Introducing Distributed Computing Concepts Into Discrete Mathematics Courses

Hyunyoung Lee         Andreas Klappenecker
Department of Computer Science and Engineering
Texas A&M University
{hlee, klappi}@cse.tamu.edu

ABSTRACT
Undergraduate computer science students typically have limited exposure to fundamental concepts of parallel and distributed computing. We show how one can integrate concepts from distributed computing into a course on discrete mathematics. Arguing about executions of distributed processes can motivate the students to learn more about the theory of ordered sets. Exploring questions on the minimum size of vector clocks leads to a deeper appreciation of concepts from discrete mathematics such as Dilworth’s theorem and the dimension theory of partial orders. The combination of the two areas leads to a considerable synergistic effect.

Keywords
discrete mathematics, distributed computing, education, order relations, partial order, ordering of events

1. INTRODUCTION
Many daily tasks require a little bit of organization. For example, you would not want your coffee poured before you got your cup. Some tasks simply need to take precedence before other tasks, otherwise you might get into a mess. This becomes particularly apparent in distributed computing, which is prevalent in our life these days (for instance, consider the widespread use of hand-held computing devices such as smart phones or tablet computers). In distributed computing, correctness of many tasks relies on correct understanding of the ordering of the distributed events. Thus, an early exposure to basic distributed computing concepts such as event ordering is desirable in modern computing education.

Many modern undergraduate computer science or computer engineering courses discuss some applications of distributed and parallel computing. However, the basics of distributed computing are often taught in advanced courses, and long prerequisite chains prevent the students to take these courses early in their studies. Even worse, distributed computing might be merely taught as an elective. An obvious choice is to spread parallel and distributed computing concepts over core curriculum classes, as was suggested in several NSF initiatives. As a result, many departments integrated some parallel and distributed programming concepts into their systems courses. One of the challenges in teaching parallel and distributed programming in a short amount of time is that the students lack the proper background on the ordering of events, so have a hard time grasping the ramifications of distributed executions.

Many computer science undergraduate curricula include a required course on discrete mathematics or discrete structures that the students take in their freshman or sophomore year. One challenge in teaching such a course is that the students are not motivated enough, since they fail to recognize the importance of the theoretical foundations. In this paper, we suggest that it is prudent to incorporate some foundational material on parallel and distributed computing into a discrete mathematics course. On one hand, the purpose is to motivate the students to learn more abstract concepts of discrete mathematics; on the other hand, the goal is to incorporate some distributed computing foundations at an early stage. The synergy resulting from these seemingly competing goals is significant.

In this paper, we will focus on the area of order relations and illustrate how distributed computation concepts can enrich the presentation and increase the motivation of the students. Before getting into those details, we sketch very briefly the preparation of the students and the material that has been covered up to that point, so that the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since the reader will be able to roughly gauge what one can expect from the students at this point, see Section 2. We have chosen partial order relations to illustrate our point, since

2. BACKGROUND
The background for this paper is the course CSCE 222 “Discrete Structures for Computing” at Texas A&M University, which is essentially a course on discrete mathematics that additionally contains a brief discussion of computational models such as finite state automata and Turing machines. The course is taken by computer science undergraduates at the freshmen or sophomore level. The students have some background in programming and have taken at least one course of college-level calculus. The course is proof-
oriented and drills abstraction.

The course begins with a brief introduction to symbolic logic and an introduction to proofs. This is followed by an introduction to set theory that teaches the ZFC axioms, the basic set operations, and the existence of the natural numbers. A brief introduction to the cardinality of sets culminates in a short proof that there exist uncomputable functions. At this point, one usually gets the attention of the students.

The exposition of set theory is followed by a discussion of relations and their properties (including reflexivity, irreflexivity, symmetry, asymmetry, antisymmetry, and transitivity), and a discussion of equivalence relations.

Most courses on discrete mathematics for computer science undergraduates will cover similar material up to this point, so it should not be too hard to integrate the material presented here.

Our goal is to show that one can reach significant depth in an introductory course on discrete mathematics. We begin by discussing the basics of partial orders, strict orders, and preorders. Then we discuss Dilworth’s famous theorem, Shприрн linear extension theorem, and dimension theory. We use these results to establish causal ordering (also known as the happened-before relation). We discuss logical clocks and vector clocks. The results developed allow us to prove Charron-Bost’s celebrated bound on the length of the vector clock from first principles, see [3].

3. DISTRIBUTED COMPUTATION

After relations have been introduced, one can briefly discuss that it is possible to model a computer or a distributed system of computers in an abstract way by a transition system. We will just briefly sketch the idea; the details are nicely explained for instance in Tel [13], so we can be brief here. There are many excellent textbooks on distributed computing that provide further information, see for example Attiya and Welch [1], Garg [6], and Lynch [8].

A transition system is a triple $(C, \rightarrow, I)$ consisting of a set $C$ of all possible configurations or system states, a binary transition relation $\rightarrow$ on the set of configurations, and a subset $I$ of the configurations called the initial configurations.

