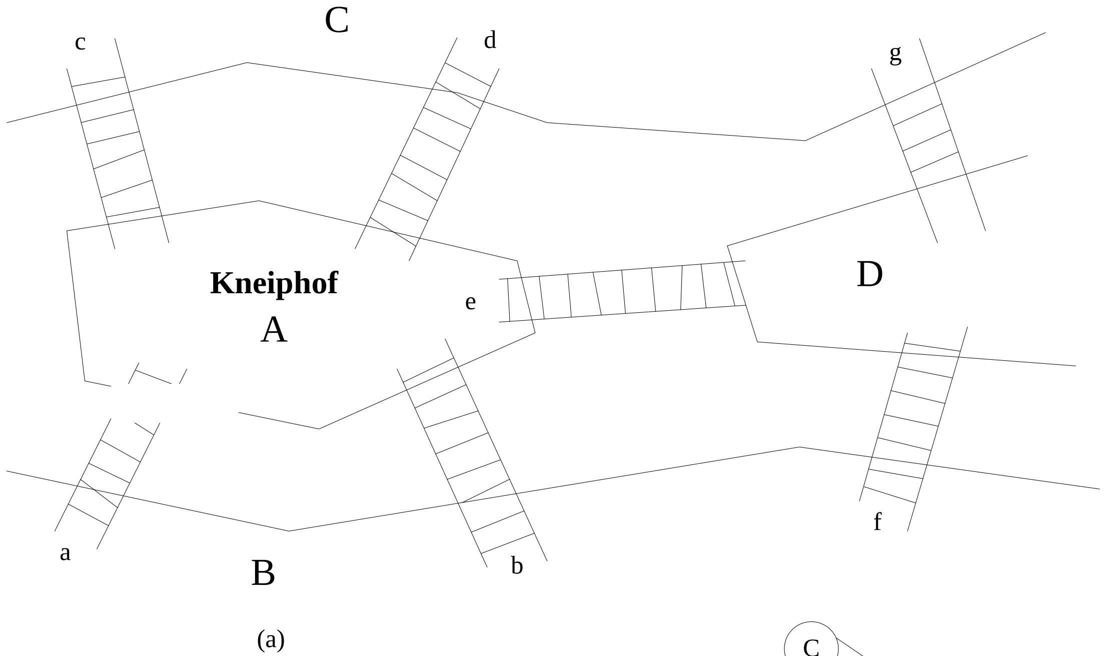
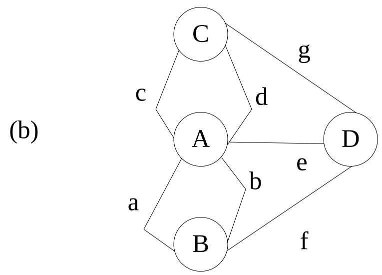
CHAPTER 6

Graphs

All the programs in this file are selected from

Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed
"Fundamentals of Data Structures in C /2nd Edition",
Silicon Press, 2008.





Euler trail, circuit (Edge)
Hamilton path, Cycle (Vertic)

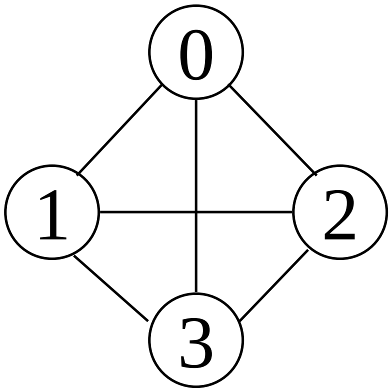
Figure 6.1:(a)Section of the river Pergel in Konigsberg; (b) Euler's graph (p.266)

Definition

- A G consists of two sets
  - – a finite, nonempty set of vertices V(G)
  - – a finite, possible empty set of edges E(G)
  - – G(V,E) represents a graph
- An is one in which the pair of vertices in a edge is unordered, $(v_0, v_1) = (v_1,v_0)$
- A is one in which each edge is a directed pair of vertices, $<v_0, v_1> != <v_1,v_0>$

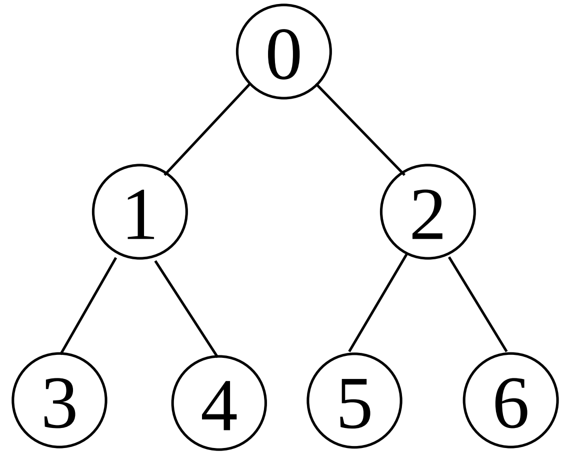
tail head

Examples for Graph



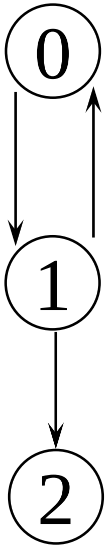
G_1

graph



G_2

graph



G_3

$V(G_1)=\{0,1,2,3\}$ $E(G_1)=\{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$

$V(G_2)=\{0,1,2,3,4,5,6\}$ $E(G_2)=\{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$

$V(G_3)=\{0,1,2\}$ $E(G_3)=\{<0,1>,<1,0>,<1,2>\}$

complete undirected graph: edges

complete directed graph: edges

Complete Graph

- A complete graph is a graph that has the maximum number of edges
  - for undirected graph with n vertices, the maximum number of edges is $n(n-1)/2$
  - for directed graph with n vertices, the maximum number of edges is $n(n-1)$
  - example: G1 is a complete graph

Adjacent and Incident

- If (v_0, v_1) is an edge in an undirected graph,
  - v_0 and v_1 are adjacent
  - The edge (v_0, v_1) is incident on vertices v_0 and v_1
- If $<v_0, v_1>$ is an edge in a directed graph
  - v_0 is adjacent v_1, and v_1 is adjacent v_0
  - The edge $<v_0, v_1>$ is incident on v_0 and v_1

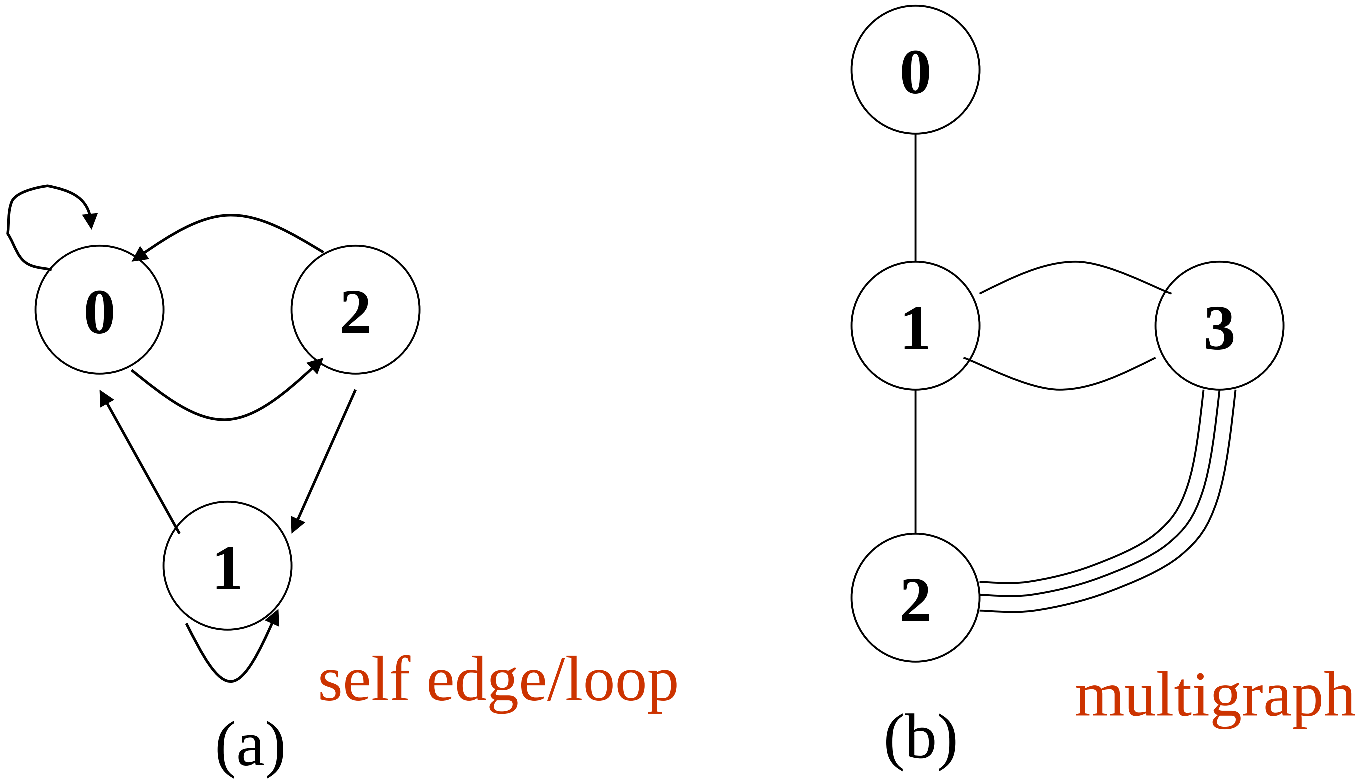
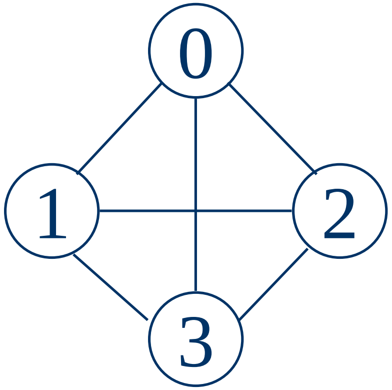


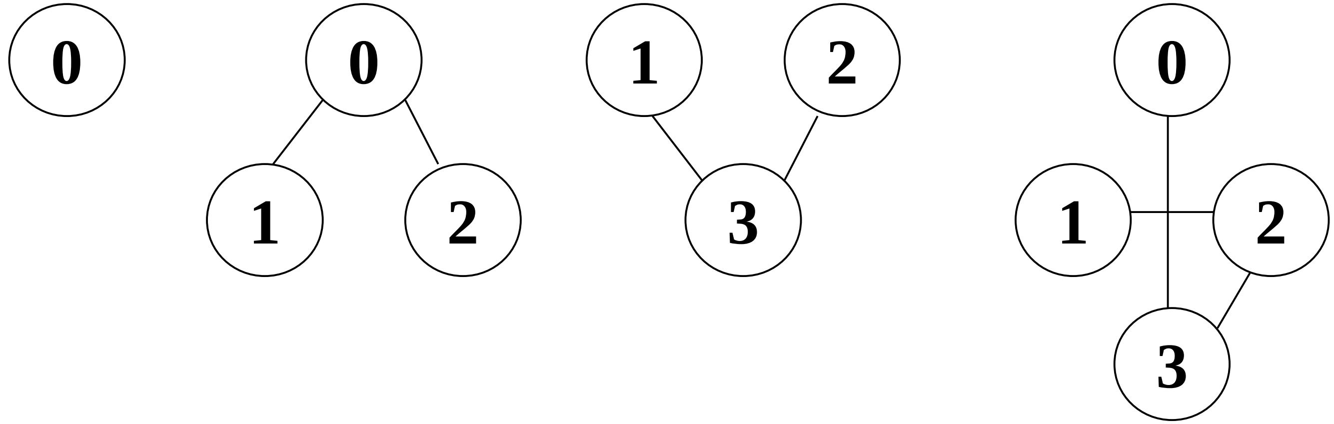
Figure 6.3: Example of a graph with feedback loops and a multigraph (p.268)

Subgraph and Path

- A of G is a graph G’ such that V(G’) is a subset of V(G) and E(G’) is a subset of E(G)
- A from vertex v_p to vertex v_q in a graph G, is a sequence of vertices, v_p, v_{i1}, v_{i2}, ..., v_{in}, v_q, such that (v_p, v_{i1}), (v_{i1}, v_{i2}), ..., (v_{in}, v_q) are edges in an undirected graph
- The length of a path is the number of edges on it

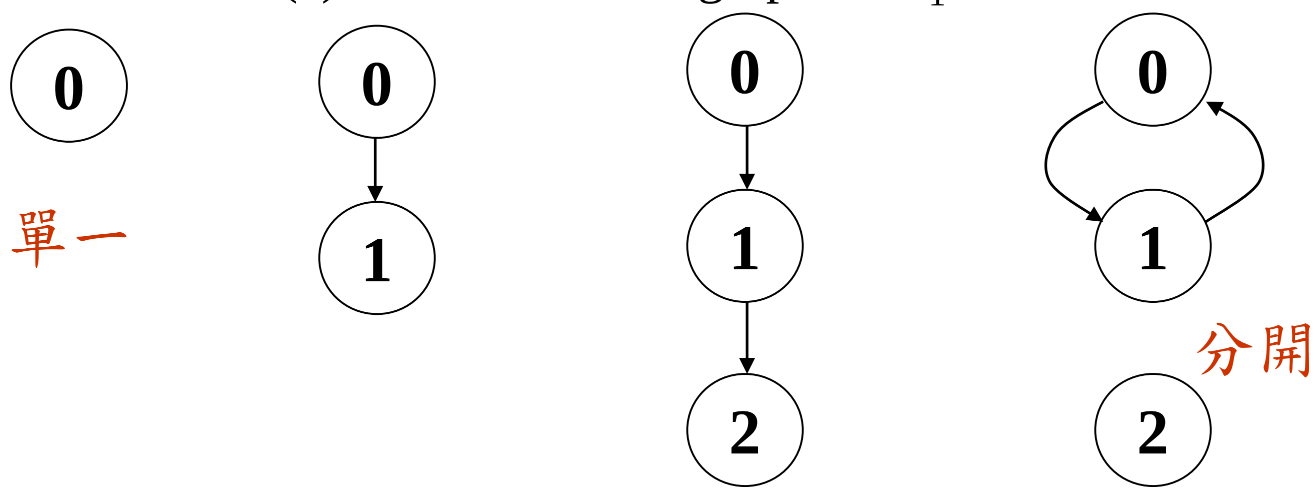


G_1



(i) (ii) (iii) (iv)

(a) Some of the subgraph of G_1



(i) (ii) (iii) (iv)

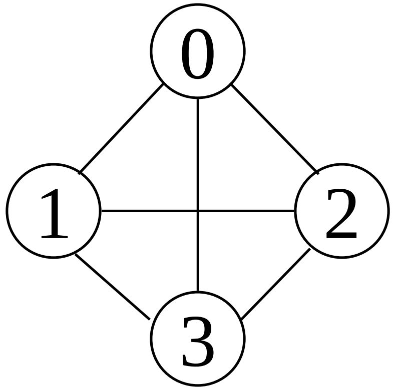
(b) Some of the subgraph of G_3

G_3

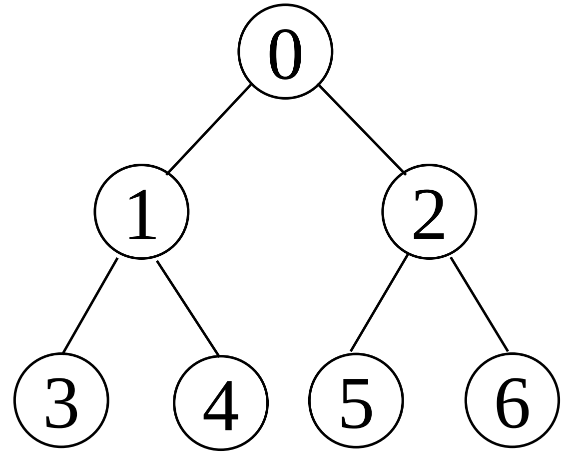
Figure 6.4: subgraphs of G_1 and G_3 (p.269)

Simple Path and Style

- A is a path in which all vertices, except possibly the first and the last, are distinct
- A is a simple path in which the first and the last vertices are the same
- In an undirected graph G, two vertices, v_0 and v_1, are if there is a path in G from v_0 to v_1
- An undirected graph is connected if, for every pair of distinct vertices v_i, v_j, there is a path from v_i to v_j



G_1



G_2

tree (acyclic graph)

Connected Component

- A connected component of an undirected graph is a maximal connected subgraph.
- A is a graph that is connected and acyclic.
- A directed graph is strongly connected if there is a directed path from v_i to v_j and also from v_j to v_i.
- A strongly connected component is a maximal subgraph that is strongly connected.

