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.)

(a) BFS(v) (Prog. 6.2) (v: starting vertex. You need to output the vertices visited in BFS order)

(b) DFS(v) (Prog. 6.1) (v: starting vertex. You need to output the vertices visited in DFS order)

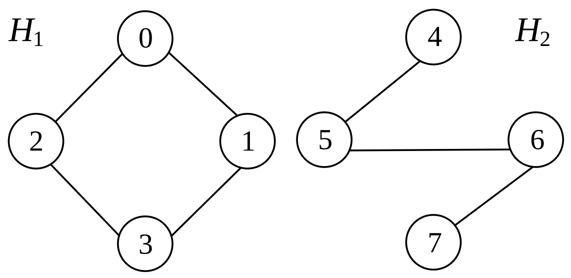
(c) Component() (Prog. 6.3 where OutputNewComponent() can be simplified to just output the vertices of the component)

(d) 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.

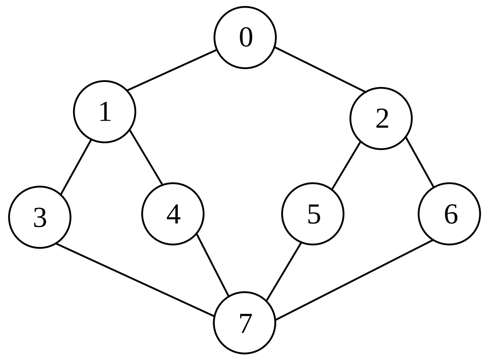
Show your results using the following three graphs (G_1, G_2, and G_3) in your program. The main() would contain similar codes segment shown below. BFS and DFS should start from 3 vertices: 0, 3, 7, respectively as shown in the code segment.

```
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
```

