Short Questions

1. (30 points) You have given an undirected complete graph with n vertices numbered $1, 2, \ldots, n$. (A complete graph has an edge between every pair of vertices). For $i < j$, the cost of edge $c(i, j) = (i - 1) \times n + j$ where $i, j \in 1, 2, \ldots, n$. Find out the cost of the shortest path from a to b using the Dijkstra algorithm; $a, b \in 1, 2, \ldots, n$. The execution of Dijkstra starts at a.

   - What is the cost of the shortest path? (15M)
   - And what is the path? (5M)
   - How many `ExctractMin` operations will be required to get the shortest path? (10M)

2. (30 points) Given an undirected connected graph $G = (V, E)$, prove that for any given proper non-empty subset $X \subset V$, a CUT that partitions the graph into $X, V \backslash X$, the lightest edge (the edge with the least cost; assume only one such edge for any cut) in the cut is part of an MST.

3. (40 points) Prove that the height of the AVL tree is $O(\log N)$ where N is the number of nodes inserted in an AVL tree.
(**HINT**: Argue $S(h)$: the minimum number of nodes in an AVL tree with height h has similarities with the Fibonacci series. Then, get an approximate lower bound to $S(h)$ and, from there, prove the claim.)

4. (30 points) Describe the implementation of a queue using two stacks. In particular, give an algorithm (pseudo code is fine. No need to give complete C code) to implement enqueue and dequeue operations using two given stack data structures. You may optionally draw a diagram explaining your logic.

Long Questions

For Q5 and Q6, the tree ADT is as follows. If you are **writing** any other additional routines, please mention them appropriately in line 22-34 below.

```
1  #ifndef TREE_H__ #define TREE_H__
2  #include <stdio.h>
3  typedef int Element;
4  typedef struct stTreeNode* BinTree;
5  typedef BinTree Position;
6  typedef BinTree BST;
7  struct stTreeNode {
8      Element Element;
9      BinTree Left;
10     BinTree Right;
11 };
12 // The next four routines are available as explained in
       the class. If required, you may use them without
       writing their code.
13 BinTree  MakeNode(Element X);
14 BinTree  Insert(BinTree T, Element X);
15 BinTree  Delete(BinTree T, Element X);
16 Position FindInTree(BinTree T, Element X);
17 // The following three routines need to be implmented
18 int GetNumNodes(BinTree T);
19 int GetNumLeaves(BinTree T);
20 // PreOrder and InOrder are the pointers to the arrays of
       n keys listing preorder and inorder traversal of the
       BST. n indicates the number of keys in the BST.
21 BinTree ConstructBinTree(Element *PreOrder, Element *
       InOrder, int n);
22
23
24
25
26
27
28
29
30
31
32
33
34
35 #endif
```

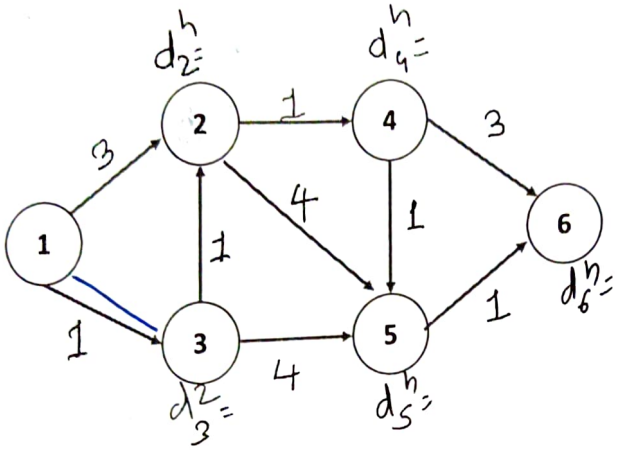
You may use the above page (pg 11) to draw any diagrams for Q5/6. These are optional. The page 11 will not be evaluated.

5. (40 points) Write code which takes a pointer to a binary tree (and not BST) as input and computes
   (a) the number of nodes. You need to write a routine `GetNumNodes`. (20M)
   (b) number of leaves. You need to write a routine `GetNumLeaves`. (20M)

6. *(80 points) Write code for* `ConstructBinTree` *that takes an preorder traversal and inorder traversal sequence for a binary search tree as input and reconstructs the tree. If you are writing additional routines to write the above function, explain them clearly.*

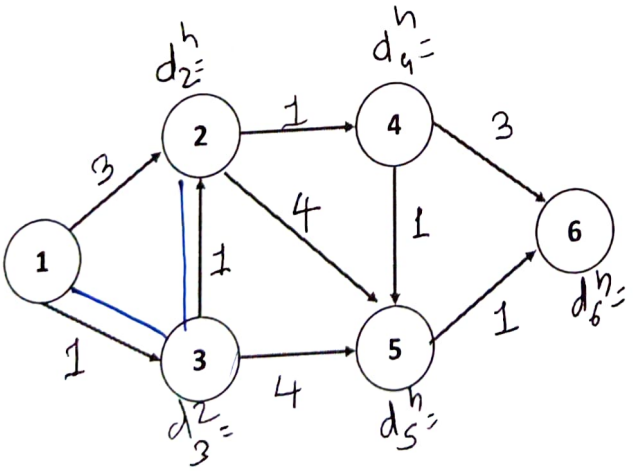
7. (40 points) Consider the following graph. Find out the shortest path from 1-6 using Bellman-Ford.

