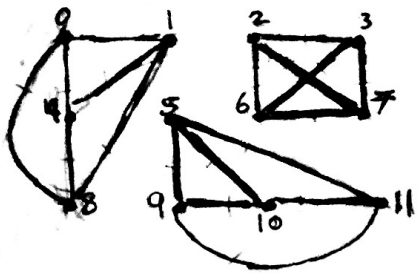
R-14.1)



Vertices = 12
Edges = 18
connected = 3

R-14.3)

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	0	0	1	0	0	0	1	0	0	0
1	1	0	0	0	1	0	0	0	1	0	0	0
2	0	0	0	1	0	0	1	1	0	0	0	0
3	0	0	1	0	0	0	1	1	0	0	0	0
4	1	1	0	0	0	0	0	0	1	0	0	0
5	0	0	0	0	0	0	0	0	0	1	1	1
6	0	0	1	1	0	0	0	1	0	0	0	0
7	0	0	1	1	0	0	1	0	0	0	0	0
8	1	1	0	0	1	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0	0	1	1
10	0	0	0	0	0	1	0	0	0	1	0	1
11	0	0	0	0	0	1	0	0	0	1	1	0

R-14.6)

From the Edge List structure ADT, new edges and vertices can be added to the graph by, "creating an Edge instance storing the given element as data, adding that instance to the positional list E, and recording its resulting Position within E as an attribute of the edge," (Book 209) in O(1) time. On the other hand, when vertex v is removed from the graph, all edges incident to v must be removed as well. To locate all incident edges to vertex, all edges of E must be traversed, thus taking O(m) time.

R-14.9)

~~[illegible]~~ No, because the get_edge(u,v) operation would take O(m) without the reference to the Edge List.

R-14.11)

a.] Adjacency List structure. Adjacency matrix allocates extra space for entries the graph may not have. Adjacency lists use less space in this case.

b.] The Adjacency List structure. Adjacency matrix uses $O(n^2 = 100,000,000)$ space while the Adjacency list uses $O(n+m = 10,000 + 20,000,000)$ space, so both structures would be okay, but the Adjacency List is better for insert/remove-vertex.

c.] The Adjacency Matrix, because it supports get-edge(u,v) operation in $O(1)$ time.

R-14.12)

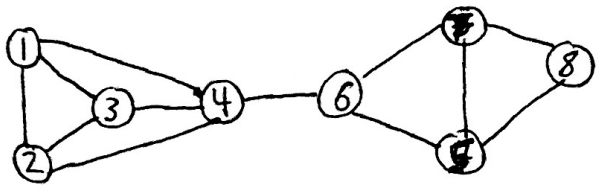
In a DFS traversal, for each vertex of the graph, the set of neighbours of the vertex must be traversed. Thus, every row of the adjacency matrix representation must be traversed, as each row corresponds to a vertex of the graph. For each of n vertices, the $n-1$ non-diagonal entries of the row must be examined, taking $O(n \cdot n)$ or $O(n^2)$ operations.

R-14.13)

Figure 14.8b has nontree edges that are forward edges, and cross-edges.

R-14.16)

a.]



b.] DFS: {1, 2, 3, 4, 6, 5, 7, 8}

c.] BFS: {1, 2, 3, 4, 6, 5, 7, 8}

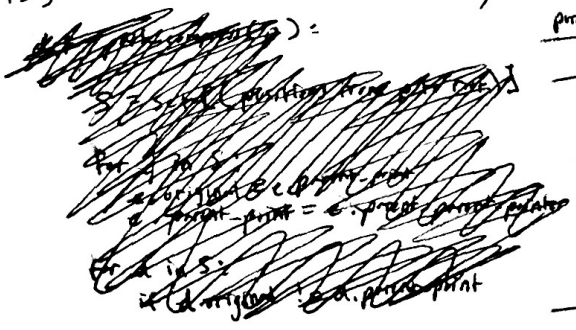
R-14.21)

Topological ordering: {BOS, ORD, JFK, SFO, DFW, LAX, MIA}

R-14.38)

```
def remove_edge(e):
    a = e._origin
    b = e._destination
    del self._outgoing[a][b]
    if self.is_directed():
        del self._incoming[b][a]
```

R-14.73)



1.)

path = 1 2 3

parent = 1 1 2

-> Karvel's compression ->

2.) path = 1 2 3

parent = 1 1 1

Compressed path:

1 -> 3 ✓ (Correct)

- Running time is O(k-2) for k positions.