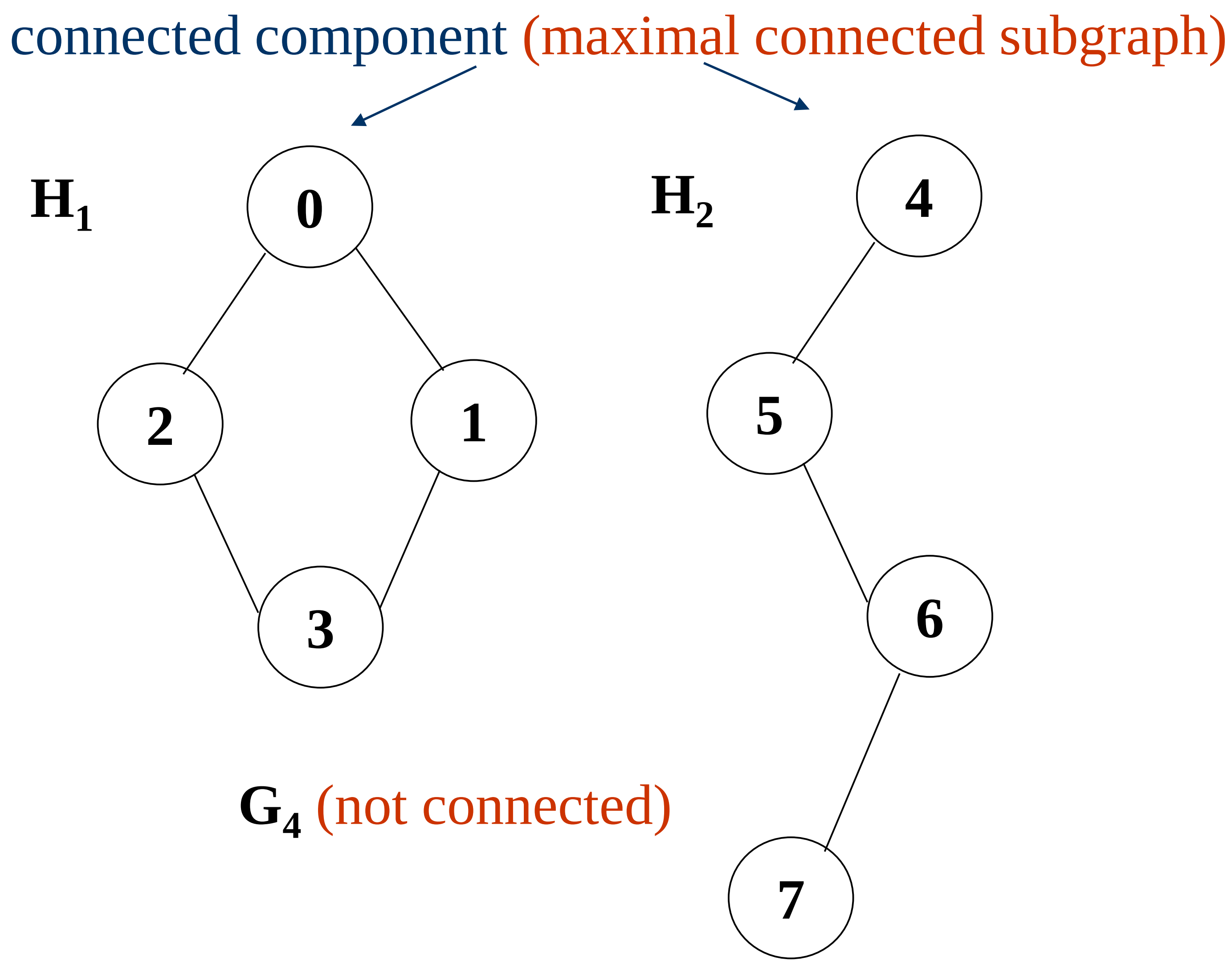


Figure 6.5: A graph with two connected components (p.270)

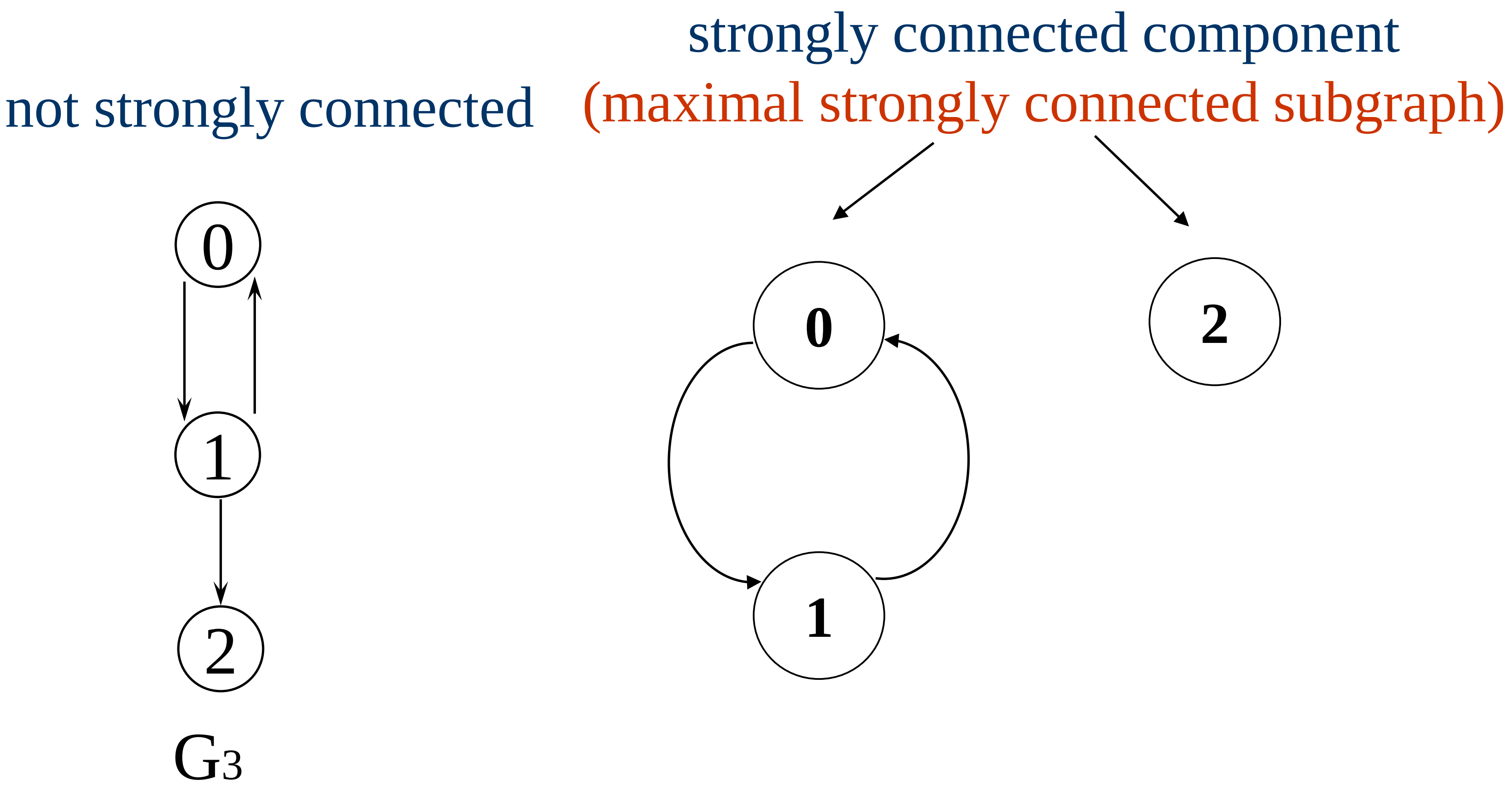


Figure 6.6: Strongly connected components of G_3 (p.270)

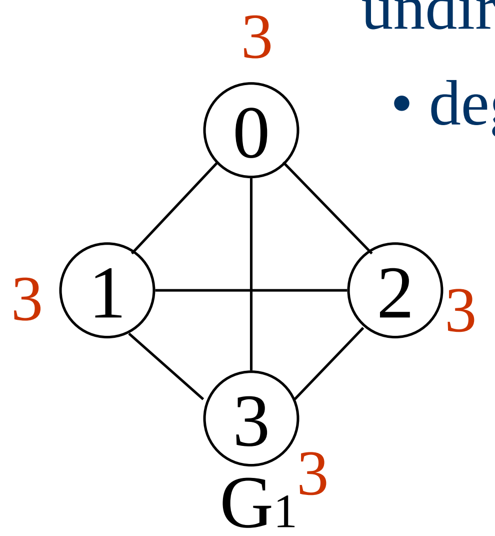
Degree

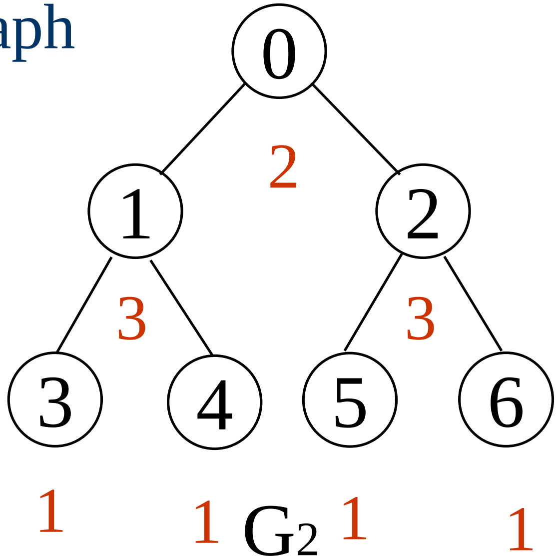
- The of a vertex is the number of edges incident to that vertex
- For directed graph,
  - the of a vertex *v* is the number of edges that have *v* as the head
  - the of a vertex *v* is the number of edges that have *v* as the tail
  - if d_i is the degree of a vertex *i* in a graph *G* with *n* vertices and *e* edges, the number of edges is

$$e = \left(\sum_{0}^{n-1} d_i\right) / 2$$

undirected graph

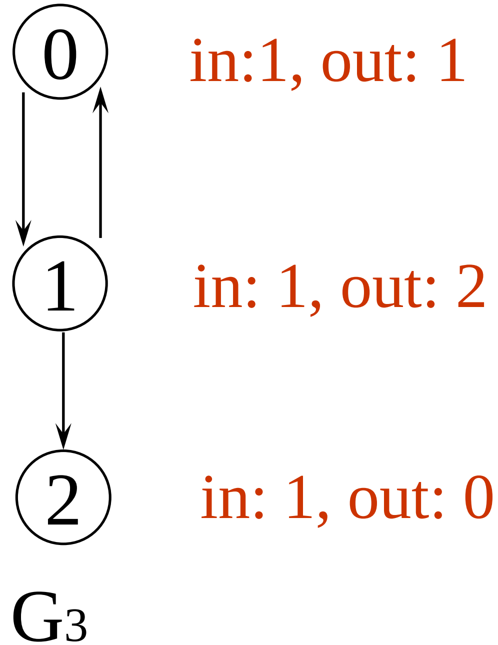
- degree





directed graph

- in-degree
- out-degree

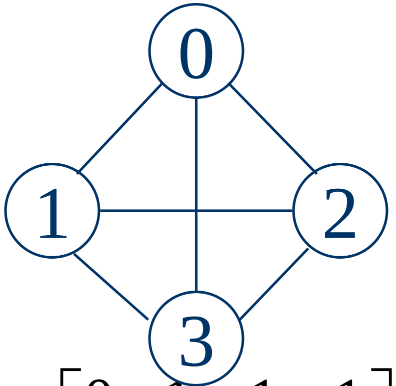


Graph Representations

Adjacency Matrix

- Let G=(V,E) be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, say adj_mat
- If the edge (v_i, v_j) is in E(G), adj_mat[i][j]=1
- If there is no such edge in E(G), adj_mat[i][j]=0
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a digraph need not be symmetric

