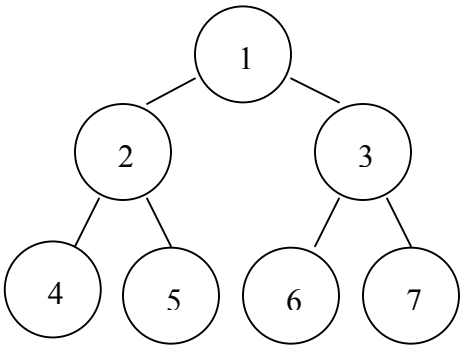
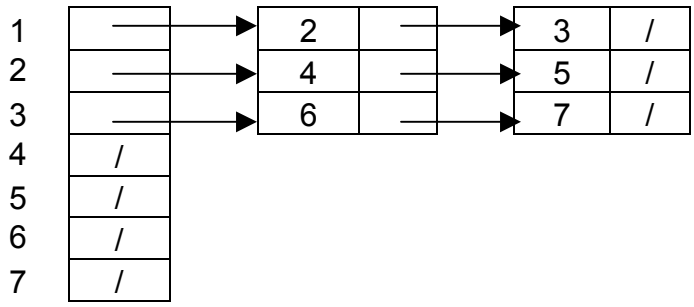
1. 22.1-2

Give an adjacency-list representation for a complete binary tree on 7 vertices. Give an equivalent adjacency-matrix representation. Assume that vertices are numbered from 1 to 7 as in a binary heap.



Adjacency List:



Adjacency Matrix:

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

2. 22.1-5

The square of a directed graph G = (V, E) is the graph $G^2 = (V, E^2)$ such that $(u,w) \in E^2$ if and only if for some $v \in V$, both $(u,v) \in E$ and $(v, w) \in E$. That is, G^2 contains an edge between u and w whenever G contains a path with exactly two edges between u and w. Describe efficient algorithms for computing G^2 from G for both the adjacency-list and adjacency-matrix representations of G. Analyze the running times of your algorithms.

G^2 for an adjacency matrix:
Computing G^2 may be done in V^3 time by matrix multiplication:

```
for i = 1 to V
  for j = 1 to V
  {
    G2[i][j] = 0;
    for k = 1 to V
      if (g[i][k] == 1 && g[k][i] == 1) {
          G2[i][j] == 1;
          break;
      }
  }
```

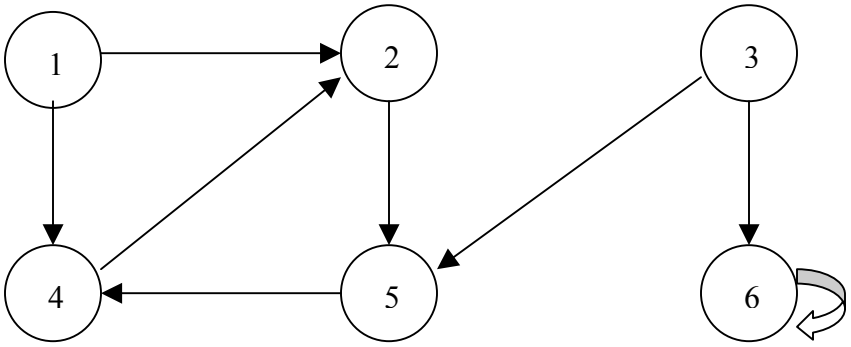
G^2 for an adjacency list:

```
Procedure G-Square (V[G], E[G]){
        V[G^2] ← V[G]
        for each u ∈ V[G]
                for each v ∈ Adj[u]
                        for each w ∈ Adj[v]
                                E[G^2] ← {(u, w)} U E[G^2]
}
```

Run time = $O(V^3)$

3. 22.2-1

Show the d and Π values that result from running breadth-first search on the directed graph of Figure 22.2(a), using vertex 3 as the source.



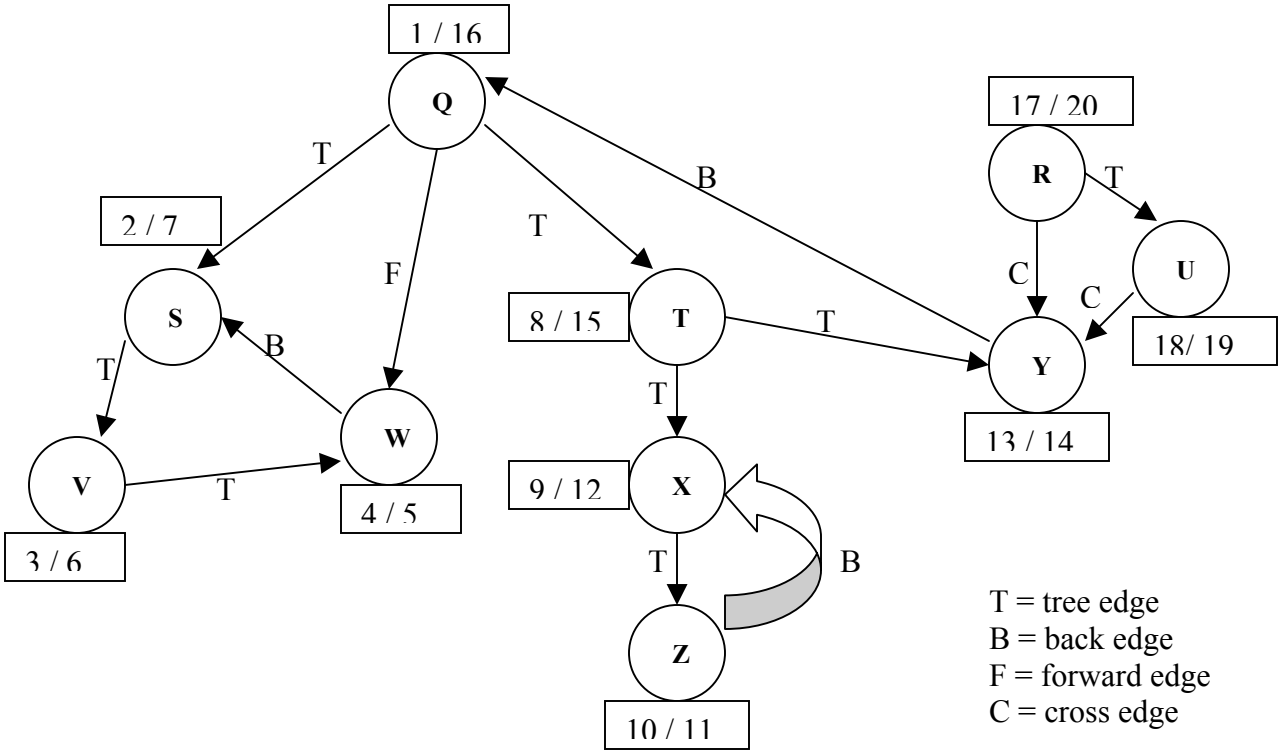
Vertex	d	Π
1	Infinite	Nil
2	3	4
3	0	Nil
4	2	5
5	1	3
6	1	3

