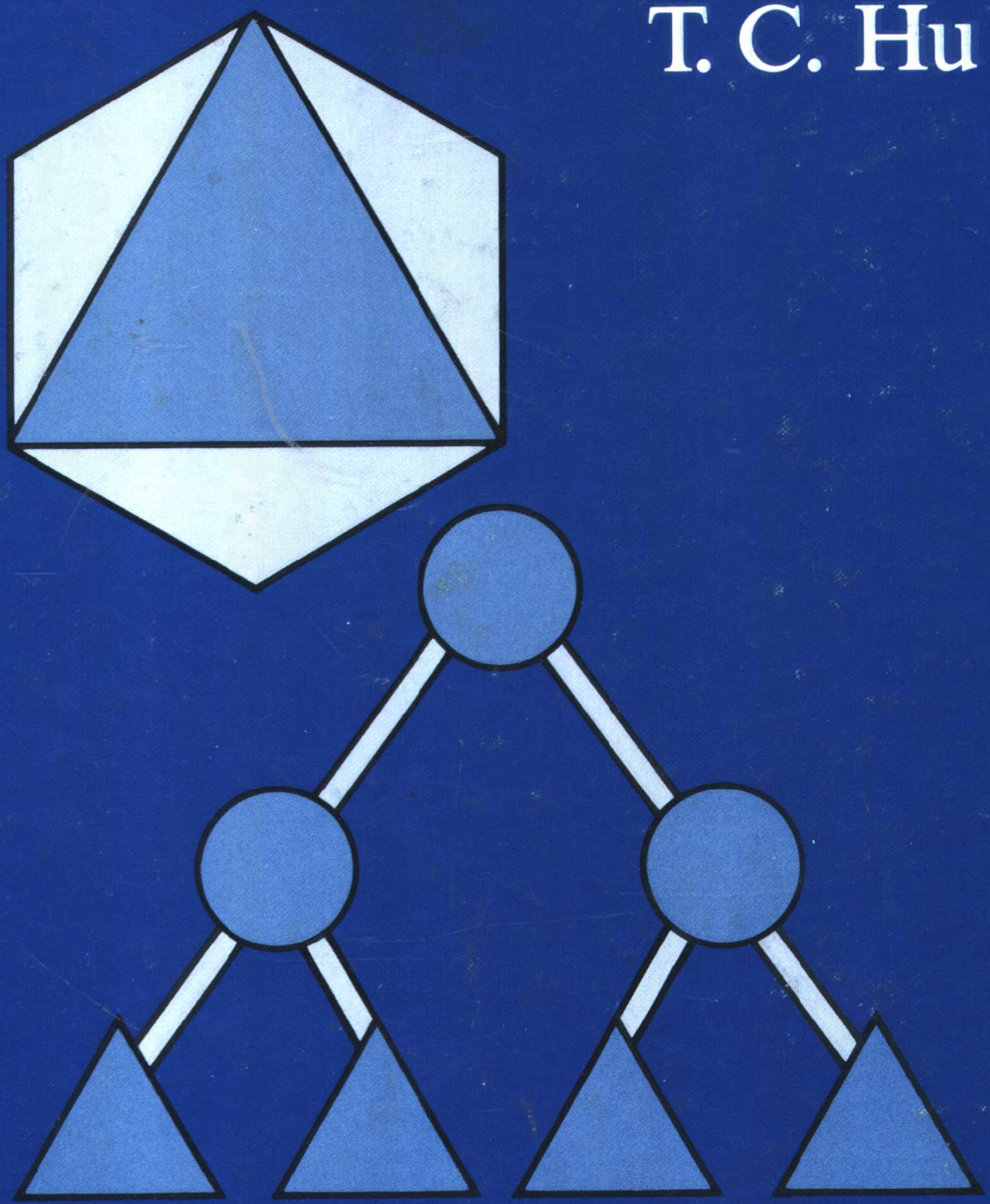
Combinatorial Algorithms

T. C. Hu



Combinatorial Algorithms

T. C. Hu
University of California, San Diego

ADDISON-WESLEY PUBLISHING COMPANY
Reading, Massachusetts • Menlo Park, California
London • Amsterdam • Don Mills, Ontario • Sydney

Reproduced by Addison-Wesley from camera-ready copy supplied by the author.