Examples for Adjacency Matrix

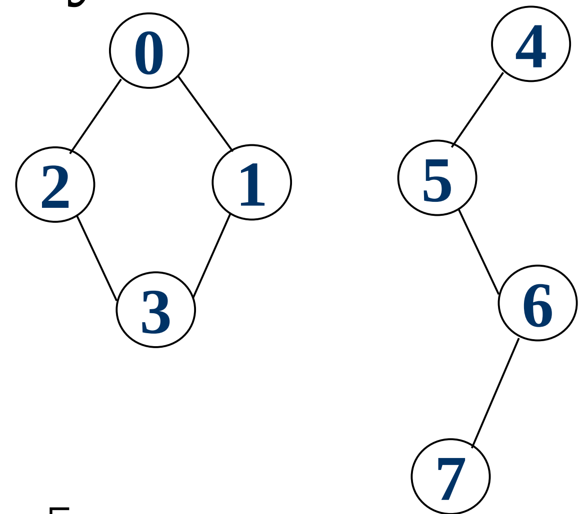


$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

G_1

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

G_2



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

G_4

symmetric

undirected: $n^2/2$
directed: n^2

Merits of Adjacency Matrix

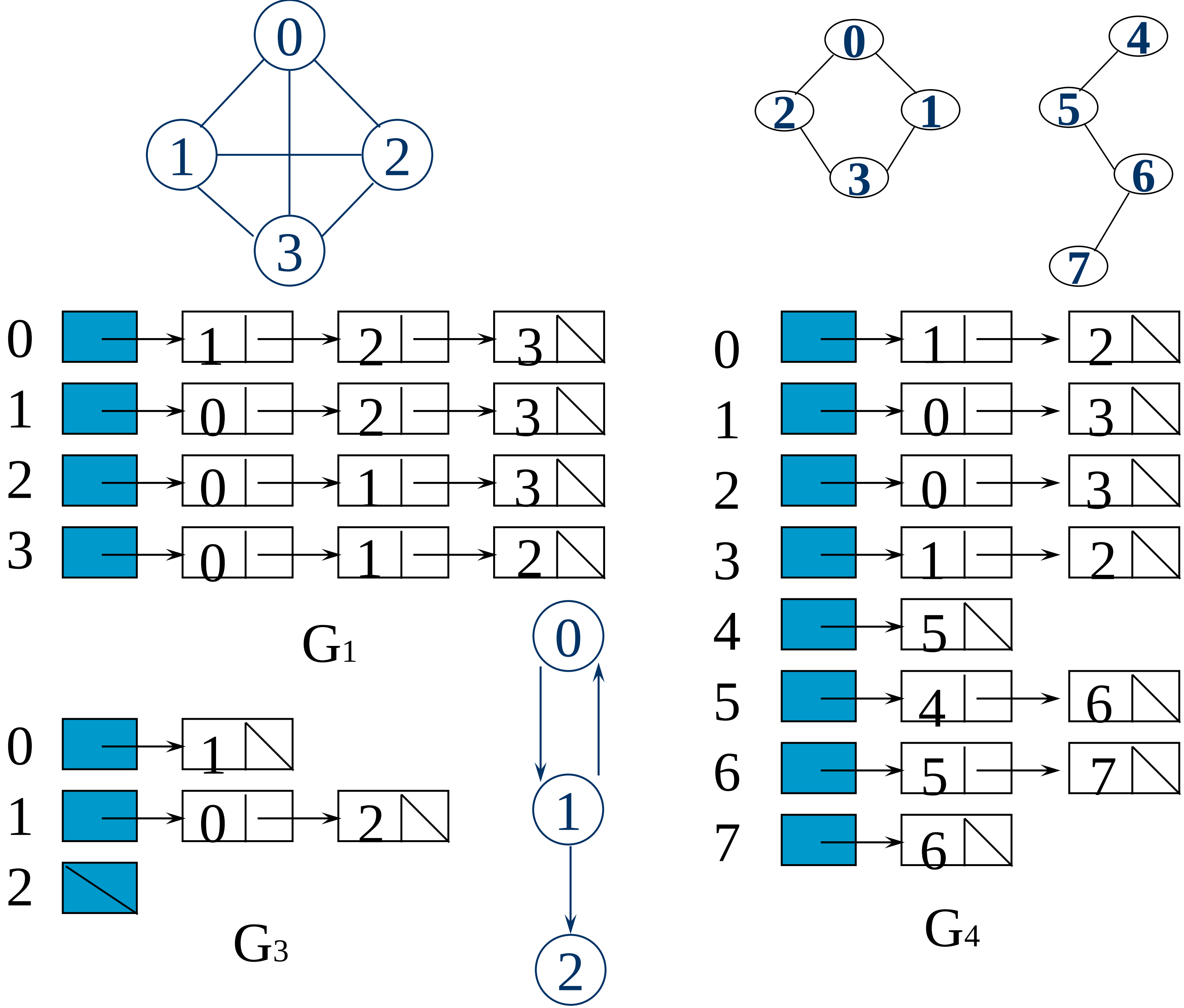
- From the adjacency matrix, to determine the connection of vertices is easy
- The degree of a vertex is $\sum_{j=0}^{n-1} adj\_mat[i][j]$
- For a digraph, the row sum is the out_degree, while the column sum is the in_degree

$$ind(vi) = \sum_{j=0}^{n-1} A[j,i] \quad outd(vi) = \sum_{j=0}^{n-1} A[i,j]$$

Data Structures for Adjacency Lists

Each row in adjacency matrix is represented as an adjacency list.

```
#define MAX_VERTICES 50
typedef struct node *node_pointer;
typedef struct node {
    int vertex;
    struct node *link;
};
node_pointer graph[MAX_VERTICES];
int n=0; /* vertices currently in use */
```

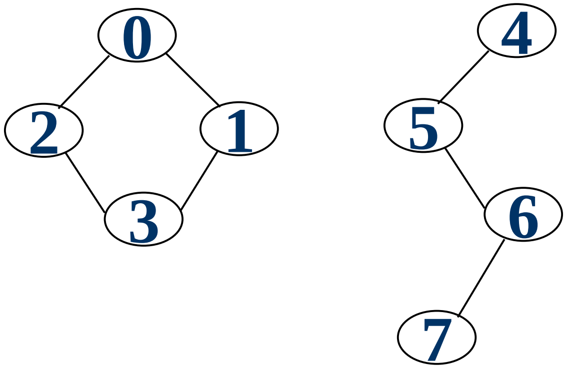


An undirected graph with n vertices and e edges ==> n head nodes and 2e list nodes

Interesting Operations

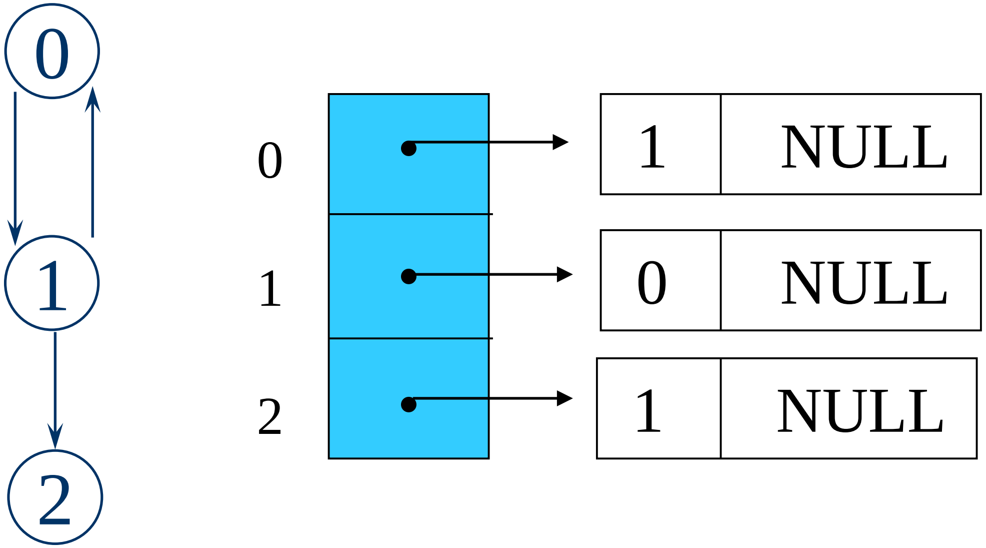
- degree of a vertex in an undirected graph
  - # of nodes in adjacency list
- # of edges in a graph
  - determined in O(n+e)
- out-degree of a vertex in a directed graph
  - # of nodes in its adjacency list
- in-degree of a vertex in a directed graph
  - traverse the whole data structure

Compact Representation



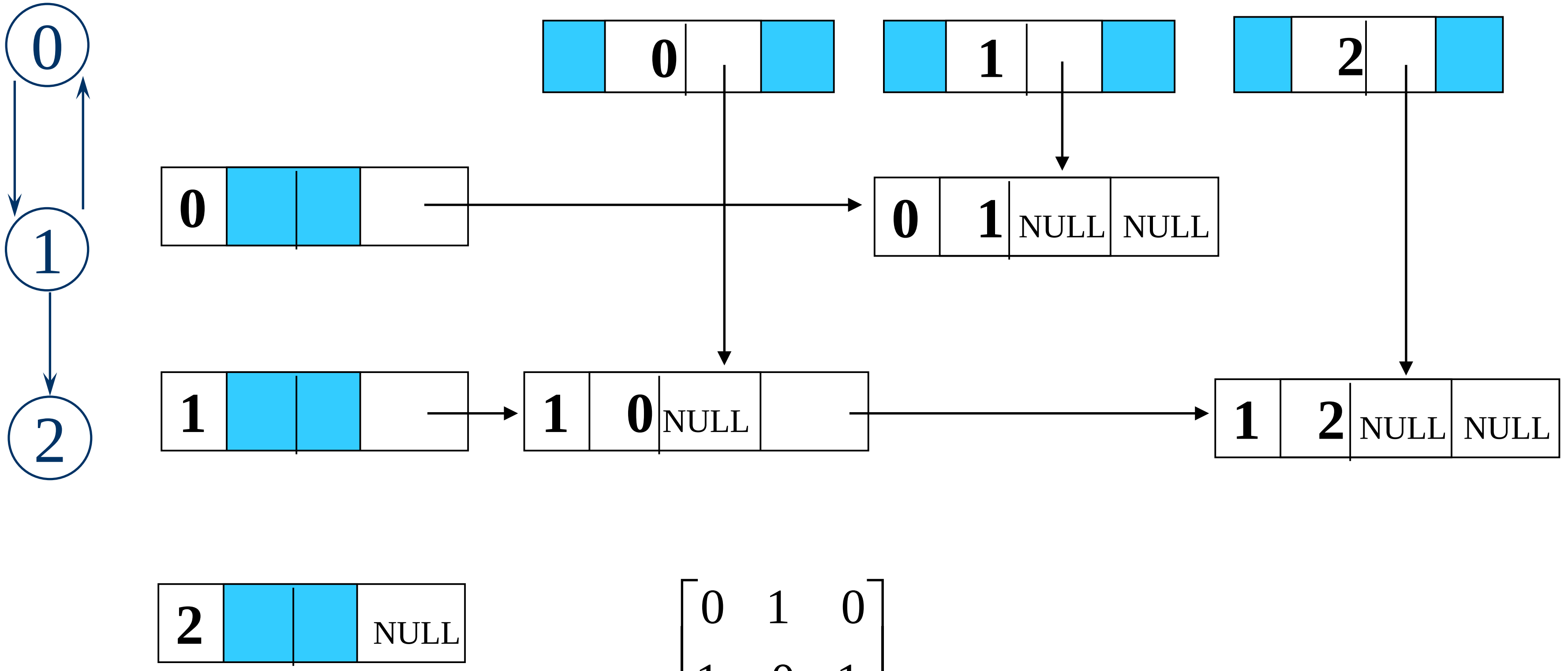
node[0] … node[n-1]: starting point for vertices
node[n]: n+2e+1
node[n+1] … node[n+2e]: head node of edge

[0]	9		[8]	23		[16]	2
[1]	11	0	[9]	1	4	[17]	5
[2]	13		[10]	2	5	[18]	4
[3]	15	1	[11]	0		[19]	6
[4]	17		[12]	3	6	[20]	5
[5]	18	2	[13]	0		[21]	7
[6]	20		[14]	3	7	[22]	6
[7]	22	3	[15]	1			



Determine in-degree of a vertex in a fast way.

Figure 6.10: Inverse adjacency list for G_3



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Figure 6.11: Orthogonal representation for graph G_3(p.276)

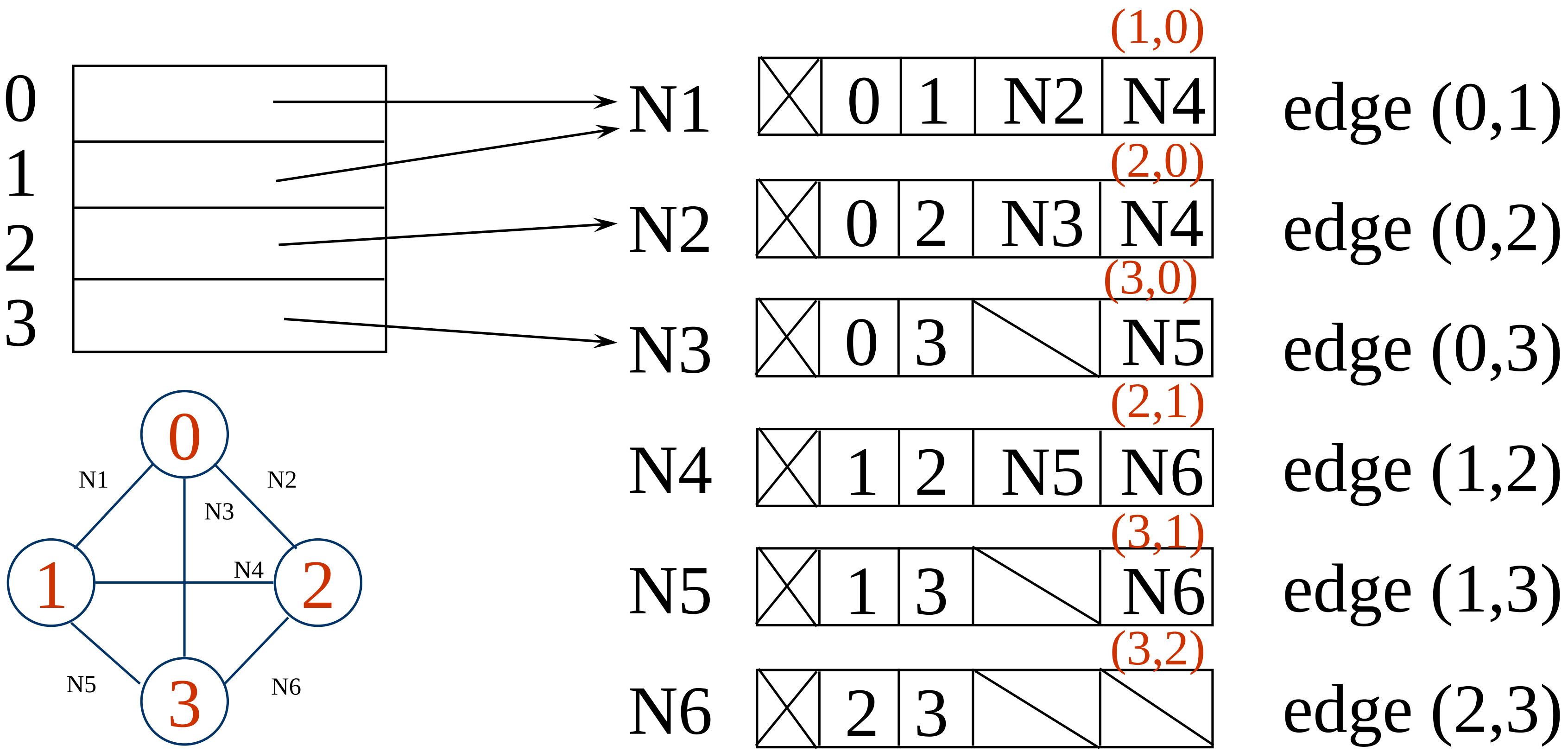
Adjacency Multilists

- An edge in an undirected graph is represented by two nodes in adjacency list representation.
- Adjacency Multilists
  - –lists in which nodes may be shared among several lists.
    (an edge is shared by two different paths)

marked	vertex1	vertex2	path1	path2

Example for Adjacency Multlists

Lists: vertex 0: M1->M2->M3, vertex 1: M1->M4->M5
vertex 2: M2->M4->M6, vertex 3: M3->M5->M6