4. 22.2-3

What is the running time of BFS if its input graph is represented by an adjacency matrix and the algorithm is modified to handle this form of input?

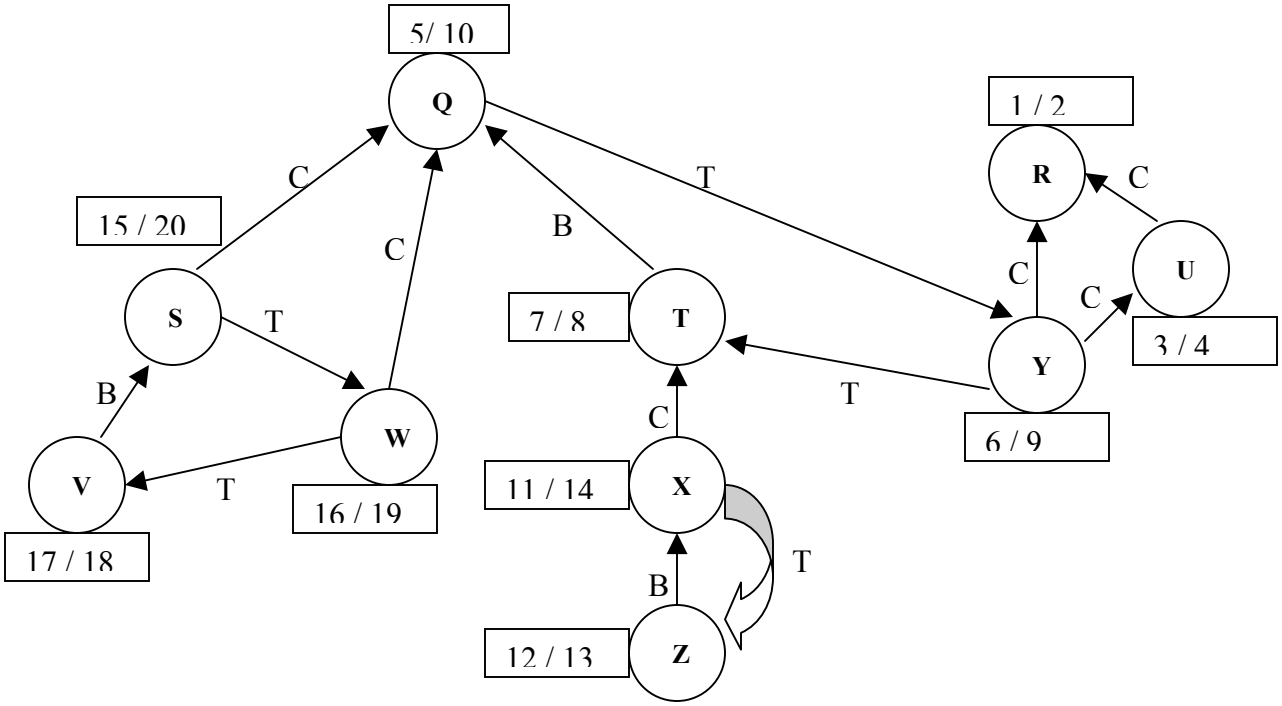
Each vertex can be explored once and its adjacent vertices must be determined too. This takes $\Theta(V^2)$ time.

5. Do a DFS on figure 22.6 (p.548). Classify each edge based on the DFS tree you determine.



6. Find the strongly connected components in figure 22.6.

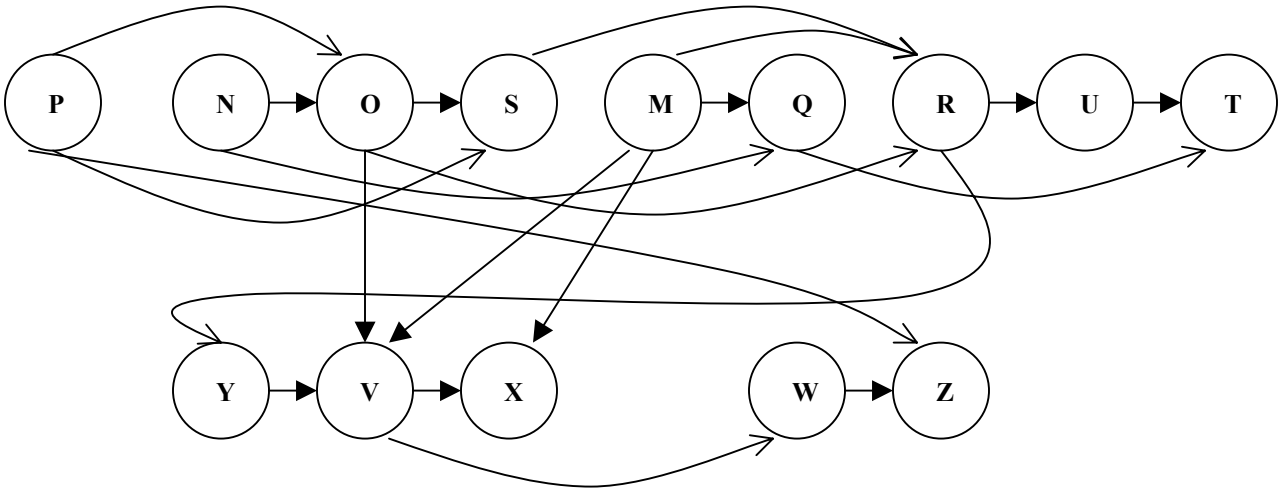
From 5., the first DFS gives the list R U Q T Y X Z S U W (reverse order of turning black)



Strongly connected components are {s, w, v}, {q, y, t}, {x, z}, {r}, {u}

7. 22.4-1

Show the ordering of vertices produced by TOPOLOGICAL-SORT when it is run on the dag of Figure 22.8, under the assumption of Exercise 22.3-2.



