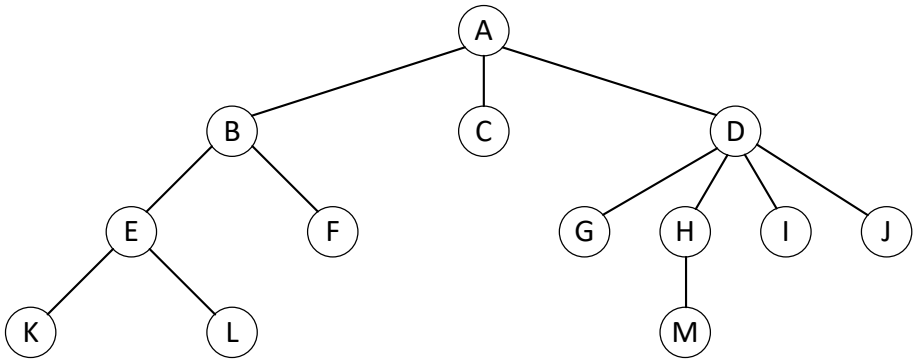
Fall 2021 CSE3080 Data Structures
Final Exam

※ Write your answers on the answer sheet. Make sure your answers are clearly recognizable.

1. Tree: concepts (6 points)

Consider the following tree.



(1) What is the degree of the tree? (2 pts)

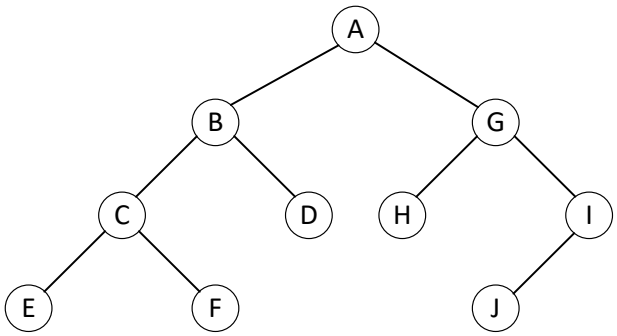
(2) How many leaf nodes are in the tree? (2 pts)

(3) Fill in the blank. (2 pts)

A ( ) is a binary tree in which all levels are completely filled except possibly the last level, and the last level has all keys as left as possible.

2. Binary Tree Traversal (8 points)

Consider the following binary tree.



The binary tree is represented using the following data structure.

```
struct node {
    char data;                  // the alphabet in the circle
    struct node *left_child;    // points to the left child
    struct node *right_child;   // points to the right child
};
typedef struct node *tree_pointer;
```

(1) Suppose we call the following function **A** and pass the tree as the argument. (`ptr` will be the pointer to the root.) What will be printed on the screen? (4 pts)

```
void A(tree_pointer ptr) {
    if(ptr) {
        A(ptr->left_child);
        printf("%c ", ptr->data);
        A(ptr->right_child);
    }
}
```

(2) Suppose we call the following function **B** and pass the tree as the argument. (`ptr` will be the pointer to the root.) What will be printed on the screen? (4 pts)

```
void B(tree_pointer ptr) {
    if(ptr) {
        B(ptr->left_child);
        B(ptr->right_child);
        printf("%c ", ptr->data);
    }
}
```

3. Graph: Concepts (20 points)

(1) A(n) (unidirected) graph is a graph whose edge is represented by a pair (u, v); if (u, v) is an edge in the graph, (v, u) is also an edge in the graph. The edges are drawn without arrows. (2 pts)

(2) Suppose a graph has *n* vertices. If the graph has *n(n-1)/2* edges, the graph is called a ( ). (2pts)

(3) A ( ) from vertex u to vertex v in graph G is a sequence of vertices u, i_1, i_2, ..., i_k, v such that (u, i_1), (i_1, i_2), ..., (i_k, v) are edges in G. (2 pts)

(4) A ( ) is a **(3)** in which all vertices except possibly the first and the last are distinct. (2 pts)

(5) A ( ) is a **(4)** in which the first and the last vertices are the same. (2 pts)

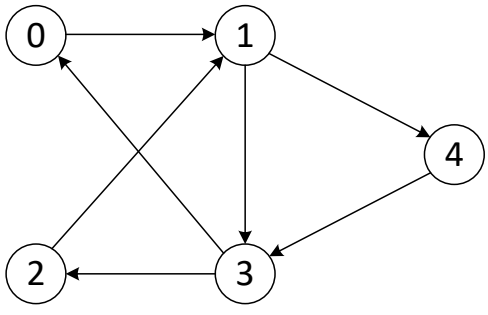
(6) A ( ) is a connected acyclic graph. (2 pts)