ISBN 0-201-03859-5
ABCDEFGHIJ-AL-8987654321

PREFACE

This book presents some combinatorial algorithms common in computer science and operations research. The presentation is to stress intuitive ideas in an algorithm and to illustrate it with a numerical example. The detailed implementation of the algorithms in PASCAL are in a separate manual. No background in linear programming and advanced data structure is needed. Most of the material can be taught to undergraduates while more difficult sections are only suitable to graduate students. Chapters can be read somewhat independently so that the instructor can select a subset of chapters for his course. This book should also be useful as a reference book since it contains much material not available in Journals or any other books.

Chapters One and Two can be used in a one quarter course in network theory or graph algorithms. Chapter One goes in-depth into several shortest-path problems and introduces a decomposition algorithm for large sparse networks. Chapter Two deals with network flows, and contains a large amount of new material, such as the algorithms of Dinic and Kazanov which have never appeared in English before, the optimum communication spanning tree and the description of PERT in terms of longest paths and cheapest cuts. Also in Chapter Two is a section on multi-terminal flows where a subset of nodes are terminal nodes.

Chapters Three and Four cover dynamic programming and backtrack (branch and bound) which are two general optimization techniques. Both topics are usually not covered in detail in computer science departments. Chapter Three introduces the concept of dynamic programming by using examples carefully selected to show the variety of problems solvable by dynamic programming. After the knapsack problem is solved, the periodic nature of the solutions is

discussed. (The solution to the two-dimensional knapsack problem is based on the papers of Gilmore and Gomory.) This chapter ends with a brief discussion of the work of Dr. F.F. Yao. Chapter Four includes standard material on backtracking as well as a detailed description of $\alpha-\beta$ pruning in a game tree. It also gives an example of the Monte Carlo technique of estimating the size of the decision tree.

Chapters Five and Six contain a large amount of new material which should be of interest to computer scientists and operations researchers. Chapter Five introduces the Huffman algorithm, the Hu-Tucker algorithm, including a new reconstruction phase, and the generalization of both algorithms to regular cost functions. This generalization is based on the paper by Hu, Kleitman and Tamaki. Chapter Five also describes and illustrates the Garsia-Wachs construction. Chapter Six deals with heuristic algorithms. It contains the one-point theorem of Magazine, Nemhauser and Trotter and the new bin-packing algorithm of Yao. The treatment of job-scheduling for the tree-constraint is a revision of the author's paper published in 1961.

The subject of Chapter Seven is matrix multiplications. This chapter contains two combinatorial results, the Strassen's result on the multiplication of two large matrices and the results on the optimum order of multiplying a chain of matrices of different dimensions. Although the problem of optimum order can be solved by an $O(n^3)$ algorithm based on dynamic programming, the problem can now be solved by an O(nlogn) algorithm based on combinatorial insights. Since the subject of finding the optimum order is a book by itself, we give the main theorems on the subject and a heuristic O(n) algorithm which has a 15% error bound.

The final chapter, Chapter Eight, introduces the concepts of NP-complete problems. The purpose here is to give the reader some intuitive notions but not a complete treatment since a book has been published dealing with this subject in detail.

It is a pleasure to thank all persons who helped to make this book possible. To the National Science Foundation and Dr. J. Chandra and Dr. P. Boggs of the U. S. Army Research Office for their financial help. To Drs. F. Chin, S. Dreyfus,

F. Ruskey, W. Savitch, A. Tucker, M. Wachs, F. Yao for reading various parts of the drafts. To Professor L. E. Trotter, Jr. and Professor Andrew Yao for reading the next-to-final version of the whole book and made many valuable suggestions. To Mrs. Mary Deo for her effort in editing the earlier versions. To Mrs. Annetta Whiteman for her excellent technical typing of so many versions of the book. To Ms. Sue Sullivan, for skillfully converting the material into the book formats using the UNIX system. To Mr. Y.S. Kuo for preparing the index and writing parts of the manual. And last but most to Dr. Man-Tak Shing, for writing the manual and his technical and general assistance throughout the writing and production.

La Jolla, California

October 19, 1981

T. C. Hu

CONTENTS

Chapter 1. Shortest Paths

Chapter 2. Maximum Flows

Chapter 3. Dynamic Programming

Chapter 4. Backtracking

Chapter 5. Binary Tree

Chapter 6. Heuristic and Near Optimum

Chapter 7. Matrix Multiplication

Chapter 8. NP-complete

CHAPTER 1. SHORTEST PATHS

There is no shortest path to success.

