Minimum Cost Spanning Tree Algorithms (Greedy Algorithms):

– Kruskal's

– Prim's

Kruskal's Algorithm

Tree is built edge by edge. Include edges in the tree in non–decreasing order of their cost. An edge is included in the tree if it does not form any cycle with the edges already in the tree.

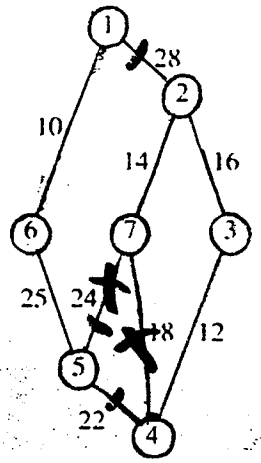
T = Empty

While T contains less than n–1 edges and E not empty
Proceed as follows:

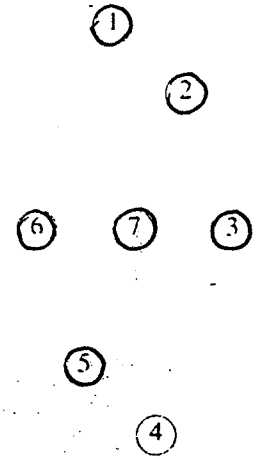
Choose an edge (v,w) from E of lowest cost,
Delete (v,w) from E,
If (v,w) does not create a cycle in T
add(v,w) to T
Else discrad (v,w)

If T contains fewer than n–1 edges then no spanning tree.

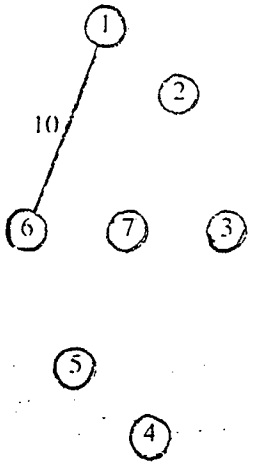
Example



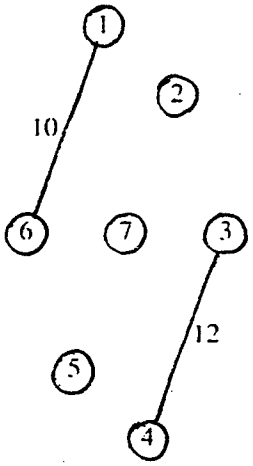
(a)



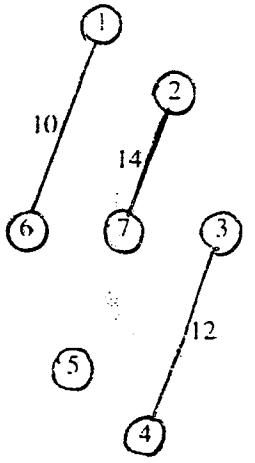
(b)



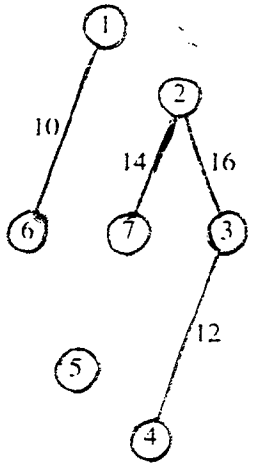
(c)



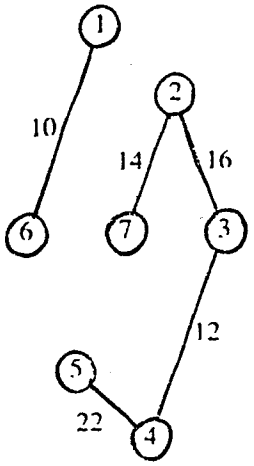
(d)



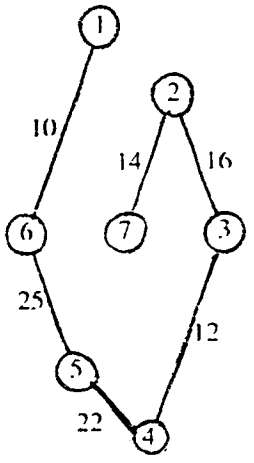
(e)



(f)



(g)



(h)

Implementation and Complexity:

Use heap sort for sorting edges. O(e log e)

Maintain information about components (forest) generated during the algorithm in form of sets of vertices to check for cycles.

