EECS2040 Data Structure Hw #5 (Chapter 6 Graph)
due date 6/13/2021
By 107061123, 孫元駿

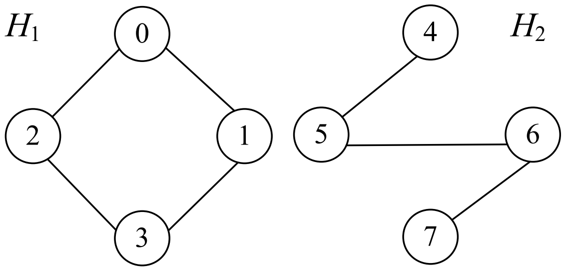
Part 2 Coding

You should submit:
(a) All your source codes (C++ file).
(b) Show the execution trace of your program.

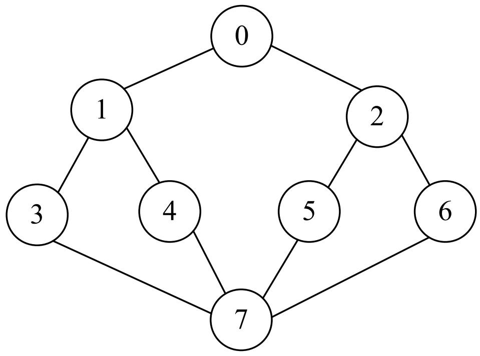
1. (50%) Graph(linked adjacency list), BFS, DFS, connected components, Computing dfn and low: Write a C++ program to perform the following basic graph functions:
   1. BFS(v) (Prog. 6.2) (v: starting vertex. You need to output the vertices visited in BFS order)
   2. DFS(v) (Prog. 6.1) (v: starting vertex. 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) (Display the computed dfn[i] and low[i] of the graph)

   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 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.

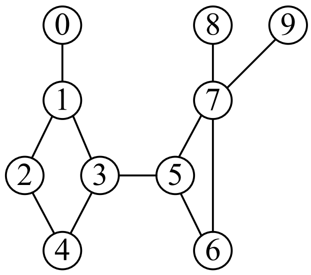
```
Graph g1(8),g2(8),g3(10);
g1.Setup1();
//BFS
g1.BFS(0);
g1.BFS(3);
g1.BFS(7);
//DFS
g1.DFS(0);
g1.DFS(3);
g1.DFS(7);
//Components & DfnLow
g1.Components();
g1.DfnLow(3);
```



G_1



G_2



G_3

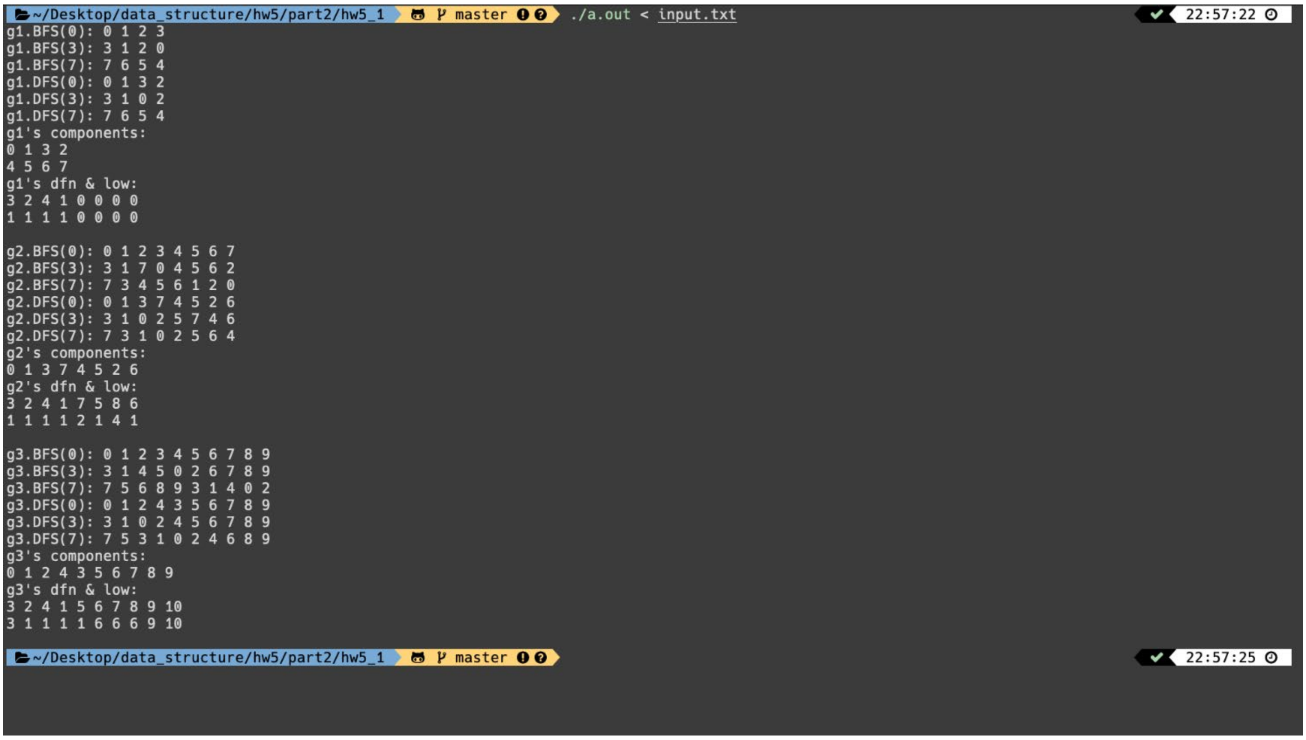
How to use my code:

