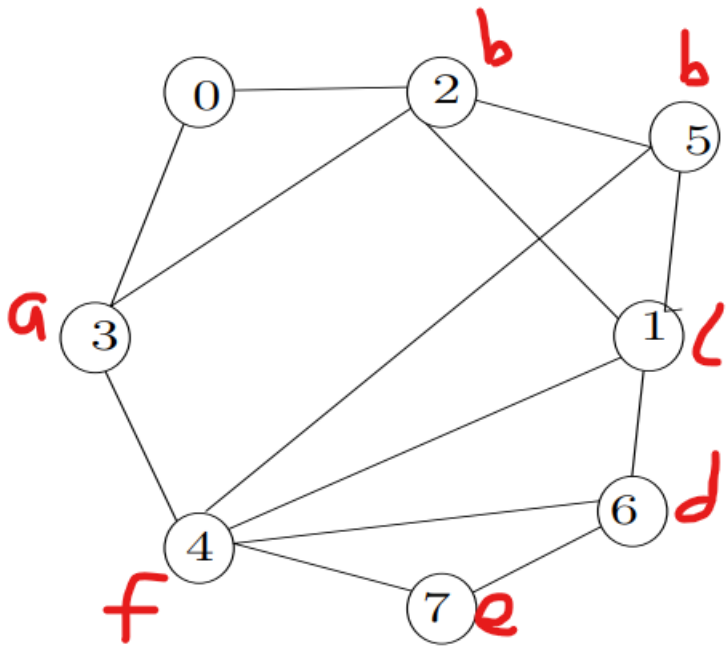
COMPSCI 711 - Assignment 1

1.

(a)

	Diameter	Arc Connectivity	Bisection width
Graph 1	2	2	4
Graph 2	3	2	4

(b)



Dilation: 2

Congestion: 1

2.

Note: This proof is taken from p. 395-396, Introduction to Parallel Computing, Second Edition

First, if the sequence s is a shifted bitonic sequence, we shift it so that s is monotonically increasing and then monotonically decreasing.

The sequences s_1, and s_2, resulting from the bitonic split are as follows

$s_1 = [min(a_0, a_{n/2}), \ldots, min(a_{n/2-1}, a_{n-1})]$

$s_2 = [max(a_0, a_{n/2}), \ldots, max(a_{n/2-1}, a_{n-1})]$

(1)

In s_1, there is an element $b_i = min(a_i, a_{n/2+i})$ such that all the elements before b_i are from the increasing part of the original sequence and all the elements after b_i are from the decreasing part.

In sequence s_2, the element $b'_i = max(a_i, a_{n/2+i})$ is such that all the elements before b'_i are from the decreasing part of the original sequence and all the elements after b'_i are from the increasing part. Thus, the sequences s_1 and s_2 are bitonic sequences.

(2)

b_i is greater than or equal to all elements of s_1, b'_i is less than or equal to all elements of s_2, and b'_i is greater than or equal to b_i. Therefore, all elements of s_1 are smaller than or equal to s_2. Note that property 2 is strictly smaller only if the s contains unique elements.

3.

Comparison of Source-Partition to Source Parallel

Relative Timestep 1

Source-partition

	0	1	2	3	4	5
0	0	5	7	4	10	8
1						
2						
3						
4						
5						

Source-parallel

	0	1	2	3	4	5
0	0	5	7	4		
1	5	0	8	4		
2	7	8	0	11		
3	4	4	11	0		
4	10	5	13	9		
5	8	3	11	7		

Relative Timestep 2

Source-partition

	0	1	2	3	4	5
0	0	5	7	4	10	8
1	5	0	8	4	5	3
2						
3						
4						
5						

Source-parallel

	0	1	2	3	4	5
0	0	5	7	4	10	8
1	5	0	8	4	5	3
2	7	8	0	11	13	11
3	4	4	11	0	9	7
4	10	5	13	9	0	2
5	8	3	11	7	2	0

Relative Timestep 3

Source-partition

	0	1	2	3	4	5
0	0	5	7	4	10	8
1	5	0	8	4	5	3
2	7	8	0	11	13	11
3						
4						
5						

Relative Timestep 4

Source-partition

	0	1	2	3	4	5
0	0	5	7	4	10	8
1	5	0	8	4	5	3
2	7	8	0	11	13	11
3	4	4	11	0	9	7
4						
5						