six edges

Adjacency Multilists

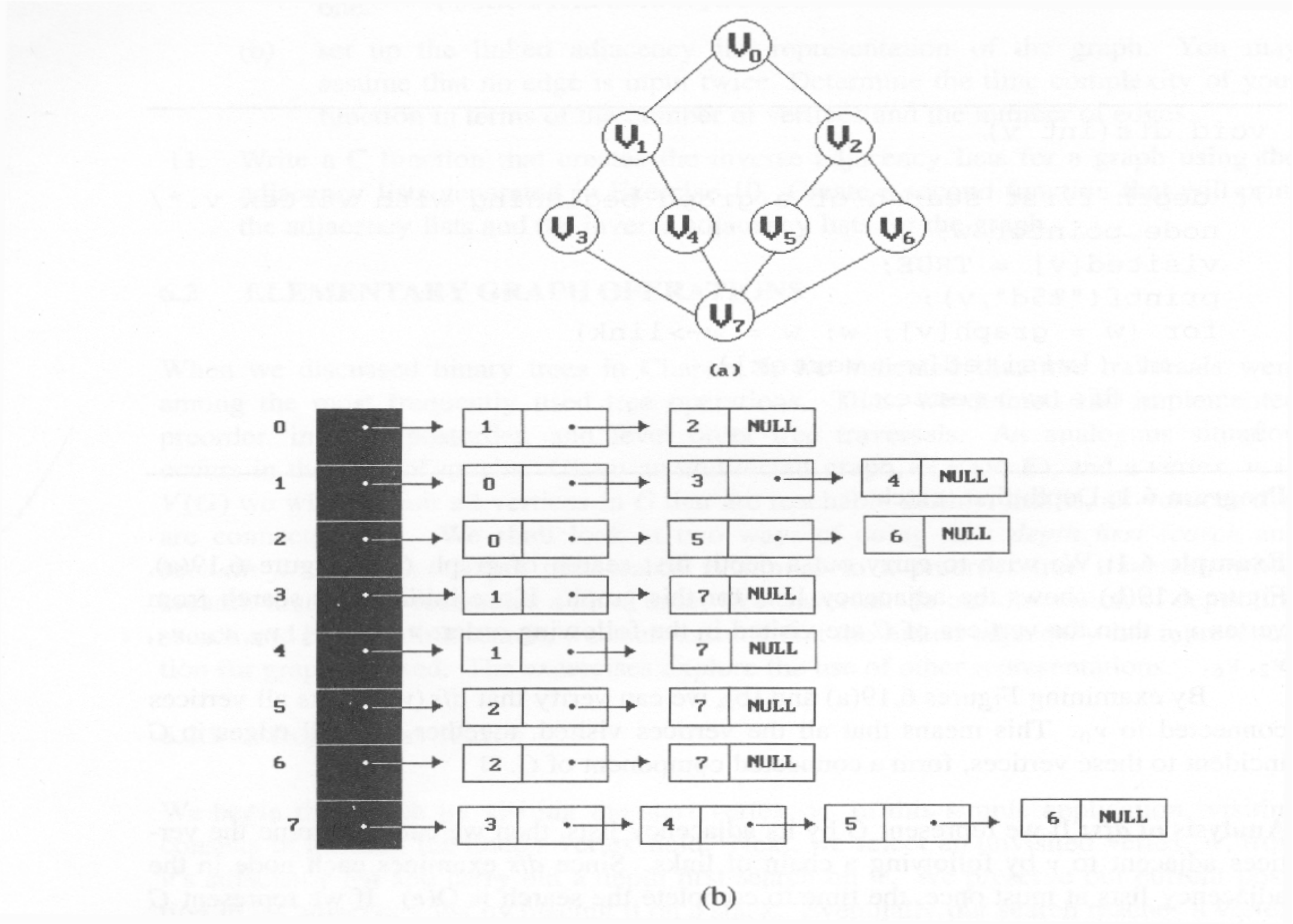
```
typedef struct edge *edge_pointer;
typedef struct edge {
    short int marked;
    int vertex1, vertex2;
    edge_pointer path1, path2;
};
edge_pointer graph[MAX_VERTICES];
```

marked	vertex1	vertex2	path1	path2

Some Graph Operations

- Traversal
  Given $G=(V,E)$ and vertex v, find all $w \in V$, such that w connects v.
  - preorder tree traversal
  - level order tree traversal
- Spanning Trees

depth first search: v0, v1, v3, v7, v4, v5, v2, v6



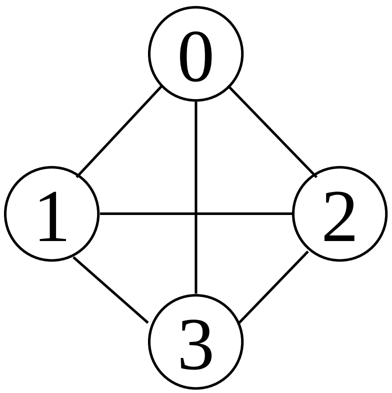
breadth first search: v0, v1, v2, v3, v4, v5, v6, v7

Figure 6.16: Graph *G* and its adjacency lists (p.281)

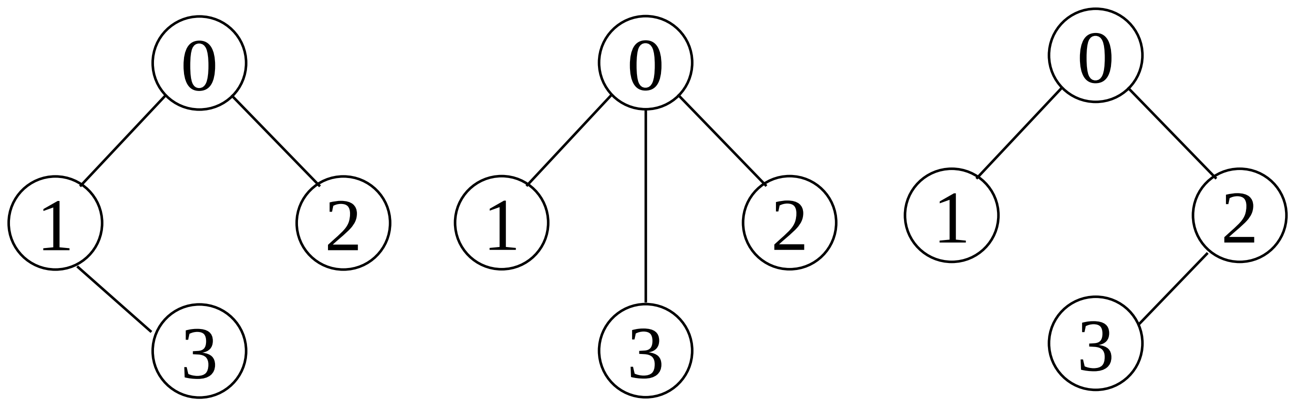
Spanning Trees

- When graph G is connected, a depth first or breadth first search starting at any vertex will visit all vertices in G
- A is any tree that consists solely of edges in G and that includes all the vertices
- E(G): T (tree edges) + N (nontree edges)
  where T: set of edges used during search
  N: set of remaining edges

Examples of Spanning Tree



G_1

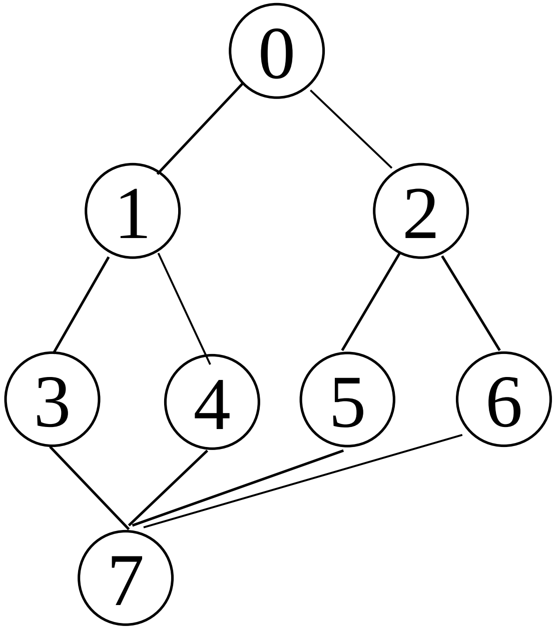


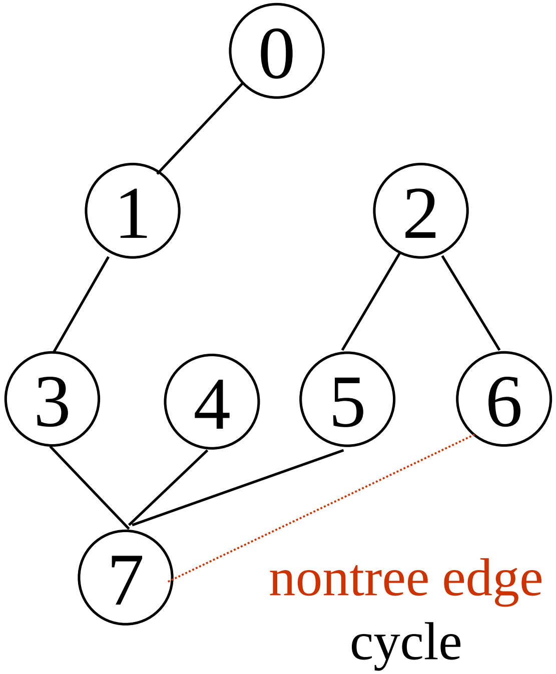
Possible spanning trees

Spanning Trees

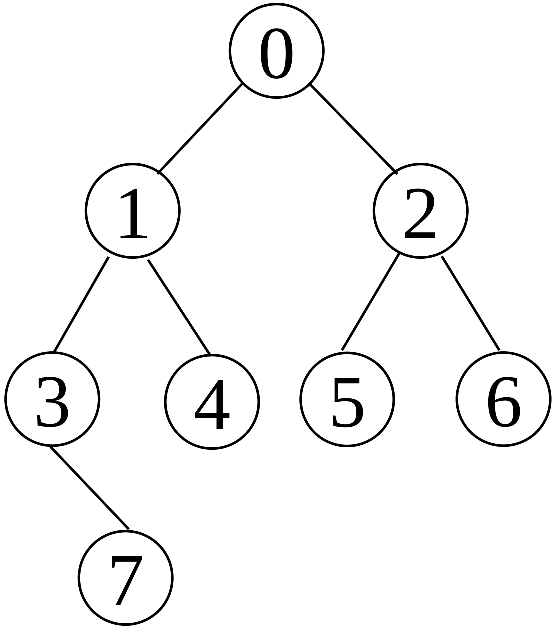
- Either dfs or bfs can be used to create a spanning tree
  - When dfs is used, the resulting spanning tree is known as a
  - When bfs is used, the resulting spanning tree is known as
- While adding a nontree edge into any spanning tree, this will create a cycle

DFS VS BFS Spanning Tree





DFS Spanning



BFS Spanning

Minimum Cost Spanning Tree

- The cost of a spanning tree of a weighted undirected graph is the sum of the costs of the edges in the spanning tree
- A minimum cost spanning tree is a spanning tree of least cost
- Three different algorithms can be used

Select n-1 edges from a weighted graph of n vertices with minimum cost.

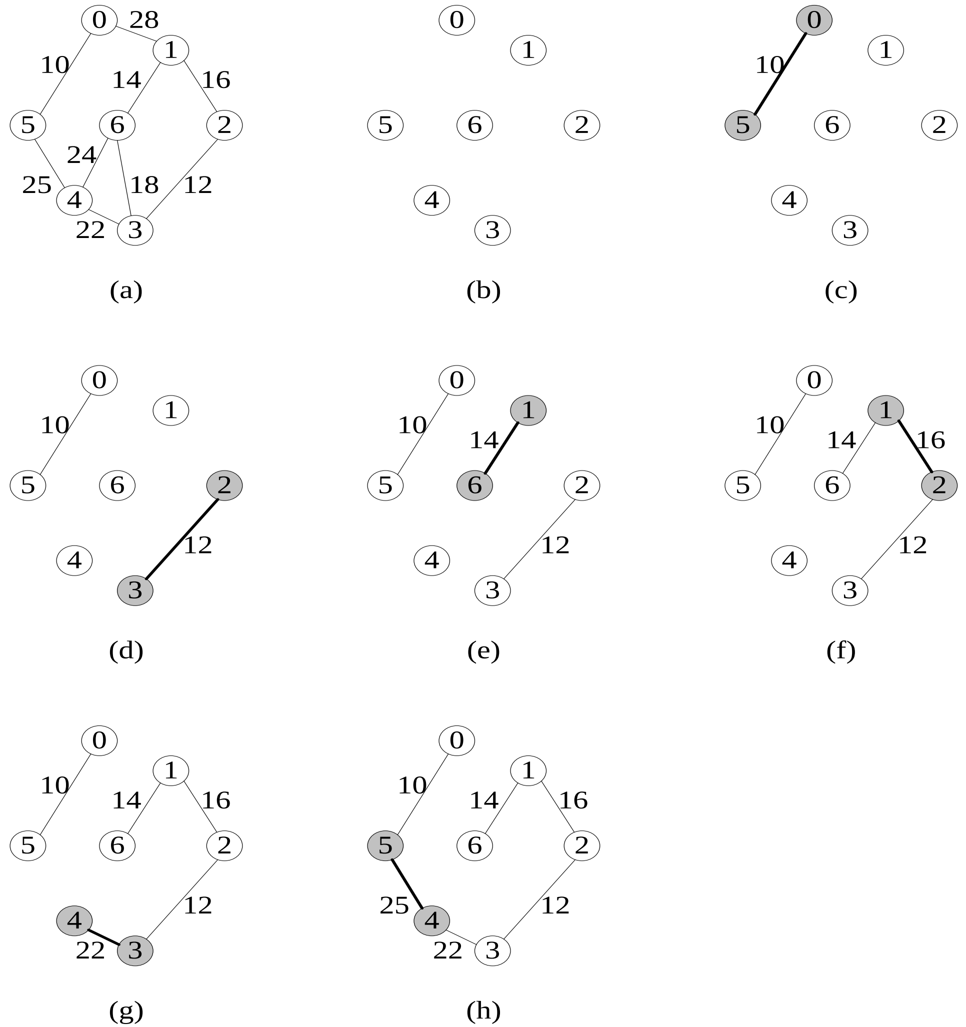
Greedy Strategy

- An optimal solution is constructed in stages
- At each stage, the best decision is made at this time
- Since this decision cannot be changed later, we make sure that the decision will result in a feasible solution
- Typically, the selection of an item at each stage is based on a least cost or a highest profit criterion

Kruskal's Idea

- Build a minimum cost spanning tree T by adding edges to T one at a time
- Select the edges for inclusion in T in nondecreasing order of the cost
- An edge is added to T if it does not form a cycle
- Since G is connected and has $n > 0$ vertices, exactly $n-1$ edges will be selected

