**EE2410 Data Structure Coding HW #5 – Graphs (Chapter 6)**

**due date 6/9/2024 (Sun.), 23:59**

You should submit:

(a) All your source codes (C++ file).

(b) Show the execution trace of your program, i.e., write a client main() to demonstrate all functions you designed using example data.

Submit your homework before the deadline (midnight of 6/9). Fail to comply (**late** homework) will have ZERO score. **Copy** homework will have ZERO score on both parties and SERIOUS consequences.

1. (40%)

Graph (linked adjacency list), BFS, DFS, connected components, Computing dfn and low:

Write a C++ program to perform the following basic graph functions: (assume the graph is represented using linked adjacency list.)

1. BFS(v) (Prog. 6.2) (v: starting vertex. You need to output the vertices visited in BFS order)
2. DFS(v) (Prog. 6.1) (v: starting vertex. You need to output the vertices visited in DFS order)
3. Component() (Prog. 6.3 where OutputNewComponent() can be simplified to just output the vertices of the component)
4. DfnLow() (Prog. 6.4, 6.5) (Display the computed dfn[i] and low[i] of the graph and the articulation points found) on a linked adjacency list based graph. Add whatever you think necessary to your class Graph to implement the required functions, e.g., setup functions for setting up various graphs required.

Show your results using the following three graphs (*G*1, *G*2, and *G*3) in your program. The main() would contain similar codes segment shown below. BFS and DFS should start from 3 vertices: 0, 3, 7, respectively as shown in the code segment.

Graph g1(8),g2(8),g3(10);

g1.Setup1();

//BFS

g1.BFS(0);

g1.BFS(3);

g1.BFS(7);

//DFS

g1.DFS(0);

g1.DFS(3);

g1.DFS(7);

//Components & DfnLow

g1.Components();

g1.DfnLow(3);



1. (30%)

Shortest paths: single source/all destination nonnegative weights (Dijkstra), single source/all destination negative weights DAG (Bellman-Ford), all pairs shortest paths (Floyd)

Write a C++ program to perform some basic graph functions:

1. Single source/all destination nonnegative weights (Dijkstra) (Prog.6.8)
2. Single source/all destination negative weights DAG (Bellman-Ford) (Prog. 6.9)
3. All pairs DAG shortest paths (Floyd) (Prog. 6.10)

Assume the graph is represented using weighted adjacency matrix. Add whatever you think necessary to your class Graph to implement the required functions, such as setups for setting up various graphs required and display corresponding adjacency matrix of the graph.

You should demonstrate your code by applying these three functions to graphs given below.

For (a) Single source/all destination nonnegative weights (Dijkstra), modify Prog. 6.8 to generate results like Fig. 6.28 in textbook (shown below) and output the computed “paths”.

You need to demonstrate your code of (a) by processing: G1, G1’, and G1” shown below.





Fig. 6.28





1. G1”. Find shortest paths from vertex 0 to all remaining vertices.

For (b) Single source/all destination negative weights DAG (Bellman-Ford), modify Prog. 6.9 to display results like Fig. 6.31(b) shown below.

You need to demonstrate your code of (b) by processing: G2 and G2’ shown below.



Fig. 6.31



Fig. 6.29

For (c) All pairs DAG shortest paths (Floyd), modify Prog. 6.10 to display results like Fig. 6.32 shown below. You need to demonstrate your code of (c) by processing G3 (below in Fig. 6.32(a)) and G2 (above in Fig. 6.31(a)).

|  |  |
| --- | --- |
| (a) Digraph G3 |  |

Fig. 6.

1. (30%)

Write a C++ program that inputs (or setups) an AOE network and outputs the following:

1. Topological order
2. The earliest and latest times of all events (ee[i], le[i])
3. The earliest and latest times of all activities (e[k], l[k])
4. A table of all activities with their early and late times together with their slack and critical activities like Figure 6.41.
5. The critical network
6. Whether or not the project length can be reduced by speeding a single activity. If so, then by how much?

Use Figure 6.39 and 6.44 as two AOE examples to illustrate your results.



Figure 6.41

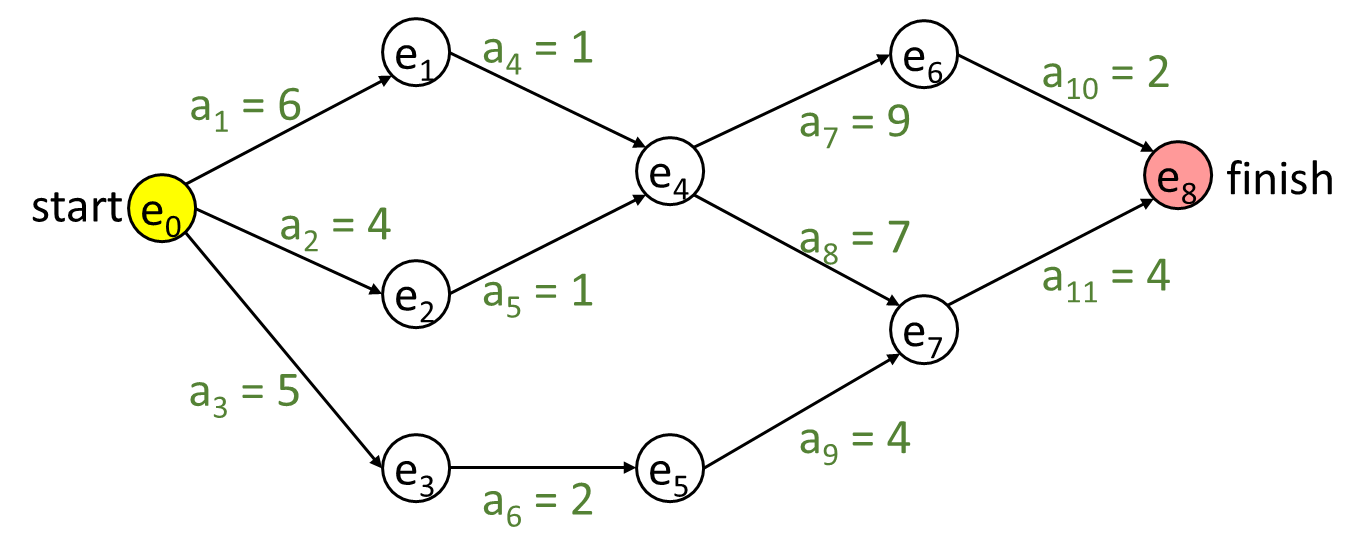


Figure 6.39



Figure 6.44