The system will start in a configuration $c_0$ from $I$ and proceed by making a transition $c_0 \Rightarrow c_1$. In subsequent steps, the computation progresses by transitioning through configurations $c_k \Rightarrow c_{k+1}$. The computation will stop if a state is reached that is not related to any other state in $C$, otherwise the computation will go on indefinitely. In other words, the computation will stop after a finite number of steps if a configuration is reached that belongs to the set $T$ of terminal configurations

$$T = \{ x \in C \mid \text{there does not exist } y \in C \text{ such that } x \Rightarrow y \}.$$ 

Therefore, an execution of a transition system is either

(a) a finite sequence $(c_0, c_1, c_2, \ldots, c_m)$ with an initial configuration $c_0$ in $I$ and such that $c_k \Rightarrow c_{k+1}$ holds for all $k$ in the range $1 \leq k \leq m$ and $c_m \in T$, or

(b) an infinite sequence $(c_0, c_1, c_2, \ldots)$ with initial configuration $c_0 \in I$ and such that $c_k \Rightarrow c_{k+1}$ holds for all $k \geq 0$.

A distributed system consists of a collection of processes and a communication subsystem. Each process is a transition system that is augmented with send and receive operations that allow the processes to interact via the communication subsystem.

![Figure 1: Example of a space-time diagram. There are three processes $p_1$, $p_2$, and $p_3$. We have send events $a$, $b$, and $f$, receive events $c$, $d$, and $h$, and internal events $e$ and $g$. The associated sequence $E$ of events is shown below.](image)

One can view executions as sequences of transitions, so this naturally induces a notion of time. Figure 1 illustrates a distributed system with three processes. The processes progress by either sending or receiving message or doing some local computation. However, one of the key features of a distributed system is that the transitions influence (and get influenced by) merely part of the configuration. Sometimes two consecutive events are independent and influence different parts of the configuration, so one can interchange them without changing the result of the computation. If two events cannot be interchanged, then there is a causal relation between these events.

This suggests that a distributed computation is best understood as an equivalence class of executions, and that it is most convenient to abandon the view that the sequence of events is totally ordered. This motivates one to study relations that allow one to model the causal relationship between events.

4. PREORDERS AND PARTIAL ORDERS

In this and the subsequent sections, we will explore the theory of ordered sets. One can find more background in the textbooks by Caspard, Leclerc, and Monjardet [2], Davey and Priestley [4], Roman [11], Schröder [12], and Trotter [14].

Let $S$ be a set. A relation $\leq$ on $S$ is called a partial order if and only if it satisfies the following three properties:

**O1.** For all $x$ in $S$, we have $x \leq x$ (reflexivity).

**O2.** For all $x$ and $y$ in $S$, $x \leq y$ and $y \leq x$ imply $x = y$ (antisymmetry).

**O3.** For all $x, y, z$ in $S$, $x \leq y$ and $y \leq z$ imply $x \leq z$ (transitivity).

If a relation $\leq$ on $S$ satisfies the properties **O1**–**O3, then $(S, \leq)$ is called a partially ordered set.

A prototypical example of a partial order is given by the power set ordered by set inclusion.

**Example 1.** Let $A$ be a set and $S = P(A)$ its power set. The set inclusion $\subseteq$ defines a partial order on $S$. Indeed, the set inclusion is reflexive, since every set is a subset of itself. For two sets $X$ and $Y$ in $S$, if $X \subseteq Y$ and $Y \subseteq X$ holds, then
the sets $X$ and $Y$ must be the same, so $\subseteq$ is antisymmetric. For all sets $X, Y, Z$ in $S$, the inclusions $X \subseteq Y$ and $Y \subseteq Z$ imply that $X \subseteq Z$, so the relation is transitive.

A relation on $S$ is called a preorder if and only if it is reflexive and transitive. The following example for a preorder considers a simple message passing distributed system, which is a basic method of communication among distributed processes.

**Example 2.** Consider a distributed computing system that consists of a set of cooperating processes that communicate by sending and receiving messages as follows.

(i) Each process can send message to itself.

(ii) If process $p_i$ sends message to process $p_j$, $p_j$ can send message to process $p_k$, then $p_i$ can send message to $p_k$ by $p_j$ forwarding the message of $p_i$ to $p_k$.

The relation “can send message to” defines a preorder (but not partial order) on the set of distributed processes.

Note that if whenever $p_i$ can send message to $p_j$, $p_j$ can also send message to $p_i$, then the “can send message to” relation is also symmetric, and thus the relation becomes an equivalence relation. Also note that if a preorder is also antisymmetric, then it is a partial order. The “can send message to” relation is not a partial order because $p_i$ being able to send a message to $p_j$ and $p_j$ being able to send a message to $p_i$ does not imply that $p_i$ and $p_j$ are the same process.

Two elements $x$ and $y$ of a partially ordered set $(S, \leq)$ are called comparable if and only if $x \leq y$ or $y \leq x$. In a partial order, there might exist elements that are not comparable, as the next example illustrates.

**Example 3.** Let $S = P(A)$ be the power set of a set $A$ ordered by inclusion. Then two elements $X$ and $Y$ of $S$ are comparable if and only if $X$ is a subset of $Y$ or $Y$ is a subset of $X$. For example, if $A = \{1, 2, 3\}$, then the subsets $X = \{1, 2\}$ and $Y = \{2, 3\}$ are not comparable, as none is a subset of the other.

