mid1

● Graded

Student

Cole Schreiner

Total Points

92.16 / 100 pts

Question 1

Q1

20 / 20 pts

✔ **– 0 pts** Correct

– 2 pts **Incorrectly looping through the list**

– 2 pts **Did not call moveFront on L**

– 2 pts **General Error**
Incorrect Function usage (For example : Wrong parameters, Incorrect parameter usage, object oriented method calling like List.Method())

– 4 pts **Returning value in place of index**
Example: returned the value rather than the index(correct index)

– 10 pts **Returned the wrong index**
Returning the incorrect index,
For example:
1. Consider given list in question, and solve for Search(L,1) - returning answer as 5 (Index of second "1" in List) . Where the correct index is of 3 (first occurrence)
2. Returning index of another value.

– 16 pts **Incorrect**

– 20 pts **No Solution**

Question 2

Q2

20 / 20 pts

✔ **– 0 pts** Correct

– 2 pts **Loop condition is incorrect**

– 2 pts **Resulting index was not initialized**

– 2 pts **Did not call moveFront on L**

– 2 pts **General Error**

The solution has an incorrect/improper statements or declarations.
1. Not returning the resulting list
2. Incorrectly using one or more ADT functions

– 10 pts **Incorrectly adding elements into resulting list**

1. The method used for inserting elements into list was incorrect. For example, this could have been appending wrong value.
2. Incorrectly removing the elements from list(Removing the non-duplicates)
3. Incorrect Sequence in resulting list compared to given list.

– 16 pts **Incorrect Solution**

Solution does not make sense

– 20 pts **No Solution**

Question 3

Q3

18 / 20 pts

– **0 pts** Correct

– **2 pts** **Wrong Format/Size of returned array**
The size of the array returned should be n+1.

– **2 pts** **Incorrectly looping through the list**

✔ – **2 pts** **Incorrect Use of Malloc/Calloc**
Malloc/Calloc was incorrectly used. For example, the solution provided may allocate the incorrect number of bytes $(n+1) \cdot sizeof(int)$.

– **2 pts** **General Error**
The solution has an incorrect/improper statements or variable declarations.
1. Not returning the resulting array.
2. Incorrectly using one or more ADT functions.
3. Not allocating the array to be returned.

– **2 pts** **Function getDist is called on wrong vertex**
The solution attempts to find the distance between the source and the incorrect or unintended vertex.

– **5 pts** **Incorrectly Calling BFS**
The solution must call BFS once per vertex in the graph. If you only called BFS once or you called BFS on the wrong vertex each time, your solution cannot be correct.

– **5 pts** **Incorrectly Updating the max distance**
The solution must only update the max distance variable when a valid path from the source node x to any other reachable vertex is longer than a previously traversed path. This can occur with improper conditional variables.

– **16 pts** **Incorrect Solution**

– **20 pts** **No Solution**

Question 4

Q4 **18.16** / 20 pts

4.1 Table **10.96** / 12 pts

– 0 pts No Solution

– 0 pts Correct answer
Each Cell Is Worth 0.26 Points

Num Cells Wrong = Points Lost
0 = -0.00
1 = -0.26
2 = -0.52
3 = -0.78
4 = -1.04
5 = -1.30
6 = -1.56
7 = -1.82
8 = -2.08
9 = -2.34
10 = -2.60
11 = -2.86
12 = -3.12
13 = -3.38
14 = -3.64
15 = -3.90
16 = -4.16
17 = -4.42
18 = -4.68
19 = -4.94
20 = -5.20
21 = -5.46
22 = -5.72
23 = -5.98
24 = -6.24
25 = -6.50
26 = -6.76
27 = -7.02
28 = -7.28
29 = -7.54
30 = -7.80
31 = -8.06
32 = -8.32
33 = -8.58
34 = -8.84
35 = -9.10
36 = -9.36