Examples for Kruskal’s Algorithm

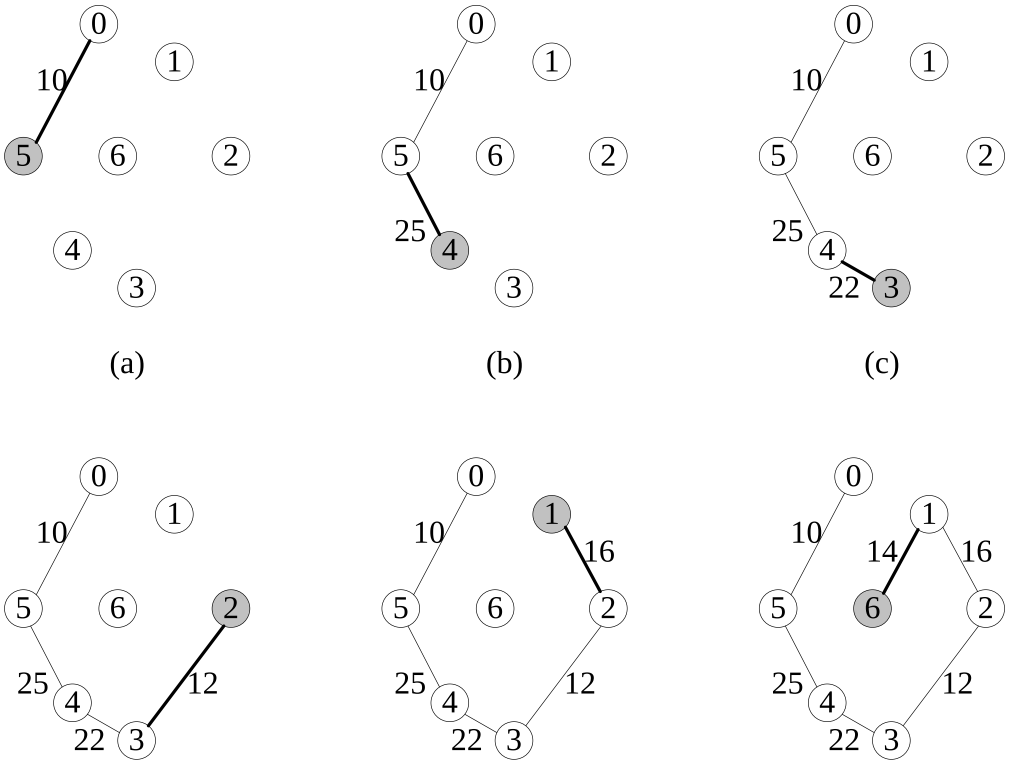


(a) (b) (c)

(d) (e) (f)

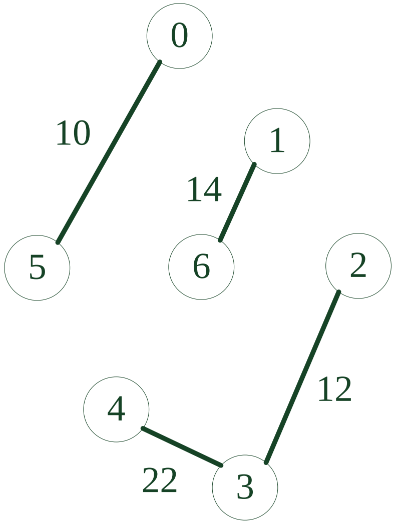
(g) (h)

Examples for Prim's Algorithm

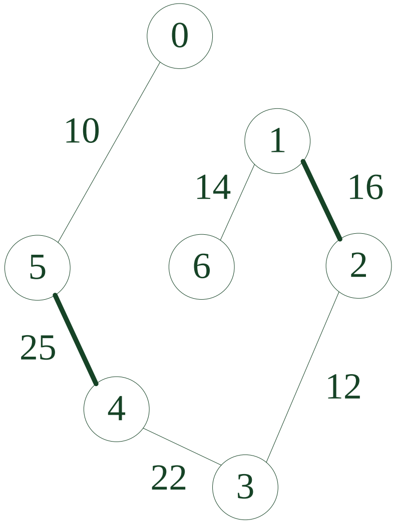


(a) (b) (c) (d) (e) (f)

Sollin's Algorithm



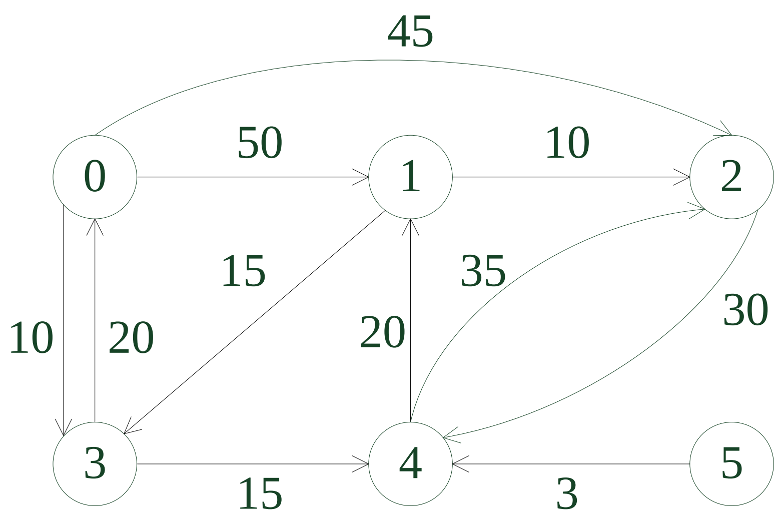
(a)



(b)

Single Source All Destinations

Determine the shortest paths from v0 to all the remaining vertices.



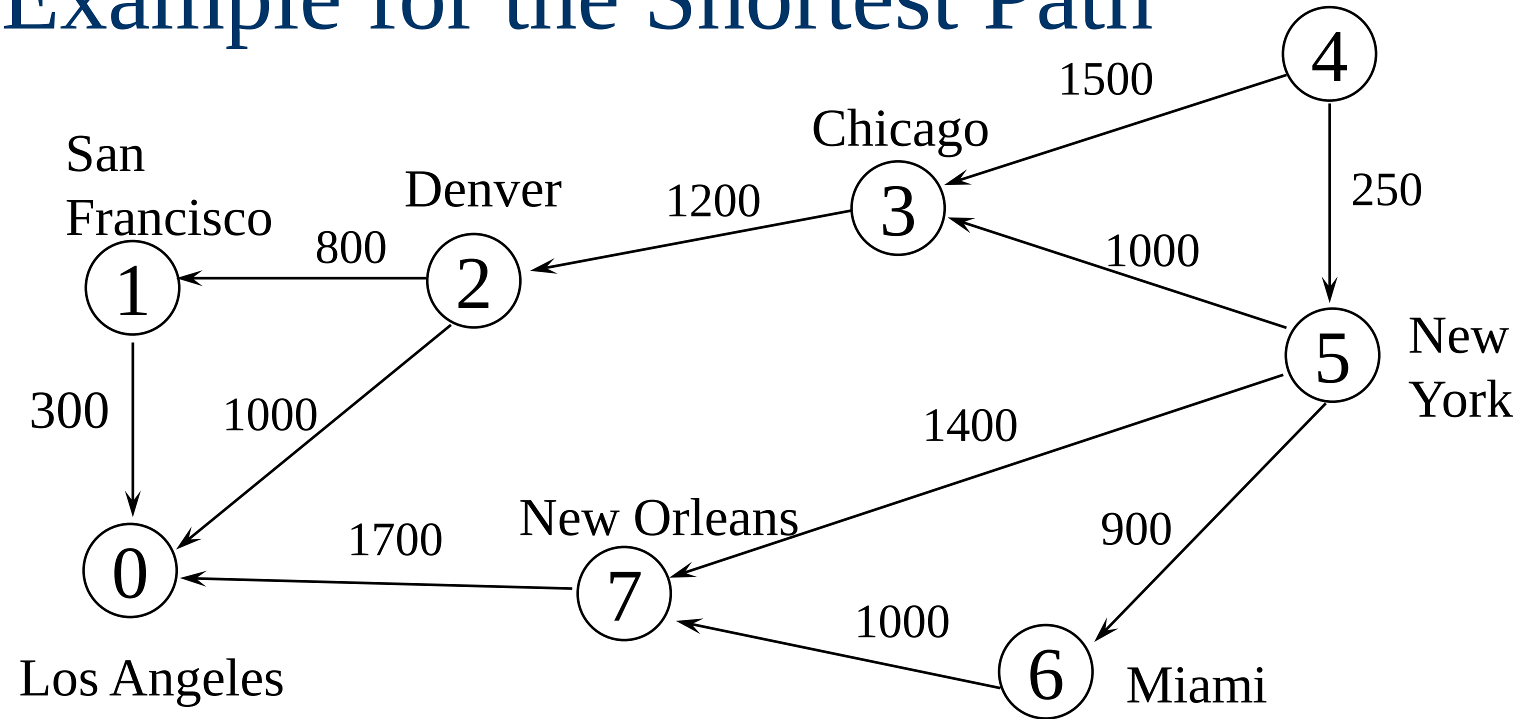
(a)

path	length
1) v0 v2	10
2) v0 v2 v3	25
3) v0 v2 v3 v1	45
4) v0 v4	45

(b)

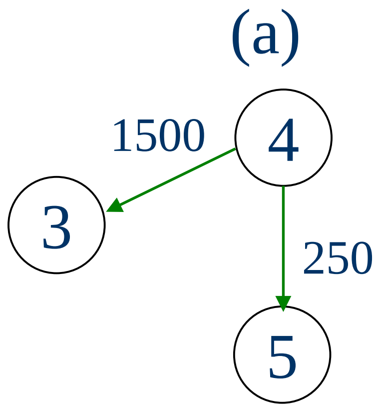
Figure 6.26: Graph and shortest paths from v_0 (p. 300)

Example for the Shortest Path

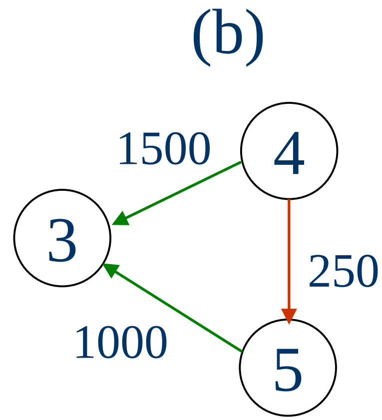


$$
\begin{array}{c|cccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & & & & & & & \\
1 & 300 & 0 & & & & & & \\
2 & 1000 & 800 & 0 & & & & & \\
3 & & & 1200 & 0 & & & & \\
4 & & & & 1500 & 0 & 250 & & \\
5 & & & & 1000 & & 0 & 900 & 1400 \\
6 & & & & & & & 0 & 1000 \\
7 & 1700 & & & & & & & 0
\end{array}
$$

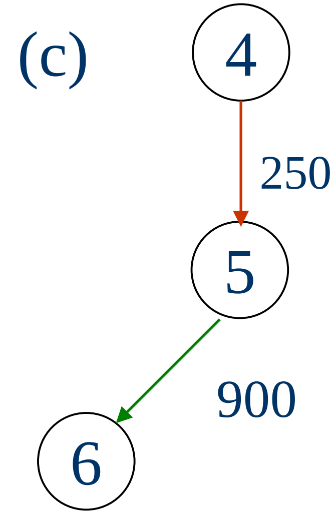
Cost adjacency matrix



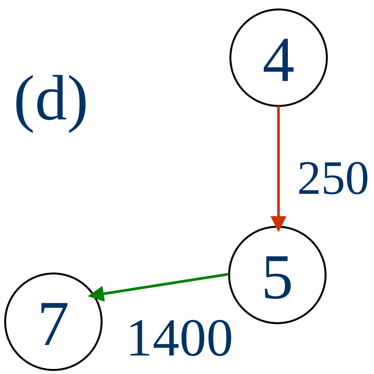
選5



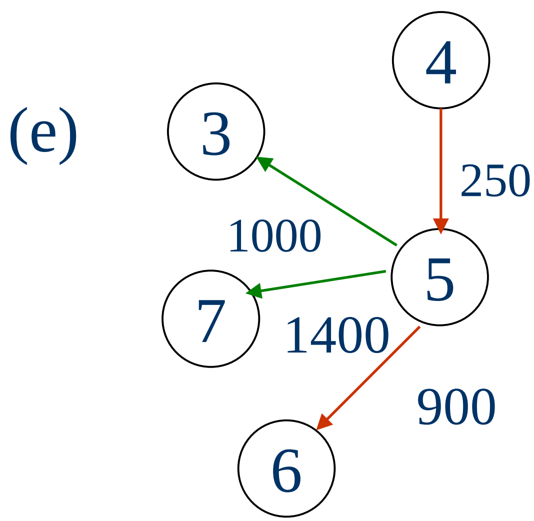
4到3由1500改成1250



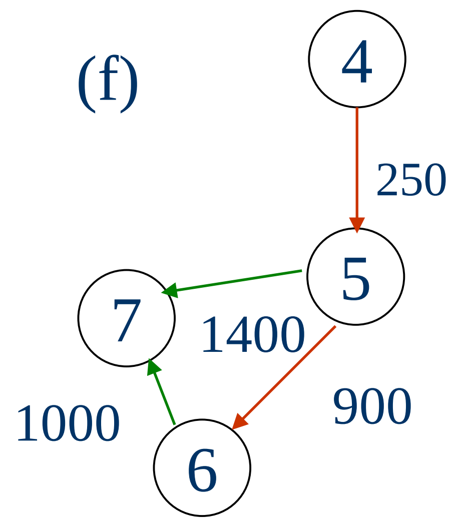
4到6由∞改成1150



4到7由∞改成1650

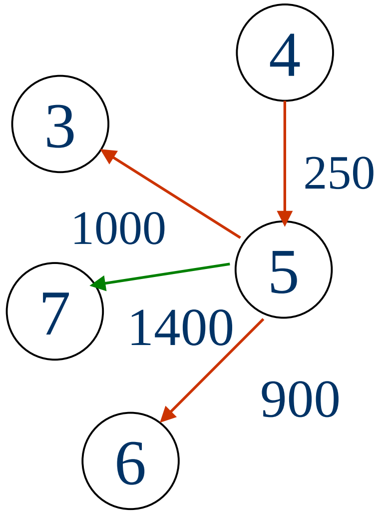


選6



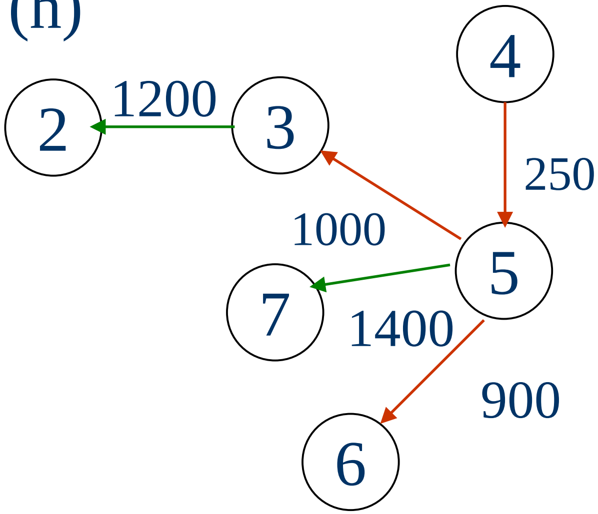
4-5-6-7比4-5-7長

(g)