All the input file are in input.txt.

First, input file will input the number of edge and set the graph up for G_1, then it will find the BFS from 0, 3, 7. Then find DFS from 0, 3, 7. Then show the components and DFN and Low.

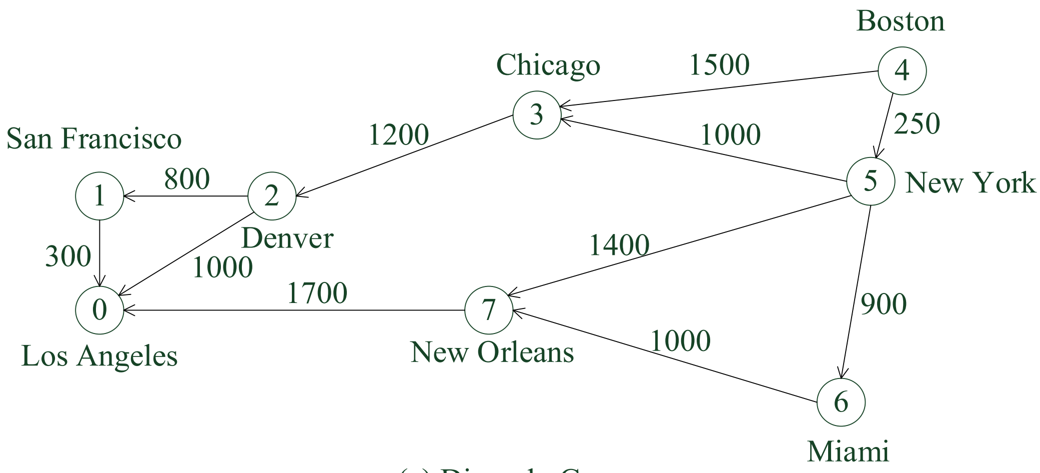
Second, input file will input the number of edge and set the graph up for G_2, then it will find the BFS from 0, 3, 7. Then find DFS from 0, 3, 7. Then show the components and DFN and Low.

Last, input file will input the number of edge and set the graph up for G_3, then it will find the BFS from 0, 3, 7. Then find DFS from 0, 3, 7. Then show the components and DFN and Low.