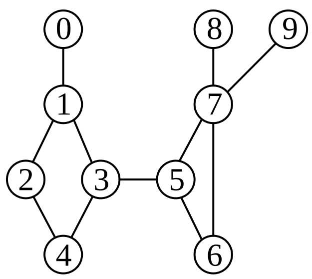
```
g1.Components();
g1.DfnLow(3);
```



G_1



G_2



G_3

C:\c++\data_structure\hw5\s

```
G1:
BFS:
Begin at vertex 0:
0 1 2 3
Begin at vertex 3:
3 1 2 0
Begin at vertex 7:
7 6 5 4

DFS:
Begin at vertex 0:
0 1 3 2
Begin at vertex 3:
3 1 0 2
Begin at vertex 7:
7 6 5 4

Components:
0 1 3 2
4 5 6 7

DfnLow:
Begin at vertex 3:
    i : 0 1 2 3 4 5 6 7
dfn[i]: 3 2 4 1 0 0 0 0
low[i]: 1 1 1 1 0 0 0 0
-------------------------------
G2:
```

```
G2:
BFS:
Begin at vertex 0:
0 1 2 3 4 5 6 7
Begin at vertex 3:
3 1 7 0 4 5 6 2
Begin at vertex 7:
7 3 4 5 6 1 2 0

DFS:
Begin at vertex 0:
0 1 3 7 4 5 2 6
Begin at vertex 3:
3 1 0 2 5 7 4 6
Begin at vertex 7:
7 3 1 0 2 5 6 4

Components:
0 1 3 7 4 5 2 6

DfnLow:
Begin at vertex 1:
    i : 0 1 2 3 4 5 6 7
dfn[i]: 2 1 3 6 7 4 8 5
low[i]: 1 1 1 1 1 1 3 1
-------------------------------
G3:
```

```
G3:
BFS:
Begin at vertex 0:
0 1 2 3 4 5 6 7 8 9
Begin at vertex 3:
3 1 4 5 0 2 6 7 8 9
Begin at vertex 7:
7 5 6 8 9 3 1 4 0 2

DFS:
Begin at vertex 0:
0 1 2 4 3 5 6 7 8 9
Begin at vertex 3:
3 1 0 2 4 5 6 7 8 9
Begin at vertex 7:
7 5 3 1 0 2 4 6 8 9

Components:
0 1 2 4 3 5 6 7 8 9

DfnLow:
Begin at vertex 3:
    i : 0 1 2 3 4 5 6 7 8 9
dfn[i]: 3 2 4 1 5 6 7 8 9 10
low[i]: 3 1 1 1 1 6 6 6 9 10

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

2. (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:

(a) Single source/all destination nonnegative weights (Dijkstra) (Prog.6.8)

(b) Single source/all destination negative weights DAG (Bellman-Ford) (Prog. 6.9)

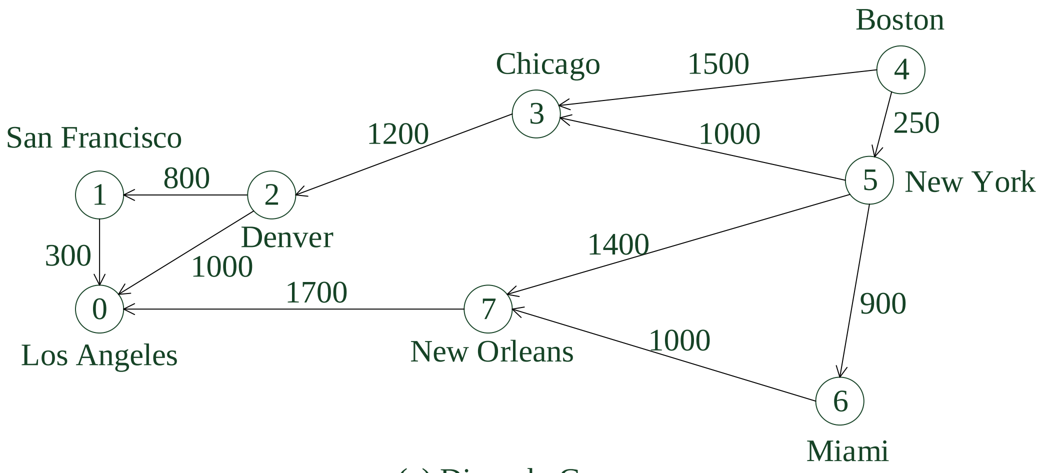
(c) All pairs DAG shortest paths (Floyd) (Prog. 6.10)