Prim's Algorithm

Tree is built edge by edge. The set of selected edges always form a tree. Always select an edge with the lowest cost from unselected edges, such that a single tree is formed (no cycles).

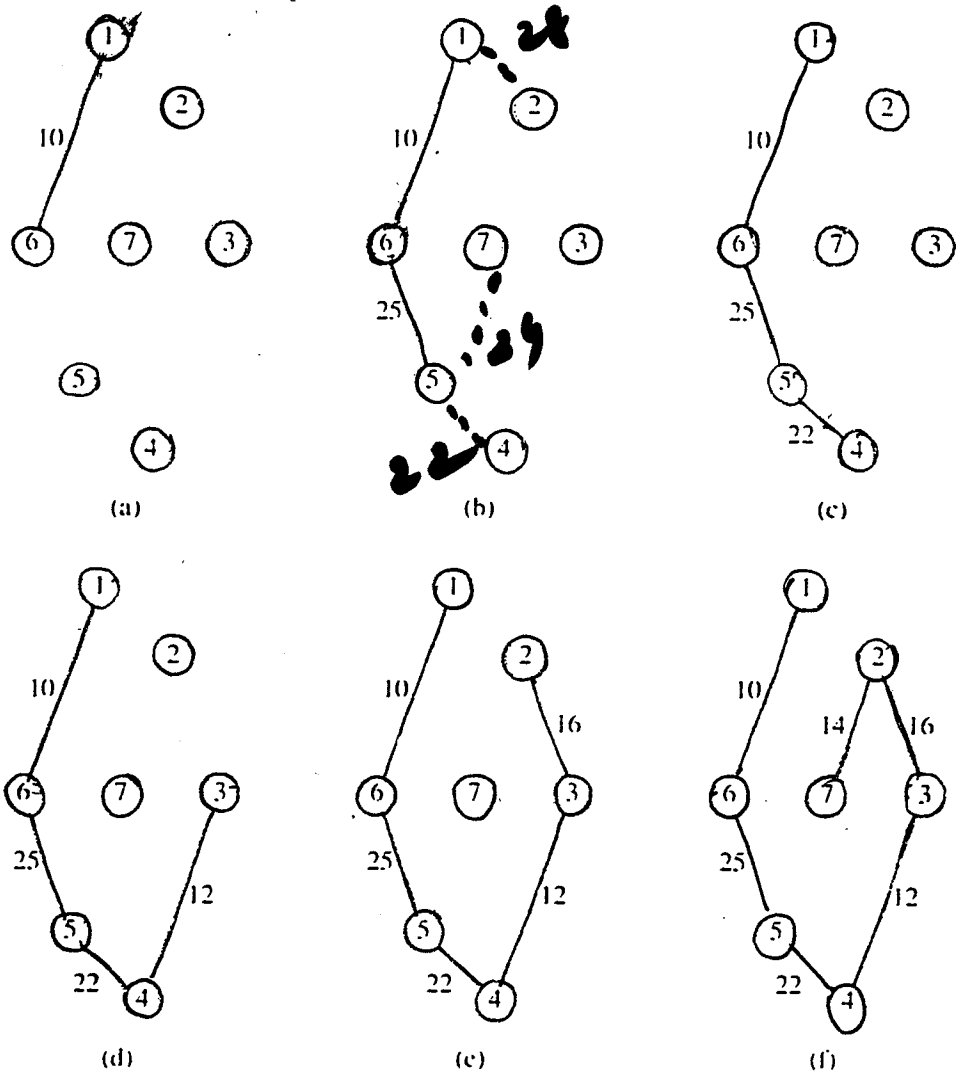
Prim's Algorithm

Tree is built edge by edge. The set of selected edges always form a tree. Always select an edge with the lowest cost from unselected edges, such that a single tree is formed (no cycles).

Complexity: $O(n^2)$

Better implementations are possible, such as using Fibonacci heaps.

