CSE 101
Winter 2026
Midterm Exam 1 **Solutions**

1. (20 Points) Using only the List ADT operations defined in the project description for pa1, write a *client* function with the heading

$$\text{List FindElement(List } L\text{, ListElement } x\text{)}$$

that returns a new List containing all positions in L at which x is found. The positions will occur in the returned List in increasing order. If List L does not contain x, the returned List will be empty. For instance, if $L = (3, 4, 4, 3, 4, 1, 1, 2, 4, 2)$ and $x = 4$, then the returned list will be $(1, 2, 4, 8)$. Function FindElement() has no preconditions, and will make no changes to the sequence contained in its argument L.

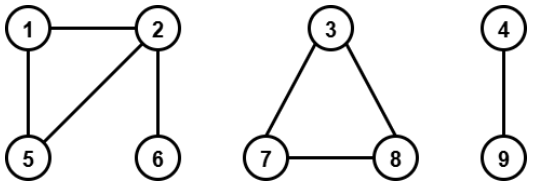
One of Several Possible Solutions:

```
List FindElement(List L, ListElement x){

   int y;
   List R = newList();

   for( moveFront(L); position(L)>=0; moveNext(L) ){
      y = get(L);
      if( x==y ){
         append(R, position(L));
      }
   }

   return R;
}
```

2. (20 Points) Given an undirected graph G, and vertices x and y in G, we say that y is reachable from x if and only if there exists a path in G from x to y. Using the Graph ADT operations defined in the project description for pa2, and the List ADT operations defined in pa1, write a *client* function with the heading

$$\text{List Reachable}(\text{Graph } G, \text{int } x)$$

that returns a new List containing the integer labels of all vertices that are reachable from vertex x. The returned List will contain the vertex labels in increasing order. For instance, if G is the graph



and $x = 6$, the returned List will be $(1, 2, 5, 6)$, while if $x = 8$, the returned List will be $(3, 7, 8)$. Function Reachable() will have no preconditions.

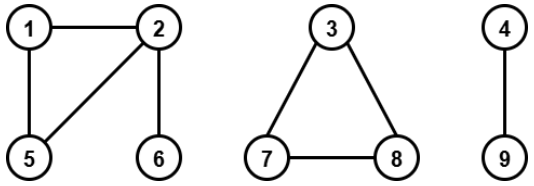
One of Several Possible Solutions:

```
List Reachable(Graph G, int x){

   int y;
   int n = getOrder(G);
   List L = newList();

   BFS(G, x);
   for( y=1; y<=n; y++ ){
      if( getDist(G, y) != INF ){
         append(L, y);
      }
   }

   return L;
}
```

3. (20 Points) Recall that when DFS is run on an undirected graph G, the trees in the associated DFS forest span the connected components of G. Using the Graph ADT operations in the project description for pa3, and the List ADT operations in the project description for pa1, write a *client* function with heading

$$\text{int numComponents(Graph } G\text{)}$$

that returns the number of connected components in G. For instance, if G is the graph

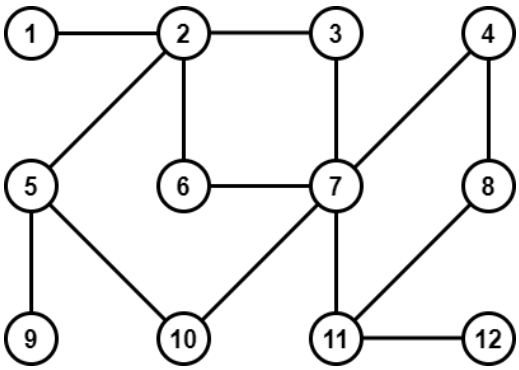


then numComponents(G) returns 3. Function numComponents() has no preconditions.