Ordering = P N O S M R Y V X W Z U Q T

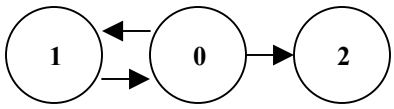
8. 22.5-1

How can the number of strongly connected components of a graph change if a new edge is added?

The number of strongly connected components can be reduced.

9. 22.5-3

Professor Deaver claims that the algorithm for strongly connected components can be simplified by using the original (instead of the transpose) graph in the second depth-first search and scanning the vertices in order of increasing finishing times. Is the professor correct?

No. Consider 

The order from the first DFS is 1 0 2.

This will lead to the entire graph being reported as a single SCC in the second DFS.

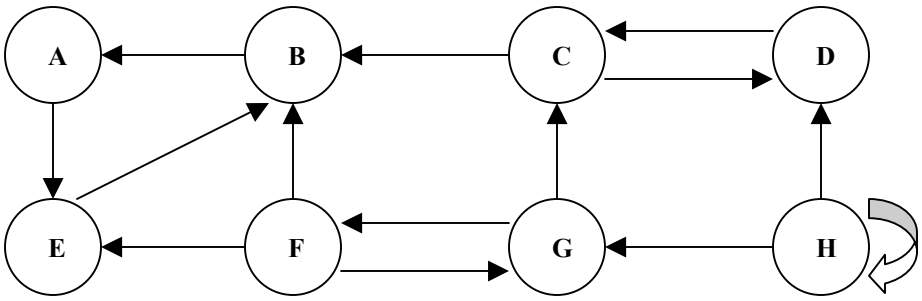
10. 22.5-4

Prove that for any directed graph G, we have $((G^T)^{SCC})^T = G^{SCC}$. That is, the transpose of the component graph of G^T is the same as the component graph of G.

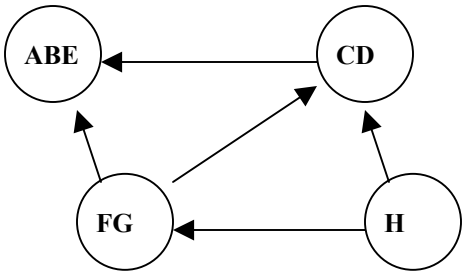
Observe that G and G^T will always have the same strongly connected components.

Using Figure 22.9 (p.553) as an example.

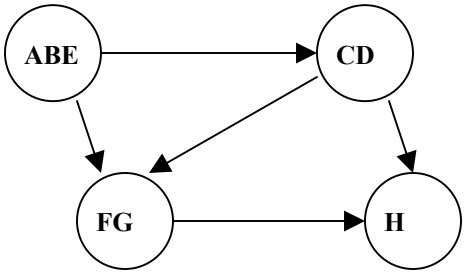
G^T =



$(G^T)^{SCC}$ =



$((G^T)^{SCC})^T$ =



$= G^{SCC}$.

11. 23.1-1

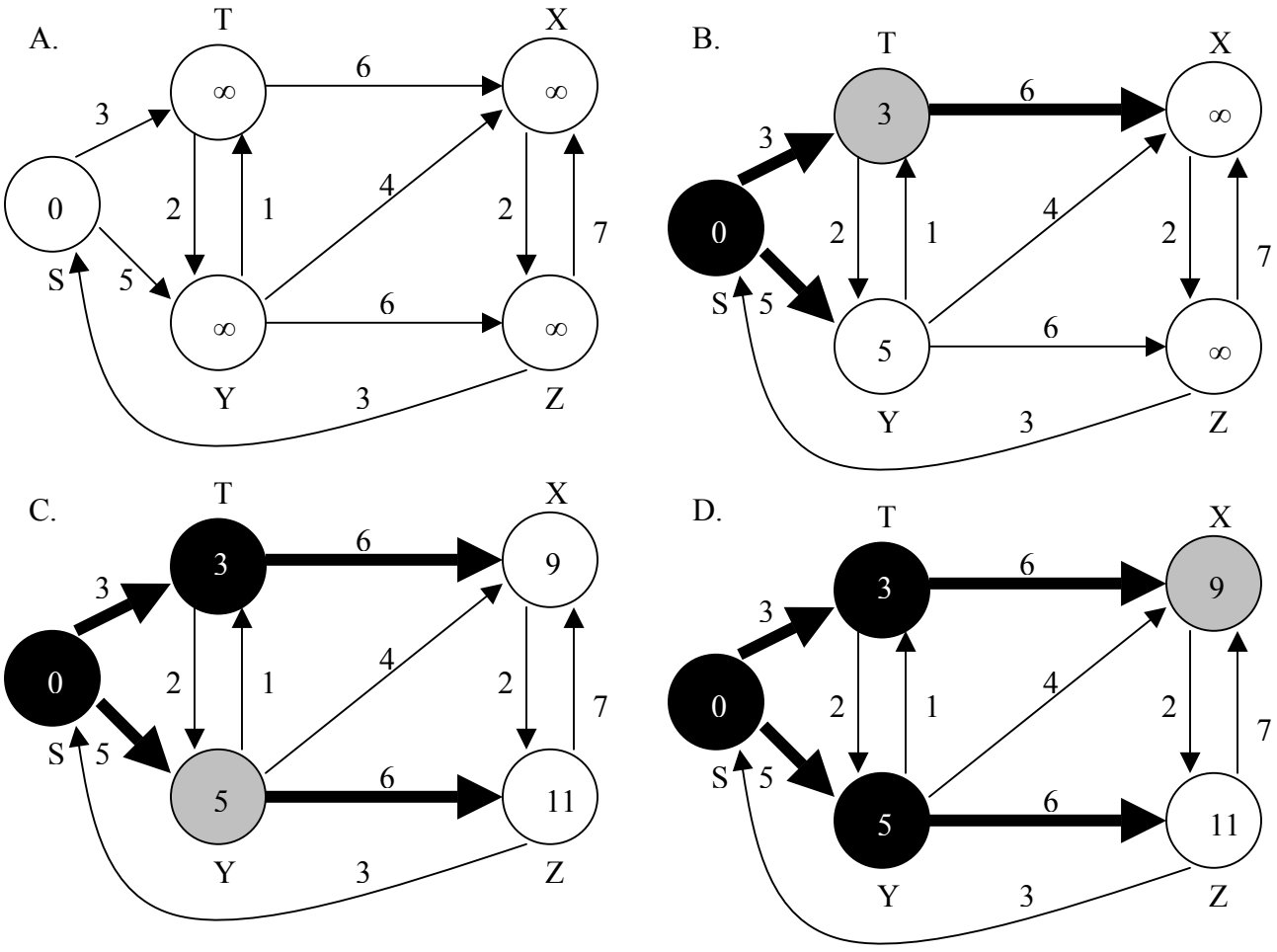
Let (u, v) be a minimum-weight edge in a graph G. Show that (u, v) belongs to some minimum spanning tree of G.