Now, consider the consensus problem in a distributed system that finds consensus among the processes on a common input value. There can be some faulty process that may send arbitrary (not consistent) values as its input value to different processes. The exponential-time algorithm [10] to solve the consensus problem consists of two phases: the first phase is to gather information on the processes’ input values and the second phase to decide on a consensus value among those gathered values according to for example the majority rule. Each process in the first phase proceeds in rounds. The gathered values by a process at the end of each round is called the “view” of the process in that round.

**Example 4.** Let $V$ be the set of all possible input values and $S = P(A)$ be the power set of $V$, then $(S, \subseteq)$ is a partially ordered set. In each round of the information gathering phase of the consensus algorithm described above, there can be some process views that are not comparable.

The term “view” of the distributed processes is often used to express a property of a distributed system such that each process can have only some part of the complete view while the computation is progressing.

A subset $T$ of a partially ordered set $S$ is called a chain if and only if every two elements of $T$ are comparable. If the entire set $S$ is a chain, then it is also called a total order or a linear order.

**Example 5.** The set $\mathbb{R}$ of real numbers with its natural order $\leq$ is a total order.

**Example 6.** The set $\{1, 2, \ldots, n\}$ of the first $n$ natural numbers is a total ordered set under the natural order $1 \leq 2 \leq \cdots \leq n$.

Let $(A, \leq_A)$ and $(B, \leq_B)$ be two partially ordered sets. One can define on the cartesian product $A \times B$ a relation $\leq$ by setting $(a_1, b_1) \leq (a_2, b_2)$ if and only if $a_1 \leq_A a_2$ or $(a_1 = a_2$ and $b_1 \leq_B b_2)$. Then $(A \times B, \leq)$ defines a partially ordered set known as the lexicographic order. The relation $\leq$ on $A \times B$ is a partial order because:

(i) It is reflexive since $\leq_A$ and $\leq_B$ are reflexive.

(ii) Suppose that $(a_1, b_1) \leq (a_2, b_2)$ and $(a_2, b_2) \leq (a_1, b_1)$.

Then we must have $a_1 = a_2$ as well as $b_1 \leq_B b_2$ and $b_2 \leq_B b_1$. By antisymmetry of $\leq_B$, we get $(a_1, b_1) = (a_2, b_2)$. Therefore, $\leq$ is antisymmetric.

(iii) Suppose that $(a_1, b_1) \leq (a_2, b_2)$ and $(a_2, b_2) \leq (a_3, b_3)$.

Then $a_1 \leq_A a_2$ or $(a_1 = a_2$ and $b_1 \leq_B b_2$) as well as $a_2 \leq_A a_3$ or $(a_2 = a_3$ and $b_2 \leq_B b_3$). If either $a_1 \leq_A a_2$ or $a_2 \leq_A a_3$, then $a_1 \leq_A a_3$; otherwise, $a_1 = a_2 = a_3$, and $b_1 \leq_B b_2$, $b_2 \leq_B b_3$, so by transitivity $b_1 \leq_B b_3$.

Therefore, $(a_1, b_1) \leq (a_3, b_3)$ and $\leq$ is transitive.

The lexicographic order is a total order if and only if both $\leq_A$ and $\leq_B$ are total orders.

**Example 7.** Let the IDs (identifiers) of distributed processes be chosen from the cartesian product $A \times \mathbb{N}$, where $A$ is the set of alphabets that are totally ordered by the dictionary order and $\mathbb{N}$ is the set of natural numbers $\{1, 2, 3, \ldots\}$. Then, the lexicographic order defined on the set of process IDs is a total order.

5. **STRICT ORDERS**

Let $S$ be a set. A relation $\prec$ on $S$ is called a strict order if and only if it satisfies the following two properties

**SO1.** The relation is irreflexive, meaning that $x \prec x$ does not hold for any $x$ in $S$.

**SO2.** The relation is transitive, that is, $x \prec y$ and $y \prec z$ imply that $x \prec z$ holds for all $x, y,$ and $z$ in $S$.

If a relation $\prec$ on $S$ satisfies the properties **SO1** and **SO2**, then $(S, \prec)$ is called a strictly ordered set. Strict orders cannot be partial orders, but the two concepts are closely related as we will see. We offer a few typical examples of strict orders.

**Example 8.** The set $\mathbb{N}$ of natural numbers with the strict inequality $\prec$ is a strictly ordered set.

**Example 9.** Let $A$ be a set and $S = P(A)$ its power set. Then the proper subset relation $\subsetneq$ on $S$ is a strict order.

The Happened-Before Relation. In distributed computing, one often considers a collection of processes that communicate by exchanging messages. A process is understood to be a totally ordered sequence of events (such as
machine instructions, the sending or receiving of messages). The events occurring on a process without involving messages are sometimes called "internal" events of the process. Lamport defined the happened-before relation $\rightarrow$ on the events in the distributed system as follows [7]:

(a) Suppose that $a$ and $b$ are internal events that belong to the same process. Then $a \rightarrow b$ if and only if the event $a$ occurs before the event $b$.

(b) Let $a$ be the event of sending a message by one process and $b$ the event of receiving the same message by another process. Then $a \rightarrow b$.