   - You need to explain how d_v^h and p_v are updated in each iteration. Note, d_v^h denotes the shortest path cost from source (in this case vertex 1) to v using atmost h edges. p_v denotes the vertex prior to v in the shortest path. (Start with h=1)



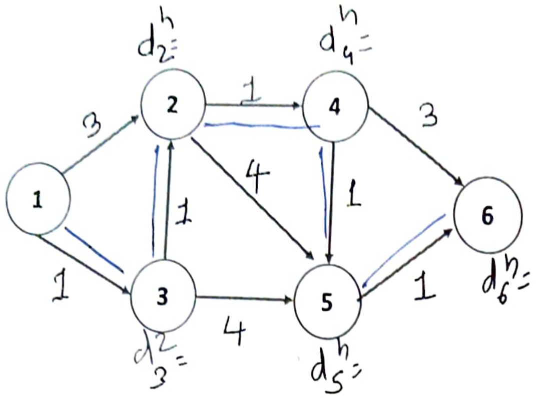
h=

v =	d_v^h	p_v
1		0
2		
3		
4		
5		
6		



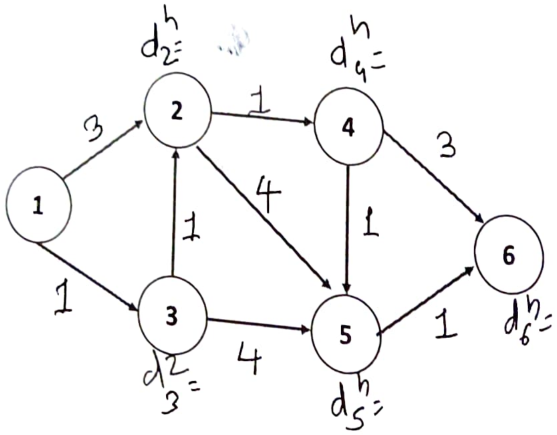
h=

v =	d_v^h	p_v
1		0
2		
3		
4		
5		
6		



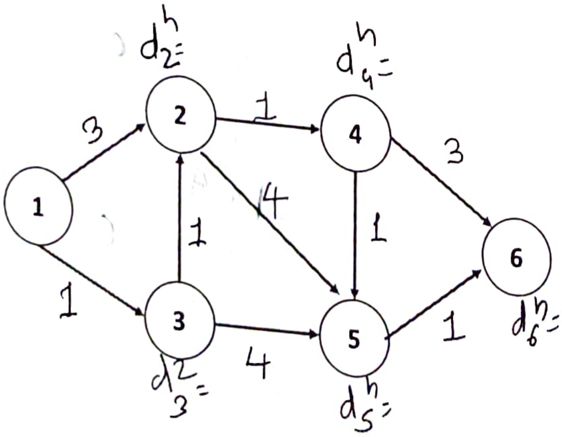
$h =$

$v =$	d_v^h	p_v
1		0
2		
3		
4		
5		
6		



$h =$

$v =$	d_v^h	p_v
1		0
2		
3		
4		
5		
6		



$h =$

$v =$	d_v^h	p_v
1		0
2		
3		
4		
5		
6		

8. (60 points) Graph ADT is as follows.

```
1  ifndef _GRAPH_H_
2  #define _GRAPH_H_
3  #include<stdio.h>
4
5  typedef struct stGraph * Graph;
6  typedef struct stNode * Node;
7  typedef int Vertex;
8
9  #define WHITE  1
10 #define GRAY   0
11 #define BLACK -1
12 #define INFTY  10000
13 #define UNKNOWN -1
14 #define NOTVERTEX -1
15
16 Graph   CreateGraph(int iNumber_of_vertices);
17 void    InsertEdge(Graph G, Vertex u, Vertex v);
18
19 struct stGraph{
20     int iN;
21     Node pVertex;
22 };
23
24 struct stNode{
25     Vertex iVertexID;
26     Node pNext;
27 };
28
29 //d[i][j] indicates the distance of edge i to j. d[i][j] =
       INFTY if there is no edge i->j
30 int LongestPath(Graph G, float **d);
31
32
33
34
35
36
37
38
39
40
41
42
43
44 #endif
```

Write a C routine, `LongestPath`, to determine the longest path length in a given directed graph. Assume the graph is acyclic. (**HINT:** Try to do a traversal as long as possible with appropriate pointer passing to the routines that you are writing and do proper bookkeeping.) If you are writing additional routines to write the above function, explain them clearly in line 31-43 above.