Let A be a subset of some MST T such that (u, v) $\notin$ A. To choose an edge to be added to A, all the edges on the cut are considered and an edge with lowest weight is selected. Since (u, v) is the minimum weight edge in the graph G, it gets selected on some cut.

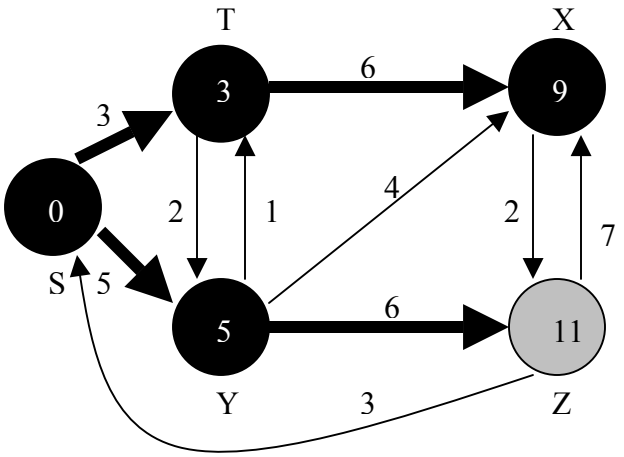
12. 24.3-1

Run Dijkstra's algorithm on the directed graph of Figure 24.2, first using vertex s as the source and then using vertex z as the source. In the style of Figure 24.6, show the d and Π values and the vertices in set S after each iteration of the while loop.

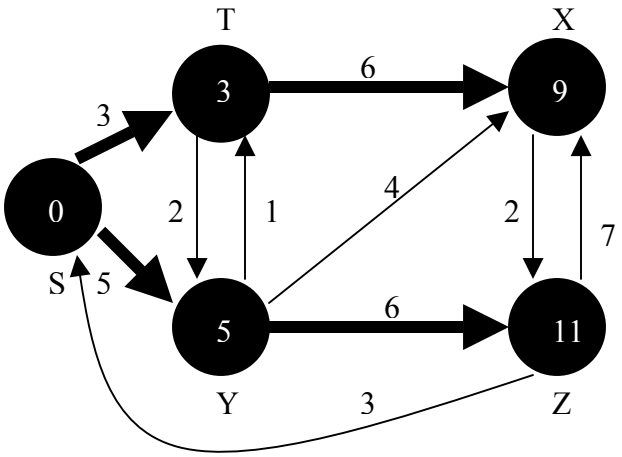
Black vertices are in the set S. The black, thick arrows are the values of Π and the values of d are included are listed inside each node.



E.

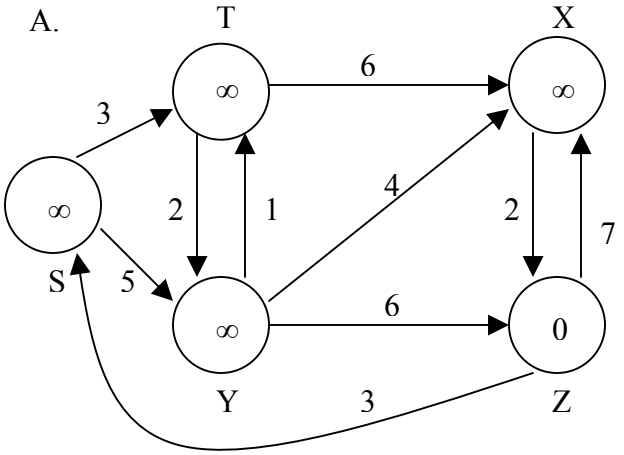


F.

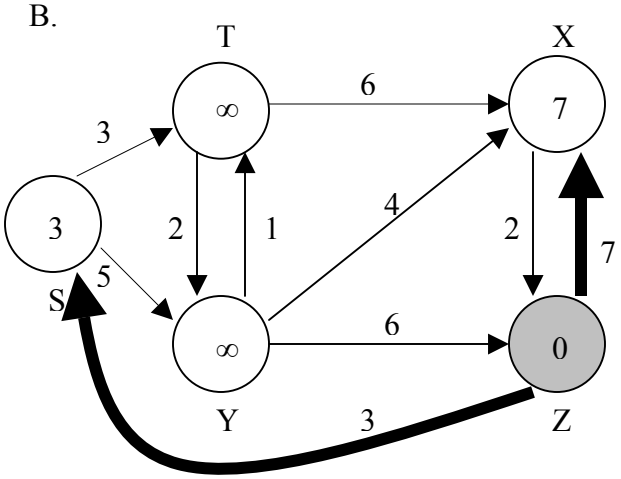


Using Vertex Z as the source.