(c) The relation $a \rightarrow a$ does not hold for any event $a$.

(d) For all events $a, b, c$, if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.

The happened-before relation $\rightarrow$ is irreflexive by property (c) and transitive by property (d), hence it is a strict order. The happened-before relation can capture the causality between two processes due to the fact that a message received must have been sent before it is received. Events that are not comparable are potentially concurrent events.

**Logical Clocks.** The happened-before relation is a theoretical concept that explains the ordering of some of the events occurring on the distributed processes. To realize the happened-before relation in the viewpoint of the distributed processes, Lamport introduced the concept of "logical clocks" and the algorithm that implements it [7], which we describe below.

A logical clock $C$ is a function that assigns a number $C(a)$ to an event $a$, and bears no relation to the physical clock time at which $a$ actually occurred. There is a fundamental condition that any correct system of logical clocks must observe.

**Clock Condition:**

For any events $a$ and $b$, if $a \rightarrow b$ then $C(a) < C(b)$.

From the definition of the happened-before relation $\rightarrow$, we see that the Clock Condition is satisfied if the two conditions hold:

**CC1.** If $a$ and $b$ are internal events that belong to the same process $p_i$ and the event $a$ occurs before the event $b$, then $C_i(a) < C_i(b)$.

**CC2.** If $a$ is the event of sending a message by process $i$ and $b$ is the event of receiving the same message by process $p_j$, then $C_i(a) < C_j(b)$.

Now consider the design of a simple distributed algorithm that implements a system of logical clocks that satisfy the two conditions CC1 and CC2. The state of each distributed process $p_i$ is kept in a software counter $t_i$, the value of which will be returned as the clock value of $C_i$.

Each process maintains its counter as follows. To satisfy CC1, each process simply needs to increment its counter between any two successive events. Furthermore, the following two rules are needed to satisfy CC2. If event $a$ by process $p_i$ is the sending of a message $m$, $p_i$ includes $C_i(a) = t_i$ as part of $m$. The counter value contained in $m$ is called the timestamp of $m$, denoted $T_m$. If event $b$ by process $p_j$ is the receipt of a message $m$, $p_j$ sets $t_j$ to be the maximum between $t_j$ and $T_m$, and sets $C_j(b)$ to be an incremented value of $t_j$. As summary,

(a) A process increments its counter between any two successive events that occur in that process.

(b) When a process sends a message, it includes its counter value in the message.

(c) Upon receipt of a message, the receiving process sets its counter to be the maximum between its counter and the message timestamp. Then, set the clock value of the receipt event to be an incremented value of its counter.

The next two propositions show that partial orders and strict orders always occur in pairs. It is a matter of personal preference whether one first specifies a partial order and derives the associated strict order or the other way round.

**Proposition 10.** Let $\leq$ be a partial order on a set $S$. We can define a relation $<$ on $S$ by setting $x < y$ if and only if $x \leq y$ and $x \neq y$. Then the relation $<$ is a strict order on $S$.

**Proof.** The relation $<$ is by construction irreflexive. Suppose that $x < y$ and $y < z$ holds. This means that $x < y$ and $y < z$, as well as $x < y$ and $y < z$. It follows from the transitivity of $\leq$ that $x < z$ holds. Seeking a contradiction, let us assume that $x = z$; this would mean that $x < y$ and $y < x$ hold, so by the antisymmetry of the partial order $\leq$, we must have $x = y$, contradicting the fact that $x \neq y$. Thus, $x < y$ and $y < z$ imply that $x < z$ and $x \neq z$, so $x < z$. Therefore, $<$ is transitive. This shows that $<$ is a strict order, as claimed. $\square$

We record a simple observation before proving our next proposition.

**Lemma 11.** A strict partial order $<$ on a set $S$ is an asymmetric relation, meaning that $x < y$ implies that $y < x$ cannot hold.

**Proof.** Seeking a contradiction, let us suppose that there are two elements $x$ and $y$ in $S$ such that $x < y$ and $y < x$ hold. By transitivity, this implies that $x < x$ must hold. However, this contradicts the irreflexivity of the relation. $\square$

**Proposition 12.** Let $<$ be a strict order on a set $S$. Define a relation $\leq$ on $S$ by setting $x \leq y$ if and only if $x < y$ or $x = y$. Then $\leq$ is a partial order relation.

**Proof.** By definition, $\leq$ is reflexive.

The relation $\leq$ is antisymmetric, since $x \leq y$ and $y \leq x$ implies $x = y$. This is evident when $x = y$. If $x$ and $y$ are distinct elements of $S$, then this follows from the asymmetry of the strict order relation $<$. Suppose that $x \leq y$ and $y \leq z$. If $x = y$ or $y = z$, then it immediately follows that $x \leq z$. If $x \neq y$ and $y \neq z$, then $x < y$ and $y < z$ holds, which implies $x < z$ by transitivity of the strict order; hence, $x \leq z$. Therefore, $\leq$ is a transitive relation. $\square$

The previous two propositions show that one can associate with each strict order a partial order by taking the union with the identity relation. Furthermore, one can obtain a strict order from a partial order by intersection with the relation of distinctness.