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".

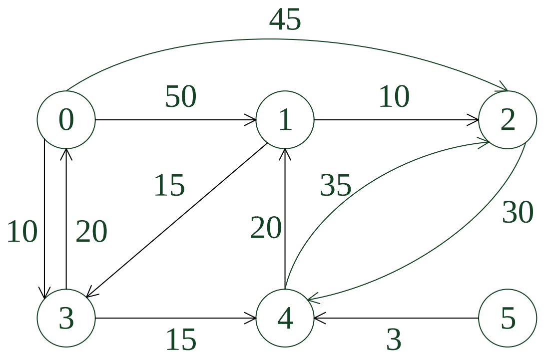
You need to demonstrate your code of (a) by processing: G_1, G_1', and G_1" shown below.



(a) Digraph G_1

Iteration	Vertex selected	Distance							
		LA [0]	SF [1]	DEN [2]	CHI [3]	BOST [4]	NY [5]	MIA [6]	NO [7]
Initial	----	∞	∞	∞	1500	0	250	∞	∞
1	5	∞	∞	∞	1250	0	250	1150	1650
2	6	∞	∞	∞	1250	0	250	1150	1650
3	3	∞	∞	2450	1250	0	250	1150	1650
4	7	3350	∞	2450	1250	0	250	1150	1650
5	2	3350	3350	2450	1250	0	250	1150	1650
6	1	3350	3350	2450	1250	0	250	1150	1650

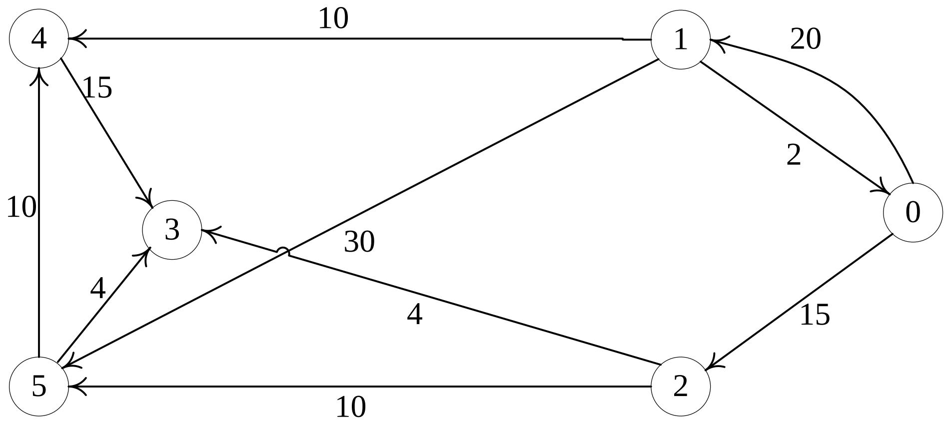
Fig. 6.28



(a) Digraph G_1'

	路徑	長度
1)	0, 3	10
2)	0, 3, 4	25
3)	0, 3, 4, 1	45
4)	0, 2	45

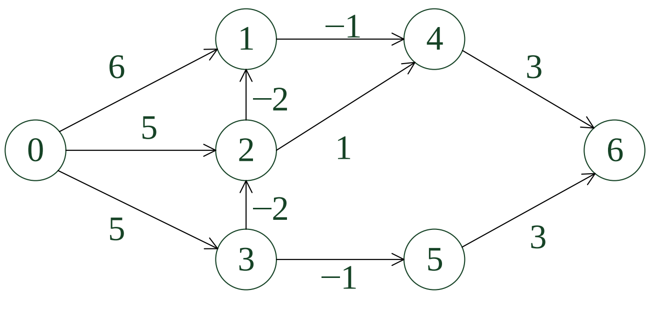
(b) 從 0 出發的最短路徑



(a) G_1”. Find shortest paths from vertex 0 to all remaining vertices.

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

You need to demonstrate your code of (b) by processing: G_2 and G_2’ shown below.

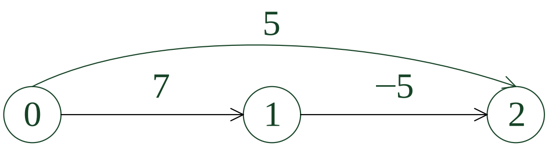


(a) Digraph G_2

	dist^k[7]						
k	0	1	2	3	4	5	6
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

(b) *dist*^{k}

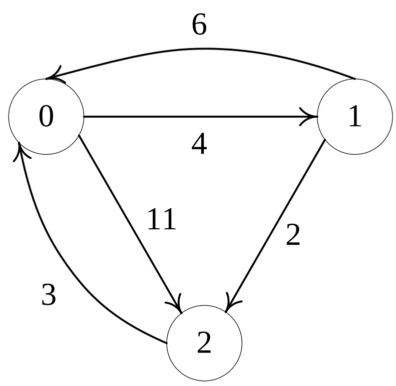
Fig. 6.31



G_2'

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 G_3 (below in Fig. 6.32(a)) and G_2 (above in Fig. 6.31(a)).



(a) Digraph G_3

A^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

(b) A^{-1}

A^{0}	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

(c) A^{0}

A^{1}	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

(d) A^{1}

A^{2}	0	1	2
0	0	4	6
1	5	0	2
2	3	7	0

(e) A^{2}

Fig. 6.

C:\c++\data_structure\hw5\su