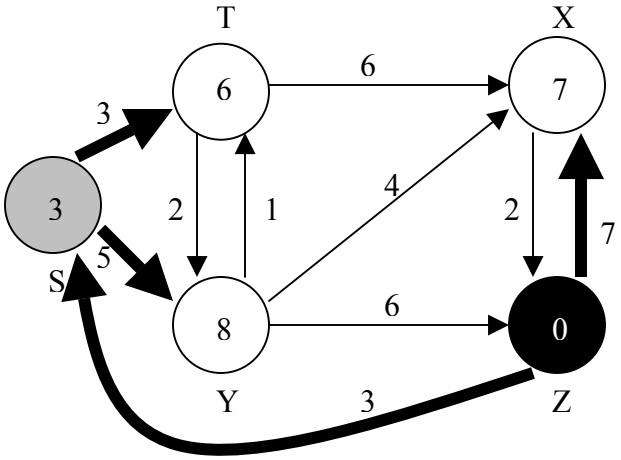
A.



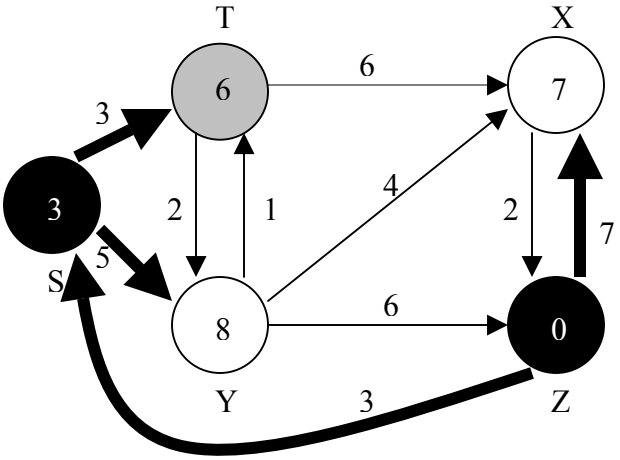
B.



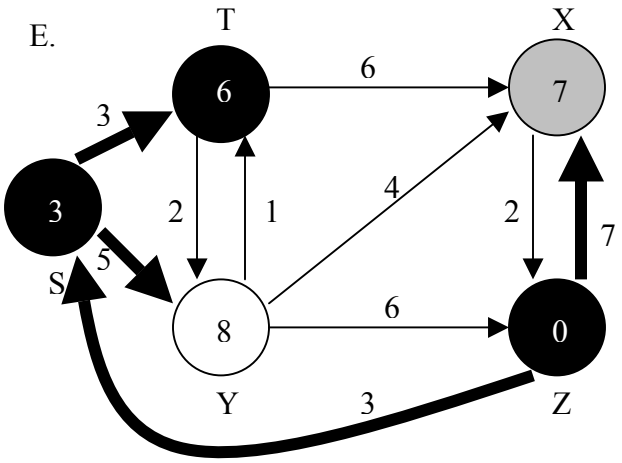
C.



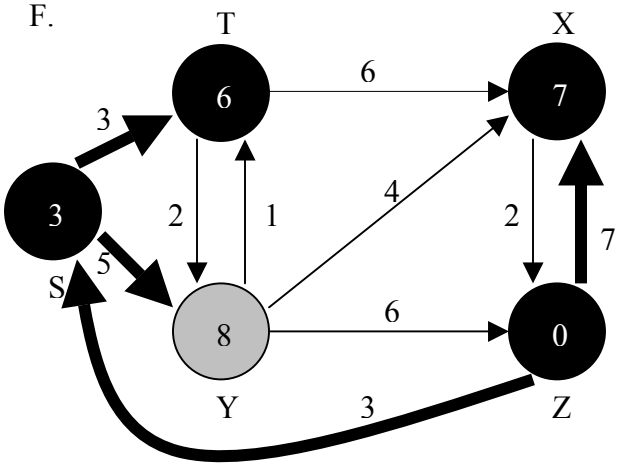
D.



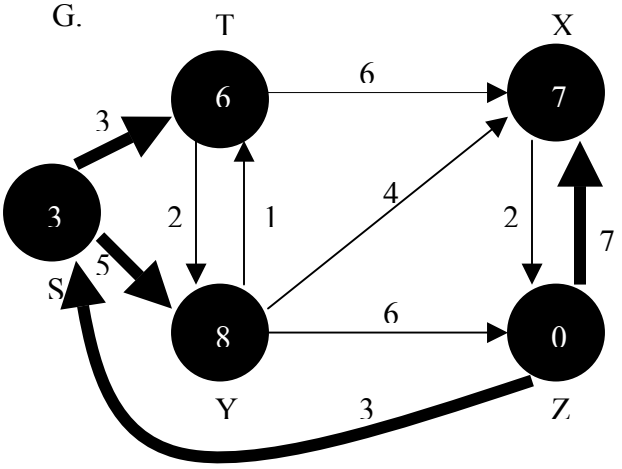
E.


T
X
6
7
3
3
2
1
4
2
7
5
S
8
6
0
Y
3
Z

F.


T
X
6
7
3
3
2
1
4
2
7
5
S
8
6
0
Y
3
Z

G.


T
X
6
7
3
3
2
1
4
2
7
5
S
8
6
0
Y
3
Z

13. a. Determine the transitive closure of the following Boolean matrix by using Warshall's algorithm.

1	0	1	1	0
0	0	1	1	0
1	0	0	0	0
1	0	1	1	1
0	0	0	1	0

T(1) =

1	0	1	1	0
0	0	1	1	0
1	0	1	1	0
1	0	1	1	1
0	0	0	1	0

T(2)=

1	0	1	1	0
0	0	1	1	0
1	0	1	1	0
1	0	1	1	1
0	0	0	1	0

T(3)=

1	0	1	1	0
1	0	1	1	0
1	0	1	1	0
1	0	1	1	1
0	0	0	1	0

T(4)=

1	0	1	1	1
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1

T(5)=

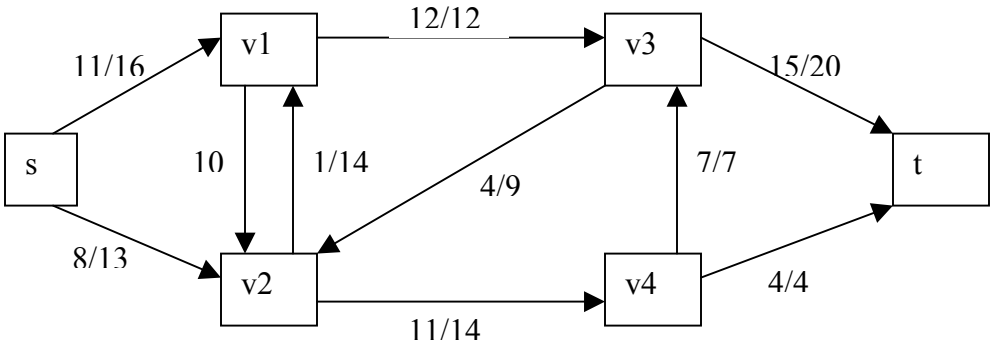
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1

b. Convert the matrix to indicate successors and use the version of Warshall's algorithm that allows path tracing.