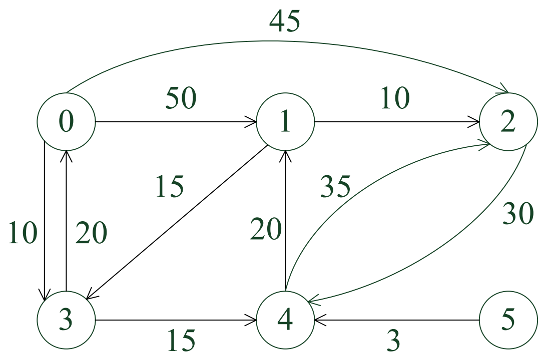
```
~/Desktop/data_structure/hw5/part2/hw5_1  master  ./a.out < input.txt                22:57:22
g1.BFS(0): 0 1 2 3
g1.BFS(3): 3 1 2 0
g1.BFS(7): 7 6 5 4
g1.DFS(0): 0 1 3 2
g1.DFS(3): 3 1 0 2
g1.DFS(7): 7 6 5 4
g1's components:
0 1 3 2
4 5 6 7
g1's dfn & low:
3 2 4 1 0 0 0 0
1 1 1 1 0 0 0 0

g2.BFS(0): 0 1 2 3 4 5 6 7
g2.BFS(3): 3 1 7 0 4 5 6 2
g2.BFS(7): 7 3 4 5 6 1 2 0
g2.DFS(0): 0 1 3 7 4 5 2 6
g2.DFS(3): 3 1 0 2 5 7 4 6
g2.DFS(7): 7 3 1 0 2 5 6 4
g2's components:
0 1 3 7 4 5 2 6
g2's dfn & low:
3 2 4 1 7 5 8 6
1 1 1 1 2 1 4 1

g3.BFS(0): 0 1 2 3 4 5 6 7 8 9
g3.BFS(3): 3 1 4 5 0 2 6 7 8 9
g3.BFS(7): 7 5 6 8 9 3 1 4 0 2
g3.DFS(0): 0 1 2 4 3 5 6 7 8 9
g3.DFS(3): 3 1 0 2 4 5 6 7 8 9
g3.DFS(7): 7 5 3 1 0 2 4 6 8 9
g3's components:
0 1 2 4 3 5 6 7 8 9
g3's dfn & low:
3 2 4 1 5 6 7 8 9 10
3 1 1 1 1 6 6 6 9 10
~/Desktop/data_structure/hw5/part2/hw5_1  master                                      22:57:25
```

2. (50%) 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), 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" (in Part 1. Problem 6.) shown below.



(a) Digraph G_1

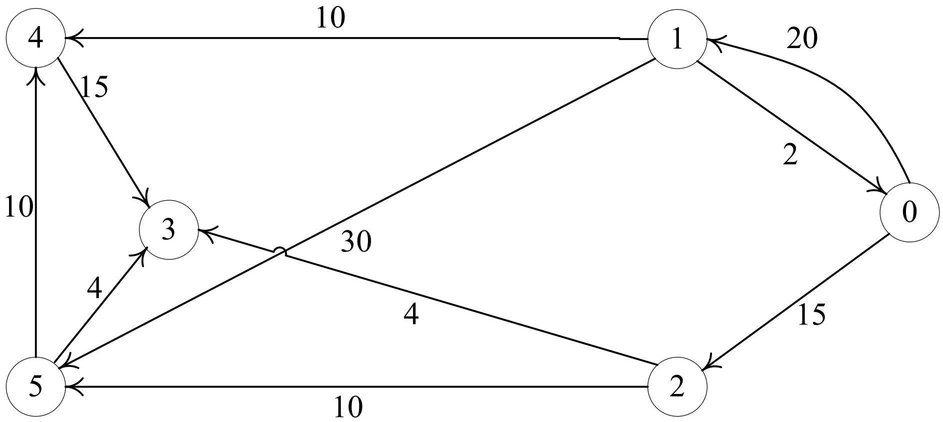
Iteration	Vertex selected	Distance LA [0]	SF [1]	DEN [2]	CHI [3]	BOST [4]	NY [5]	MIA [6]	NO [7]
Initial	----	∞	∞	∞	1500	0	250	∞	∞
1	5	∞	∞	∞	1250	0	250	1150	1650
2	6	∞	∞	∞	1250	0	250	1150	1650
3	3	∞	∞	2450	1250	0	250	1150	1650
4	7	3350	∞	2450	1250	0	250	1150	1650
5	2	3350	3350	2450	1250	0	250	1150	1650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