Example



Topological Sort

– Assume a DAG

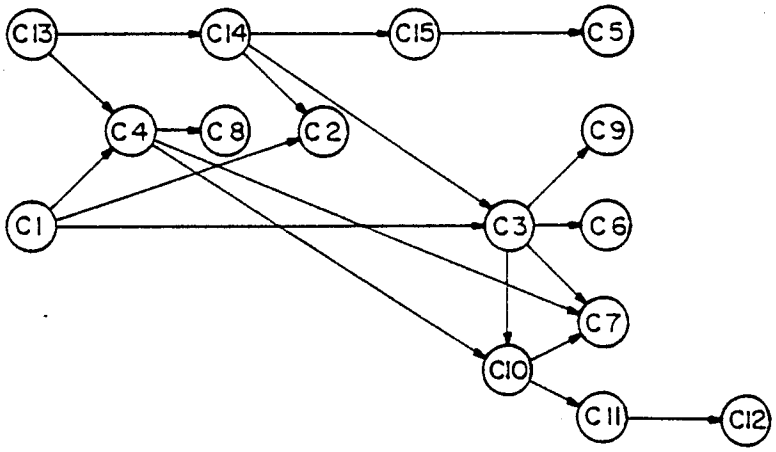
– Topological Order is a linear list of all the vertices, such that if there is an edge from v to w, the v preceds w in the list.

– Use:
Scheduling Tasks, Identifying Pre–requisites

Example

Course Number	Course Name	Prerequisites
C1	Introduction to Programming	None
C2	Numerical Analysis	C1, C14
C3	Data Structures	C1, C14
C4	Assembly Language	C1, C13
C5	Automata Theory	C15
C6	Artificial Intelligence	C3
C7	Computer Graphics	C3, C4, C10
C8	Machine Arithmetic	C4
C9	Analysis of Algorithms	C3
C10	Higher Level Languages	C3, C4
C11	Compiler Writing	C10
C12	Operating Systems	C11
C13	Analytic Geometry and Calculus I	None
C14	Analytic Geometry and Calculus II	C13
C15	Linear Algebra	C14

(a) Courses Needed for a Computer Science Degree at Some Hypothetical University



Algorithm

If every vertex has a predecessor
  Halt (cycles exist)

For all vertices:
  Pick a vertex (v) with no predecessors
   output v that vertex
    delete v and all the edges leading out of v;

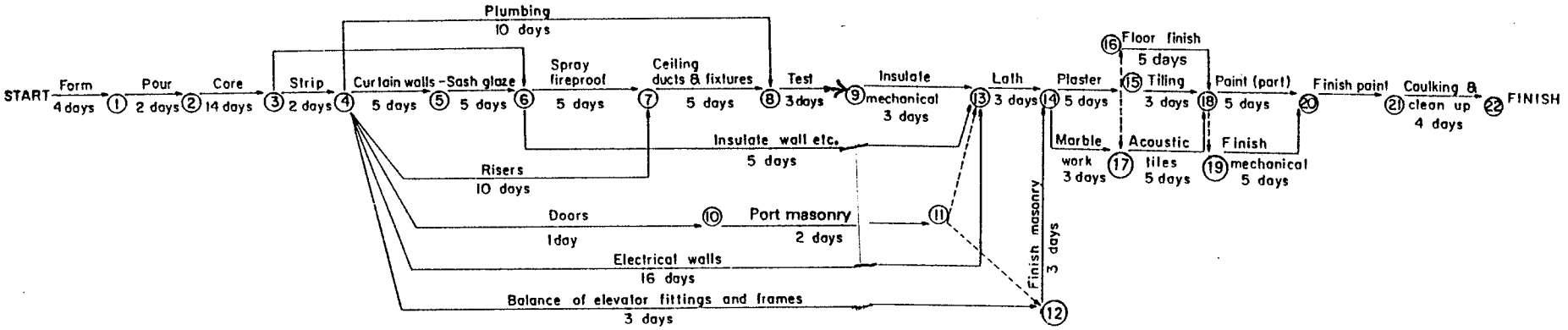


Figure 6.34 AOE network for the construction of a typical floor in a multistory building [Engineering News-Record (McGraw-Hill Book Company, Inc., Jan. 26, 1961).]

Activity Networks and PERT Diagrams

Use Network (Directed Graph) Models. Represent subprojects, tasks, or activities either as:

– vertices of a graph, with directed edges showing the precedence relation among tasks. Activity time is labelled on vertices. Activity On Vertices (AOV) network,