0	-1	2	3	3
2	-1	2	3	3
0	-1	0	0	0
0	-1	2	3	4
3	-1	3	3	3

14. 26.1-2

Prove that for any vertex v other than the source or sink, the total positive flow entering v must equal the total positive flow leaving v.



Capacity constraint: For all u, v ∈ V, we require f(u, v) <= c(u, v)

In the above figure it can be seen that for each edge f(u, v) <= c(u, v) for edge (s, v1), f(u, v1) = 11 and c(u, v) = 16. 11 <= 16. Similarly for other edges.

Skew Symmetry: For all u, v ∈ V, we require f(u, v) <= -f(u, v)

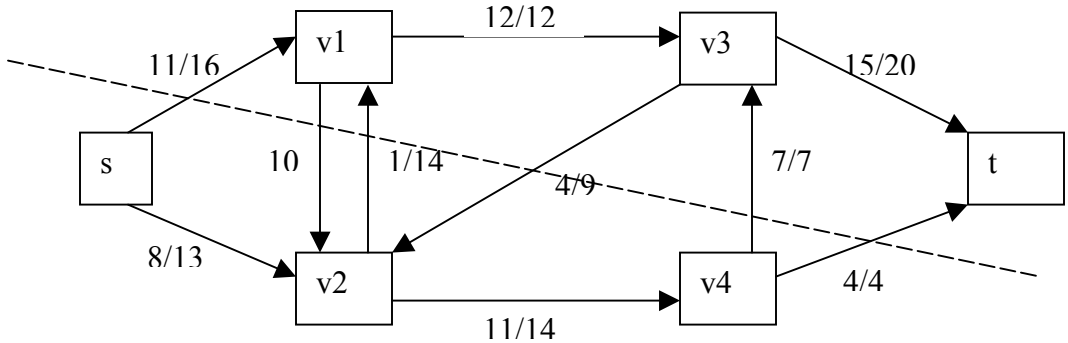
It can be seen for each edge (u, v) for flow (u, v) signifies a negative flow –f(v, u) from (v, u)

Flow conservation: For all u, v ∈ V – {s, t}, we require Σ f(u, v) = 0

It can be seen for each vertex v excluding s and t sum of the flows into it is equal to the flow out of it and hence flow is conserved. For vertex v1, f(s, v1) + f(v2, v1) = f(v1, v3). 11 + 1 = 12.

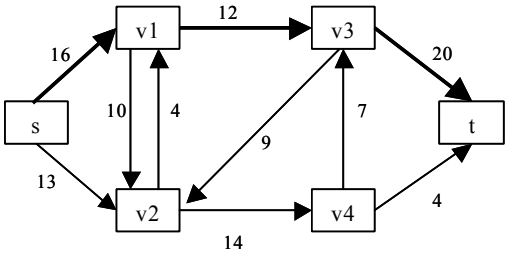
15. 26.2-1

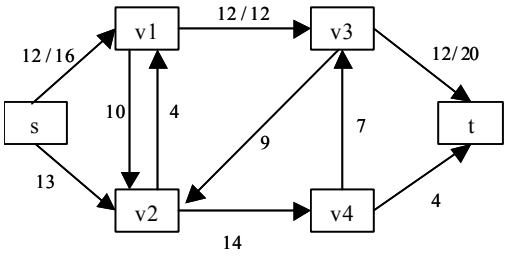
In Figure 26.1(b), which is the flow across the cut ({s, v_2, v_4}, {v_1, v_3, t})? What is the capacity of this cut?

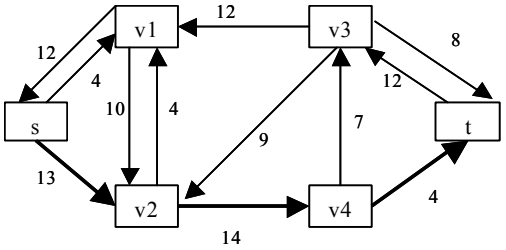


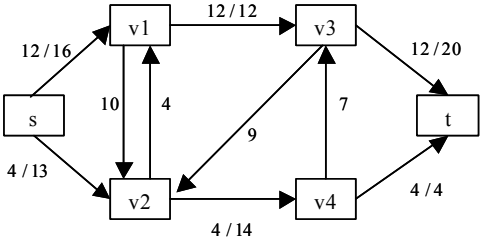
Net flow across the cut is f(s, v1) + f(v2,v1) + f(v2,v3) + f(v3,v4) + f(v4,t) = 11 + 1 + -4 + 7 + 4 = **19** and its capacity is = 16 + 14 +7 + 4 = **41**