One of Several Possible Solutions:

```
int numComponents(Graph G){

   int i, count, n = getOrder(G);
   List S = newList();

   for( i=1; i<=n; i++){
      append(S, i);
   }

   DFS(G, S);

   count = 0;
   for( moveFront(S); position(S)>=0; moveNext(S) ){
      if( getParent(G, get(S))==NIL ){
         count++;
      }
   }
   freeList(&S);

   return count;
}
```

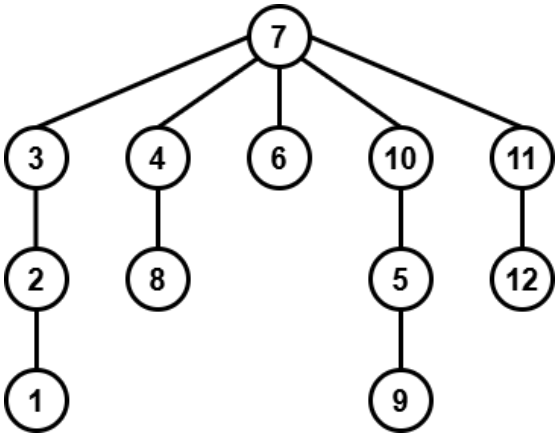
4. (20 Points) Run the BFS algorithm on the graph pictured below, with vertex $s = 7$ as the source. Fill in the table giving the adjacency list representation, colors, distances from the source, and parents in the BFS tree. List the vertices in the order that they enter the queue. Draw the resulting BFS tree.



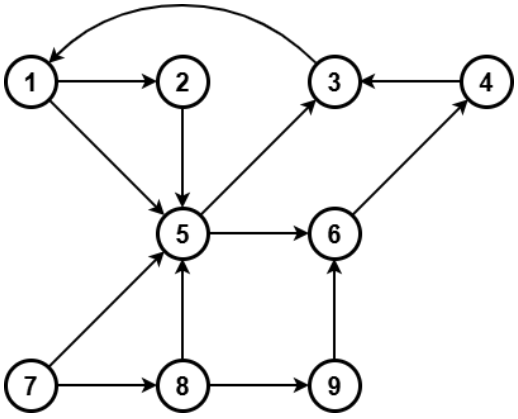
vertex	*adj*	*color*	*distance*	*parent*
1	2	black	3	2
2	1 3 5 6	black	2	3
3	2 7	black	1	7
4	7 8	black	1	7
5	2 9 10	black	2	10
6	2 7	black	1	7
7	3 4 6 10 11	black	0	nil
8	4 11	black	2	4
9	5	black	3	5
10	5 7	black	1	7
11	7 8 12	black	1	7
12	11	black	2	11

Queue: 7 3 4 6 10 11 2 8 5 12 1 9

BFS Tree:

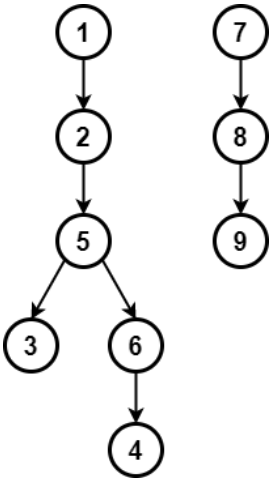


5. (20 Points) Run the DFS algorithm on the digraph pictured below. Process vertices in the main loop of DFS() by increasing vertex label. Process vertices in the for loop of Visit() by increasing vertex labels. As vertices finish, push them onto a stack. Fill in the table below giving the adjacency list representation, discover times, finish times and parents in the DFS forest. Draw the resulting DFS forest, and show the state of the stack when DFS is complete. Classify all edges as of type *tree*, *back* *forward* or *cross*.



vertex	*adj*	*discover*	*finish*	*parent*
1	2 5	1	12	nil
2	5	2	11	1
3	1	4	5	5
4	3	7	8	6
5	3 6	3	10	2
6	4	6	9	5
7	5 8	13	18	nil
8	5 9	14	17	7
9	6	15	16	8

DFS Forest:



Stack:

7
8
9
1
2
5
6
4
3

Edge Classification:

Tree: (1, 2) (2, 5) (5, 3) (5, 6) (6, 4) (7, 8), (8, 9)

Back: (3, 1)

Forward: (1, 5)

Cross: (4, 3) (7, 5) (8, 5) (9, 6)