Containers and Relations

Carlos Moreno
cmoreno@uwaterloo.ca
EIT-4103

https://ece.uwaterloo.ca/~cmoreno/ece250

These slides, the course material, and course web site are based on work by Douglas W. Harder

Containers and Relations

Standard reminder to set phones to silent/vibrate mode, please!

Containers and Relations

- In today's class:
  - We'll investigate Containers and the constraints on these according to the data they store.
  - In particular, we'll focus on the relationships in the data that constraint the data structures — we'll investigate:
    - Linear (or total) ordering
    - Partial ordering
    - Hierarchical ordering
    - Equivalence relations
    - Weak ordering.
    - Adjacency relations

Containers and Relations

- Let's start with a (rather open) question:

  How do we store data in memory for an algorithm or for algorithms to work with it?

Containers and Relations

- Let's start with a (rather open) question:

  How do we store data in memory for an algorithm or for algorithms to work with it?

- Can we really answer that question as posed?

Containers and Relations

- Say that we try to answer it — something like «well, we use classes to represent things from the real world that we need to deal with, and that simply creates a memory layout for an object to be represented in the memory of a computer. The rest follows directly — if we have many of those objects, we just store one after the other»

Containers and Relations

- Say that we try to answer it — something like «well, we use classes to represent things from the real world that we need to deal with, and that simply creates a memory layout for an object to be represented in the memory of a computer. The rest follows directly — if we have many of those objects, we just store one after the other»
- Well, ok, we create a class that “contains” them (i.e., a class to represent a collection of them)

Containers and Relations

- Let's give that class some fancy name, say, let's call it a *Container*, and problem solved!

Containers and Relations

- Let's give that class some fancy name, say, let's call it a *Container*, and problem solved!

- Ok, so I didn't mean to sound sarcastic with the fancy name — we will actually call them containers :-)

Containers and Relations

- Let's give that class some fancy name, say, let's call it a *Container*, and problem solved!

- Ok, so I didn't mean to sound sarcastic with the fancy name — we will actually call them containers :-)
- The “delusional” part is pretending that such a simple approach like storing one object after the other in memory will solve our data storage and processing problems!

Containers and Relations

- The trick of storing objects sequentially could work in many cases — that's what an array or a linked list is; they're good to store a group of values or a group of objects (e.g., an array of 100+ students for the purpose of processing grades and automating the grade reporting process)

Containers and Relations

- However, let's picture this:
  We create a class to represent a City (perhaps with geographic coordinates — latitude and longitude).
- Then one class to represent roads, avenues and freeways/highways; perhaps we use a class Polygon with a linked list of vertices to represent the geometry of the particular road.

Containers and Relations

- And then, well, we just set up a container to hold a collection of each of those ...
- Quite easy, right? Just declare:

```
Single_list<City> north_american_cities;
Single_list<Road> north_american_roads;
```

- How far do we think these will take us?? (Hint: try to find the shortest, or fastest, route from, say, Waterloo to Quebec City)

Containers and Relations

- The take-away from this example is that depending on what we need to do with the data, we'll need to store it differently.
- And it's really the types of relationships between data items that most often determine what we can or cannot do easily, as well as how should we store the data to accomplish those “typical” tasks.

Containers and Relations

- Notice that some of the operations on the containers themselves may be affected by the type of data — or more specifically, by the existing relationships between data items.
- For example, a typical operation we may expect from a container is a method max(), that returns the highest element currently stored by the container.
- Would that make sense for the list of cities??

Containers and Relations

- It's not because in C++ or in Java or in whatever language we can't figure out how to do that: it's really a matter of the relationships between data items (cities, in this example): There is no *greater-than* relationship for cities!