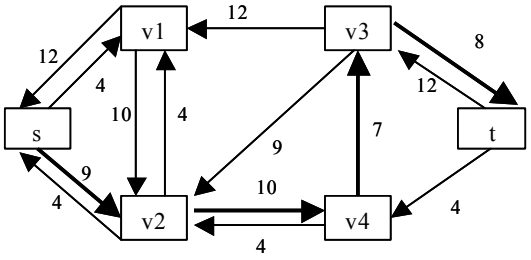
16 26.2-2

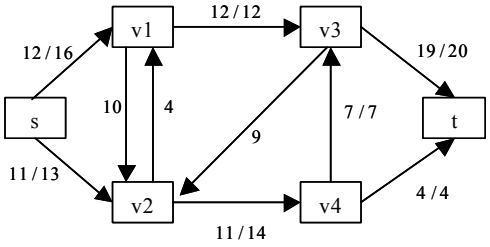


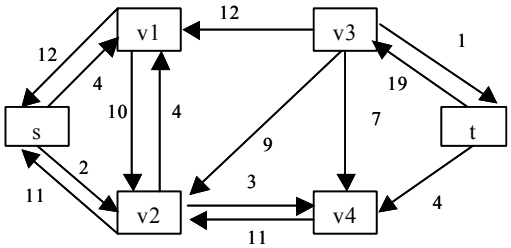




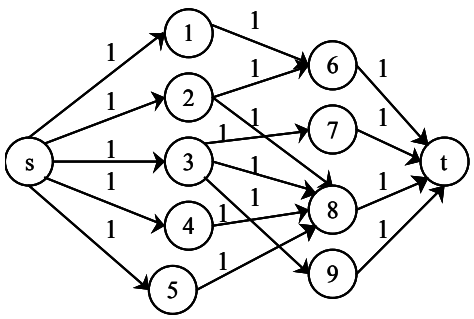








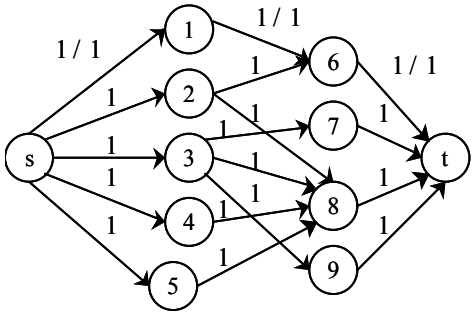
17 26.3-1



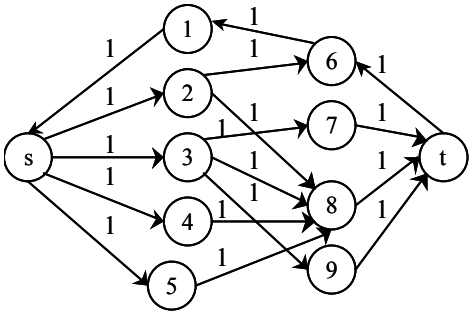
Augmenting path

$S \rightarrow 1 \rightarrow 6 \rightarrow t$

Flow N/W:

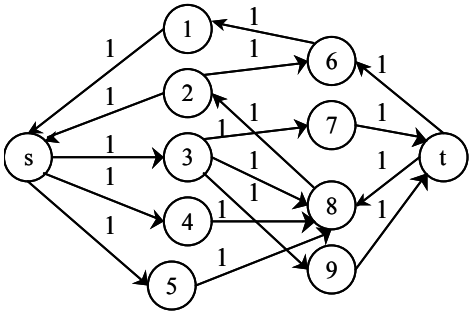


Residual N/W:



Augmenting path

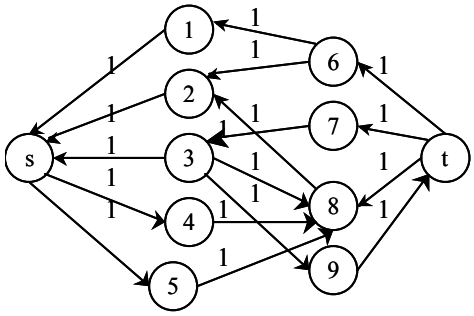
S → 2 → 8 → t



Augmenting path

S → 3 → 7 → t

Final Residual graph:



18 15.2-1

Finding the optimal parenthesization of a matrix-chain product whose sequence of dimensions is <5, 10, 3, 12, 5, 50, 6>

The m table

0	150	330	405	1655	2010
0	0	360	330	2430	1950
0	0	0	180	930	1770
0	0	0	0	3000	1860
0	0	0	0	0	1500
0	0	0	0	0	0

The s table

0	1	2	2	4	2
0	0	2	2	2	2
0	0	0	3	4	4
0	0	0	0	4	4
0	0	0	0	0	5
0	0	0	0	0	0

The optimal parenthesization

(A1 * A2) * ((A3 *A4) * (A5 * A6)))

19 15.2-2

```
MATRIX-CHAIN-MULTIPLY (A, s, i, j)
{
   if i =j
      C ← Ai
   else
     A ← MATRIX-CHAIN-MULTIPLY (A, s, i, s[i, j])
     B ← MATRIX-CHAIN-MULTIPLY (A, s, s[i,j]+1, j)
     C ←MATRIX-MULTIPLY (A, B)
  return (C)
}
```

20 15.2-4

We can find the sum by making a note of the access pattern for the m table

Example: For n = 5

	1	2	3	4	5
1	4	3	2	1	0
2	X	4	3	2	1
3	X	X	4	3	2
4	X	X	X	4	3
5	X	X	X	X	4

X – Don't Care; The numbers in the cells indicate the number of accesses to the cell

It can be seen that

$$\sum_{i=1}^{n} \sum_{j=i}^{n} R(i,j) = \sum_{i=1}^{n} i(i-1)$$

$$= \sum_{i=1}^{n} i(i^2-1)$$

$$= n/6\,((n+1)\,(2n+1)) - n/2\,(n+1)$$

$$= (n^3 - n)/3$$

21 15.4-1

x = <1,0,0,1,0,1,0,1> y = <0,1,0,1,1,0,1,1,0>

		0	1	2	3	4	5	6	7	8	9
		y_j	0	1	0	1	1	0	1	1	0
0	x_i	0	0	0	0	0	0	0	0	0	0
1	1	0	↑0	↖(1)	←1	↖1	↖1	←1	↖1	↖1	←1
2	0	0	↖1	↑1	↖(2)	←2	←2	↖2	←2	←2	↖2
3	0	0	↖1	↑1	↖2	↑2	↑2	↖(3)	←3	←3	↖3
4	1	0	↑1	↖2	↑2	↖3	↖3	↑3	↖(4)	↖4	←4
5	0	0	↖1	↑2	↖3	↑3	↑3	↖4	↑4	↑4	↖5
6	1	0	↑1	↖2	↑3	↖4	↖4	↑4	↖5	↖(5)	↑5
7	0	0	↖1	↑2	↖3	↑4	↑4	↖5	↑5	↑5	↖(6)
8	1	0	↑1	↖2	↑3	↖4	↖5	↑5	↖6	↖6	↑6

