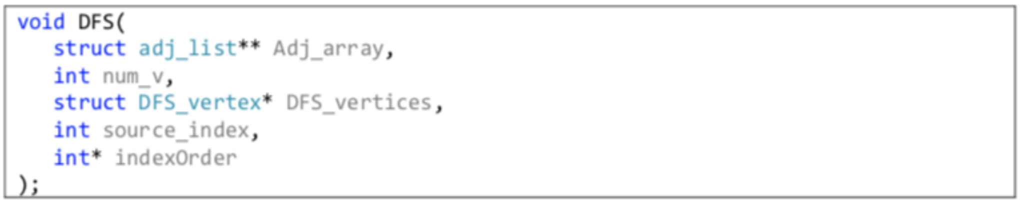
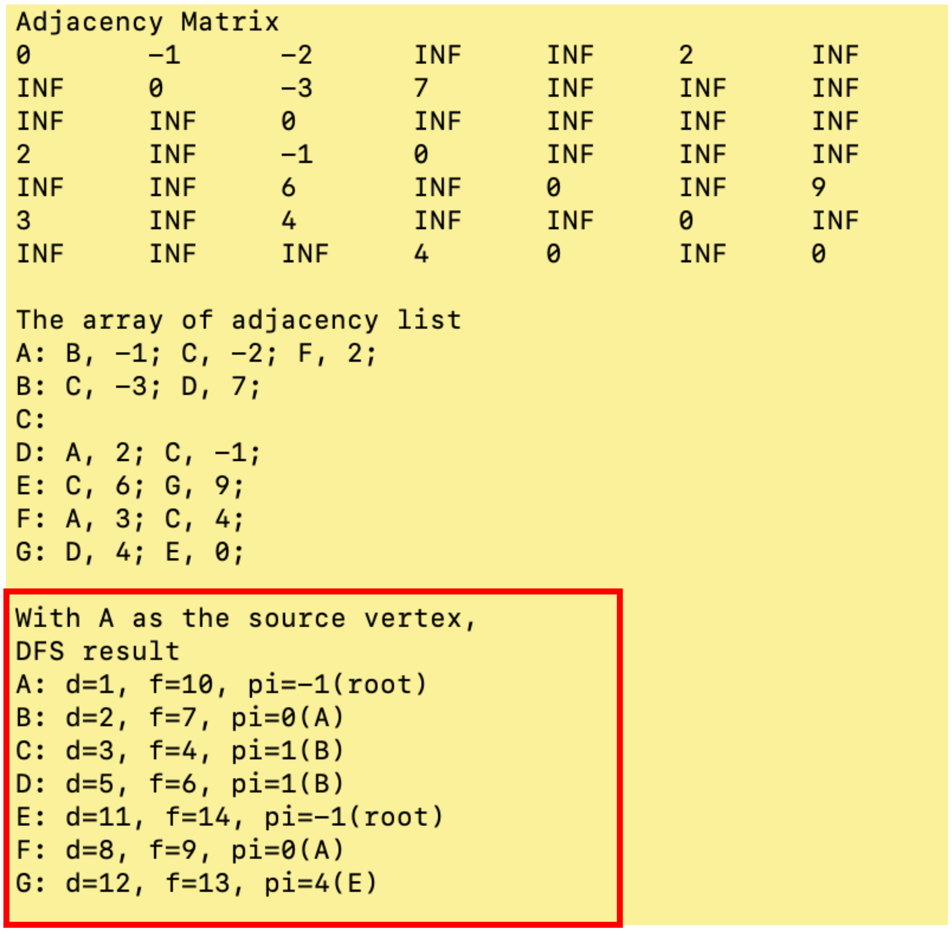
1. Implement DFS () function that does Depth-First Search algorithm with a source vertex, and print result by calling printDFS () function.