§ 1.1 GRAPH TERMINOLOGY

When we try to solve a problem, we often draw a graph. A graph is often the simplest and easiest way to describe a system, a structure, or a situation. The Chinese proverb "A picture is worth one thousand words" is certainly true in mathematical modeling. This is why graph theory has a wide variety of applications in physical, biological, and social sciences. Due to the wide variety of applications, we also have diverse terminology. Papers on graph theory are full of definitions, and every author has his own definitions. Here, we introduce a minimum number of definitions which are intuitively obvious. The notation and terminology adopted here is similar to that of Knuth [18].

A graph consists of a finite set of vertices and a set of edges joining the vertices. We shall draw small circles to represent vertices and lines to represent edges. A system or a structure can often be represented by a graph where the lines indicate the relations among the vertices (the elements of the system). For example, we can use vertices to represent cities and edges to represent the highways connecting the cities. We can also use vertices to represent persons and draw an edge joining two vertices if the two persons know each other.

The reader should keep in mind that graph theory is a theory of relations, *not* a theory of definitions; however, a minimum number of definitions is needed here. Vertices are also called nodes, and edges are also called arcs, branches, or links. We usually assume that there are n vertices in the graph G and at most one edge joining any two vertices and there is no edge joining a node to itself. The vertices are denoted by V_i ($i = 1,2,...,n$) and the edge joining V_i and V_j is denoted by e_{ij}. Two vertices are *adjacent* if they are joined by an edge (the two vertices are also called *neighbors*); two edges are adjacent if they are both incident to the same vertex. A vertex is of *degree* k if there are k edges incident to it.

A sequence of vertices and edges

$$(V_1,e_{12},V_2,e_{23},V_3,...,V_n)$$

is said to form a *path* from V_1 to V_n. We can represent a path by only its vertices as

$$(V_1,V_2,...,V_n)$$

or by only the edges in the path as

$$(e_{12}, e_{23}, \ldots, e_{n-1,n}).$$

A graph is *connected* if there is a path between any two nodes of the graph. A path is of length k if there are k edges in the path. A path is a *simple path* if all the vertices $V_1, V_2, \ldots, V_{n-1}, V_n$ are distinct. If $V_1 = V_n$, then it is called a cycle. In other words, a cycle is a path of length three or more from a vertex to itself. If all vertices in a cycle are distinct, then the cycle is a simple *cycle.* Unless otherwise stated, we shall use the word "path" to mean a simple path, "cycle" to mean a simple cycle, and "graph" to mean a connected graph.

If an edge has a direction (just like a street may be a one-way street), then it is called a *directed edge.* If a directed edge is from V_i to V_j, then we cannot follow this edge from V_j to V_i. Thus in the definition of a path, we want an edge to be undirected or to be a directed edge from V_i to V_{i+1}. In all other definitions, the directions of edges are ignored. A graph is called a *directed graph* if all edges are directed and a *mixed graph* if some edges are directed and some are not. A cycle formed by directed edges is called a *directed cycle* (or *circuit*). A directed graph is called *acyclic* if there are no directed cycles. The words "graph" and "edge" are used for an undirected graph and an undirected edge throughout this section.

A *tree* is a connected graph with no cycles. If a graph has n vertices, then any *two* of the following conditions characterize a tree and automatically imply the third condition.

1. The graph G is connected.
2. The graph has n-1 edges.
3. The graph contains no cycles.

We shall denote a graph by G = (V; E) where "V" is the set of nodes or vertices, and "E" is the set of edges in the graph. A graph $G' = (V'; E')$ is a *subgraph* of G = (V; E) if $V' \subseteq V$ and $E' \subseteq E$.

A subgraph which is a tree and which contains all the vertices of a graph is called a *spanning tree* of the graph. We shall illustrate these intuitive definitions of graph theory in Figure 1.1.

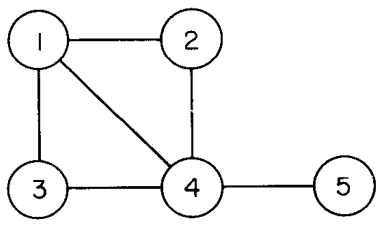


Figure 1.1

There are three paths between V_1 and V_5, namely (V_1, V_2, V_4, V_5), (V_1, V_4, V_5), (V_1, V_3, V_4, V_5). The edges e_{14}, e_{24}, e_{34}, and e_{45} form a spanning tree and so do the edges e_{12}, e_{24}, e_{34}, and e_{45}. Also, we may pick e_{12}, e_{13}, e_{34}, and e_{45} to be the spanning tree. Here the node V_1 is of degree 3 in the graph G but is of degree 2 in the last spanning tree. If the edge e_{45} was directed from V_4 to V_5, then there are still three paths from V_1 to V_5 but none from V_5 to V_1.