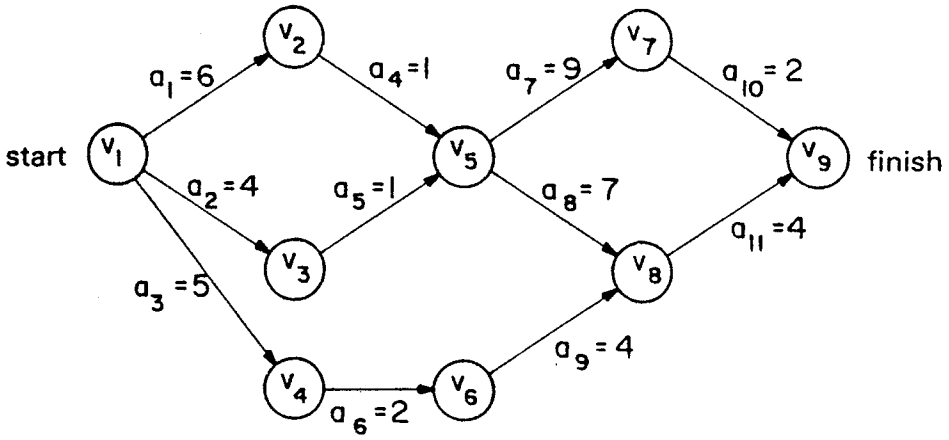
– or as weighted edges, with weights showing the execution time of tasks. The vertices show the synchronization points. All in–edge activities must complete before out–edge activities start. Therefore, a vertex represents end of some events and start of other events. Activity On Edge (AOE). PERT (Performance Evaluation and Review Technique) Diagram.
In an AOE: Activity is an edge, event is a vertex.

Activity Networks and PERT Diagrams

– Evaluating Performance of Projects

– Identifying Bottleneck and potential improvements to speed–up in order to reduce the completion time of a project.

Examples of AOE



(a) AOE Network. Activity Graph of a Hypothetical Project.

event	interpretation
v_1	start of project
v_2	completion of activity a_1
v_5	completion of activities a_4 and a_5
v_8	completion of activities a_8 and a_9
v_9	completion of project

(b) Interpretation for Some of the Events in the Activity Graph of (a).

Analyzing PERT Diagram and Critical Path

– Critical Path: A path of longest length in PERT diagram. Length is the sum of weights on the edges (or vertices). It gives the minimum time to complete the project.

– **Earliest Time** of an event (represented by a vertex), is the length of the longest path from the starting vertex to that vertex.

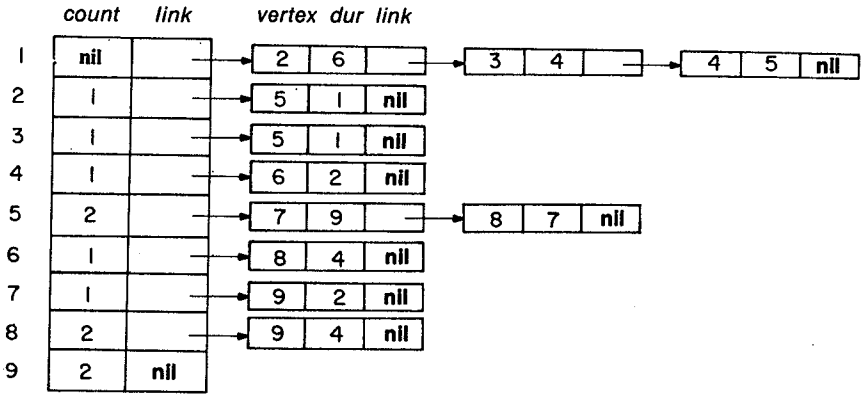
– **Earliest Start Time** for all the activities represented by edges leaving a vertex is equal to the Earliest Time of that vertex.

– **Latest Time of an activity**: How late an activity start without increasing the project duration.

– Critical Activity: If
Earliest Start Time = Latest Time

– Criticality of an activity: **Latest Time –Earliest Start Time**

We should try to speed up critical activities to speed–up the completion of the project.



(a) Adjacency Lists for Figure 6.34(a)

ee	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	stack
initial	0	0	0	0	0	0	0	0	0	1
output v_1	0	6	4	5	0	0	0	0	0	4 3 2
output v_4	0	6	4	5	0	7	0	0	0	6 3 2
output v_6	0	6	4	5	0	7	0	11	0	3 2
output v_3	0	6	4	5	5	7	0	11	0	2
output v_2	0	6	4	5	7	7	0	11	0	5
output v_5	0	6	4	5	7	7	16	14	0	8 7
output v_8	0	6	4	5	7	7	16	14	16	7
output v_7	0	6	4	5	7	7	16	14	18	9
output v_9										

(b) Computation of *ee*

Figure 6.35 Action of modified topological order

Network Flow Problem

Maximizing flow of material, information etc., in a Network, from a source node to a sink node.