Relationships

- We'll look at the following types of relationships:
  - Linear or Total ordering
  - Partial ordering
  - Hierarchical ordering
  - Equivalence relations
  - Weak ordering
  - Adjacency relations

Relationships

- Linear or Total ordering:
  - A binary relationship (we'll use ◄ to denote it) on the elements of a set A with the following properties: for every pair of elements x, y, z ∈ A, it holds that:
    - If x ◄ y and y ◄ x, then x = y (antisymmetry)
    - If x ◄ y and y ◄ x, then x ◄ z (transitivity)
    - Either x ◄ y or y ◄ x (totality)

Relationships

- The most typical (and perhaps simplest) example of linear or total ordering is the order in the real (or integer, or rational) numbers given by the relationship < or ≤
- The intuitive meaning is that we can place the elements in order (given antisymmetry and transitivity) using a one-dimensional (linear) arrangement (given the totality)

Relationships

- A non-example is the complex numbers, or the set of points in the plane or in the space.
- Even if we attempt something like ordering by magnitude, it's easy to verify that it won't work (i.e., it won't satisfy the three properties)
- We simply can not place them in a linear arrangement where the order is given by the relationship.

Relationships

- Lexicographical ordering:
  - An interesting extension is the notion of orderings *induced* by a given total ordering.
  - For example, the English alphabet is a totally ordered set (the relationship being defined by the order of the letters).
  - This *induces* a total ordering in words (sequences of letters) — we usually/informally refer to this as "alphabetic" order; the "formal" term should be *lexicographical* order.

Relationships

- Lexicographical ordering:
  - Lexicographical order in a pair of elements of a totally ordered set is given by:

    $x_1 x_2 \blacktriangleleft y_1 y_2$ if:

    - $x_1 \blacktriangleleft y_1$

      OR

    - $x_1 = y_1$ AND $x_2 \blacktriangleleft y_2$
  - The definition can be recursively extended to sequences of more than two elements.

Relationships

- Typical operations on a container for totally ordered elements (a sequential container) are:
  - Determine first and last elements
  - What is the k^{th} element?
  - Given an element in the container:
    - What is the previous element?
    - What is the next element?

Relationships

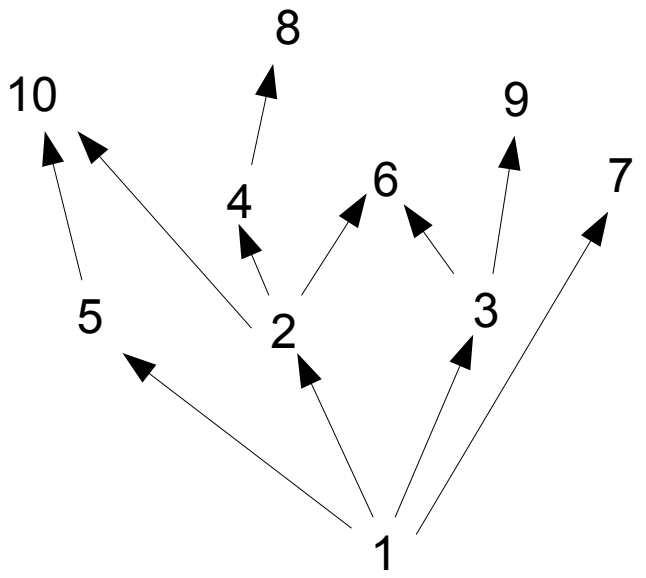
- Partial ordering:
  - Similar to total ordering, except that it does not require totality; that is, not every pair of elements need to be related by the binary relationship.
  - This leads to a relationship that is *somewhat similar* to a hierarchical relationship (emphasis in the *somewhat* similar — they're two different things)

Relationships

- One of the typical examples for this ordering relation is divisibility in the set of positive integer numbers — the relation $n \blacktriangleleft m$ represents the condition "*n* divides *m*"
- Notice that we can't place the numbers in linear order due to lack of totality — for example, given 5 and 7, neither $5 \blacktriangleleft 7$ nor $7 \blacktriangleleft 5$, so which one would we put first??
- (we can't say «we put 5 first since 5 is less than 7», since that obeys *a different* relationship)

Relationships

- However, we could draw them in a “graph” or “lattice” graphical representation:



Relationships

- And seeing that graphical representation, does that give you any ideas? That is, can you suggest some other example(s) of partial ordering?

