CSE 101
Spring 2022
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

```
bool isPalindrome(List L)
```

Your function will return `true` if the integer sequence represented by *L* is a palindrome (i.e. is identical to its own reversal), and will return `false` if *L* is not a palindrome.

One of several possible solutions:

```
bool isPalindrome(List L){

   bool eq;
   List R = newList();
   for(moveFront(L); index(L)>=0; moveNext(L)){
      prepend(R, get(L));
   }
   eq = equals(L, R);
   freeList(&R);

   return eq;
}
```

Another solution:

```
bool isPalindrome(List L){

   bool eq = true;
   List C = copyList(L);
   while( eq && length(C)>1 ){
      eq = ( front(C)==back(C) );
      deleteFront(C);
      deleteBack(C);
   }
   freeList(&C);

   return eq;
}
```

2. (20 Points) Using only the List ADT operations defined in the project description for pa1, write a client function with the heading

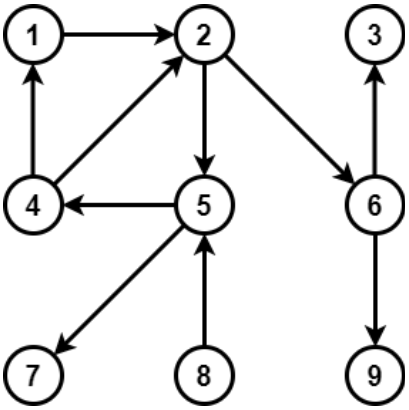
```
void Replace(List L, int x, int y)
```

Your function will replace all occurrences of x in L with y. If x is not in L, your function will make no changes to the integer sequence in L.

One of several possible solutions:

```
void Replace(List L, int x, int y){

   for(moveFront(L); index(L)>=0; moveNext(L)){
      if( get(L)==x ){
         set(L, y);
      }
   }
}
```

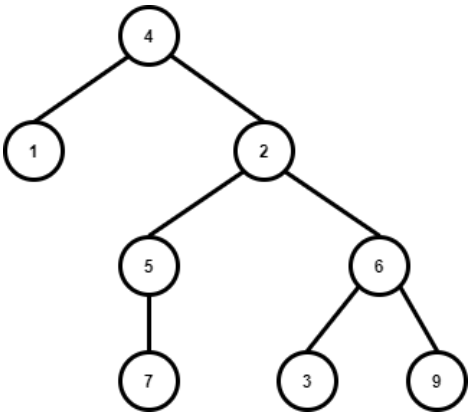
3. (20 Points) Run the BFS algorithm on the digraph pictured below, with vertex **4** as the source. Fill in the table giving the adjacency list representation, colors, distances from the source, and parents in the BFS tree. List the discovered vertices in the order that they enter the queue. Draw the resulting BFS tree.



vertex	*adj*	*color*	*distance*	*parent*
1	**2**	**black**	**1**	**4**
2	**5 6**	**black**	**1**	**4**
3		**black**	**3**	**6**
4	**1 2**	**black**	**0**	**nil**
5	**4 7**	**black**	**2**	**2**
6	**3 9**	**black**	**2**	**2**
7		**black**	**3**	**5**
8	**5**	**white**	**infinity**	**nil**
9		**black**	**3**	**6**

Queue: 4 1 2 5 6 7 3 9

BFS Tree:



4. (20 Points) Given a connected (undirected) graph G, the *diameter* of G is the maximum possible distance between any two vertices x and y in G, i.e.

$$\text{diameter}(G) = \max\{\, \delta(x, y) \mid x, y \in V(G) \,\}$$

Using only the Graph ADT functions defined in the project description for pa2, write a client function with the heading

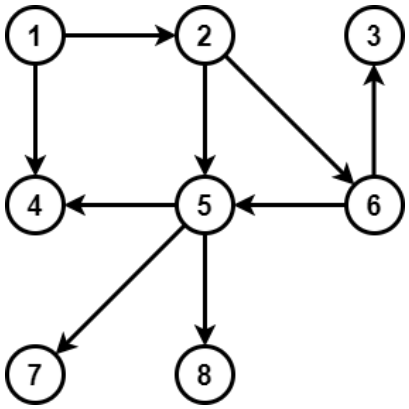
```
int diameter(Graph G)
```

Your function will compute and return the diameter of its input graph G.

One of several possible solutions:

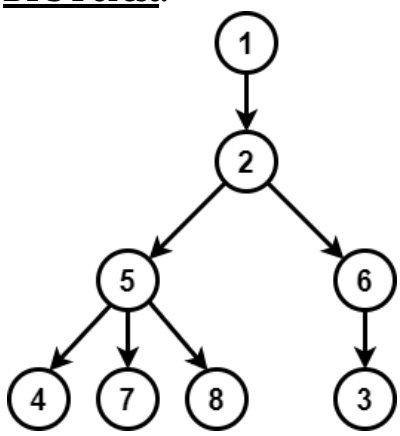
```
int diameter(Graph G){

   int max, d, x, y;

   max = 0;
   for (x=1; x<=getOrder(G); x++){
      BFS(G, x);
      for(y=1; y<=getOrder(G); y++){
         d = getDist(G, y);
         if( d>max ){
            max = d;
         }
      }
   }
   return max;

}
```

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 label. 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 4**	**1**	**16**	**nil**
2	**5 6**	**2**	**15**	**1**
3		**12**	**13**	**6**
4		**4**	**5**	**5**
5	**4 7 8**	**3**	**10**	**2**
6	**3 5**	**11**	**14**	**2**
7		**6**	**7**	**5**
8		**8**	**9**	**5**

DFS Forest:



Stack: 1 2 6 3 5 8 7 4

Edge Classification:

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

Back:

Forward: (1, 4)

Cross: (6, 5)