Examples: Liquids flowing through pipes, Parts through assembly lines, Current through electrical networks, information through communication networks.

Example:

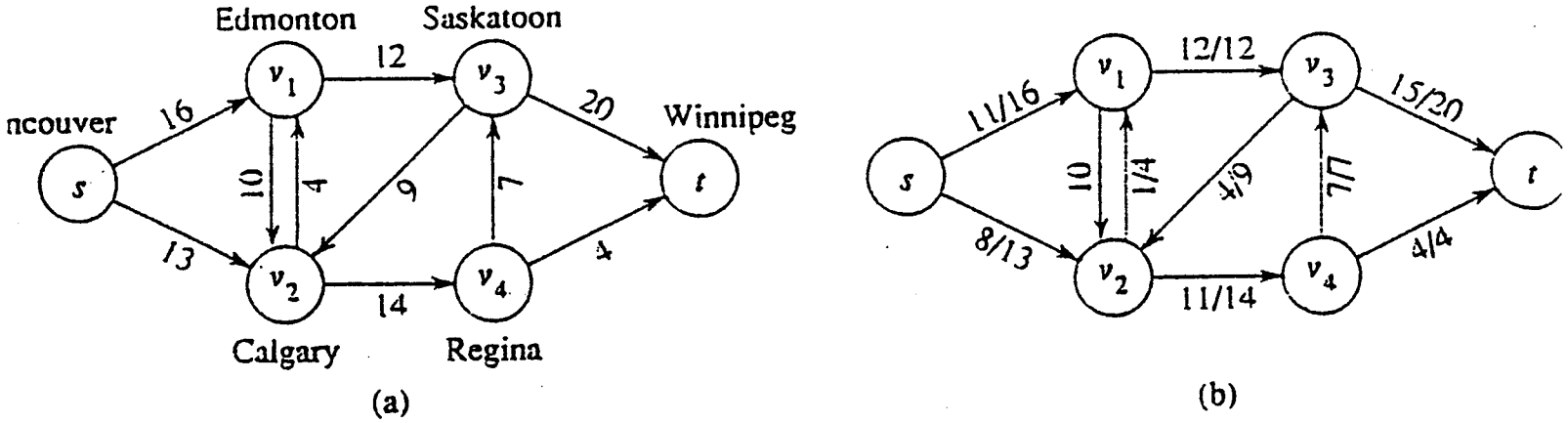


Figure 27.1 (a) A flow network $G = (V, E)$ for the Lucky Puck Company's truck ing problem. The Vancouver factory is the source s, and the Winnipeg warehous is the sink t. Pucks are shipped through intermediate cities, but only $c(u, v)$ crate per day can go from city u to city v. Each edge is labeled with its capacity. (b) A flow f in G with value $|f| = 19$. Only positive net flows are shown. If $f(u, v) > 0$ edge (u, v) is labeled by $f(u, v)/c(u, v)$. (The slash notation is used merely to sep arate the flow and capacity; it does not indicate division.) If $f(u, v) \leq 0$, edg (u, v) is labeled only by its capacity.

Network Flow Problem

Network G(V,E) is a connected directed graph.

– Each edge (u,v) has a nonnegative capacity :

$c(u,v) \geq 0$

– Each edge (u,v) has a flow : $f(u,v) \leq c(u,v)$.
f represents the net flow from vertex v to vertex u.

– Flow f has a skew symmetry property:
$f(u,v) = -f(v,u)$. Flow in the reverse direction.

Network Flow Problem

Problem is to maximize flow from source to sink, under the following constraints.

1. Total Flow out of Source =

Total flow into the Sink =

2. inflow of a node = outflow of that node (except source and sink). In other words, for all ***u*** (excluding source and sink), the value of flow *f*,

$$| f | = \sum_{v \varepsilon V} f(u, v) = 0$$

The Ford–Fulkerson Method

1. Initialize flow f on all the edges to 0.

2. From the source to the sink, find a path along which we can push more flow. Augment the flow along this path as follows:

The amount of flow that can be pushed on a path $\boldsymbol{p}$:

$\boldsymbol{Min}\{\boldsymbol{c(u,v) - f(u,v)}$: where $\boldsymbol{(u,v)}$ is in path $\boldsymbol{p}\}$

Therefore, increase flow of each link on $\boldsymbol{p}$ by this amount.
You can use $\boldsymbol{-f(v,u)}$ to check for the possible existence of other paths.