It is customary to denote the strict order associated with $\leq$, $\leq$, and $\sqsubset$ respectively by $<, \leq, \subset$, and $\sqsubset$. Sometimes
it is convenient to reverse the symbols. For instance, we write \( x \geq y \) for \( y \leq x \), and \( x > y \) for \( y < x \).

The union of the happened-before relation with the identity relation ("the same event") on the set of distributed events is a partial order relation. Indeed, any event is the same event as itself, thus reflexivity holds. If, for any events \( x \) and \( y \), "\( x \) happened-before \( y \), or \( x \) and \( y \) are the same event" and "\( y \) happened-before \( x \), or \( y \) and \( x \) are the same event", then \( x \) and \( y \) must be the same event, thus satisfying antisymmetry. Furthermore, if, for any events \( x \), \( y \), and \( z \), "\( x \) happened-before \( y \), or \( x \) and \( y \) are the same event" and "\( y \) happened-before \( z \), or \( y \) and \( z \) are the same event", then "\( x \) happened-before \( z \), or \( x \) and \( z \) are the same event", thus transitivity holds.

As noted before, the happened-before relation captures the causality between message send and receive events, and is expressed using the logical clock \( C \) on each process. Thus, if an event \( a \) happened before another event \( b \), then the logical clock value for \( a \) is smaller than that for \( b \) (Clock Condition: if \( a \rightarrow b \), then \( C(a) < C(b) \)).

However, it does not characterize incomparable (or concurrent) events among processes. For some applications such as version control or synchronizing among the processes, it will be helpful to characterize concurrent events as well, so as to enhance the clock condition above to be a bidirectional.

Vector clocks [9, 5] can express concurrency as well as version control or synchronizing among the processes, it will be helpful to characterize concurrent events as well, so as to enhance the clock condition above to be a bidirectional. Each process \( V \) maintains its vector clock according to the following algorithm:

(a) Initially, each element in \( V_C \) is set to zero.

(b) Each time an internal event occurs, increment \( V_C[i] \) (its own logical clock in the vector).

(c) When sending a message, the process first increments \( V_C[i] \), and then piggybacks its entire vector clock in the message.

(d) When receiving a message with the sender’s vector clock \( V_C \), the process first increments \( V_C[i] \); and then sets each element in \( V_C \) to be the maximum between \( V_C \) and \( V_C \), i.e., \( \forall j, i : V_C[i] = \max(V_C[i], V_C[j]) \).

Now we define a relation \( \leq \) on the vector clock values as follows:

\[ V_C \leq V_C \iff \forall k, V_C[k] \leq V_C[k]. \]

For example, for vector clocks of size 4, \([5, 3, 0, 6] \leq [5, 4, 1, 6], [5, 3, 0, 6] \geq [5, 2, 0, 5], [5, 3, 0, 6] \geq [5, 2, 1, 6] \) are incomparable.

The relation \( \leq \) on the vector clock values is a partial order relation since (i) for any vector clock \( V_C \), \( V_C \leq V_C \) (reflexivity), (ii) for any two vector clocks \( V_C \) and \( V_C \), if \( V_C \leq V_C \) and \( V_C \leq V_C \), then \( V_C \) and \( V_C \) have exactly the same clock values in all elements (antisymmetry), and (iii) for any three vector clocks \( V_C \), \( V_C \), and \( V_C \), if \( V_C \leq V_C \) and \( V_C \leq V_C \), then \( V_C \leq V_C \) (transitivity).

If two elements \( x \) and \( y \) of a partially ordered set \((S, \leq)\) are not comparable, then they are called incomparable. A subset \( A \) of a partially ordered set \( S \) is called an antichain if and only if any two elements of \( A \) are incomparable. The next example shows the rather extreme case that a partially ordered set \( S \) itself is an antichain; this is also known as the discrete preorder on \( S \).

**Example 13.** Let \( S \) be a set. The identity relation on \( S \) is a partial order called the discrete preorder on \( S \). This relation is explicitly given by \( \{(x, x) | x \in S \} \). It relates every element \( x \) of \( S \) to itself, but distinct elements are incomparable.

Let \( S \) be a partially ordered set. The maximal cardinality of an antichain in \( S \) is called the width of \( S \). Dilworth showed the remarkable result that there exists \( n = m \) pairwise disjoint chains that partition \( S \).

**Theorem 14 (Dilworth’s Theorem).** Let \((S, \leq)\) be a finite partially ordered set. Then the minimal number \( n \) of chains that partition \( S \) is equal to the maximal cardinality \( m \) of an antichain in \( S \).

**Proof.** Since each chain can contain at most one element of an antichain, we have \( m \geq n \). We show that \( m \geq n \) using a proof by induction.

If \( |S| = 0 \), then no chains are needed to cover \( S \) and the maximal cardinality of an antichain is 0, so the claim holds in this case.

Let us assume that the claim holds for all partially ordered sets of cardinality \(|S| < k \). Consider a partially ordered set of cardinality \(|S| = k \), and let \( C \) be a maximal chain of \( S \).