22 15.4-2

Print LCS without the b table

```
print_lcs (i,j)
{
        int i,j;                        /* i, j are the lengths of the two lists */
        if ((i = = 0) || (j = = 0))
            return;
        if(x[i]  = = y [j])             /* x, y are the two lists whose LCS is to be found */
        {
            print_lcs (i -1, j -1);
            printf("%d", x[i]);
        }
       else if (c[i-1][j] >= c[i][j])   /* c is the c table as in the algorithm */
        {
            print_lcs (i -1, j);
        }
        else
        {
            print_lcs (i, j-1);
        }
        return;
}
```

23 15.4-5

Algorithm

- a[1 ..n] is the input sequence
- length[1..n] contains the length of the monotonically increasing subsequences up to a[i]= { i = 1 ..n}
- lms is the length of the longest monotonically increasing subsequence

```
for i = 2 to n do
        Begin
            for j = 1 to i -1 do
                Begin
                    Search for the j such that length[j] is the largest and a[i] can be
                    included in the subsequence it represents.
                End
                length[i] = length[j] + 1
                if  lms < length[i] then lms = length[i]
        End
```

24 15-1

Bitonic TSP

Points $P_0 \ldots P_{n-1}$ are sorted by increasing X- Coordinate

C(i, j) = Cost of achieving optimal pair of paths such that are paths ends with P_i, the other with P_j (i < j)

Base Case
C(0,1) = dist (0, 1)

General Case
C(i-1, i) = min {C(j, i-1) + dist(j, i)}
0≤ j< i-1

C(i, j) = C(i, j-1) + dist(j-1, j) where i <j-1

Final solution
min {C(i, n-1) + dist(i,n-1)}
0≤ i< n-1

25 16.1-1

```
/* f [1…n] contains finishing times (sorted) of activities
   s [1…n] contains the starting times of those activities
   m[1…n] contains the number of activities from 1 .. i that can be scheduled m_i in the
            problem
   fm[1…n] indicates the finishing times of the tasks scheduled in each of m[1…n]  */


Begin
        m[1] = 1
        fm[1] = f[1]
        for i = 2 to n do
                Begin
                        if( fm[i-1] ≤ s[i] then
                           Begin
                                m[i] = m[i-1] + 1
                                fm[i] = f[i]
                           End
                        else
                            Begin
                                fm[i] = fm[i-1]
                                m[i] = m[i-1]
                            End
                End i
End
```

26 16.1-3

```
n ← length [s]
for i ← 1 to n
        A[i] ← { ∅ }                    //each A[i] (lecture Hall) has a set of activities
        LIST_INSERT( L, i );
k ← 0
while L ≠ ∅
        do k ← k + 1
           i ← head [L]
           for j ← i + 1 to tail[L]
                do if s_j ≥ f_i
                        then A[k] ← A[k] U { j }
                                i ← j
                                LIST_DELETE( L, j )
return L // the final value of 'k' has the number of lecture halls
```

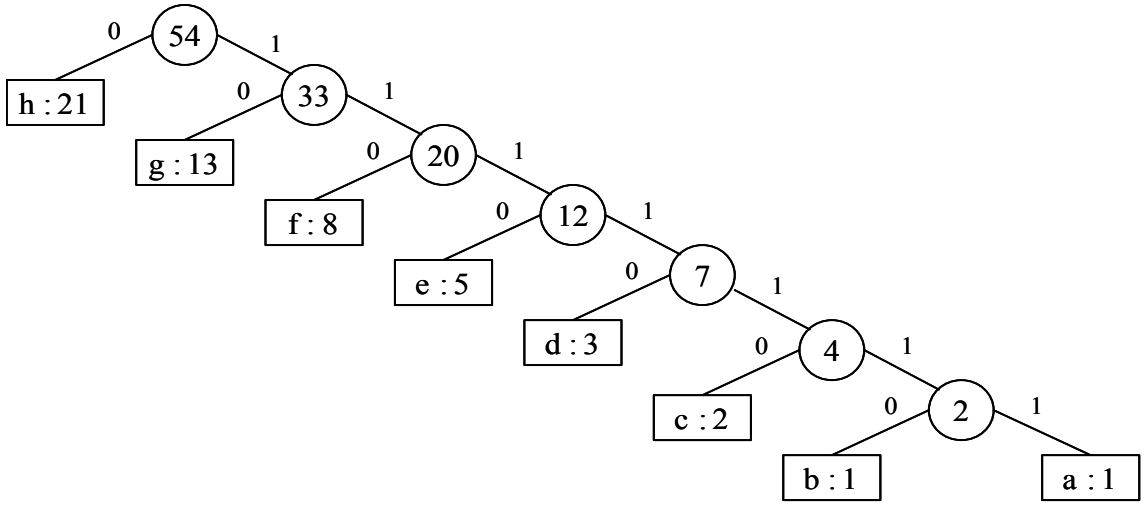
27 16.2-4

The greedy strategy would be to fill up gas at the last moment i.e., Travel to the farthest gas station that can be reached from the current gas station (without falling short)

28 16.2-5

Sort the points in ascending order of their k values
The greedy strategy would be to enclose the leftmost unenclosed point and all points that lie within a unit distance of this point. The next interval will begin at the closest point to the right of this interval

29 16.3-2



Generalization:

code = { k-1 1's followed by a '0', if k ≤ n-1
       { k 1's, k = n

30

String ababbabbababbababbabb
Pattern ababbabbababbababbabb

Fail link table 1	Fail link table 2
0 a -1	0 a -1
1 b 0	1 b 0
2 a 0	2 a -1
3 b 1	3 b 0
4 b 2	4 b 2
5 a 0	5 a -1
6 b 1	6 b 0
7 b 2	7 b 2
8 a 0	8 a -1
9 b 1	9 b 0
10 a 2	10 a -1
11 b 3	11 b 0
12 b 4	12 b 2
13 a 5	13 a -1
14 b 6	14 b 0
15 a 7	15 a 7
16 b 3	16 b 0
17 b 4	17 b 2
18 a 5	18 a -1
19 b 6	19 b 0
20 b 7	20 b 2