In most applications, we associate numbers with edges or vertices. Then the graph is called a *network*. All the definitions of graph theory apply to networks as well. In network theory, we usually use "nodes" and "arcs" instead of "vertices" and "edges".

§1.2 SHORTEST PATH

One of the fundamental problems in network theory is to find shortest paths in a network. Each arc of the network has a number which is the length of the arc.

In most cases, the arcs have positive lengths, but the arcs may have negative lengths in some applications. For example, the nodes may represent the various states of a physical system, where the length associated with the arc e_{ij} denotes the energy absorbed in transforming the state V_i to the state V_j. An arc with negative length then indicates that energy is released in transforming the state V_i into the state V_j. If the total length of a circuit or cycle is negative, we say that the network contains a negative circuit.

The length of a path is the sum of lengths of all the arcs in the path. There are usually many paths between a pair of nodes, say V_s and V_t, but a path with the minimum length is called a *shortest path* from V_s to V_t.

The problem of finding a shortest path is a fundamental problem and often occurs as a subproblem of other optimization problems. In some applications, the numbers associated with arcs may represent characteristics other than lengths and we may want optimum paths where optimum is defined by a different criterion. But the shortest path problem is the most common problem in the whole class of optimum path problems. The shortest path algorithm can usually be modified slightly to find other optimum paths. Thus we shall concentrate on the shortest paths.

If we denote a path from V_1 to V_k by $(V_1, V_2, \ldots, V_k)$, then $e_{i,i+1}$ must be either a directed arc from V_i to V_{i+1} or an undirected arc joining V_i and V_{i+1} $(i = 1,\ldots,k\text{-}1)$. In most applications, we can think of an undirected arc between V_i and V_j as two directed arcs, one from V_i to V_j and the other from V_j to V_i. We usually are interested in three kinds of shortest-path problems:

(1) The shortest path from one node to another.

(2) The shortest paths from one node to all the other nodes.

(3) The shortest paths between all pairs of nodes.

Since all algorithms solving Problem (1) and Problem (2) are essentially the same, we shall discuss the problem of finding shortest paths from one node to all other nodes in the network.

The problem of finding shortest paths is well-defined if the network does not contain a negative cycle (or a negative circuit). Note that a network can have some directed arcs with negative length and yet does not contain a negative cycle. We shall first study the case that all arcs have positive length.

Let us denote the length of the arc from V_i to V_j by d_{ij}, and assume that

$$d_{ij} > 0 \quad \text{for all } i,j \quad \text{condition 1}$$

$$d_{ij} \neq d_{ji} \quad \text{for some } i,j \quad \text{condition 2}$$

$$d_{ij} + d_{jk} \leqslant d_{ik} \quad \text{for some } i,j,k \quad \text{condition 3}$$

For convenience, we assume that $d_{ij} = \infty$ if there is no arc leading from V_i to V_j and $d_{ii} = 0$ for all i.

Condition 3 makes the shortest-path problem nontrivial. Otherwise, the shortest path from V_i to V_j consists of the single arc e_{ij}.

Assume that there are n nodes in the network and we want the shortest paths from V_0 to V_i ($i = 1, 2, \ldots, n-1$). If there are two or more shortest paths from V_0 to a node, then any one path is equally acceptable.

Usually, we would like to know the length of a shortest path as well as the intermediate nodes in the path.

Let us make some observations first. Let P_k be a path from V_0 to V_k, where V_i is an intermediate node on the path. Then the subpath from V_0 to V_i contains fewer arcs than the path P_k. Since all arcs have positive lengths, the subpath must be shorter than P_k. We state this as observation 1.

Observation 1. The length of a path is always longer than the length of any of its subpaths. (Note this is true only when all arcs are positive.)

Let V_i be an intermediate node on the path P_k (from V_0 to V_k). If the path P_k is a shortest path, then the subpath from V_0 to V_i must *itself* be a shortest path. Otherwise, a shorter path to V_i followed by the original route from V_i to V_k constitutes a path shorter than P_k. This would contradict that P_k is a shortest path. We state this as observation 2.