(7) A ( ) of a vertex is the number of edges incident to that vertex. (2 pts)

(8) Represent **Graph A** using an **adjacency list**. The adjacency list should be represented as an array of linked lists. In a linked list, the vertices must be sorted in the ascending order of the vertex ID. (Draw the adjacency list.) (3 pts)

(9) Represent **Graph A** using an **adjacency matrix**. (Draw the adjacency matrix.) (3 pts)

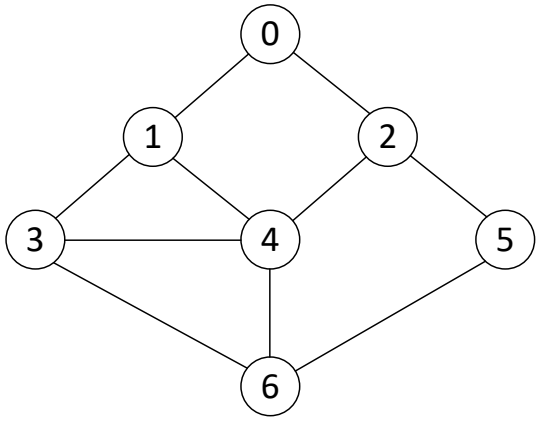
Graph A



4. Graph Search (8 points)

(1) We are going to do a depth-first search (DFS) on **Graph B** starting from vertex 0. Write the vertex IDs in the order they are visited. When there are multiple candidate vertices to visit next, we choose the one with the lower vertex ID. (e.g., If both vertex 1 and vertex 2 can be the next node to visit, we visit vertex 1.) (4 pts)

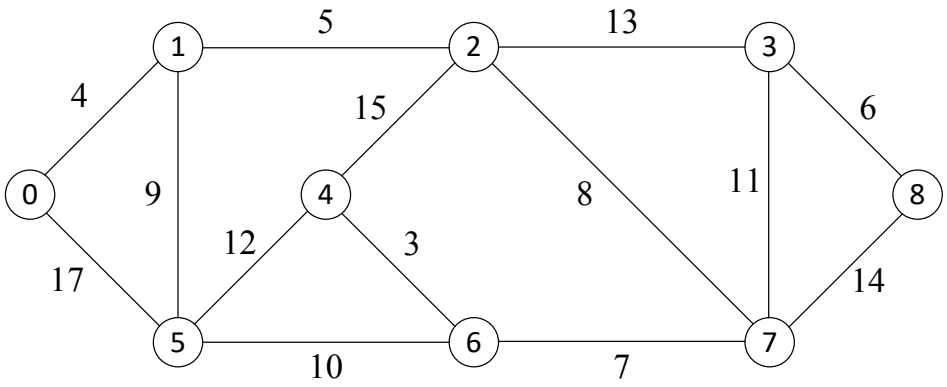
Graph B



(2) We are going to do a breadth-first search (BFS) on **Graph B** starting from vertex 0. Write the vertex IDs in the order they are visited. When there are multiple candidate vertices to visit next, we choose the one with the lower vertex ID. (e.g., If both vertex 1 and vertex 2 can be the next node to visit, we visit vertex 1.) (4 pts)

5. Minimum Spanning Trees (10 points)

Graph C



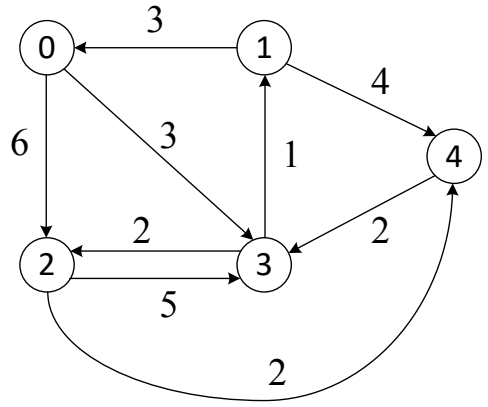
(1) For **Graph C**, find the minimum spanning tree using **Prim's algorithm,** starting from vertex 0. Write the **order** of edges included in the minimum spanning tree. You can write the edge between vertex **i** and **j** as **(i, j)**. (5 pts)

(2) For **Graph C**, find the minimum spanning tree using **Kruskal's algorithm**. Write the **order** of edges included in the minimum spanning tree. (5 pts)

6. Single Source Shortest Paths – Nonnegative Weights (10 points)

For **Graph D,** we would like to find the shortest path from vertex 0 to all other vertices using Dijkstra's algorithm. Initially (Step 0), the set **SPT** includes vertex 0, and the **distance table** is as given below.

