Note: YOU MUST DO THE HOMEWORK BY YOURSELF. If you have difficulties in solving a question you may discuss it with friends, BUT you MUST phrase, write and formulate the answers by yourself, after you understand the solution. The language of the solution must be entirely your own. If you got some idea from a friend or a certain paper please cite her/him/it and give her/him/it credit.

Notice 2: Please type (Word, or LaTeX, etc.) and submit your homework as pdf file to the email address that was given.

Notice 3: Put your name and id number on each page of the solution.

1. The roman senate wants to reach agreement (consensus) on whether the Roman empire should change the currency to use Euros (vote for 0) or stay with the Roman coins (vote for 1). If all are against then they should decide against and if all are in favor they should decide in favor, otherwise any decision is acceptable.

The senators gather in a ring around the roman forum, and the announcer declares the method (distributed algorithm) by which they will reach consensus. Senators then converse, each only with his two neighbors on the ring, according to the announced method (distributed algorithm).

However, each senator might prefer that either 0 or 1 be decided, and thus may cheat in an attempt to increase the probability of its preferred value being decided. You have to devise a consensus algorithm for the announcer to declare to the senators, such that, no senator gains anything from deviating (cheating) from your algorithm. Your algorithm should satisfy both the agreement and the validity conditions. The cheating is constrained as follows:

(a) If the algorithm fails to reach agreement, all the senators are sent to prison for the rest of their life. Meaning, senators will not cheat if the cheating will definitely i.e., with probability 1, cause the algorithm to fail. However, senators do take chances, if there is some positive probability to succeed.

(b) A senator cheats only if the cheating increases the probability of its preferred value being decided.

Further make the following assumptions: (i) each senator has a unique name, (ii) each knows n, the total number of senators (iii) the senators converse in synchronous rounds (i.e., assume a synchronous ring), and (iv) assume all senators start together at the same round.

For any algorithm that you provide give its message and time complexities.
(a) Provide an algorithm when each senator prefers a certain value (0 or 1) not necessarily its own, and a subset of the senators may cooperate in cheating, to increase the probability of a certain value. Each subset assumes that it is the only one to cheat. What is the largest size sub-set under which your algorithm works correctly? It should be the maximum size for which there is a solution. Notice, members of any subset may communicate only through the ring.

(b) It is known that the number of senators in the senate is odd ($n \mod 2 = 1$). How do the above questions change given this assumption?

2. Based on the centralized Depth First Search (DFS) procedure, describe a distributed DFS traversal algorithm for an asynchronous network. In a traversal algorithm a central node, called root, initiates a token which has to visit all the nodes of the network, one at a time. Argue that in order to visit all the nodes the token must traverse all the links of the network. (links might be traversed more than one time)

(a) (Half a page) Verify that your algorithm uses $O(|E|)$ messages and $O(|E|)$ time, where $|E|$ is the total number of links in the network.

(b) (At most one page) Modify your algorithm to reduce its time complexity to $O(n)$, where $n$ is the total number of nodes in the network. (Hint: in the modification there must be times at which there are more than one message in transit in the network.)

(c) Modify the algorithm of (Section a) to implement a Directed DFS on a strongly connected unidirectional network. In a unidirectional network some or all the links can carry messages only in one, predetermined, direction. A unidirectional network is strongly connected if there is a path from any node to any other node. What is the message complexity of the new algorithm? (Hint: use the unique path connecting the root with the token location, to perform the backtracking). A solution exists even if all nodes are identical (no unique ids) except the initiator which is singled out. If you cannot find such a solution, try to provide a solution assuming nodes have unique ids.

