We work only with asynchronous message passing. Message complexity is the number of messages exchanged in the network and time complexity will be defined as the maximal length maximal causal chains in computations.

Unless specified otherwise, processes do not have a unique ID and do not have access to any topological information save for the list of their neighbours and whatever global constraint we may impose on the network.

1 Wave algorithms

Let us recall that a wave algorithm is a distributed algorithm, together with a distinguished set of internal events called decide events, satisfying the following desiderata:

**Termination** Each execution is finite.

\[ \forall C. |C| < \infty \]

**Decision** Each computation contains at least one decide event.

\[ \forall C. \exists d \in C. d \text{ is a decide event} \]

**Dependence** For every decide event \( d \in C \), \( d \) is causally preceded by an event in \( C_p \) for every \( p \).

\[ \forall C. \forall d \in C. d \text{ is a decide event} \Rightarrow \forall q. \text{processor}. \exists f \in C_q. f \preceq d \]

**Question 1**

a) Let \( X = \{v_1, \ldots, v_n\} \) and \( (X, \leq) \) be a partial order with a unique infimum. Suppose that each process \( p \) holds \( v_p \) and that we have an algorithm return the infimum value of \( X \) in at least one process. Show that this algorithm is a wave algorithm.

b) Conversely, show that every wave algorithm can be used to compute the infimum of \( X \).

c) Consider an arbitrary associative, commutative and idempotent operator \( a \ast b \). Show that every wave algorithm can be used to compute \( v_1 \ast v_2 \ast \cdots \ast v_n \). This is a generalization of the previous question.
Let us recall that the *phase algorithm* is a decentralized wave algorithm working over directed graphs. All processes are supposed to know the diameter $D$ of the network before proceeding as described in Algorithm 1.

**Algorithm 1: Phase algorithm**

```plaintext
const int: $D$ (* network diameter *)
var set : $Out_p$ (* set of successor vertices *)
var set : $In_p$ (* set of predecessor vertices *)
var int : $Rec_p[] ← 0$ (* number of messages received from $q ∈ In_p$ *)
var int : $Sent_p ← 0$ (* number of messages sent to each neighbour *)

if I am an initiator then
    for $r ∈ Out_p$ do
        send to $r$
        increment $Sent_p$

while $\min_{q ∈ In_p} Rec_p[q] < D$ do
    receive from $q_0$
    increment $Rec_p[q_0]$
    if $\min_q Rec_p[q] ≥ Sent_p$ and $Sent_p < D$ then
        for $r ∈ Out_p$ do
            send to $r$
            increment $Sent_p$
    decide
```

**Question 2**

a) Show that the phase algorithm is indeed a wave algorithm. You can use that $f_{pq}^{(i)} \preceq g_{pq}^{(i)}$, where $f_{pq}^{(i)}$ is the $i$-th send event from $p$ to $q$ and $g_{pq}^{(i)}$ is the $i$-th receive event of $q$ from $p$. What is its message complexity?

b) Does $D$ need to be exactly the diameter of the graph?

c) *(Hard)* Assume that processes are given unique identifiers. Modify the phase algorithm so that it computes a good over-approximation of $D$.

## 2 Traversal algorithms

Recall that a *traversal* algorithm is a wave algorithm satisfying the following additional desiderata:

- There should be a unique initiator $i$ and a unique decision event $d$ with $d ∈ C_i$ for every computation $C$.
- Upon receiving a single message, a process should either send a unique message or decide.
Question 3

a) Show that in traversal algorithms, time complexity and message complexity are “equivalent” notions.

We call an algorithm \( f\)-traversal if it is traversal, and after \( f(x) \) rounds of sending/receiving, the message has gone through at least \( \min(x + 1, N) \) processors.

Question 4

a) For which \( f \) the obvious traversal algorithm for the ring topology is an \( f \)-traversal algorithm?

b) Design a \( 2x \)-traversal algorithm for complete bipartite graphs (Hint: here, you are allowed to make messages carry an additional bit of information)

Recall that the following constraints are satisfied in an execution of Tarry’s algorithm:

- \( p \) does not forward the token twice to the same processor if others are available.
- Otherwise, \( p \) forwards the token to its first corresponding process (its father) if it is not the initiator; otherwise, it decides.

Question 5

a) Write down Tarry’s algorithm.

b) Prove that Tarry’s algorithm is indeed a traversal algorithm by proving each of the following steps:

When the algorithm terminates,

(a) All channels incident to the initiator have been used once in each direction.

(b) For each visited process \( p \), all channels incident to \( p \) have been used once in each direction.

(c) All processes have been visited and each channel has been used once in both directions.

What is its complexity?

c) Based on this algorithm, construct an algorithm to compute a spanning tree.

3 Computing sums

Notice that our instrumentalization of wave algorithms in the first section enabled us to do reduction only when the operator we used was idempotent.
Question 6

a) Suppose that $A$ is a wave algorithm. Additionally, assume that for every computation, active channels give rise to a spanning tree over the network of processors. Assuming that each processor holds an integer $v_p$, explain how to modify $A$ to compute $\sum_p v_p$.

b) Can you use Tarry’s algorithm to compute a sum using this reduction?

c) Do you know any other suitable wave algorithm for this purpose?

Question 7

a) Suppose that you are given a wave algorithm and that each processor possesses a unique identifier, design an algorithm to compute a sum as in the previous question.

b) Show that the messages can get quite bigger.

Question 8

a) Assuming the following statement, explain why one cannot apply a similar trick in the most general case.

There exists no anonymous distributed and decentralized distributed algorithm computing the size of the network, even if it is known that $D \leq 2$.

b) Prove the lemma. (Hint: you can restrict the class of networks whose every vertex has degree exactly 3.)

\[ ^{1}\text{Notice that this works for any commutative semigroup operation.} \]