```
'i' means infinte

(a):
G1:
Path: 4->5; Distance: 250
Path: 4->5->6; Distance: 1150
Path: 4->5->3; Distance: 1250
Path: 4->5->7; Distance: 1650
Path: 4->5->3->2; Distance: 2450
Path: 4->5->3->2->1; Distance: 3250
Path: 4->5->7->0; Distance: 3350

G1':
Path: 0->3; Distance: 10
Path: 0->3->4; Distance: 25
Path: 0->3->4->1; Distance: 45
Path: 0->2; Distance: 45

G1'':
Path: 0->2; Distance: 15
Path: 0->2->3; Distance: 19
Path: 0->1; Distance: 20
Path: 0->2->5; Distance: 25
Path: 0->1->4; Distance: 30
---------------------------------------------------------
(b):
```

```
---------------------------------------------------------
(b):
G2:
k   0   1   2   3   4   5   6
-------------------------------
1   0   6   5   5   i   i   i
2   0   3   3   5   5   4   i
3   0   1   3   5   2   4   7
4   0   1   3   5   0   4   5
5   0   1   3   5   0   4   3
6   0   1   3   5   0   4   3


G2':
k   0   1   2
-------------------------------
1   0   7   5
2   0   7   2
---------------------------------------------------------
(c):
```

(c):

G3:

A(-1)	0	1	2
0	0	4	11
1	6	0	2
2	3	i	0

A0	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

A1	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

A2	0	1	2
0	0	4	6
1	5	0	2
2	3	7	0

G2:

A(-1)	0	1	2	3	4	5	6
0	0	6	5	5	i	i	i
1	i	0	i	i	-1	i	i
2	i	-2	0	i	1	i	i
3	i	i	-2	0	i	-1	i
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A0	0	1	2	3	4	5	6
0	0	6	5	5	i	i	i
1	i	0	i	i	-1	i	i
2	i	-2	0	i	1	i	i
3	i	i	-2	0	i	-1	i
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A0	0	1	2	3	4	5	6
0	0	6	5	5	i	i	i
1	i	0	i	i	-1	i	i
2	i	-2	0	i	1	i	i
3	i	i	-2	0	i	-1	i
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A1	0	1	2	3	4	5	6
0	0	6	5	5	5	i	i
1	i	0	i	i	-1	i	i
2	i	-2	0	i	-3	i	i
3	i	i	-2	0	i	-1	i
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A2	0	1	2	3	4	5	6
0	0	3	5	5	2	i	i
1	i	0	i	i	-1	i	i
2	i	-2	0	i	-3	i	i
3	i	-4	-2	0	-5	-1	i
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A3	0	1	2	3	4	5	6
0	0	1	3	5	0	4	i
1	i	0	i	i	-1	i	i
2	i	-2	0	i	-3	i	i
3	i	-4	-2	0	-5	-1	i
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A4	0	1	2	3	4	5	6
0	0	1	3	5	0	4	3
1	i	0	i	i	-1	i	2
2	i	-2	0	i	-3	i	0
3	i	-4	-2	0	-5	-1	-2
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A5	0	1	2	3	4	5	6
0	0	1	3	5	0	4	3
1	i	0	i	i	-1	i	2
2	i	-2	0	i	-3	i	0
3	i	-4	-2	0	-5	-1	-2
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

A6	0	1	2	3	4	5	6
0	0	1	3	5	0	4	3
1	i	0	i	i	-1	i	2
2	i	-2	0	i	-3	i	0
3	i	-4	-2	0	-5	-1	-2
4	i	i	i	i	0	i	3
5	i	i	i	i	i	0	3
6	i	i	i	i	i	i	0

3. (30%)

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

(a) Topological order

(b) The earliest and latest times of all events (ee[i], le[i])

(c) The earliest and latest times of all activities (e[k], l[k])

(d) A table of all activities with their early and late times together with their slack and critical activities like Figure 6.41.

(e) The critical network

(f) 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.

	最早時間	最晚時間	餘裕	臨界度
Activity	*e*	*l*	*l* - *e*	*l* – *e* = 0
a_1	0	0	0	Yes
a_2	0	2	2	No
a_3	0	3	3	No
a_4	6	6	0	Yes
a_5	4	6	2	No
a_6	5	8	3	No
a_7	7	7	0	Yes
a_8	7	7	0	Yes
a_9	7	10	3	No
a_{10}	16	16	0	Yes
a_{11}	14	14	0	Yes

Figure 6.41

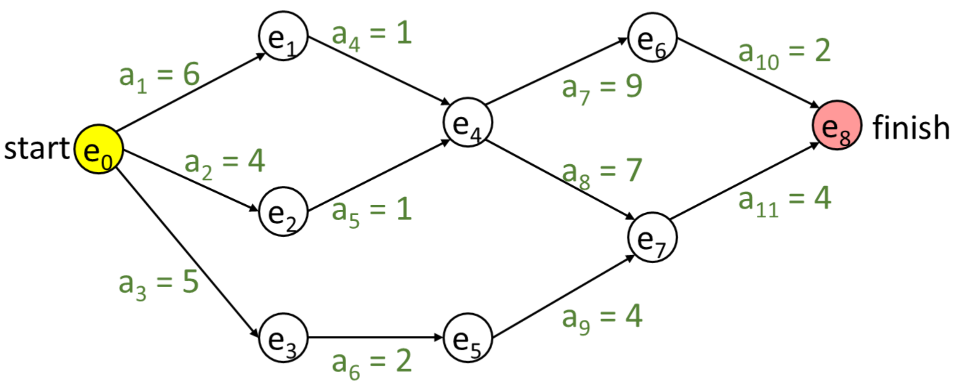


Figure 6.39

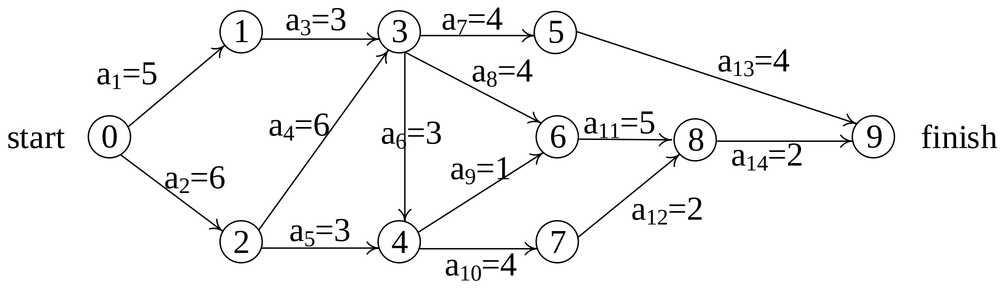


Figure 6.44

