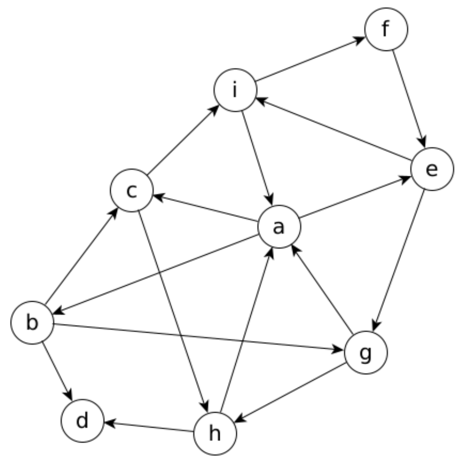
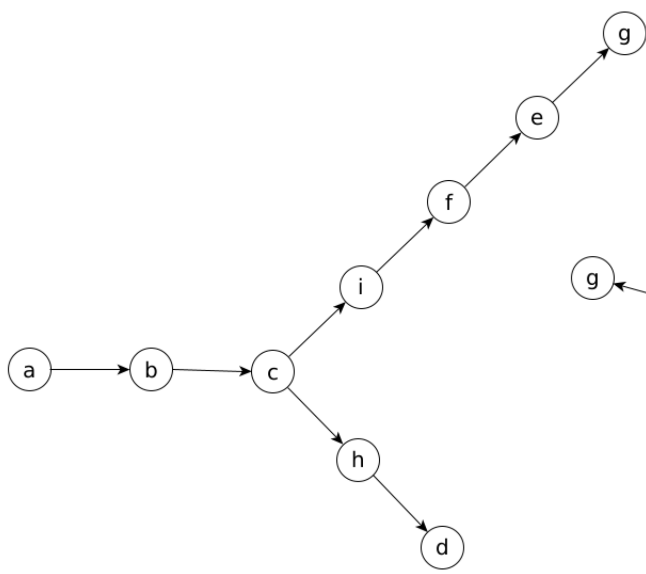
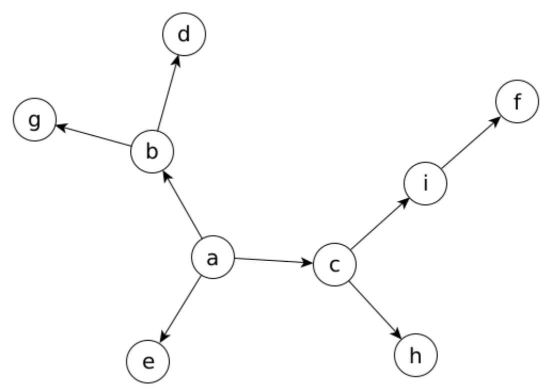
1. **Algorithms:** (*60 Points*)

   (a) Given the following directed graph display the spanning tree/forest for a depth-first search/traversal and the spanning tree/forest for a breadth-first search/traversal. As usual, the vertices are to be processed in alphabetical order.



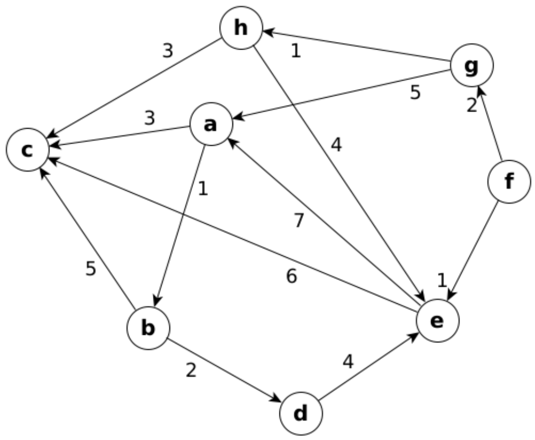
Solution:





(b) Given the following directed weighted graph, use Dijkstra's algorithm to find the shortest path to all vertices from the starting vertex f. Display each iteration, active vertex, and weight label in chart form as was done in the text and in class.

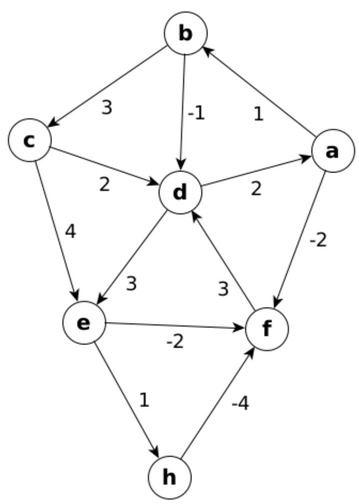
Solution:



Iteration:	Init	1	2	3	4	5	6	7
Active Vertex:		f	e	g	h	c	a	b
a	∞	∞	8	7	7	7		
b	∞	∞	∞	∞	∞	∞	8	
c	∞	∞	7	7	6			
d	∞	∞	∞	∞	∞	∞	∞	10
e	∞	1						
f	0							
g	∞	2	2					
h	∞	∞	∞	3				