   In the template code, the DFS() function is declared as



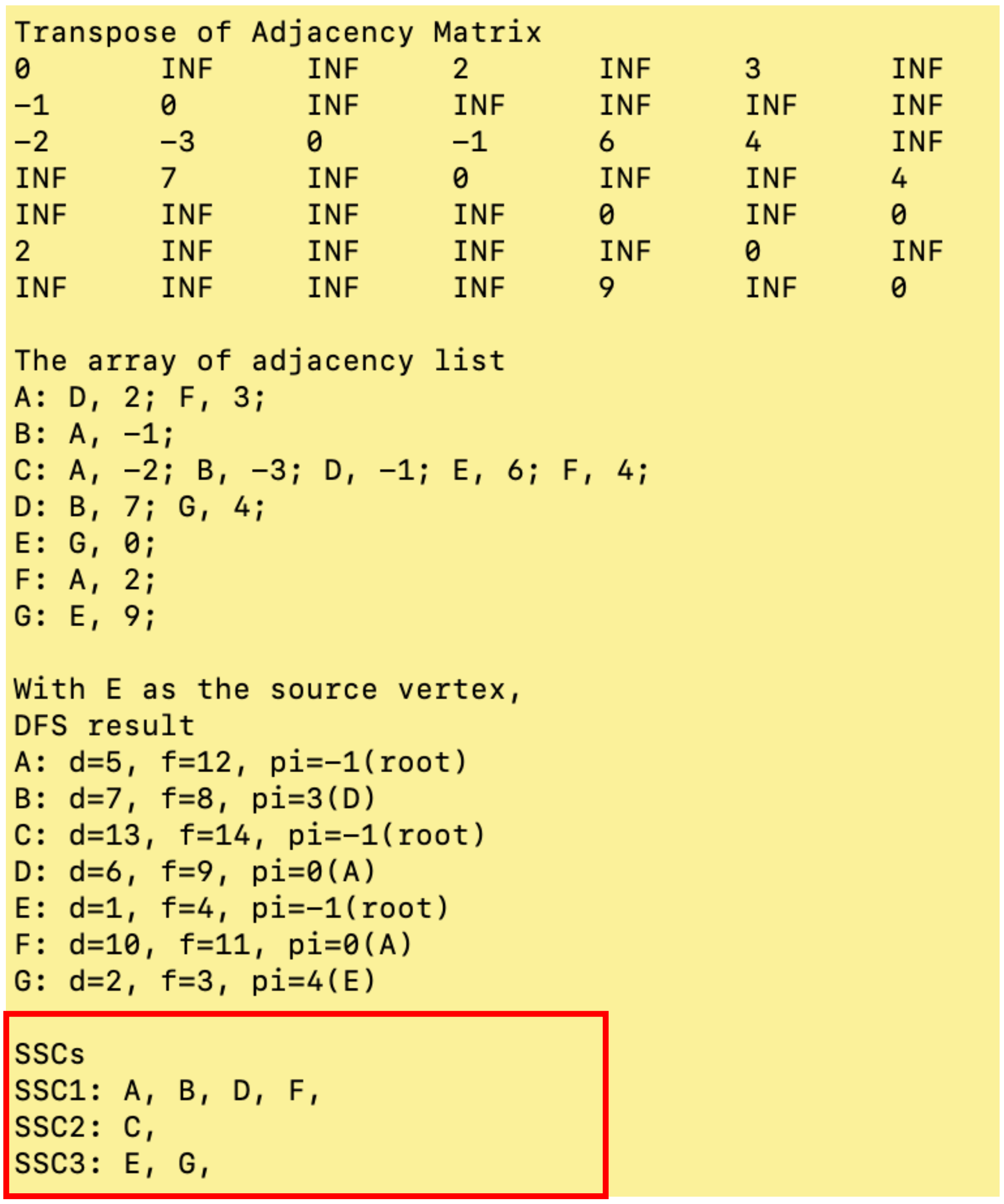
```
void DFS(
    struct adj_list** Adj_array,
    int num_v,
    struct DFS_vertex* DFS_vertices,
    int source_index,
    int* indexOrder
);
```

   where Adj_array is the *Adj* array; num_v is the number of all vertices; DFS_vertices is an array of a structure in order to store discovery time, finish time, color, and index of parent vertex of each vertex; source_index is the index of source vertex; and indexOrder is the order of index which is usually in increasing order representing alphabetical order but sometimes it is needed to indicate a specific order of vertices, such as doing second DFS with the largest finish time to find Strongly Connected Components (SSCs).