Graph D



Step 0: SPT = { 0 }

vertex	distance
1	INF
2	6
3	3
4	INF

(1) Write SPT and distance table after Step 1. (2.5 pts)

(2) Write SPT and distance table after Step 2. (2.5 pts)

(3) Write SPT and distance table after Step 3. (2.5 pts)

(4) Write SPT and distance table after Step 4. (2.5 pts)

7. Single Source Shortest Paths – General Weights (10 points)

For **Graph E,** we would like to find the shortest path from vertex 0 to all other vertices using Bellman-Ford algorithm.

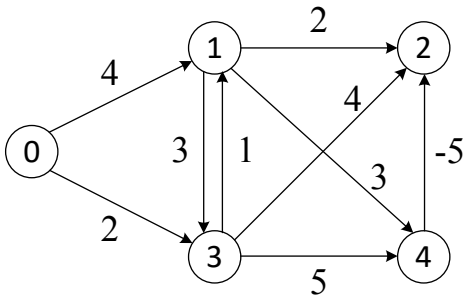
In each step, the edges are processed in the following order.

(4, 2) – (3, 4) – (3, 2) – (3, 1) – (1, 4) – (1, 3) – (1, 2) – (0, 3) – (0, 1)

Each row in the table represents the distance of the shortest path to each vertex after k-th step.

Fill in the table to complete the algorithm.

Graph E

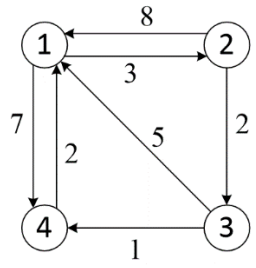


k	1	2	3	4
0	inf	inf	inf	inf
1				
2				
3				
4				

8. All-Pairs Shortest Paths (12 points)

For **Graph F,** we would like to find the shortest path between all pairs of vertices. We define a matrix $A^k[][]$, that represents the length of shortest paths from a source vertex to a destination vertex. $A^k[i][j]$ is defined as the shortest path length from vertex i to vertex j, only taking vertices 1 through k as intermediate vertices. (The vertex numbering starts at 1.)

Graph F



Initially (Step 0), A^0 matrix looks like the following.

	1	2	3	4
1	0	3	inf	7
2	8	0	2	inf
3	5	inf	0	1
4	2	inf	inf	0

(1) Write A^1 after adding vertex 1 as an intermediate vertex. (3 pts)

(2) Write A^2 after adding vertex 2 as an intermediate vertex. (3 pts)

(3) Write A^3 after adding vertex 3 as an intermediate vertex. (3 pts)

(4) Write A^4 after adding vertex 4 as an intermediate vertex. (3 pts)

9. Sorting (8 points)

Following is an example code implementing Quick Sort.

```
#define SWAP(a,b) {int tmp; tmp = a; a = b; b = tmp;}
void quicksort(int A[], int left, int right) {
    int pivot;
    if(right – left > 0) {
        pivot = partition(A, left, right);
        quicksort(A, left, pivot-1);
        quicksort(A, pivot+1, right);
    }
}
int partition(int A[], int left, int right) {
    int i, pivot;
    pivot = left;
    for(i = left; i < right; i++) {
        if(A[i] < A[right]) {
            SWAP(A[i], A[pivot]);
            pivot++;
        }
    }
    SWAP(A[right], A[pivot]);
    return pivot;
}
```

(1) Suppose the input array A looks like the following:

50	91	19	74	2	4	52	17	80	44

What is the return value of function **partition** when A is the array above, left is 0 and right is 9? In other words, what is the return value of `partition(A, 0, 9)`? (4 pts)

(2) What does array A look like after the first call to function **partition**? We assume that the input array is the same as in problem (1). Describe the contents of the array. (4 pts)

10. Hashing (8 points)

Suppose we are trying to store integers in a table of 13 buckets using hashing. The hash function we use is "modulo 13" which means we divide the integer by 13, and the remainder will be the bucket address for the integer. (We assume each bucket has a single slot.)

For overflow handling, we use linear probing.

Starting from an empty bucket, we store the following 13 integers in the order from left to right.

79 – 50 – 75 – 96 – 70 – 83 – 85 – 88 – 10 – 90 – 4 – 59 – 86

Fill in the bucket below to represent the contents of the bucket after all 13 integers are stored. (1 pt deducted for each wrong value.)

addr	value
[0]	
[1]	
[2]	
[3]	
[4]	
[5]	
[6]	
[7]	
[8]	
[9]	
[10]	
[11]	
[12]	

- End of the Exam -

Great work! Thanks for taking the course.