3. Use the DFS algorithm of the previous question (asynchronous, Bidirectional network, with unique id to each node) to derive an $O(|E| + n \log n)$ messages election algorithm. HINT: Each initiator of the algorithm would start a DFS traversal. The question is how would colliding DFS traversals efficiently eliminate one another until only one DFS is left, which then captures the entire network and is elected as the leader.
4. ID:____________________

\( n \) processors on a unidirectional ring (\( p_i \) can send a message only to \( P_{(i+1)\mod n} \), \( i = 0 \ldots n – 1 \) and the links are FIFO) implement a shared virtual single writer multi reader atomic register as follows:

- To read the value of the register processor \( p_j \) reads its local copy of the register.
- To start the write(\( v \)) operation to the register the writer, \( p_0 \) (assuming \( p_0 \) is the writer) sends a message ”\( p_0 \) writes \( v \)” to \( p_1 \) (assuming there are at least 2 processors in the ring).
- If \( p_i \) receives a message ”\( p_j \) writes \( v \)” \( j \neq i \) then, it updates the value of its local copy of the register to \( v \) and forwards the message to \( P_{(i+1)\mod n} \).
- If \( p_i \) receives a message ”\( p_i \) writes \( v \)” (which means that \( i = 0 \)) then, it updates the value of its local copy of the register to \( v \) and discards the message. The write operation is now complete.

For each of the following statements provide either a short explanation why it is true, or provide a counter example.

(a) This is an implementation of a regular register.
(b) This is an implementation of an atomic register.
5. *k*-atomic registers (15%)

Company Bugtel built a shared memory system in which processors communicate only by reading and writing shared *k*-atomic-registers. The linearize specification of a Bugtel *k*-atomic-register, is such that (a) the reads and writes are ordered in a linearizable order and (b) a read returns the value of one of the *k* last write operations. In other words, for any execution, there is some way of totally ordering the overlapping reads and writes operations (the standard linearization order). Call this order, serialized-order. Then the value returned by each read is any value written by one of the *k* writes that precedes this read in the serialized-order. (As usual operations take effect sometimes after their invocation and before .)

Question: Is it possible to build a standard atomic read write register system from the Bugtel *k*-atomic-registers. Prove your answer, by either proving why it is impossible, or providing a construction (which may use any construction we studied in class as a building block).
6. (20%)

Here is a presentation of a wait free implementation of *Fifo Queue* from *fetch-and-add* and *swap* objects. The implementation is based on Herlihy and Wing's construction from . A queue is defined by the two operations performed on it, Enq and Deq. *Enq(Q, v)* inserts *v* into queue *Q*’s head. *Deq(Q)* returns the item at queue *Q*’s tail. Assuming that every item in the queue is distinct, the sequential specification of the queue is as follows:

(a) If *x* = *Deq(Q)* then *Enq(Q, x)* is before that *Deq* operation.
(b) If *x* = *Deq(Q)* is before *y* = *Deq(Q)* then *Enq(Q, x)* is before *Enq(Q, y)*.

The implementation uses one *fetch-and-add* object - *counter* - that can also be read by applying *F&A(0)*. In addition it uses an unbounded array of *swap* objects - *items*. The *Enq* operation listed in figure 1 is simple. A process first *F&A(1)* to the counter. It then assigns the item into the queue by swapping its value into the appropriate entry - the counter value - in the *items* array.

The *Deq* operation starts by reading the *counter* current value. This value serves as the upper bound on the number of items within the queue. The process then scans the *items* array from *items[1]* up to the upper bound, by swapping *null* in every entry until it returns a non *null* value - which is the return value of the operation. The time of the *Deq* operation is therefore unbounded though it is finite. The code for the operation is listed in figure 1.

Question: What is wrong with this implementation, give a scenario that shows the incorrect behavior of the algorithm.
shared counter a fetch-and-add
    object initialized to 0.
    items[1..] array of swap
    objects initialized to null.
local head, i, m of type integer; item of type data-item

Op Enq(item)
    head := F&A(counter, 1) /* F&A returns the value after the addition
    Swap(items[head], item)
end Enq

Op Deq() Returns(item)
    m := F&A(counter, 0)
    for i := 1 to m
        item := Swap(items[i], null);
        if item ≠ null
            return (item);
    end for
    return (empty);
end Deq

Figure 1: The Enq and Deq operations