Observation 2. Any subpath (of a shortest path) must itself be a shortest path.

(Note that this does not depend on arcs having positive lengths.)

Observation 3. Any shortest path contains at most n-1 arcs. (This depends on arcs not forming negative cycles and that there are n nodes in the network.)

Based on these three observations, we can develop an algorithm to find the shortest paths from V_0 to all the other nodes in the network.

Imagine that all shortest paths from V_0 to all other nodes have been ordered according to their path lengths. *For convenience of discussion, we can rename the nodes such that the shortest path to* V_1 *is the shortest among all shortest paths.* We shall write

$$P_1 \leqslant P_2 \leqslant P_3 \leqslant \ldots \leqslant P_{n-1}$$

to denote that the lengths of these paths are monotonically increasing.

The algorithm will find P_1 first, P_2 second,..., until the longest of the shortest paths is found.

Let us motivate the ideas behind the algorithm. How many arcs are in the path P_1? If P_1 contains more than one arc, then it contains a subpath which is shorter than P_1 (observation 1). Thus P_1 must contain only one arc.

If P_k contains more than k arcs, then it contains at least k intermediate nodes on the path. Each of the subpaths to an intermediate node must be shorter than P_k, and we would have k paths shorter than P_k, a contradiction. Thus *the shortest path* P_k *contains at most* k *arcs.* We shall state this as observation 4.

Observation 4. The shortest path P_k contains at most k arcs.

To find P_1, we need only examine one-arc paths; the minimum among these must be P_1.

To find P_2, we need only one-arc or two-arc paths. The minimum among these must be P_2. If P_2 is a two-arc path where the last arc is e_{j2} $(j \neq 1)$, then the single arc e_{0j} is a subpath of P_2 and hence shorter than P_2. Thus, the path P_2 must be either a one-arc path or a two-arc path where the arc e_{12} is the last arc on the path P_2.

In what follows, we shall write numbers on the nodes and call these numbers labels. There are two kinds of labels, temporary and permanent labels. The permanent label on a node is the true shortest distance from the origin V_0 to that node. A temporary label is the length of a path from the origin to that node. Since the path may or may not be a shortest path, a temporary label is an upper bound on the true shortest distance.

When we search for P_1, we write on every node V_i the length of the arc d_{0i}. These are called the temporary labels of V_i (since these labels may be changed later). Among all the temporary labels, we select the minimum and change the label to be permanent. Thus we have V_1 permanently labelled. (A node

permanently labelled is called a permanent node.)

To find P_2, we do not have to find all two-arc paths; only those where the first arc is e_{01}. All the lengths of one-arc paths have already been written on the nodes as temporary labels. So we can compare d_{0i} (the length of one arc) with $d_{01} + d_{1i}$ (the length of the two-arc path), and the minimum of the two is written on V_i as the temporary label of V_i. Then among all temporary labels, the minimum is P_2.

The permanent label on a node V_i indicates the true shortest distance from V_0 to V_i. The temporary label on a node V_j indicates either the distance of the arc e_{0j} or the distance of a path from V_0 to a permanent node V_i followed by the arc e_{ij}.

Imagine that all arcs of the network were colored green. Whenever an arc is used in a shortest path, we recolor it brown. So we use one brown arc to reach V_1, one or two brown arcs to reach V_2, ..., at most k brown arcs to reach V_k. (The choice of color is such that the brown arcs form a tree, see EX. 2.) We see that the path P_{k+1} cannot contain nodes with temporary labels as intermediate nodes. Thus we can limit our search to those paths consisting of a sequence of brown arcs followed by one green arc reaching the node V_{k+1}. Two or more green arcs indicate a subpath of shorter distance than P_{k+1}.

To find the path P_{k+1} containing one green arc and possibly some brown arcs, we limit our search to the neighbors of V_0, V_1, ..., V_k. The search is made easy if we adopt the following rule:

Whenever a node V_i receives a permanent label, say l_i^*, we shall check all temporary labels of neighbors V_j of V_i to see if $l_i^* + d_{ij}$ is less than the current temporary label of V_j. If it is less, we shall replace the current temporary label by the smaller value. If not, we leave the temporary label unchanged.

To find P_{k+1}, we just find the minimum of temporary labels of all neighbors of V_0, V_1,...,V_k and change the minimum to a permanent label.