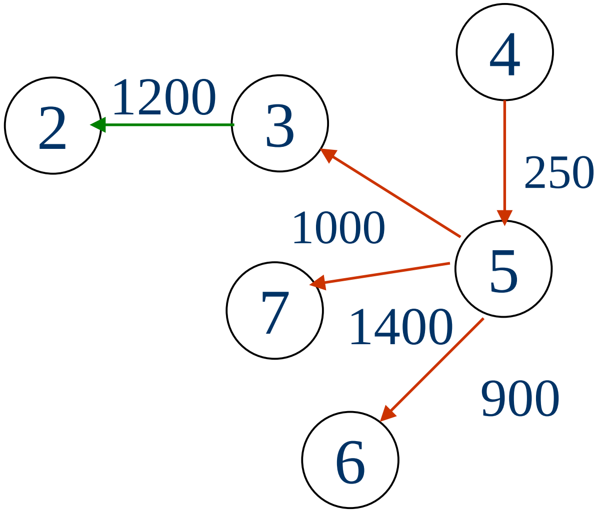
選3

(h)



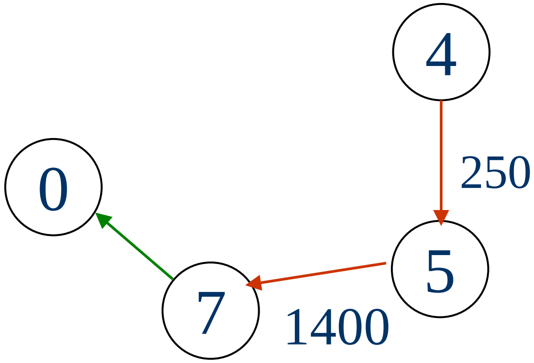
4到2由∞改成2450

(i)

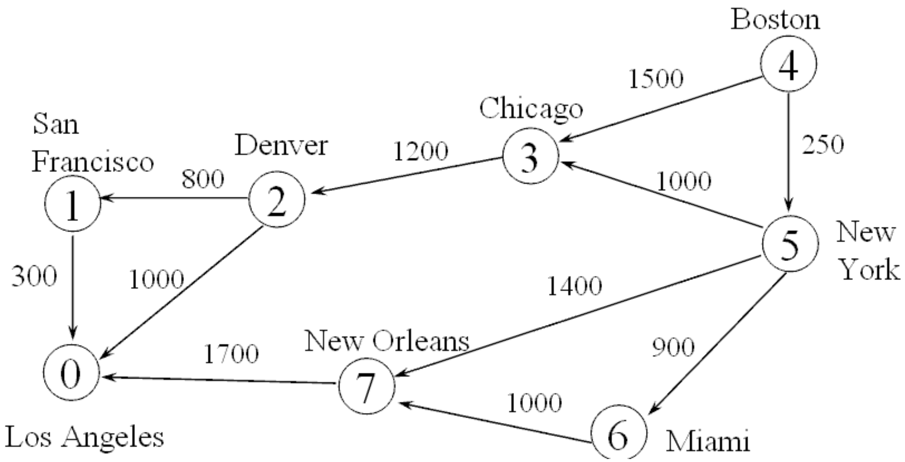


選7

(j)



4到0由∞改成3350

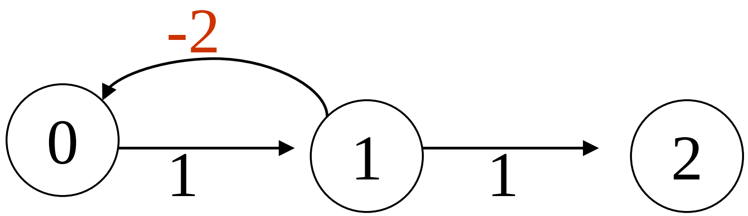


Iteration	S	Vertex Selected	LA [0]	SF [1]	DEN [2]	CHI [3]	BO [4]	NY [5]	MIA [6]	NO
Initial	--	----	$+\infty$	$+\infty$	$+\infty$ (b)	1500	0	250 (c)	$+\infty$ (d)	$+\infty$
1	{4} (a)	5	$+\infty$	$+\infty$	$+\infty$	1250	0	250	1150 (f)	1650
2	{4,5} (e)	6	$+\infty$	$+\infty$	(h) $+\infty$	1250	0	250	1150	1650
3	{4,5,6} (g)	3	(j) $+\infty$	$+\infty$	2450	1250	0	250	1150	1650
4	{4,5,6,3} (i)	7	3350	$+\infty$	2450	1250	0	250	1150	1650
5	{4,5,6,3,7}	2	3350	3250	2450	1250	0	250	1150	1650
6	{4,5,6,3,7,2}	1	3350	3250	2450	1250	0	250	1150	1650
7	{4,5,6,3,7,2,1}									

All Pairs Shortest Paths (*Continued*)

- The cost of the shortest path from i to j is $A^{n-1}[i][j]$, as no vertex in G has an index greater than n-1
- $A^{-1}[i][j]=cost[i][j]$
- Calculate the A^0, A^1, A^2, ..., A^{n-1} from A^{-1} iteratively
- $A^k[i][j]=\min\{A^{k-1}[i][j], A^{k-1}[i][k]+A^{k-1}[k][j]\}$, k>=0

Graph with negative cycle



$$\begin{bmatrix} 0 & 1 & \infty \\ -2 & 0 & 1 \\ \infty & \infty & 0 \end{bmatrix}$$

(a) Directed graph

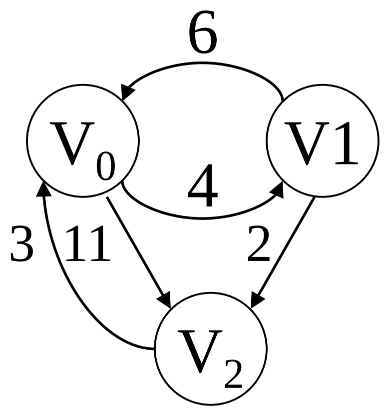
(b) A^{-1}

The length of the shortest path from vertex 0 to vertex 2 is -∝.

0, 1, 0, 1,0, 1, …, 0, 1, 2

Algorithm for All Pairs Shortest Paths

```
void allcosts(int cost[][MAX_VERTICES],
         int distance[][MAX_VERTICES], int n)
{
  int i, j, k;
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
         distance[i][j] = cost[i][j];
  for (k=0; k<n; k++)
    for (i=0; i<n; i++)
      for (j=0; j<n; j++)
        if (distance[i][k]+distance[k][j]
             < distance[i][j])
            distance[i][j]=
                distance[i][k]+distance[k][j];
}
```

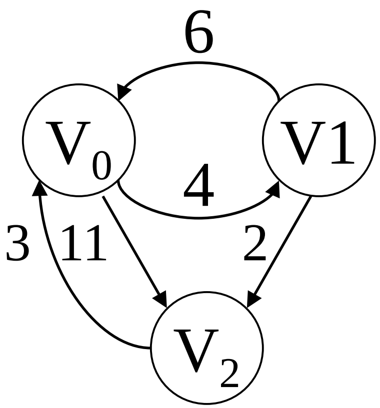


(a)Digraph G

	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

(b)Cost adjacency matrix for G

Figure 6.33: Directed graph and its cost matrix (p.310)



A^{-1}

	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

A^{0}

	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

A^{1}

	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

A^{2}

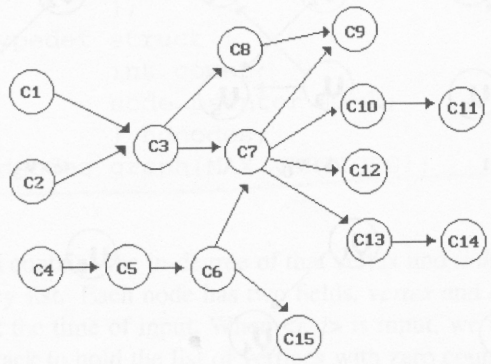
	0	1	2
0	0	4	6
1	5	0	2
2	3	7	0

Activity on Vertex (AOV) Network

- definition
  A directed graph in which the vertices represent tasks or activities and the edges represent precedence relations between tasks.
- predecessor (successor)
  vertex i is a predecessor of vertex j iff there is a directed path from i to j. j is a successor of i.
- partial order
  a precedence relation which is both transitive $(\forall i, j, k, i \bullet j \ \& \ j \bullet k \Rightarrow i \bullet k)$ and irreflexive $(\forall x \neg x \bullet x)$.
- acylic graph
  a directed graph with no directed cycles

Course number	Course name	Prerequisites
C1	Programming I	None
C2	Discrete Mathematics	None
C3	Data Structures	C1, C2
C4	Calculus I	None
C5	Calculus II	C4
C6	Linear Algebra	C5
C7	Analysis of Algorithms	C3, C6
C8	Assembly Language	C3
C9	Operating Systems	C7, C8
C10	Programming Languages	C7
C11	Compiler Design	C10
C12	Artificial Intelligence	C7
C13	Computational Theory	C7
C14	Parallel Algorithms	C13
C15	Numerical Analysis	C5

(a) Courses needed for a computer science degree at a hypothetical university



(b) AOV network representing courses as vertices and edges as prerequisites

Topological order:
linear ordering of vertices of a graph
∀i, j if i is a predecessor of j, then i precedes j in the linear ordering

C1, C2, C4, C5, C3, C6, C8, C7, C10, C13, C12, C14, C15, C11, C9

C4, C5, C2, C1, C6, C3, C8, C15, C7, C9, C10, C11, C13, C12, C14

Figure 6.37: An AOV network (p.316)

```
for (i = 0; i <n; i++) {
    if every vertex has a predecessor {
        fprintf(stderr, "Network has a cycle. \n " );
        exit(1);
    }
    pick a vertex v that has no predecessors;
    output v;
    delete v and all edges leading out of v
    from the network;
}
```

Program 6.13: Topological sort (p.318)

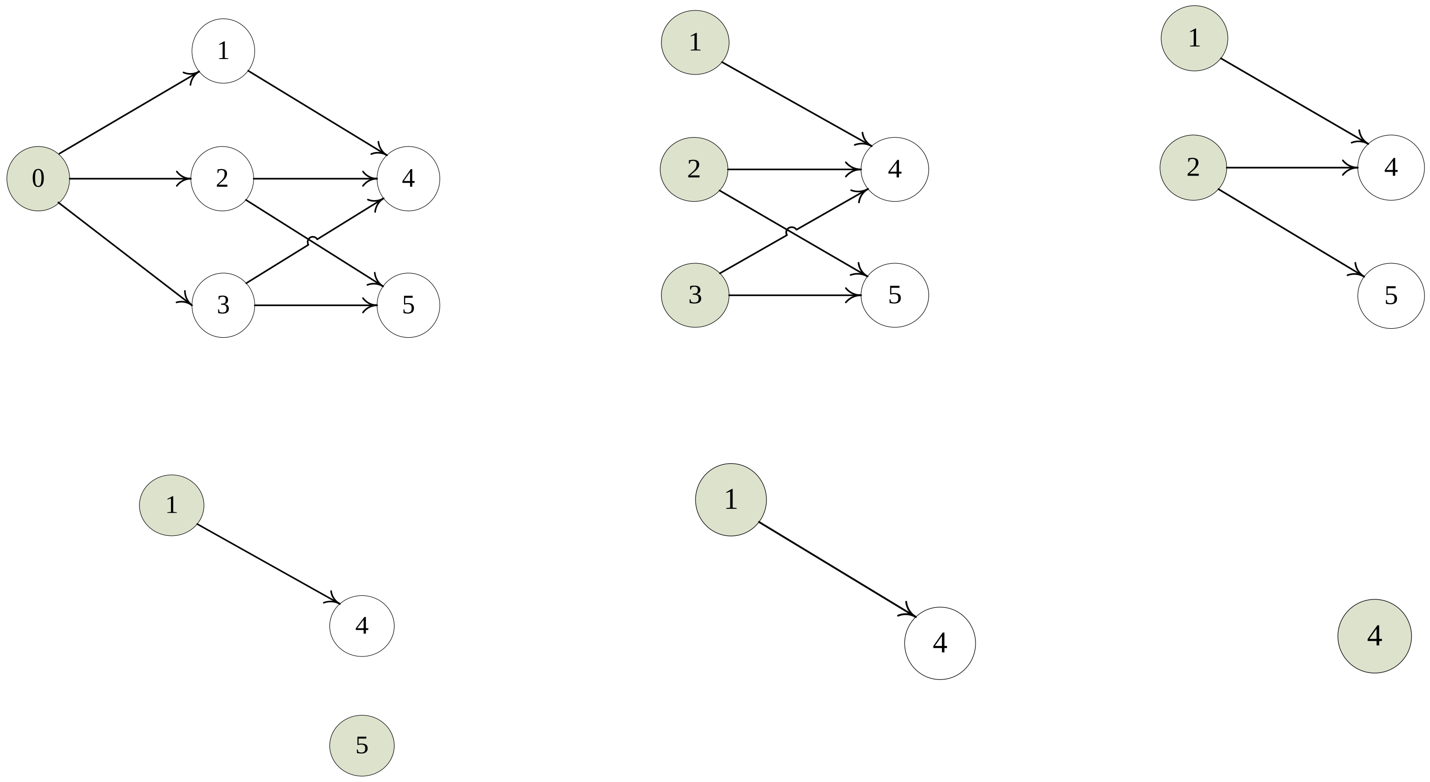


Figure 6.38: Simulation of Program 6.13 on an AOV network (p.318)

Issues in Data Structure Consideration

- Decide whether a vertex has any predecessors.
  - –Each vertex has a count.
- Decide a vertex together with all its incident edges.
  - –Adjacency list

