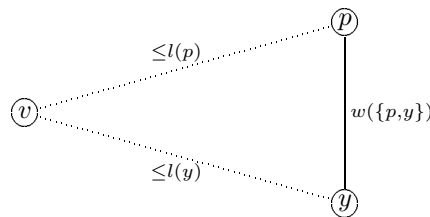
Clearly the travelling time between A and C cannot exceed $\min\{z, x + y\}$.

Suppose now that v is a vertex of a weighted connected graph $G = (V, E)$. For every vertex $x \in V$, suppose it is known that the shortest path from v to x does not exceed $l(x)$.

Let y be a vertex of the graph, and let p be a vertex adjacent to y . Then clearly the shortest path from v to y does not exceed

$$\min\{l(y), l(p) + w(\{p, y\})\}.$$

This is illustrated by the picture below.



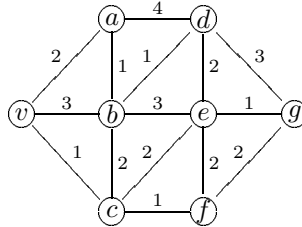
We can therefore replace the information $l(y)$ by this minimum.

DIJKSTRA'S SHORTEST PATH ALGORITHM. Consider a connected graph $G = (V, E)$ and a weight function $w : E \rightarrow \mathbb{N}$.

- (1) Let $v \in V$ be a vertex of the graph G .
- (2) Let $l(v) = 0$, and write $l(x) = \infty$ for every vertex $x \in V$ such that $x \neq v$.
- (3) Start a partial spanning tree by taking the vertex v .
- (4) Consider all vertices $y \in V$ not in the partial spanning tree and which are adjacent to the vertex v . Replace the value of $l(y)$ by $\min\{l(y), l(v) + w(\{v, y\})\}$. Look at the new values of $l(y)$ for every vertex $y \in V$ not in the partial spanning tree, and choose a vertex v_1 for which $l(v_1) \leq l(y)$ for all such vertices y . Add the edge $\{v, v_1\}$ to the partial spanning tree.
- (5) Consider all vertices $y \in V$ not in the partial spanning tree and which are adjacent to the vertex v_1 . Replace the value of $l(y)$ by $\min\{l(y), l(v_1) + w(\{v_1, y\})\}$. Look at the new values of $l(y)$ for every vertex $y \in V$ not in the partial spanning tree, and choose a vertex v_2 for which $l(v_2) \leq l(y)$ for all such vertices y . Add the edge giving rise to the new value of $l(v_2)$ to the partial spanning tree.
- (6) Repeat the argument on v_2, v_3, \dots until we obtain a spanning tree of the graph G . The unique path from v to any vertex $x \neq v$ represents the shortest path from v to x .

REMARK. In (2) above, we take $l(x) = \infty$ when $x \neq v$. This is not absolutely necessary. In fact, we can start with any sufficiently large value for $l(x)$. For example, the total weight of all the edges will do.

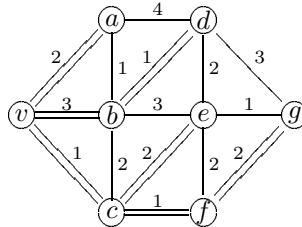
EXAMPLE 19.3.2. Consider the weighted graph described by the following picture.



We then have the following situation.

	$l(v)$	$l(a)$	$l(b)$	$l(c)$	$l(d)$	$l(e)$	$l(f)$	$l(g)$		new edge
	(0)	∞	∞	∞	∞	∞	∞	∞		
v		2	3	(1)	∞	∞	∞	∞	$v_1 = c$	$\{v, c\}$
$v_1 = c$		(2)	3		∞	3	2	∞	$v_2 = a$	$\{v, a\}$
$v_2 = a$			3		6	3	(2)	∞	$v_3 = f$	$\{c, f\}$
$v_3 = f$			(3)		6	3		4	$v_4 = b$	$\{v, b\}$
$v_4 = b$					4	(3)		4	$v_5 = e$	$\{c, e\}$
$v_5 = e$					(4)			4	$v_6 = d$	$\{b, d\}$
$v_6 = d$								(4)	$v_7 = g$	$\{f, g\}$

The bracketed values represent the length of the shortest path from v to the vertices concerned. We also have the following spanning tree.



Note that this is not a minimal spanning tree.

PROBLEMS FOR CHAPTER 19

1. Rework Question 6 of Chapter 17 using the Depth-first search algorithm.
2. Consider the graph G defined by the following adjacency table.

1	2	3	4	5	6	7	8
2	1	2	1	2	7	3	1
4	3	4	2			6	7
8	4	7	3				8
				5			

Apply the Depth-first search algorithm, starting with vertex 7 and using the convention that when we have a choice of vertices, then we take the one with lower numbering. How many components does G have?

3. Rework Question 6 of Chapter 17 using the Breadth-first search algorithm.
4. Consider the graph G defined by the following adjacency table.

1	2	3	4	5	6	7	8	9
5	4	5	2	1	3	2	2	1
9	7	6	7	3	5	4	4	3
	8	9	8	6	9			6

Apply the Breadth-first search algorithm, starting with vertex 1 and using the convention that when we have a choice of vertices, then we take the one with lower numbering. How many components does G have?

5. Find the shortest path from vertex a to vertex z in the following weighted graph.