```
C:\c++\data_structure\hw5\fi
Fig 6.39:
Topological order:
0 3 5 2 1 4 7 6 8

Earliest event time:
vertex:  0  1  2  3  4  5  6  7  8
ee:      0  6  4  5  7  7 16 14 18
Latest event time:
vertex:  0  1  2  3  4  5  6  7  8
le:      0  6  6  8  7 10 16 14 18

Activity table:
Activity  Earliest  Latest  Slack  Criticality
--------------------------------------------------
  1     0       0       0       1
  2     0       2       2       0
  3     0       3       3       0
  4     6       6       0       1
  5     4       6       2       0
  6     5       8       3       0
  7     7       7       0       1
  8     7       7       0       1
  9     7       10      3       0
  10    16      16      0       1
  11    14      14      0       1

Critical path (represented by activity number):
1 4 7 8 10 11
--------------------------------------------------
--------------------------------------------------
Fig 6.44:
Topological order:
0 2 1 3 5 4 7 6 8 9

Earliest event time:
vertex:  0  1  2  3  4  5  6  7  8  9
ee:      0  5  6 12 15 16 16 19 21 23
Latest event time:
vertex:  0  1  2  3  4  5  6  7  8  9
le:      0  9  6 12 15 19 16 19 21 23

Activity table:
Activity  Earliest  Latest  Slack  Criticality
--------------------------------------------------
  1     0       4       4       0
  2     0       0       0       1
  3     5       9       4       0
  4     6       6       0       1
  5     6       12      6       0
  6     12      12      0       1
  7     12      15      3       0
  8     12      12      0       1
  9     15      15      0       1
  10    15      15      0       1
  11    16      16      0       1
  12    19      19      0       1
  13    16      19      3       0
  14    21      21      0       1

Critical path (represented by activity number):
2 4 6 8 9 10 11 12 14

Process returned 0 (0x0)   execution time : 0.037 s
Press any key to continue.
```