CISC235
Winter 2007
Homework for weeks 8 and 9
in preparation for quiz 4
Solutions

Note: I received help for preparing these solutions from Michael Xiao and Fangpeng Dong. All questions are from section 8.14 of the text.

2. In a graph G = (V, E) the sum of degrees of all vertices is twice of the number of edges. This follows from the fact that every edge is incident to exactly two vertices.

3. The complexity of breadthFirstSearch() is $O(|V| + |E|)$.

4. To show that a simple graph is connected if it has a spanning tree, all we need to do is use the definition of a spanning tree. By definition a spanning tree is incident to every vertex in the graph it spans. Also by definition a tree is a connected graph. Therefore if a (simple) graph has a spanning tree then it is connected.

5. In a tree, each vertex except the root has only one edge which is connected to its parent. So there is a one-to-one map between vertices and edges, except for the root. Therefore, a tree with n vertices has $n - 1$ edges.

6. To apply Dijkstra's algorithm to undirected graphs, just turn every edge in the undirected graph into two directed edges with opposite directions. In fact in class I presented a high level description of the algorithm on an undirected graph without modification.

7. To solve a one to one shortest path algorithm we can modify Dijkstra's algorithm so that it exits when the desired destination is reached. In the worst case there will be no early exit.

9. Observe that no non-cylclic path can have more than $|V| - 1$ edges. Running Floyd's algorithm for a maximum of $|V| - 1$ iterations guarantees that all legitimate shortest paths are found and also guarantees finishing.

10. Ford's algorithm iterates as long as no improvements are found. So one iteration is enough when the initial currDist values can't be improved. This occurs when no vertex is reachable from the vertex *first*. Two iterations are enough when no improvements can be found after the first iteration. This can occur if all shortest paths use only the edges in the order that they are processed in the while loop.

11. No, Ford's algorithm cannot be applied to an undirected graph without changes, because a negative weighted edge in an undirected graph forms a negative cycle in its directed interpretation.

15. We add an array path[$|V|$][$|V|$]. path[i][j] saves the name of js predecessor vertex in the shortest path from i to j. weight[i][j] is the shortest length from i to j.

```
WFIalgorithm(matrix weight)
for i = 1 to |V |
      for j = 1 to |V |
            if there is a edge from i to j
               path[i][j] = i;     // There is a path from i to j.
            else path[i][j] = NOT_A_VERTEX
      end for
end for

for i = 1 to |V |
      for j = 1 to |V |
            for k = 1 to |V |
                        if weight[j][k] > weight[j][i] + weight[i][k]
                                    weight[j][k] = weight[j][i] + weight[i][k];
                                    path[j][k] = path[i][k];
             end for
       end for
 end for
```

Now we can use the path matrix to retrieve shortest paths using a recursive function that prints the shortest path between any vertex pair (u,v).

```
void printThePath (int u, int v, const matrix<int> P)
    int k;

    k = path[u][v];
    if (k == NOT_A_VERTEX) return;
    printThePath(u,k,P);
    if(k != u)
        cout << " to " <<  k ;
```

12. To compute the all-to-one shortest paths we can simply reverse the direction of every edge in the digraph and just use the given algorithm. However the question asks us to modify the algorithm, not the graph. Here is one way to do this:

```
ModifiedFordAlogrithm( weighted simple digraph, vertex last )
for all vertices v
         currDivst (v) = INFINITY;
currDist (last) = 0;
while there is an edge(vu) such that currDist(v) > currDist(u)+weight(edge(vu))
          currDist(v) = currDist(u) + weight(edge(vu))
```

The order of edges: ab be cd cg ch da de di ef gd hg if

	Init	1	2	3	4
a	∞			0	
b	∞		-1		
c	∞			3	2 1
d	∞		8 2		
e	∞	4			
f	0				
g	∞			1	
h	∞			0	
i	∞	1			

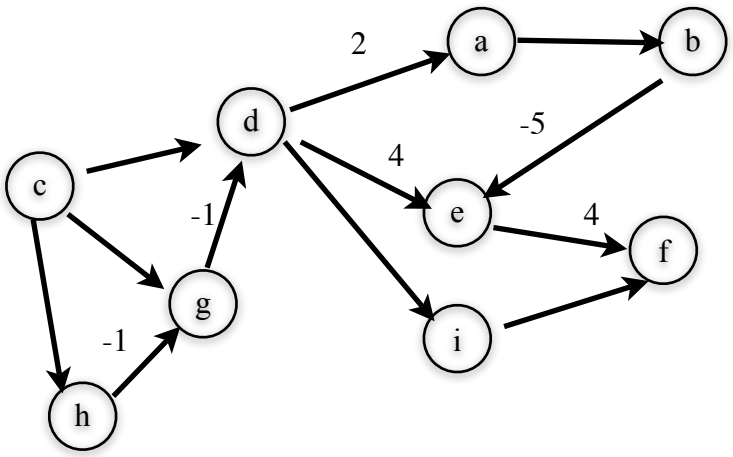

Figure 1: Figure 8.8 from the text. Note unmarked edge weights are all equal to 1.

16. After WFIalgorithm finishes, if there is an element on the diagonal which has a negative value, we see that there is an negative cycle.

17. It's not clear to me that you would save much by checking to see if the weight is finite. We add one comparison to every iteration with the hope of not doing a comparison and an addition. For non-infinite values we save an addition operation for infinite values we save a comparison operation, I think on the whole things may balance out. Another issue that was raised in class is the possibility that summing infinite edge weights may result in arithmetic overflow.

18. The problem with this approach is that it cannot guarantee the short path in the new graph is the same as the short path in the original graph. For example, consider the following graph G: the short path from a to d is $a \to b \to c \to d$ in the original graph (left), but after the graph is processed in the given way (right), the shortest path is changed to $a \to d$