Relative Timestep 5

Source-partition

	0	1	2	3	4	5
0	0	5	7	4	10	8
1	5	0	8	4	5	3
2	7	8	0	11	13	11
3	4	4	11	0	9	7
4	10	5	13	9	0	2
5						

Relative Timestep 6

Source-partition

	0	1	2	3	4	5
0	0	5	7	4	10	8
1	5	0	8	4	5	3
2	7	8	0	11	13	11
3	4	4	11	0	9	7
4	10	5	13	9	0	2
5	8	3	11	7	2	0

Source-partitioned Dijkstra's can only utilise 6 processes at a time. This is because the number of processes is restricted by the number of nodes in the graph. One process performs Dijkstra's single-source algorithm for each node. Therefore, there is no inter-process communication so no communication overhead. The parallel run-time when p < n is $O(n^2)$. However, as can be seen in the example the isoefficiency is $O(n^3)$. This algorithm does not scale well (performs about as well as non-parallel) as the number of processors scale.

Source-parallel Dijkstra's can efficiently utilise all 24 processors. It has 4 partitions, each with 6 processes each. It is significantly more performant than source-partitioned Dijkstra's algorithm. In contrast to source-partitioned Dijkstra's, there is communication overhead of $O(n \log p)$. The isoefficiency of this formulation is $O((p \log p)^{1.5})$

4.

Note: Algorithm derived from slide 45 of *The Evolution of P Systems, to Hyperday P Systems, to P Modules* by Michael J.Dinneen, Yun-Bum Kim and Radu Nicolescu

The following rules assume that the clocks for processors are the same frequency, and are synchronised.

For state s_0:

1. $s_0 \rightarrow s_1 ac_{\uparrow}, \alpha = min, \beta = repl$

For state s_1:

1. $s_1 ac \rightarrow s_1 hac_{\uparrow}, \alpha = max, \beta = repl$
2. $s_1 c \rightarrow s_1, \alpha = max$
3. $s_1 a \rightarrow s_2, \alpha = min$

At the start of the process, the nodes are empty. Then they all transition to state 1 and send a message to their parents. The parents collect these messages to determine whether they have a child. Note that a represents if a node is still actively processing. If a node has no children after this first stage then it will just have a as its state. By rule (3) it will then transition to an inactive state. s_2 is the inactive state of the system.

Otherwise, the node does have a child (or many), in which case it increments it's height (1). If there are multiple children, this information is discarded as it is not useful via (2). When (1) occurs, it also sends a message to its parents. This is to inform them that they need to increment their height.

This will loop until all nodes end in state s_2. The number of h's is the height of the node. There is a side-effect which is that all other nodes will also store their height.

5.

(a)

Algorithm

```
FUNCTION MinimalVertexCover(G)
    set MIS = MaximalIndependentSet(G)
    return V - MIS
end FUNCTION
```

Run Luby's Maximal Independent Set Algorithm to retrieve a maximally independent set W.

The set $W' = V - W$ is a minimal vertex cover.

Proof W' is a vertex cover

Let $G = (V, E)$ be a graph.

Let W be a maximally independent set of G.

Let $W' = V - W$

By definition, there are no edges between any vertices in W.

Adding any vertex from W' to W does not give an independent set.

There is an edge between some vertex in W' and any vertex in W. Therefore, W' is a vertex cover.

Proof W' is a minimal vertex cover

Suppose there is a vertex v in W' that can be removed such that W' is still a vertex cover.

Vertex v can now be added to W.

By definition, there exists an edge between v and some vertex in W'. However, there also exists an edge between v and some vertex in W. Therefore, W is not an independent set.

Therefore, by contradiction, there can not exist a vertex which can be removed from W' such that W' is still a vertex cover. W' is a minimal vertex cover.

(b)

```
FUNCTION MinimalFeedbackVertexSet(G)
    set MFVS = {}

    while there exists a cycle in G:
        Assign random numbers to each vertex in the cycle
        set V = Vertex with smallest number in the cycle

        MVFS.add(V)
        set G = G - V
    end while

    return MVFS
end FUNCTION
```