– 1.04 pts 4 cells wrong

4.2 **Queue** 4 / 4 pts

– 0 pts Correct

✔ – 0.8 pts At most 2 out of order

– 1.6 pts At most 4 out of order

– 2.4 pts At most 6 out of order

– 3 pts More than 6 out of order

– 4 pts No Solution

+ 4 pts swapped

4.3 **Tree** 3.2 / 4 pts

– 0 pts Correct

✔ – 0.8 pts At most 2 out of order

– 1.6 pts At most 4 out of order

– 2.4 pts At most 6 out of order

– 3 pts More than 6 out of order

– 4 pts No Solution

Question 5

Q5 **16** / 20 pts

5.1 **Table** **8** / 8 pts

✔ **– 0 pts** Correct

Each Cell Is Worth 0.177 Points

Num Cells Wrong = Points Lost
0 = -0.00
1 = -0.18
2 = -0.35
3 = -0.53
4 = -0.71
5 = -0.89
6 = -1.06
7 = -1.24
8 = -1.42
9 = -1.59
10 = -1.77
11 = -1.95
12 = -2.12
13 = -2.30
14 = -2.48
15 = -2.65
16 = -2.83
17 = -3.01
18 = -3.19
19 = -3.36
20 = -3.54
21 = -3.72
22 = -3.89
23 = -4.07
24 = -4.25
25 = -4.42
26 = -4.60
27 = -4.78
28 = -4.96
29 = -5.13
30 = -5.31
31 = -5.49
32 = -5.66
33 = -5.84
34 = -6.02
35 = -6.19
36 = -6.40

– 8 pts **No Solution**

5.2 **Stack** **2.4** / 4 pts

– 0 pts Correct

– 0.8 pts At most 2 out of order

✔ – 1.6 pts At most 4 out of order

– 2.4 pts At most 6 out of order

– 3 pts More than 6 out of order

– 4 pts No Solution

5.3 **DFS Forrest** **4** / 4 pts

✔ – 0 pts Correct

– 0.8 pts At most 2 out of order

– 1.6 pts At most 4 out of order

– 2.4 pts At most 6 out of order

– 3 pts More than 6 out of order

– 4 pts No Solution

5.4 **Edge Classification** **1.6** / 4 pts

– 0 pts Correct

– 0.8 pts At most 2 out of order

– 1.6 pts At most 4 out of order

✔ – 2.4 pts At most 6 out of order

– 3 pts More than 6 out of order

– 4 pts No Solution

CSE 101
Winter 2024
Midterm Exam 1

Name: Cole Schreiner

Email: CaSchrei **@ucsc.edu**

This exam is presented *as is*, which means that *no explanations or clarifications will be given during the exam period.* If you do not understand a problem, do the best work you can and state any assumptions you make. Read all instructions carefully, show all work and, unless otherwise stated, justify all answers. This is a closed book, closed note exam. Calculators are allowed, but not necessary. **Note that the back pages of this exam will not be scanned, so any part of a solution you place on a back page will not be graded.**

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

   int Search(List L, int x))

   If the int x is an element of List L, your function will return the first (i.e. leftmost) index in L at which x occurs. If x is not an element of L, the function will return -1, indicating that the target x was not found. For instance, if $L = [5, 6, 4, 1, 3, 1, 2]$, then Search$(L, 1)$ will return 3, and Search$(L, 7)$ will return -1. Function Search() will make no changes to the sequence represented by its argument L, and has no preconditions.

```
int Search (List L, int x)
{

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

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

List Unique(List L)

Your function will return a new List containing the unique elements of List L, with no repetition. The returned List will retain the leftmost occurrence of each unique element in L, and discard the other occurrances. For instance, if $L = [4, 3, 2, 2, 1, 1, 1, 3, 4, 1]$, then Unique($L$) will return $[4, 3, 2, 1]$. Unique() will make no changes to the sequence represented by its List L, and has no preconditions. (Hint: you may use function Search() from Problem 1).

```
List Unique(List L)
{
   List U = newList();
   moveFront(L)
   if( length(U)<=0)
   { append(U, get(L)); }
   for(moveFront(L); index(L)>=0; moveNext(L))
   {
      int x = get(L);
      Search(U, x);
      if(Search(U,x)==-1)
      {
         append(U,x);
      }
   }
   return U;
}
```

3. (20 Points) Given a connected (undirected) graph G, and a vertex x in G, the *eccentricity* of x is the maximum possible distance from x, to any other vertex y in G, i.e.

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

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

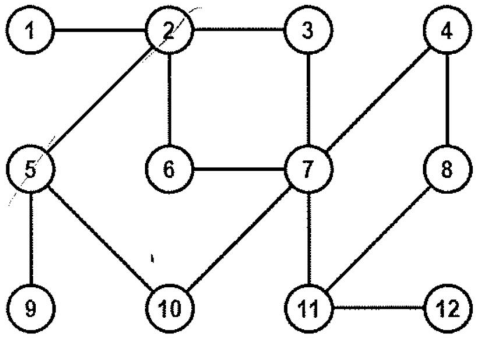
$$\text{int} * \ \text{Eccentricities(Graph } G\text{)}$$

Your function will return a newly allocated int array of length $n + 1$, where n is the number of vertices in G. For each x in the range $1 \leq x \leq n$, index x of the returned array will contain eccentricity(x). Index 0 may contain anything, i.e. it is not used. For instance, if G is the graph

then the returned array will be of length 6, and will contain ($*$, 4, 3, 2, 3, 4), where $*$ denotes arbitrary contents.