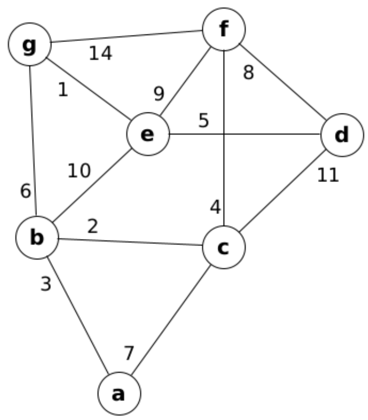
(c) Given the following directed weighted graph, use Ford's algorithm to find the shortest path to all vertices from the starting vertex b. Display each iteration and sequence of weight label changes in chart form as was done in the text and in class.

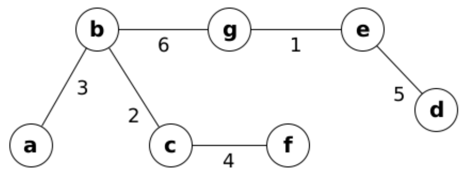
Solution:



Iteration:	Init	1		2
a	∞	1		No Changes
b	0			
c	∞	3		
d	∞	-1		
e	∞	7	2	
f	∞	0	-1	
h	∞	3		

(d) Given the following weighted graph, use Kruskal's algorithm to find the minimum spanning tree of the graph. Display the final resulting minimum spanning tree.
Solution:





2. **Complexities:** (*15 Points*)

(a) What is the complexity of the depth-first search/traversal?
Solution: $O(|V| + |E|)$

(b) What is the complexity of the breadth-first search/traversal?
Solution: $O(|V| + |E|)$

(c) What is the complexity of Dijkstra's algorithm for finding the shortest path from one vertex to all the other vertices in a graph?
Solution: $O(|V|^2)$, also accepted $O(|E| + |V|)\lg(|V|)$ although our implementation did not use a heap.

(d) What is the complexity of Ford's algorithm for finding the shortest path from one vertex to all the other vertices in a graph?
Solution: $O(|V||E|)$

(e) What is the complexity of Kruskal's algorithm for finding a minimal spanning tree for a graph?
Solution: $O(|E|\lg(|V|))$

(f) What is the complexity of Dijkstra's algorithm for finding a minimal spanning tree for a graph?
Solution: $O(|E||V|)$

(g) What is the complexity of the Ford-Fulkerson algorithm for finding the maximum flow through a network?
Solution: $O(|V||E|^2)$

3. **Code:** (*30 Points*)

(a) Given our graph class structure, write a depth-first search/traversal that will return a list of edges for the traversal order to follow in a depth-first search. The list of edges should be a vector of pairs of templated type, the type that is storing the vertex label.

Solution:

```
template <class T> vector<pair<T, T>> depthFirstSearchG(Graph<T> &G) {
  vector<T> vlist = G.getVertexList();
  vector<int> num(vlist.size());
  vector<pair<T, T>> Edges;
  int count = 1;

  while (find(num.begin(), num.end(), 0) < num.end()) {
    int pos = find(num.begin(), num.end(), 0) - num.begin();
    DFS(G, num, vlist, pos, count, Edges);
  }

  return Edges;
}

template <class T>
void DFS(Graph<T> &G, vector<int> &num, vector<T> &vlist, int pos, int &count,
         vector<pair<T, T>> &Edges) {
  vector<T> Adj = G.getAdjacentList(vlist[pos]);
  num[pos] = count++;

  for (size_t i = 0; i < Adj.size(); i++) {
    T vert = Adj[i];

    size_t vPos = find(vlist.begin(), vlist.end(), vert) - vlist.begin();
    if (vPos < vlist.size() && num[vPos] == 0) {
      Edges.push_back({vlist[pos], vert});
      DFS(G, num, vlist, vPos, count, Edges);
    }
  }
}
```

(b) Given our weighted graph class structure, write a function that will use Kruskal's algorithm to return the minimal spanning tree of the input (parameter) weighted graph. The return type should be a weighted graph, templated of course.

Solution:

```
template <class T, class W> WGraph<T, W> KruskalAlgorithm(WGraph<T, W> &G) {
  WGraph<T, W> MST;
  vector<pair<T, pair<T, W>>> edges = G.getEdgeList();

  sort(edges.begin(), edges.end(),
       [](auto &a, auto &b) { return a.second.second < b.second.second; });

  int MSTedgecount = 0;
  int Gvertcount = G.size();
  for (size_t i = 0; i < edges.size() && MSTedgecount < Gvertcount - 1; i++) {
    if (MST.getEdgePos(edges[i].first, edges[i].second.first) != -1)
      continue;

    WGraph<T, W> TestMST = MST;
    TestMST.addEdge(edges[i]);
    if (!detectCycles(TestMST)) {
      MST.addEdge(edges[i]);
      MSTedgecount++;
    }
  }
  return MST;
}
```