3. Keep repeating Step 2, until no more paths are possible.

Example

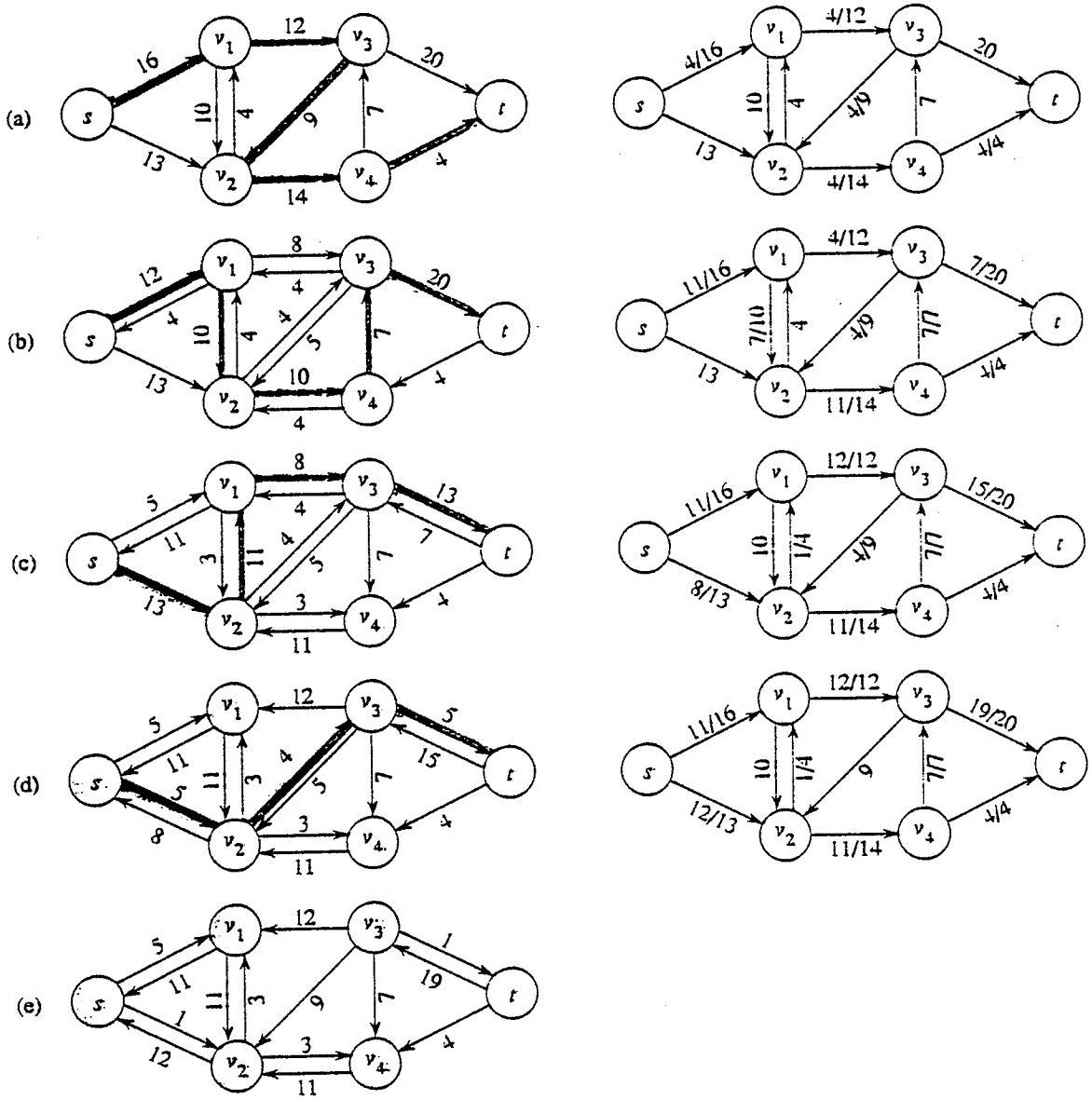


Figure 27.6 The execution of the basic Ford-Fulkerson algorithm. (a)–(d) Successive iterations of the **while** loop. The left side of each part shows the residual network G_f from line 4 with a shaded augmenting path p. The right side of each part shows the new flow f that results from adding f_p to f. The residual network in (a) is the input network G. (e) The residual network at the last **while** loop test. It has no augmenting paths, and the flow f shown in (d) is therefore a maximum flow.