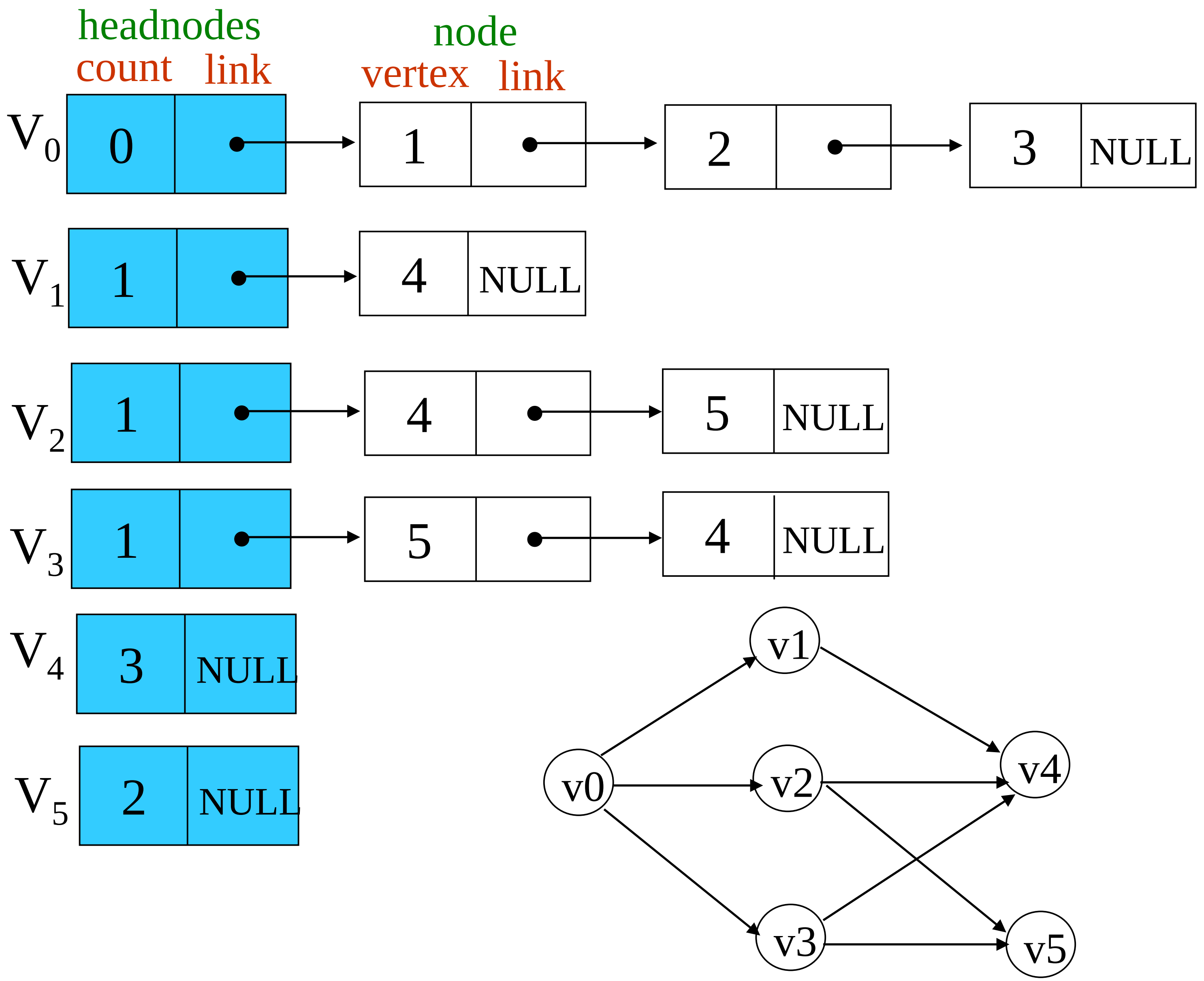


Figure 6.39: Adjacency list representation of Figure 6.30(a) (p.320)

Activity on Edge (AOE) Networks

- directed edge
  - –tasks or activities to be performed
- vertex
  - –events which signal the completion of certain activities
- number
  - –time required to perform the activity

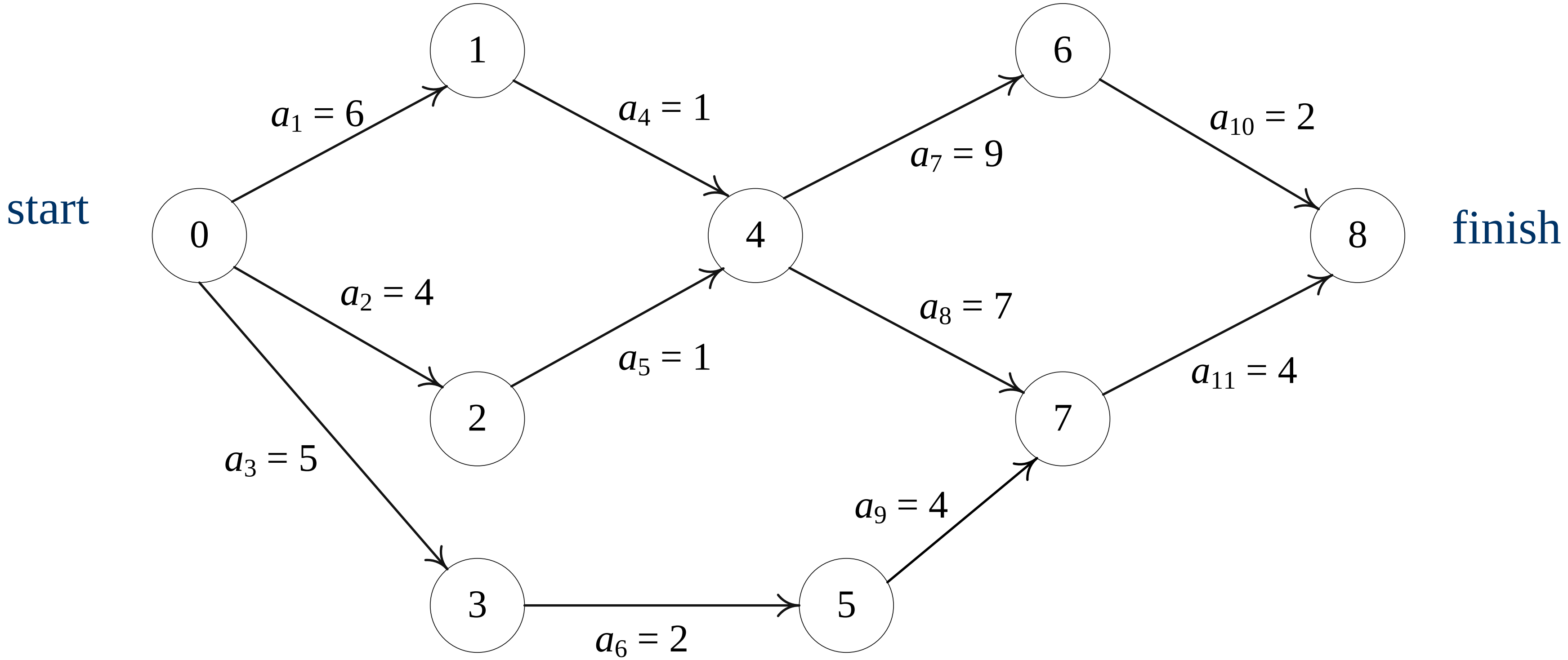


Figure 6.40: An AOE network(p.322)

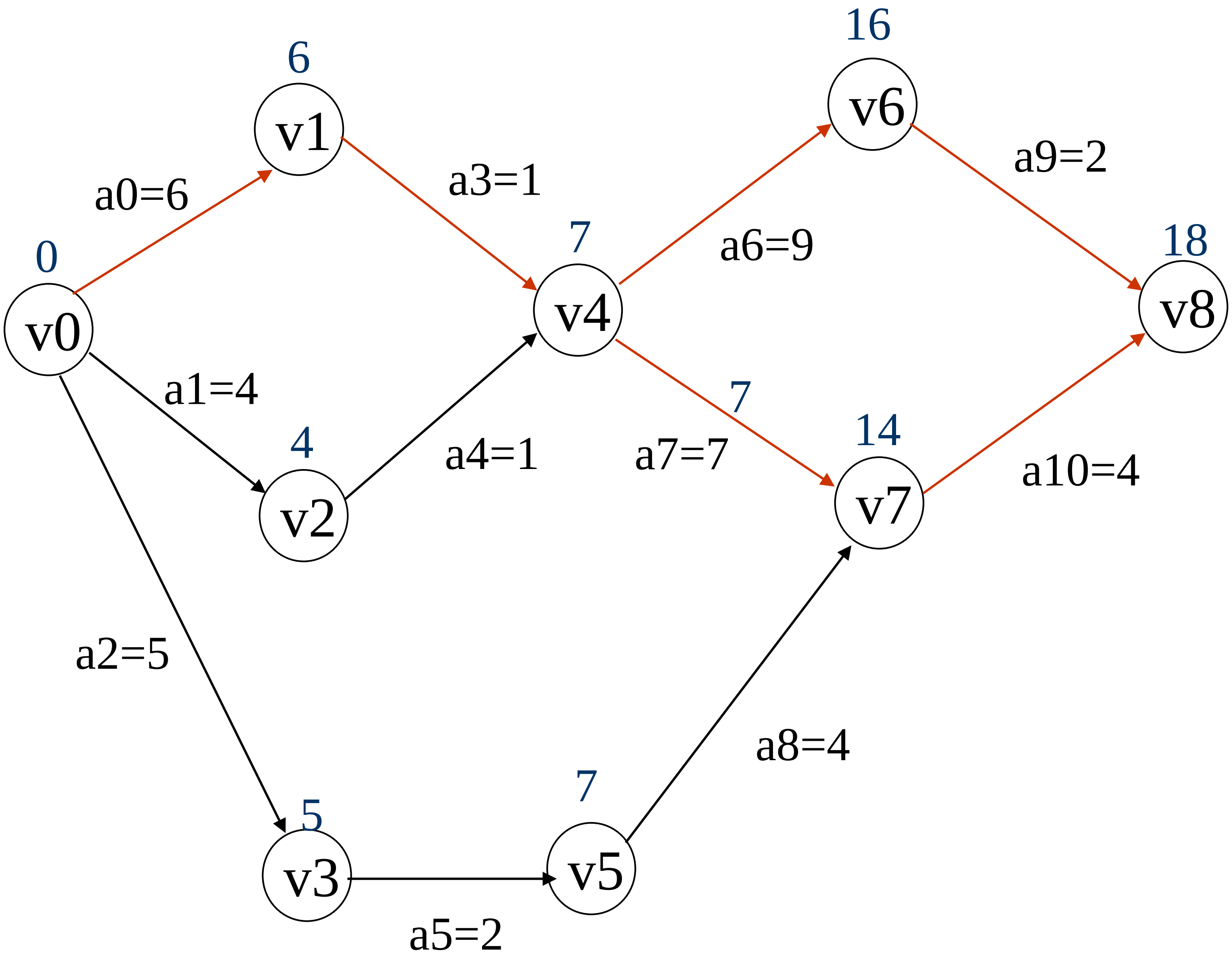
Application of AOE Network

- Evaluate performance
  - –minimum amount of time
  - –activity whose duration time should be shortened
  - –…
- Critical path
  - –a path that has the longest length
  - –minimum time required to complete the project
  - –v0, v1, v4, v7, v8 or v0, v1, v4, v6, v8 (18)

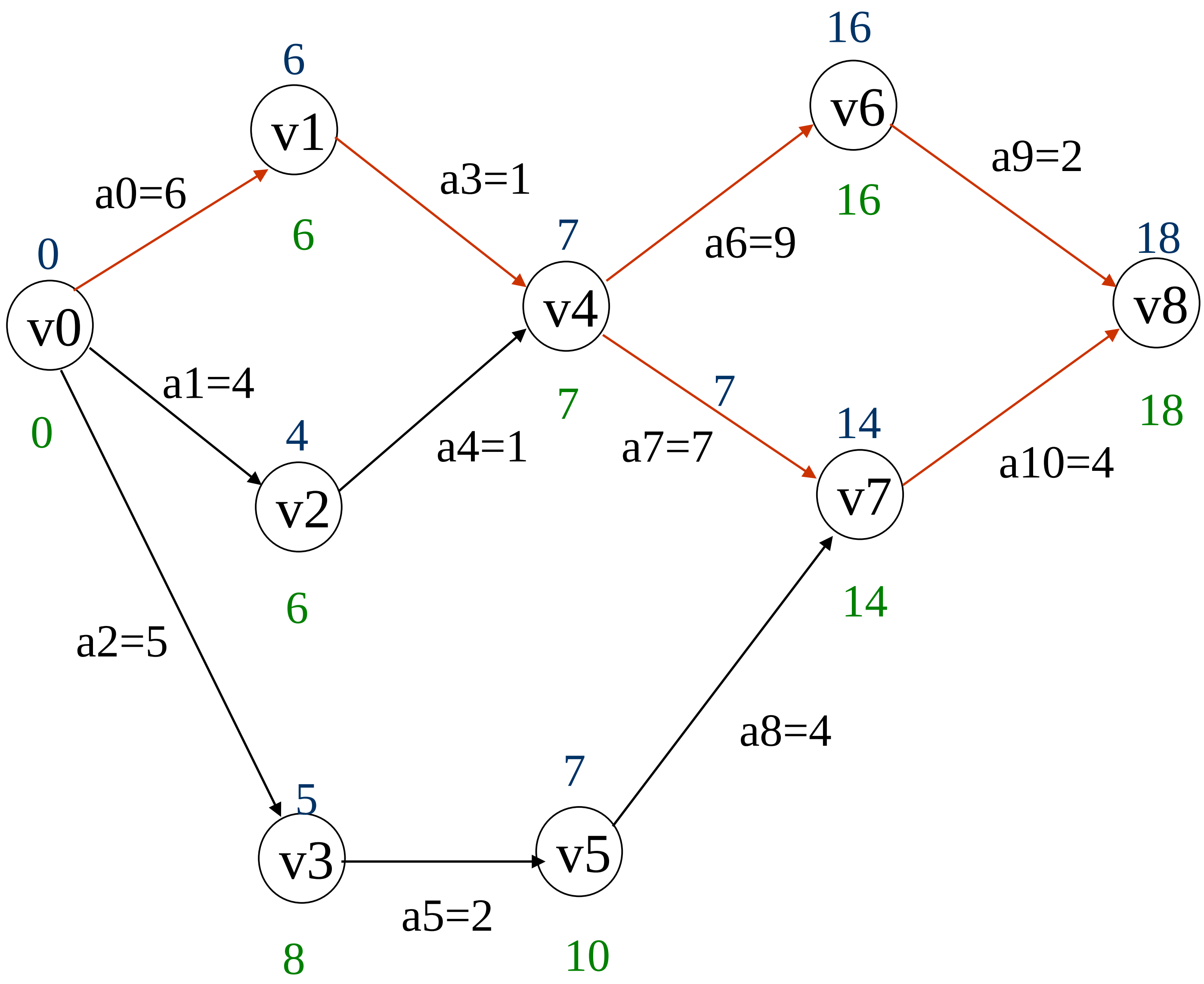
other factors

- Earliest time that vi can occur
  - –the length of the longest path from v0 to vi
  - –the earliest start time for all activities leaving vi
  - –early(6) = early(7) = 7
- Latest time of activity
  - –the latest time the activity may start without increasing the project duration
  - –late(5)=8, late(7)=7
- Critical activity
  - –an activity for which early(i)=late(i)
  - –early(7)=late(7)
- late(i)-early(i)
  - –measure of how critical an activity is
  - –late(5)-early(5)=8-5=3