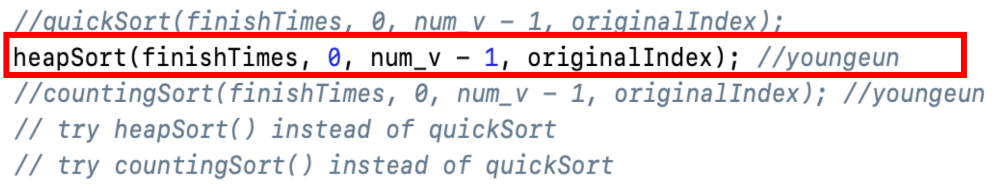
```
Adjacency Matrix
0       -1      -2      INF     INF     2       INF
INF     0       -3      7       INF     INF     INF
INF     INF     0       INF     INF     INF     INF
2       INF     -1      0       INF     INF     INF
INF     INF     6       INF     0       INF     9
3       INF     4       INF     INF     0       INF
INF     INF     INF     4       0       INF     0

The array of adjacency list
A: B, -1; C, -2; F, 2;
B: C, -3; D, 7;
C:
D: A, 2; C, -1;
E: C, 6; G, 9;
F: A, 3; C, 4;
G: D, 4; E, 0;

With A as the source vertex,
DFS result
A: d=1, f=10, pi=-1(root)
B: d=2, f=7, pi=0(A)
C: d=3, f=4, pi=1(B)
D: d=5, f=6, pi=1(B)
E: d=11, f=14, pi=-1(root)
F: d=8, f=9, pi=0(A)
G: d=12, f=13, pi=4(E)
```

2. To find SSCs, you need to do second DFS with the transpose of the graph according to the finish times obtained from the first DFS. The procedure is already included in the main() function. Understand the code; implement int findSSCs(DFS_vertices_second, num_v, SSCs) function that actually finds SSCs, saves them in array SSCs, and returns the number of SSCs found; and print result by calling printSSCs() function.

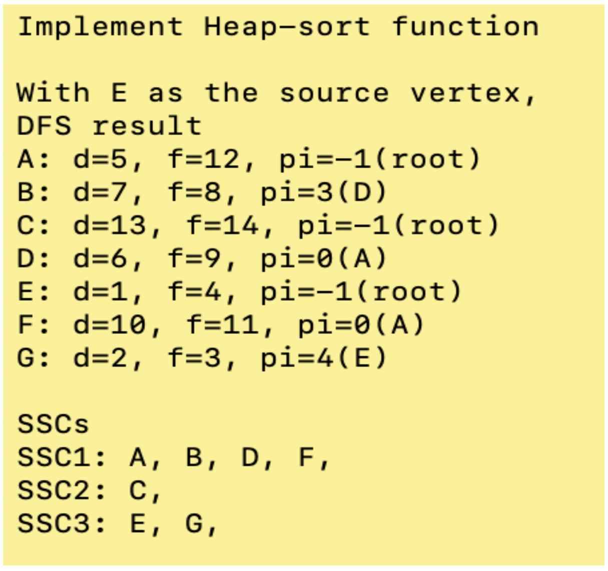


```
Transpose of Adjacency Matrix
0       INF     INF     2       INF     3       INF
-1      0       INF     INF     INF     INF     INF
-2      -3      0       -1      6       4       INF
INF     7       INF     0       INF     INF     4
INF     INF     INF     INF     0       INF     0
2       INF     INF     INF     INF     0       INF
INF     INF     INF     INF     9       INF     0

The array of adjacency list
A: D, 2; F, 3;
B: A, -1;
C: A, -2; B, -3; D, -1; E, 6; F, 4;
D: B, 7; G, 4;
E: G, 0;
F: A, 2;
G: E, 9;

With E as the source vertex,
DFS result
A: d=5, f=12, pi=-1(root)
B: d=7, f=8, pi=3(D)
C: d=13, f=14, pi=-1(root)
D: d=6, f=9, pi=0(A)
E: d=1, f=4, pi=-1(root)
F: d=10, f=11, pi=0(A)
G: d=2, f=3, pi=4(E)

SSCs
SSC1: A, B, D, F,
SSC2: C,
SSC3: E, G,
```

3. Implement Heap-sort function; replace quickSort () with the Heap-sort function; and show that it still finds the correct SSCs.

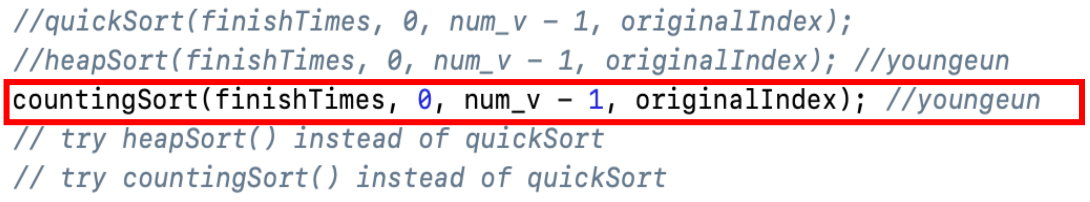


```
//quickSort(finishTimes, 0, num_v - 1, originalIndex);
heapSort(finishTimes, 0, num_v - 1, originalIndex); //youngeun
//countingSort(finishTimes, 0, num_v - 1, originalIndex); //youngeun
// try heapSort() instead of quickSort
// try countingSort() instead of quickSort
```