(a) Digraph G_1'

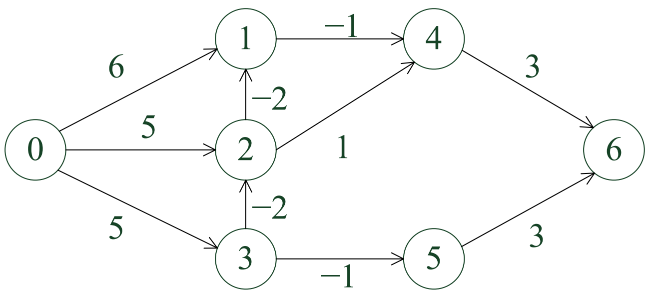
	路徑	長度
1)	0, 3	10
2)	0, 3, 4	25
3)	0, 3, 4, 1	45
4)	0, 2	45

(b) 從 0 出發的最短路徑



G_1”

For (b), 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.

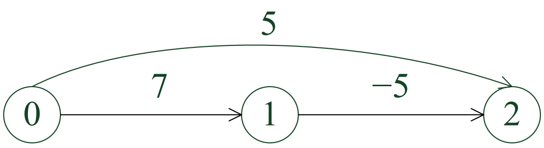


(a) Digraph G_2

k	$dist^k[7]$ 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

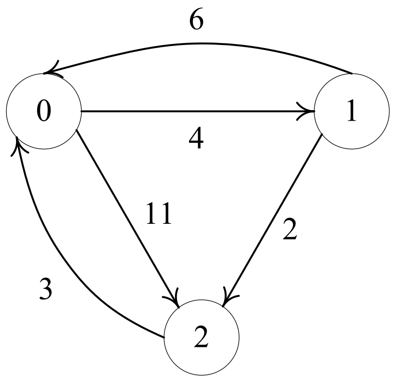
(b) $dist^k$

Figure 6.31



G_2'

For (c), 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) and G_2 (above).



(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}

Figure 6.32

How to use my code:

All the input file are in input.txt.

First, input file will input the number of edge and set the graph up for G_1, G_1’, and G_1”, then it will find the ShortestPath, and show the path and the distance.

Second, input file will input the number of edge and set the graph up for G_2 and G_2’, then it will find the distance by Bellman–Ford algorithm and show the distance by processing.

Last, input file will input the number of edge and set the graph up for G_3 and G_2, then it will find the distance by Floyd-Warshall algorithm and show the matrix by processing.

X means infinite.

因為印出 process 太多了，所以我分頁截圖

Part a

```
~/Desktop/data_structure/hw5/part2/hw5_2  master  ./a.out < input.txt
Part a
        [0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]
0  --     X    X    X 1500    0  250    X    X
1   5     X    X    X 1250    0  250 1150 1650
2   6     X    X    X 1250    0  250 1150 1650
3   3     X    X 2450 1250    0  250 1150 1650
4   7 3350    X 2450 1250    0  250 1150 1650
5   2 3350 3250 2450 1250    0  250 1150 1650
6   1 3350 3250 2450 1250    0  250 1150 1650
[0]: 4 -> 5 -> 7 -> 0 / 3350
[1]: 4 -> 5 -> 3 -> 2 -> 1 / 3250
[2]: 4 -> 5 -> 3 -> 2 / 2450
[3]: 4 -> 5 -> 3 / 1250
[4]: 4 / 0
[5]: 4 -> 5 / 250
[6]: 4 -> 5 -> 6 / 1150
[7]: 4 -> 5 -> 7 / 1650

        [0]  [1]  [2]  [3]  [4]  [5]
0  --     0   50   45   10    X    X
1   3     0   50   45   10   25    X
2   4     0   45   45   10   25    X
3   2     0   45   45   10   25    X
4   1     0   45   45   10   25    X
[0]: 0 / 0
[1]: 0 -> 3 -> 4 -> 1 / 45
[2]: 0 -> 2 / 45
[3]: 0 -> 3 / 10
[4]: 0 -> 3 -> 4 / 25
[5]: X / Infinite

        [0]  [1]  [2]  [3]  [4]  [5]
0  --     0   20   15    X    X    X
1   2     0   20   15   19    X   25
2   3     0   20   15   19    X   25
3   1     0   20   15   19   30   25
4   5     0   20   15   19   30   25
[0]: 0 / 0
[1]: 0 -> 1 / 20
[2]: 0 -> 2 / 15
[3]: 0 -> 2 -> 3 / 19
[4]: 0 -> 1 -> 4 / 30
[5]: 0 -> 2 -> 5 / 25

Part b
      [0] [1] [2] [3] [4] [5] [6]
(0):    0   6   5   5   X   X   X
(1):    0   3   3   5   2   4   5
(2):    0   1   3   5   0   4   3
(3):    0   1   3   5   0   4   3
(4):    0   1   3   5   0   4   3
(5):    0   1   3   5   0   4   3
(6):    0   1   3   5   0   4   3

      [0] [1] [2]
```