Now we can formalize the algorithm and use it in a numerical example. We will use l_i to denote the temporary shortest distances and l_i^* to denote the true shortest distances.

Dijkstra's Algorithm

Step 0. All nodes V_i receive temporary labels l_i with value equal to d_{0i} ($i = 1,2,...,n-1$). For convenience, we can take $d_{0i} = \infty$ if there is no arc joining V_0 and V_i.

Set $l_i = d_{0i}$ $(i = 1,...,n-1)$

Step 1. Among all temporary labels l_i

Pick $l_k = \min_i l_i$.

Change l_k to l_k^*.

Stop if there is no temporary label left.

Step 2. Let V_k be the node that just received a permanent label in Step 1. Replace all temporary labels of the neighbors of V_k by the following rule:

$l_i \leftarrow \min[l_i, l_k^* + d_{ki}]$

Return to Step 1.

Consider the network shown in Figure 1.2 where the numbers are arc lengths.

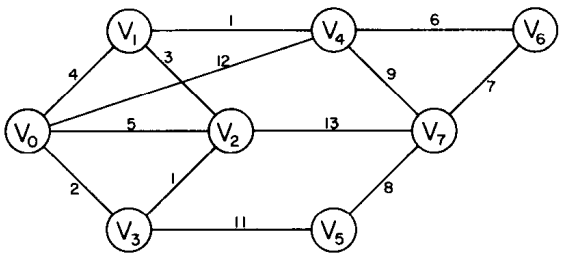


Figure 1.2

We shall put temporary labels inside each node and when the label becomes permanent, we shall add a star on the number. Whenever arcs are used in a shortest path, we shall use heavy lines to represent them.

Step 0. All nodes receive temporary labels equal to d_{0i}, and the node V_0 gets permanent label 0. This is shown in Figure 1.3.

Step 1. Among all temporary labels, V_3 has the minimum value 2, so V_3 receives a permanent label.

Step 2. The node V_3 has neighbors V_2 and V_5

$l_2 \leftarrow \min[l_2, l_3^* + d_{32}] = \min[5, 2+1] = 3.$

$l_5 \leftarrow \min[l_5, l_3^* + d_{35}] = \min[\infty, 2+11] = 13.$

The result is shown in Figure 1.4.

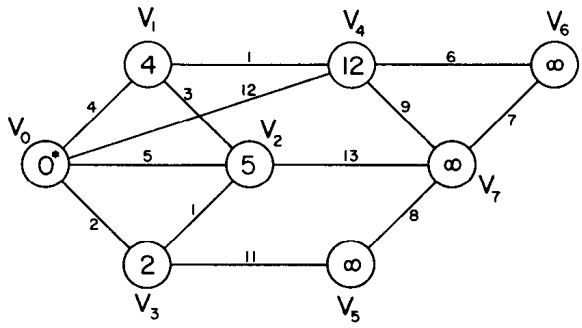


Figure 1.3

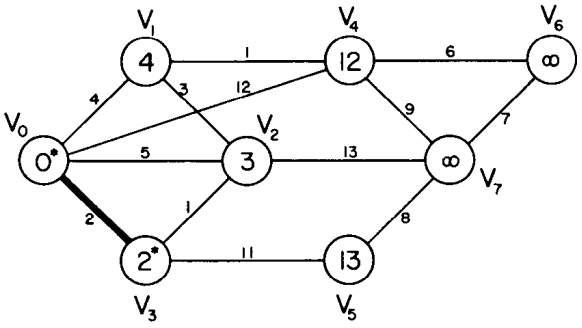


Figure 1.4

Step 1. Among all temporary labels, the node V_2 has the minimum label 3. So V_2 receives a permanent label.

Step 2. The neighbors of V_2 are V_1,V_7. (Note V_3 is also a neighbor, but since V_3 has become permanent it is excluded.)

$l_1 \leftarrow \min[l_1, l_2^* + d_{21}] = \min[4, 3+3] = 4$

$l_7 \leftarrow \min[l_7, l_2^* + d_{27}] = \min[\infty, 3+13] = 16$

Step 1. The node V_1 receives the permanent label 4.

Step 2. $l_4 \leftarrow \min[l_4, l_1^* + d_{14}] = \min[12, 4+1] = 5.$

This is shown in Figure 1.5.

Step 1. V_4 gets a permanent label.

Step 2. $l_6 \leftarrow \min[l_6, l_4^* + d_{46}] = \min[\infty, 5+6] = 11$