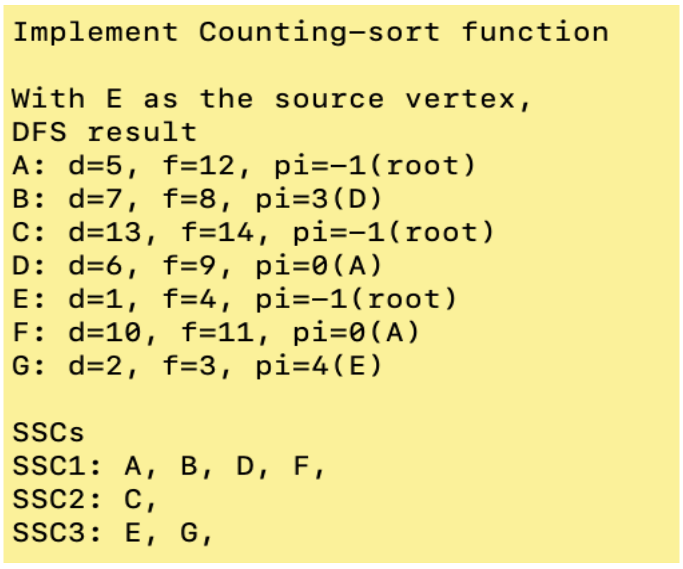


```
Implement Heap-sort function

With E as the source vertex,
DFS result
A: d=5, f=12, pi=-1(root)
B: d=7, f=8, pi=3(D)
C: d=13, f=14, pi=-1(root)
D: d=6, f=9, pi=0(A)
E: d=1, f=4, pi=-1(root)
F: d=10, f=11, pi=0(A)
G: d=2, f=3, pi=4(E)

SSCs
SSC1: A, B, D, F,
SSC2: C,
SSC3: E, G,
```

4. Implement Counting-sort function, replace quickSort () with the Counting-sort function, and show that it still finds the correct SSCs.

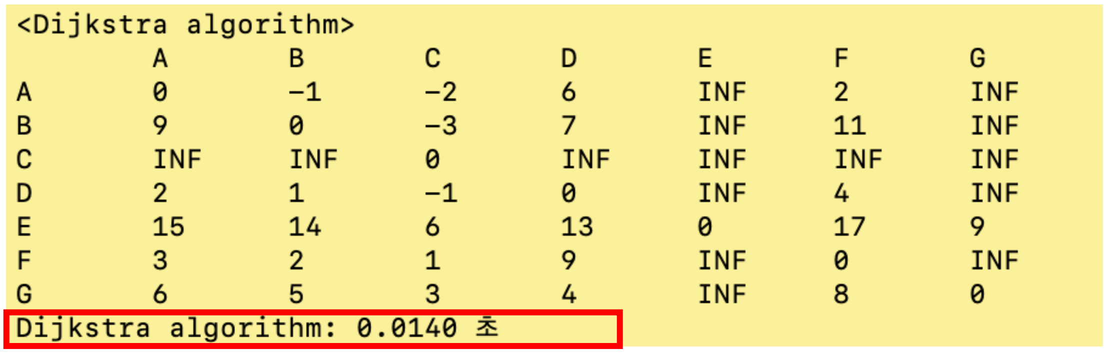


```
//quickSort(finishTimes, 0, num_v - 1, originalIndex);
//heapSort(finishTimes, 0, num_v - 1, originalIndex); //youngeun
countingSort(finishTimes, 0, num_v - 1, originalIndex); //youngeun
// try heapSort() instead of quickSort
// try countingSort() instead of quickSort
```