**Case 1.** The chain \( C \) meets every maximal antichain in \( S \) of cardinality \( m \). Then the cardinality of a maximal antichain in \( S \) is \( m - 1 \). By induction hypothesis, \( S \) can be partitioned into \( m - 1 \) chains. So adding the chain \( C \), we can conclude that \( S \) can be partitioned into \( n = m \) chains.

**Case 2.** The chain \( C \) does not meet the maximal antichain \( A = \{a_1, \ldots, a_m\} \) in \( S \). Define the sets

\[ A_d = \{x \in S \mid \text{there exists an } a \in A \text{ such that } x \leq a\}, \]

and

\[ A_u = \{x \in S \mid \text{there exists an } a \in A \text{ such that } x \geq a\}. \]

Since \( A_d \) and \( A_u \) both include the antichain \( A \), they still have width \( m \).

Let \( x \) be an element of \( S \). If \( x \) were incomparable with every element in \( A \), then \( A \cup \{x\} \) would be an antichain, which contradicts the maximality of the antichain \( A \). Therefore, \( S = A_d \cup A_u \). If there would be an element \( x \in S \) and different elements \( a \) and \( a' \) in \( A \) such that \( a \leq x \leq a' \), then this would contradict that \( A \) is an antichain. Therefore, \( A_d \cap A_u = A \).

The greatest element of \( C \) cannot belong to \( A_d \), since it would be less than some element of \( A \), contradicting the maximality of the chain \( C \). Thus, \( A_d \) is a proper subset of \( S \); hence, by induction hypothesis, \( A_d \) can be partitioned into \( m \) chains \( C_d \) with \( a \in A \) such that \( a \in C_d \). Similarly, \( A_u \) is a proper subset of \( S \) that can be partitioned by induction hypothesis into \( m \) chains \( C_u \) with \( a \in A \) such that \( a \in C_u \). Then \( C_d \cap C_u = \{a\} \) holds for each \( a \in A \).

The element \( a \) must be the greatest element of \( C_d \) and the least element of \( C_u \), so \( C_u \cup C_d \) is a chain in \( S \). Therefore, \( S \) can be partitioned into the \( m \) chains \( C_u \cup C_d \) with \( a \in A \), as claimed. □
7. EXTENSIONS OF PARTIAL ORDERS

Given two partial orders \( P \) and \( Q \) on a set \( S \), we say that \( Q \) extends \( P \) if and only if \( P \subset Q \) holds. In this section, we are going to show that each partial order can be extended to a total order. These results have numerous applications.

**Proposition 15.** Let \( \leq \) be a partial order on a set \( S \). Let \( a \) and \( b \) be incomparable elements of \( S \). Then there exists a partial order \( \leq \) extending \( \leq \) such that \( a \nleq b \).

**Proof.** We are going to define the partial order \( \leq \) on \( S \) by including all pairs from the relation \( \leq \), the pair \((a, b)\), and additional pairs that restore transitivity. Let us define the sets

\[
A = \{ x \in S \mid x \leq a \} \quad \text{and} \quad B = \{ x \in S \mid b \leq x \}.
\]

We define the relation \( \leq \) to be the cartesian product \( A \times B \). Hence, the elements in \( A \) precede the elements in \( B \) under \( \leq \); in particular, \( a \nleq b \). We define \( \leq \) as the union of the relations \( \leq \) and \( \nleq \). In other words, the relation \( \leq \) consists of the set of pairs

\[
\{(x, y) \in S \times S \mid x \leq y\} \cup (A \times B).
\]

It remains to show that \( \leq \) is a partial order. The relation \( \leq \) is reflexive, since it contains the reflexive relation \( \leq \).

Seeking a contradiction, let us assume that \( \leq \) is not antisymmetric, that is, there must exist distinct elements \( x \) and \( y \) in \( S \) such that \( x \leq y \) and \( y \leq x \). These relations can arise from \( x \leq y \) or \( x \nleq y \) and from \( y \leq x \) or \( y \nleq x \), so we distinguish four cases. (a) If \( x \leq y \) and \( y \leq x \), then \( x = y \) by antisymmetry of the relation \( \leq \), which contradicts that \( x \) is distinct from \( y \). (b) If \( x \leq y \) and \( y \nleq x \), then \( b \leq x \), \( x \leq y \), \( y \leq a \), so by transitivity \( b \leq a \), which contradicts the incomparability of \( a \) and \( b \). (c) If \( x \nleq y \) and \( y \leq x \), then \( b \leq x \), \( x \leq y \), \( y \nleq a \), so by transitivity \( b \nleq a \), which contradicts the incomparability of \( a \) and \( b \). (d) If \( x \nleq y \) and \( y \nleq x \), then \( b \leq x \) and \( x \nleq a \), so by transitivity \( b \nleq a \), which once again contradicts the incomparability of \( a \) and \( b \). Since we arrived at a contradiction in each of the four cases, we can conclude that \( \leq \) is antisymmetric.