Part b

```
Part b
     [0] [1] [2] [3] [4] [5] [6]
(0):   0   6   5   5   X   X   X
(1):   0   3   3   5   2   4   5
(2):   0   1   3   5   0   4   3
(3):   0   1   3   5   0   4   3
(4):   0   1   3   5   0   4   3
(5):   0   1   3   5   0   4   3
(6):   0   1   3   5   0   4   3

     [0] [1] [2]
(0):   0   7   5
(1):   0   7   2
(2):   0   7   2

Part c
Process 0
[0]    0   4  11
[1]    6   0   2
[2]    3   X   0
Process 1
[0]    0   4  11
[1]    6   0   2
[2]    3   7   0
Process 2
[0]    0   4   6
[1]    6   0   2
[2]    3   7   0
Process 3
[0]    0   4   6
[1]    5   0   2
[2]    3   7   0

Process 0
[0]    0   6   5   5   X   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X   1   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 1
[0]    0   6   5   5   X   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X   1   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 2
[0]    0   6   5   5   5   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X  -3   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
```

Part c_1

```
Part c
Process 0
[0]    0   4  11
[1]    6   0   2
[2]    3   X   0
Process 1
[0]    0   4  11
[1]    6   0   2
[2]    3   7   0
Process 2
[0]    0   4   6
[1]    6   0   2
[2]    3   7   0
Process 3
[0]    0   4   6
[1]    5   0   2
[2]    3   7   0

Process 0
[0]    0   6   5   5   X   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X   1   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 1
[0]    0   6   5   5   X   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X   1   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 2
[0]    0   6   5   5   5   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X  -3   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 3
[0]    0   3   5   5   2   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X  -3   X   X
[3]    X  -4  -2   0  -5  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 4
[0]    0   1   3   5   0   4   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X  -3   X   X
[3]    X  -4  -2   0  -5  -1   X
```

Part c_2

```
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X   1   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 2
[0]    0   6   5   5   5   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X  -3   X   X
[3]    X   X  -2   0   X  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 3
[0]    0   3   5   5   2   X   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X  -3   X   X
[3]    X  -4  -2   0  -5  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 4
[0]    0   1   3   5   0   4   X
[1]    X   0   X   X  -1   X   X
[2]    X  -2   0   X  -3   X   X
[3]    X  -4  -2   0  -5  -1   X
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 5
[0]    0   1   3   5   0   4   3
[1]    X   0   X   X  -1   X   2
[2]    X  -2   0   X  -3   X   0
[3]    X  -4  -2   0  -5  -1  -2
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 6
[0]    0   1   3   5   0   4   3
[1]    X   0   X   X  -1   X   2
[2]    X  -2   0   X  -3   X   0
[3]    X  -4  -2   0  -5  -1  -2
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0
Process 7
[0]    0   1   3   5   0   4   3
[1]    X   0   X   X  -1   X   2
[2]    X  -2   0   X  -3   X   0
[3]    X  -4  -2   0  -5  -1  -2
[4]    X   X   X   X   0   X   3
[5]    X   X   X   X   X   0   3
[6]    X   X   X   X   X   X   0

~/Desktop/data_structure/hw5/part2/hw5_2   master
```