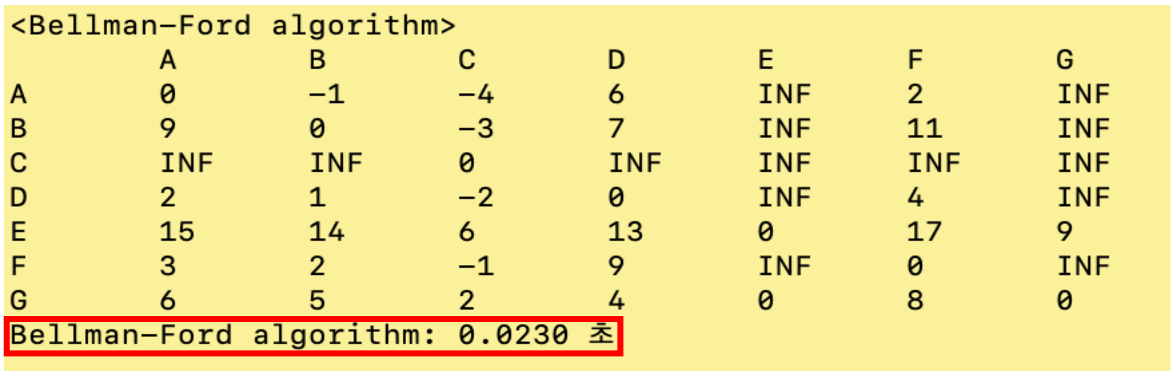
```
Implement Counting-sort function

With E as the source vertex,
DFS result
A: d=5, f=12, pi=-1(root)
B: d=7, f=8, pi=3(D)
C: d=13, f=14, pi=-1(root)
D: d=6, f=9, pi=0(A)
E: d=1, f=4, pi=-1(root)
F: d=10, f=11, pi=0(A)
G: d=2, f=3, pi=4(E)

SSCs
SSC1: A, B, D, F,
SSC2: C,
SSC3: E, G,
```

5. Follow the instructions below. (Sample output is shown in the next page)
   A. Implement Dijkstra's algorithm function; apply it for $|V|$ times to get all pairs shortest path; and print result (table & run time). (15 points)



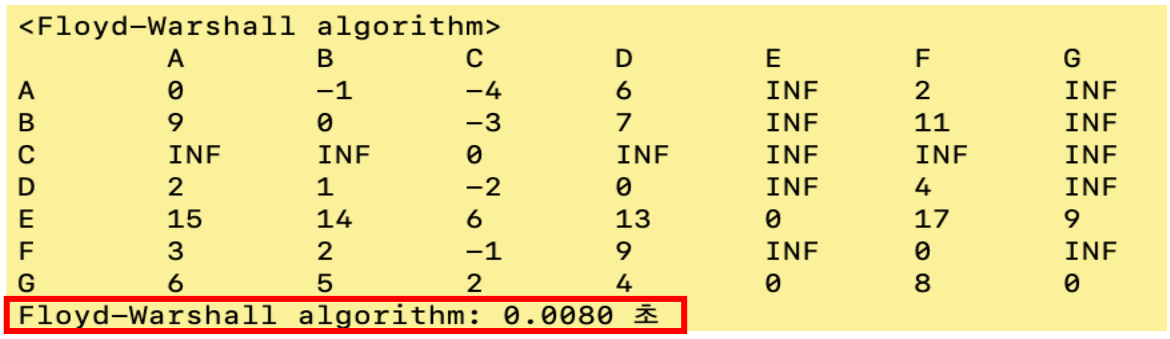
```
<Dijkstra algorithm>
        A       B       C       D       E       F       G
A       0       -1      -2      6       INF     2       INF
B       9       0       -3      7       INF     11      INF
C       INF     INF     0       INF     INF     INF     INF
D       2       1       -1      0       INF     4       INF
E       15      14      6       13      0       17      9
F       3       2       1       9       INF     0       INF
G       6       5       3       4       INF     8       0
Dijkstra algorithm: 0.0140 초
```

   B. Implement Bellman-Ford algorithm function; apply it for $|V|$ times to get all pairs shortest paths; and print result (table & run time). (15 points)



```
<Bellman-Ford algorithm>
        A       B       C       D       E       F       G
A       0       -1      -4      6       INF     2       INF
B       9       0       -3      7       INF     11      INF
C       INF     INF     0       INF     INF     INF     INF
D       2       1       -2      0       INF     4       INF
E       15      14      6       13      0       17      9
F       3       2       -1      9       INF     0       INF
G       6       5       2       4       0       8       0
Bellman-Ford algorithm: 0.0230 초
```

   C. Implement and run Floyd-Warshall algorithm function to get all pairs shortest paths, and print result (table & run time). (15 points)



```
<Floyd-Warshall algorithm>
        A       B       C       D       E       F       G
A       0       -1      -4      6       INF     2       INF
B       9       0       -3      7       INF     11      INF
C       INF     INF     0       INF     INF     INF     INF
D       2       1       -2      0       INF     4       INF
E       15      14      6       13      0       17      9
F       3       2       -1      9       INF     0       INF
G       6       5       2       4       0       8       0
Floyd-Warshall algorithm: 0.0080 초
```