For the transitivity, we need to show that \( x \leq y \) and \( y \leq z \) implies \( x \leq z \). We will distinguish for \( x \leq y \) the cases \( x \nleq y \) and \( x \leq y \) and for \( x \leq z \) the cases \( y \nleq z \) and \( y \leq z \), leading to a total of four cases. (a) If \( x \leq y \) and \( y \nleq z \), then \( x \nleq z \) by transitivity. (b) If \( x \nleq y \) and \( y \leq z \), then \( x \nleq y \), \( y \nleq a \), so by transitivity \( x \nleq a \), which implies \( x \nleq z \), so \( z \) is an element of \( B \). As \( B \) is a set, we have \( x \nleq z \). Hence, \( x \leq z \). (c) If \( x \nleq y \) and \( y \nleq z \), then \( b \nleq y \) and \( y \nleq z \), which implies \( b \nleq z \), so \( z \) is an element of \( B \). As \( B \) is a set, we have \( x \nleq z \). (d) If \( x \nleq y \) and \( y \nleq z \), then \( x \in A \) and \( z \in B \), so \( x \nleq z \). In all four cases, we can conclude that \( x \leq z \). □

The next axiom is known as Zorn’s Lemma. It is equivalent to the axiom of choice. It is frequently used to establish the existence of maximal elements in infinite partially ordered sets.

**Axiom 1 (ZORN’S LEMMA).** Let \((S, \leq)\) be a partially ordered set. If every nonempty chain in \( S \) has an upper bound, then the set \( S \) has a maximal element.

A linear extension of a partial order \( P \) is a total order that extends \( P \). The next theorem shows that every partial order has a linear extension.

**Theorem 16.** Every partial order can be extended to a total order.

**Proof.** Let \( P \) be a partial order on a set \( S \). We can define the family \( F \) of partial orders by

\[
F = \{ Q \mid Q \text{ is a partial order on } S \text{ containing } P \}.
\]

The set family \( F \) is partially ordered by set inclusion. For every chain \( C \) in \( F \), a least upper bound is given by \( \bigcup C \). By Zorn’s lemma, there must exist a maximal element \( M \) in \( F \). It follows from Proposition 15 that \( M \) must be a total order. □

**Remark 17.** One can include a discussion of topological sorting as an application.

8. DIMENSION

**Proposition 18.** Let \( E \) denote the set of all linear extensions of a partial order \( P \). Then the intersection of all linear extensions satisfies \( \bigcap E = P \).

**Proof.** The set \( E \) is not empty by Theorem 16. Since \((x, y) \in P \) implies \((x, y) \in E_i \) for all linear extensions \( E_i \in E \), we have

\[
P \subseteq \bigcap E.
\]

If \( x \) and \( y \) are two incomparable elements of \( P \), then there exist a linear extension \( E_1 \) of \( P \) satisfying \( x \notin E_1 y \) and another linear extension \( E_2 \) of \( P \) satisfying \( y \notin E_2 x \). Therefore, \( x \) and \( y \) are incomparable in \( \bigcap E \). It follows that \( \bigcap E \subseteq P \). □

Let \( d \) be the smallest cardinal such that there exist a set \( R \) of dimension \( d \) linear extension of \( P \) such that \( \bigcap R = P \). Then \( d \) is called the dimension of the partially ordered set \( P \). The dimension of \( P \) is denoted by \( \dim P \).

This combinatorial notion of dimension might appear a bit strange. We will now relate it to more familiar notions.

Suppose that \((A, \leq)\) and \((B, \subseteq)\) are partially ordered sets. A function \( f : A \to B \) is called an embedding if and only if \( x \leq y \) if and only if \( f(x) \subseteq f(y) \). An embedding \( f \) of partial orders is necessarily an injective function, since \( f(x) = f(y) \) implies \( x \subseteq y \) and \( y \subseteq x \), so \( x = y \). Thus, if one restricts the codomain to the image of \( f \), then \( f \) becomes an order isomorphism, which explains why the term “embedding” is used.

The next theorem shows that if we can embed a partial order into a cartesian product \( P = \prod_{k \in I} C_k \) of totally ordered sets \((C_k, \leq_k)\), where the product order \((P, \leq)\) is defined as \((x_k : k \in I) \leq (y_k : k \in I)\) if and only if \( x_k \leq_k y_k \) holds for all \( k \in I \). The theorem says that it is only possible to embed a partial order of dimension \( n \) into, say, the partially ordered set of \( \mathbb{N}^d \) of vectors of natural numbers when the dimension \( d \) is large enough, namely \( d \geq n \).

**Theorem 19 (ORE).** If a partially ordered set \((S, \leq)\) can be embedded into a cartesian product of \( d \) totally ordered sets, then \( d \geq \dim P \).

**Proof.** Suppose that \((C_k, \leq_k) \mid k \in I\) are \( |I| \) totally ordered sets and that \( f : S \to \prod_{k \in I} C_k \) is an embedding into the cartesian product of the totally ordered sets. In other words, for \( x, y \in S \), we have \( x \leq y \) if and only if \( f(x)_k \leq_k f(y)_k \) holds for all \( k \in I \).
We define a relation \( \sqsubset_k \) on \( S \) as the union of \( \leq \) with \( \{(x, y) \in S \times S \mid f(x)_k < f(y)_k \text{ and } x, y \text{ incompr. in } \leq\} \). Then \( (S, \sqsubset_k) \) is a partial order. By Theorem 16, we can extend \( \sqsubset_k \) to a total order \( E_k \).