```
int * Eccentricities (Graph G)
{
  int max, y;
  BFS(G, x);
  max = getDist(G, 1)
  for (y = 2; y <= getOrder(G); y++)
  {
    if (getDist(G, y) > max)
    {
      max = getDist(G, y);
    }
  }
  return max;
}
```

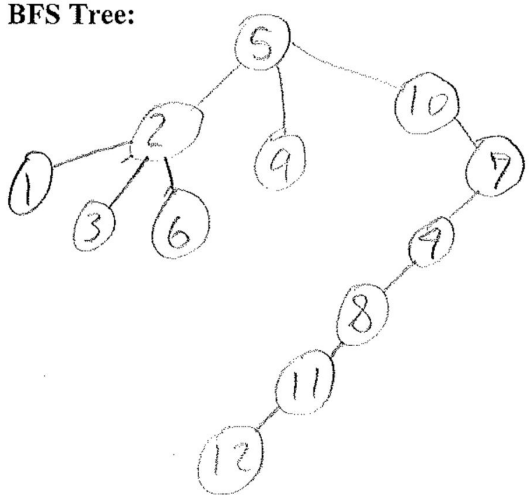
4. (20 Points) Run the BFS algorithm on the graph pictured below, with vertex $s = 5$ as the source. Fill in the table giving the adjacency list representation, colors, distances from the source, and parents in the BFS tree. List the vertices in the order that they enter the queue. Draw the resulting BFS tree.



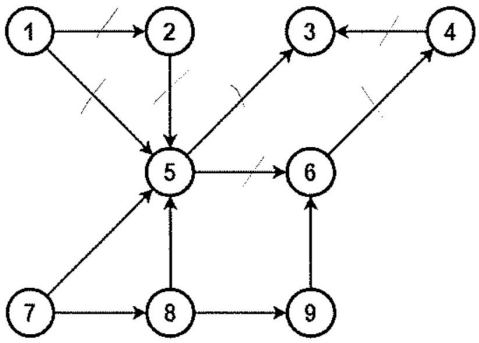
vertex	*adj*	*color*	*distance*	*parent*
→1	2			
→2	1 3 5 6			
→3	2 7			
→4	7 8			
→5	2 9 10		0	
→6	2 7			
→7	3 4 6 10			
→8	4 11			
→9	5			
→10	5 7			
→11	7 8 12			
→12	11			

Queue:

BFS Tree:



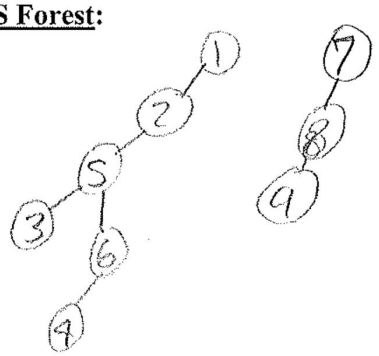
5. (20 Points) Run the DFS algorithm on the digraph pictured below. Process vertices in the main loop of DFS() by increasing vertex label. Process vertices in the for loop of Visit() by increasing vertex labels. As vertices finish, push them onto a stack. Fill in the table below giving the adjacency 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	h
2	5	2	11	1
3		4	5	5
4	3	7	8	6
5	3 6	3	10	2
6	4	6	9	5
7	5 8	13	18	h
8	5 9	14	17	7
9	6	15	16	8

Time: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

DFS Forest:



Stack:

1
2
5
6
4
3
9
8
7

Edge Classification:

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

Back:

Forward: (1,5)

Cross: (3,6)