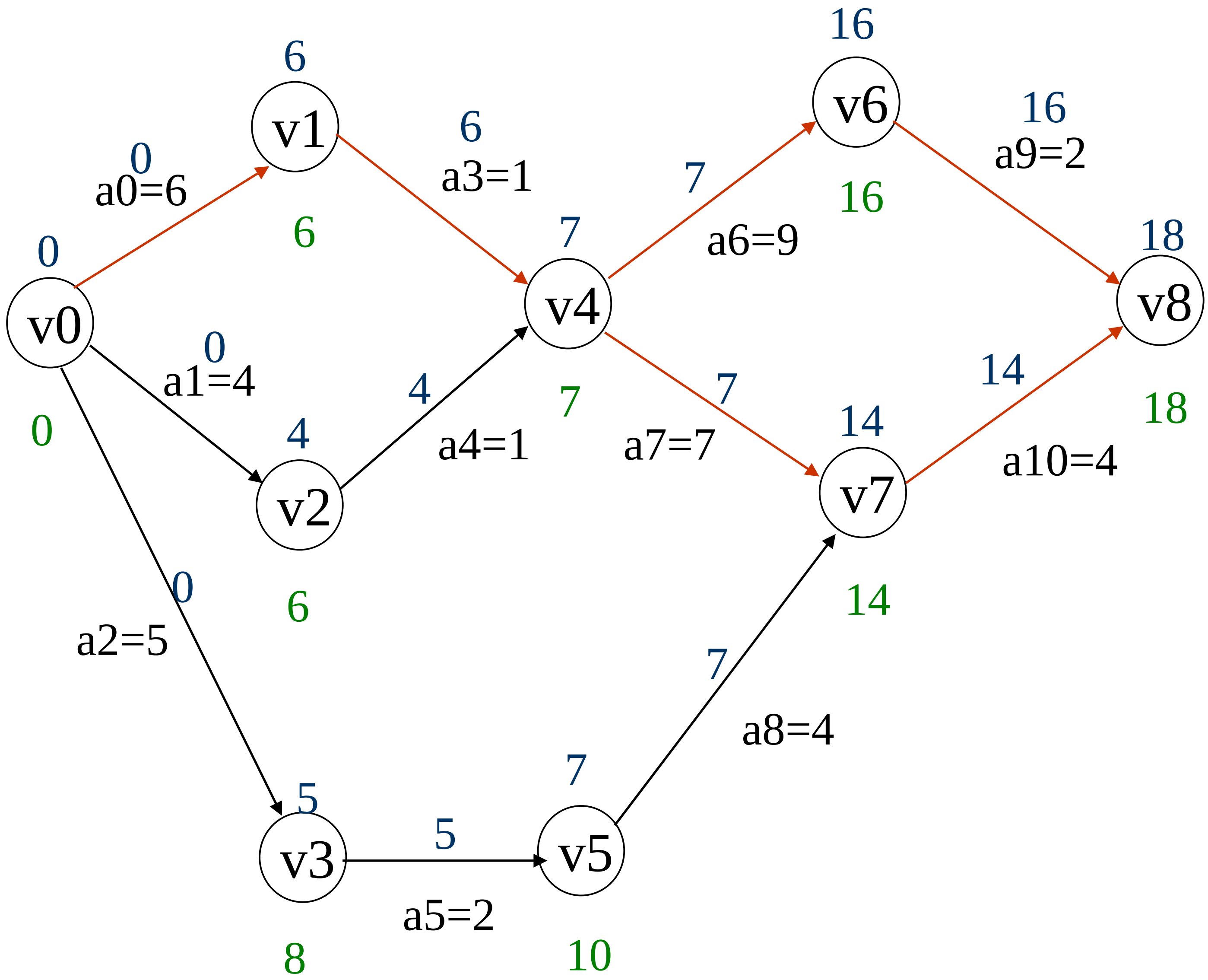
earliest, early



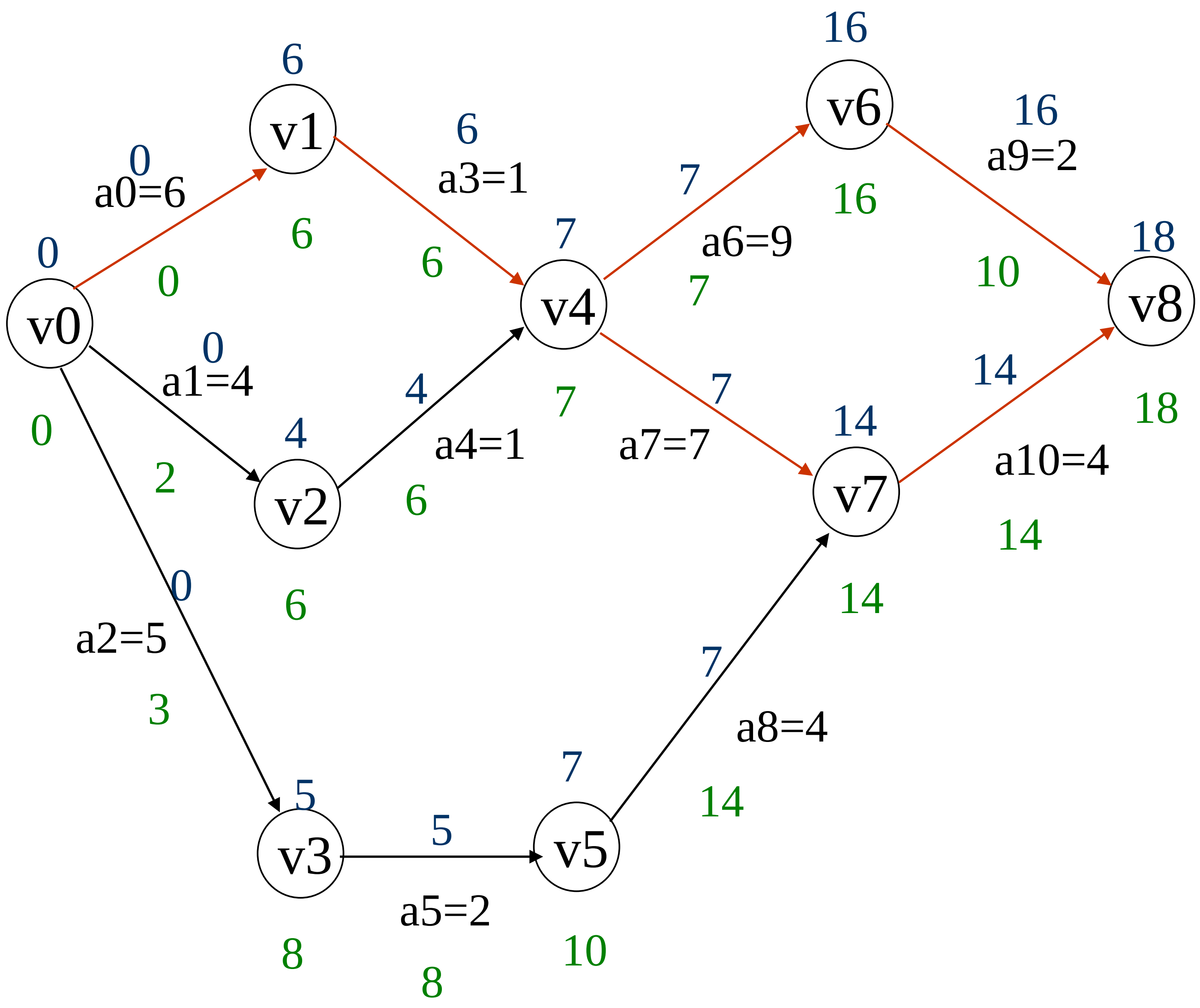
earliest, early, latest, late



earliest, early, latest, late



earliest, early, latest, late



Determine Critical Paths

- Delete all noncritical activities
- Generate all the paths from the start to finish vertex.

Calculation of Earliest Times

- earliest[j]
  - –the earliest event occurrence time

$$earliest[0]=0$$

$$earliest[j]=\max_{i \in p(j)}\{earliest[i]+\text{duration of } \langle i,j \rangle\}$$

- latest[j]
  - –the latest event occurrence time

$$v_k \xrightarrow{a_i} v_l$$

$$early(i)=earliest(k)$$

$$late(i)=latest(l)-\text{duration of } a_i$$

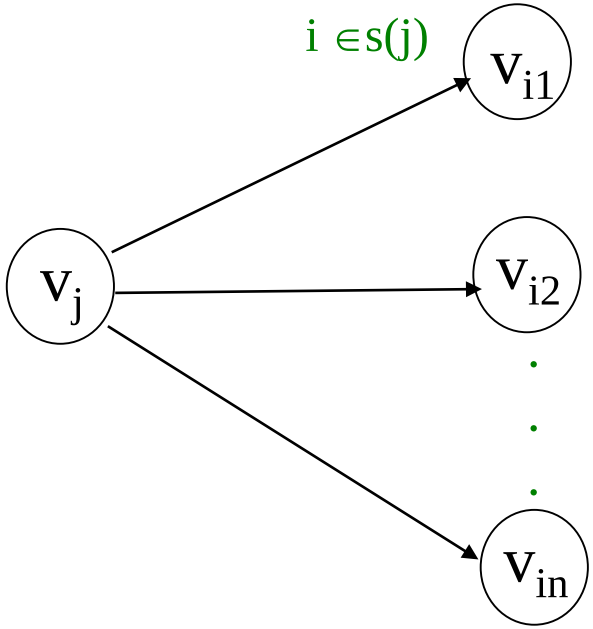
Calculation of Latest Times

- latest[j]
  - the latest event occurrence time

latest[n-1]=earliest[n-1]

latest[j]=min{latest[i]-duration of <j,i>}

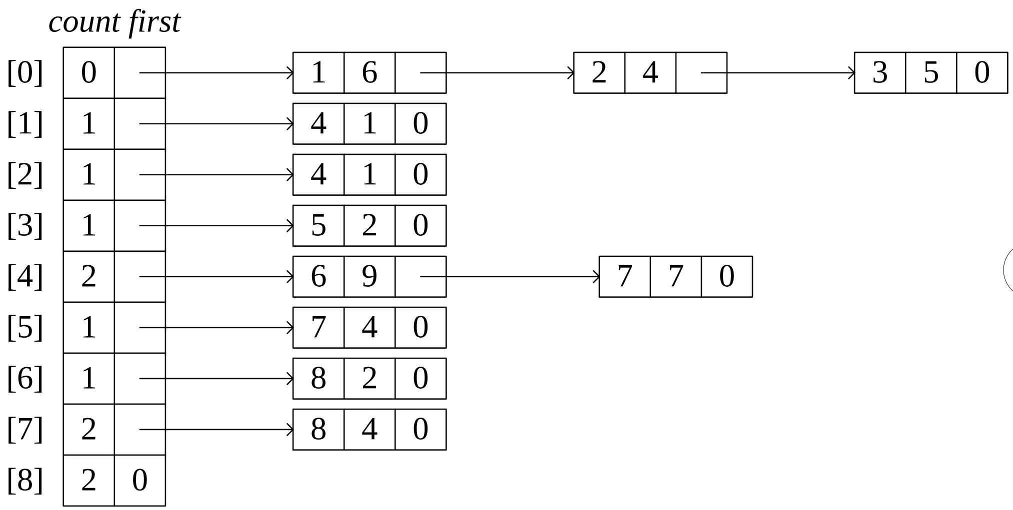
$i \in s(j)$

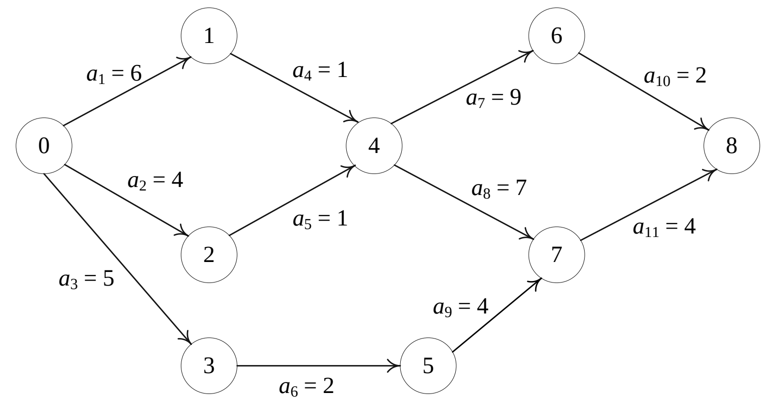


backward stage

if (latest[k] > latest[j]-ptr->duration)

latest[k]=latest[j]-ptr->duration





(a) Adjacency lists for Figure 6.40(a)

ee	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	堆疊
起始	0	0	0	0	0	0	0	0	0	[0]
輸出 0	0	6	4	5	0	0	0	0	0	[3, 2, 1]
輸出 3	0	6	4	5	0	7	0	0	0	[5, 2, 1]
輸出 5	0	6	4	5	0	7	0	11	0	[2, 1]
輸出 2	0	6	4	5	5	7	0	11	0	[1]
輸出 1	0	6	4	5	7	7	0	11	0	[4]
輸出 4	0	6	4	5	7	7	16	14	0	[7, 6]
輸出 7	0	6	4	5	7	7	16	14	18	[6]
輸出6	0	6	4	5	7	7	16	14	18	[8]
輸出 8										

(b) Computation of ee

Figure 6.41: Computing latest for AOE network of Figure 6.41(a)(p.325)

latest[8]=earliest[8]=18

latest[6]=min{earliest[8] - 2}=16

latest[7]=min{earliest[8] - 4}=14

latest[4]=min{earliest[6] - 9;earliest[7] -7}= 7

latest[1]=min{earliest[4] - 1}=6

latest[2]=min{earliest[4] - 1}=6

latest[5]=min{earliest[7] - 4}=10

latest[3]=min{earliest[5] - 2}=8

latest[0]=min{earliest[1] - 6;earliest[2]- 4; earliest[3] -5}=0

Activity	Early	Late	Late-Early	Critical
a_0	0	0	0	Yes
a_1	0	2	2	No
a_2	0	3	3	No
a_3	6	6	0	Yes
a_4	4	6	2	No
a_5	5	8	3	No
a_6	7	7	0	Yes
a_7	7	7	0	Yes
a_8	7	10	3	No
a_9	16	16	0	Yes
a_{10}	14	14	0	Yes

Figure 6.42: Early, late and critical values(p.327)

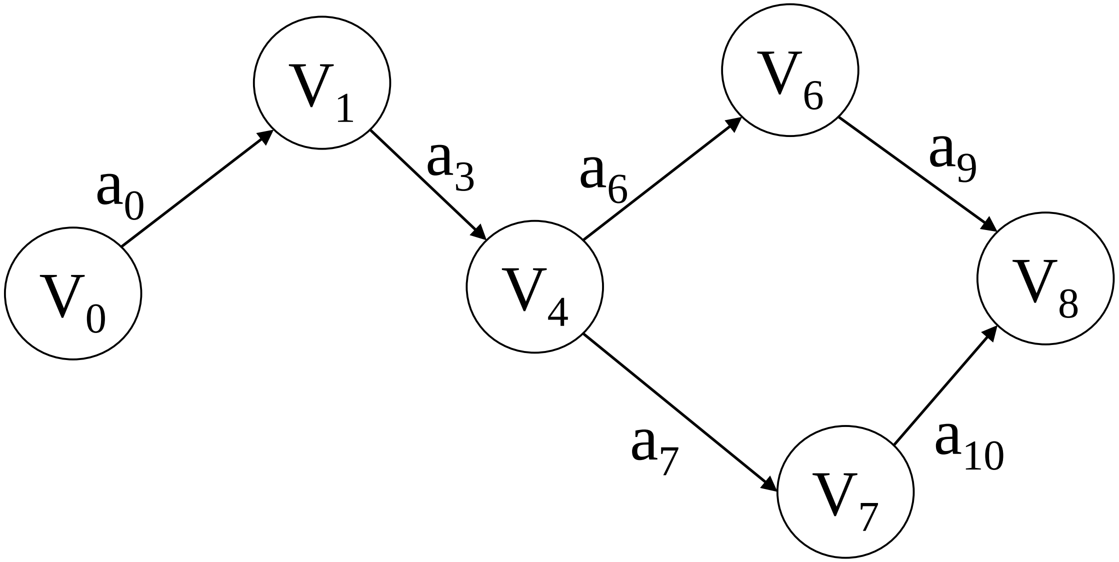


Figure 6.43: Graph with noncritical activities deleted (p.328)