It remains to show that the intersection \( \bigcap \{E_k \mid k \in I\} = P \). Indeed, \( P \) is contained in \( E_k \) for all \( k \in I \) by construction, so \( P \) is a subset of \( \bigcap \{E_k \mid k \in I\} \). If \( x \nleq y \), then there must exist some index \( k \in I \) such that \( f(x)_k > f(y)_k \), as \( f \) is an embedding into the cartesian product. It follows that \( (y, x) \in E_k \); hence, \( (x, y) \notin E_k \). Therefore, the intersection \( \bigcap \{E_k \mid k \in I\} \) of the total orders \( E_k \) does not contain pairs that are not related in \( (S, \leq) \). Therefore, we can conclude that \( \bigcap \{E_k \mid k \in I\} = P \). By definition of the dimension of a partial order, it follows that \( d \geq \dim P \).

Let \((P, \leq)\) be a partially ordered set. Let us write \( p \parallel q \) when \( p \) and \( q \) are incomparable elements, so \( p \nleq q \) and \( q \nleq p \).

**Lemma 20.** Let \((P, \leq)\) be a finite partially ordered set. Let \( C \) be a chain in \((P, \leq)\) and
\[
B = \{p \in P \mid \text{for all } c \in C, p \parallel c\}
\]
the set of all elements that are incomparable with all elements in the chain \( C \). Then there exists a linear extension \( \sqsubseteq \) of \( \leq \) such that \( b \sqsubseteq c \) for all \( b \in B \) and all \( c \in C \).

**Proof.** Suppose that \( C = \{c_1, \ldots, c_k\} \) are the elements of the chain \( C \) such that \( c_1 < c_2 < \cdots < c_k \). For all integers \( m \) in the range \( 1 \leq m \leq k \), let \( A_m \) be the set of all elements \( p \in P - C \) such that \( c_m \) is the largest element in \( C \) such that \( c_m < p \). Furthermore, define \( A_0 \) to be the set of the remaining elements,
\[
A_0 = P - (C \cup A_1 \cup \cdots \cup A_k).
\]
If \( i \) and \( j \) are integers in the range \( 0 \leq i < j \leq k \), then the elements \( a_i \in A_i \) and \( a_j \in A_j \) satisfy \( a_i \nleq a_j \).

For each integer \( i \) in the range \( 0 \leq i \leq k \), we find a linear extension \( \sqsubseteq \) of \( \leq \) restricted to the set \( A_i = \{a_1^i, a_2^i, \ldots, a_n^i\} \), with superscripts chosen such that \( a_1^i \sqsubset a_2^i \sqsubset \cdots \sqsubset a_n^i \). Then the linear extension \( \sqsubseteq \) of \( \leq \) is given by
\[
a_1^0 \sqsubset a_2^0 \sqsubset \cdots \sqsubset a_n^0 \sqsubset c_1 \sqsubset a_1^1 \sqsubset a_2^1 \sqsubset \cdots \sqsubset a_n^1 \sqsubset \cdots \sqsubset c_k \sqsubset a_1^k \sqsubset a_2^k \sqsubset \cdots \sqsubset a_n^k.
\]
This yields the claim, since the set \( B \) is contained in \( A_0 \).

**Theorem 21.** Let \( P \) be a finite partially ordered set. Then \( \dim P \leq \width P \).

**Proof.** By Dilworth’s theorem there exists a decomposition of \( P \) into \( w = \width P \) chains \( C_w \),
\[
P = C_1 \cup \cdots \cup C_w.
\]
For each integer \( i \) in the range \( 1 \leq i \leq w \) there exists by the previous lemma a linear extension \( \leq_i \) of \( \leq \) such that \( p \leq_i c_i \) holds for all \( p \in B_i \) and all \( c_i \in C_i \), where the set \( B_i = \{p \in P - C_i \mid p \parallel c_i \text{ for all } c_i \in C_i\} \).

Let \( p \) and \( q \) be incommparable elements in \( P \). We have \( p \in C_i \) and \( q \in C_j \) for some integers \( i \) and \( j \) in the range \( 1 \leq i < j \leq w \). Then \( q < p \) and \( p < j \). It follows that the partial order \( \leq \) is the intersection of the total orders \( \leq_{1, \ldots, \leq_w} \); whence \( \dim P \leq w \), as claimed.

**Theorem 22 (Charron-Bost).** A vector clock that allows one to characterize concurrency for computations distributed over \( n \) processes must in general use vectors of dimension \( d \geq n \).

**Proof.** A computation distributed over \( n \) processes can have at most \( n \) concurrent events. Therefore, the dimension \( D \) of the partial order of events is bounded by \( D \leq n \), cf. Theorem 21. For a generic computation, this bound will be attained. By Theorem 19, embedding the partial order of events into \( \mathbb{N}^d \) requires \( d \geq n \).

**9. CONCLUSIONS**

In this paper, we showed how distributed computing concepts can be used as motivating examples of order relations that are fundamental in discrete mathematics. The early introduction of distributed computing concepts may motivate the students to further explore the area of parallel and distributed computing during their undergraduate studies. The course module presented in this paper can be adapted in various ways. It can be shortened by omitting some of the proofs or extended by adding more applications in distributed computing.

**10. REFERENCES**