Relationships

- Typical operations on containers with partially ordered elements:
  - Given two elements: does one precede the other?
  - Find the elements which have no predecessors.
  - Given an element in the container, determine the set of elements that immediately precede it

Relationships

- Hierarchical ordering
  - This can be seen as a constrained type of partial ordering — one where there may be no loops:
    - That is, if you follow the paths formed by consecutive arrows representing relationship between pairs of elements, you can never get to the same element following different paths.
  - And also, there must be a single “root” (a single element that has no predecessors).

Relationships

- As we will see (later on during the course), hierarchical relations are represented by *Trees*.

Relationships

- An example of hierarchical relation is that of ancestry — but only if we restrict it to one ancestor; for example, a maternal ancestor. (why do we need this restriction?)

Relationships

- Another example: structure of directories and sub-directories on a file system.

Relationships

- Typical operations on hierarchical data:
  - Given two elements, does one precede the other?
  - Are two given elements at the same depth?
  - Given two elements, find the nearest common predecessor.

Relationships

- Equivalence relations:
  - An equivalence relation on a set A, denoted ~, has the following three properties for every x, y, $z \in$ A:
    - Reflexivity ($x \sim x$)
    - Symmetry ($x \sim y \Rightarrow y \sim x$)
    - Transitivity (if $x \sim y$ and $y \sim z$, then $x \sim z$)

Relationships

- Equivalence relations:
  - An interesting (and important) characteristic of every equivalence relation is that it partitions the set into disjoint subsets, called *equivalence classes*.
  - Every element in an equivalence class is related to every other element in the same equivalence class, and is related to no other element in the set (i.e., in any other equivalence class)

    (why? Can you prove this, based on the properties that define equivalence relation?)

Relationships

- Equivalence relations:
  - A consequence of this is that you can identify an entire class (one of the equivalence classes) by picking one element of that class — a *class representative*.

Relationships

- Examples of Equivalence relations:
  - People with the same age — clearly, this partitions the population (or whatever subset that we're considering) into disjoint (non-overlapping) sets: people who are 1 year old, people who are 2 year old, and so on; everyone in one of these groups is related to everyone else in that group (since they have the same age!), and is related to no-one in any other groups (since they do not have the same age).

Relationships

- Examples of Equivalence relations:
  - Notice that in this example, we don't need to pick a class representative to identify one class — we could simply identify the classes by the age (when we say “people who are 20 years of age”, we are unambiguously identifying that subset of people)
  - Still, we could pick one of those persons (any one), and we would still unambiguously identify the entire class by using that class representative)

Relationships

- Examples of Equivalence relations:
  - A more mathematical example: the numbers modulo *m*.
  - The relation being: two numbers are related (equivalent) if they have the same value modulo *m*. (that is, numbers that have the same remainder when dividing by *m*).
  - This equivalence relation partitions the set of integer numbers into *m* equivalence classes.

Relationships

- Examples of Equivalence relations:
  - For example, if m = 100, then two numbers are related if the last two digits are the same (assuming digits in decimal representation, of course)
  - We observe that in this case, by picking a class representative (one of the numbers), we identify the entire equivalence class:
    - The number 73 (in this context) automatically identifies the set $\{73 + 100\,k \mid k \in \mathbf{Z}\}$

Relationships

- There is one example that we have seen and we have been using (quite a lot, actually!) in class (in the previous few classes).
- Can you think of such example? And explain why it constitutes an equivalence relation?

Relationships

- Typical operations for an equivalence relation:
  - Are two given elements related?
  - Iterate through all elements related to a particular element (equivalently, iterate through all the elements of an equivalence class, as specified by any given class representative)

Relationships

- Weak ordering:
  - The mathematical definition is somewhat tricky/involved (though it is quite neat! I invite you to look it up on, say, Wikipedia — not today, though! Kudos to Wikipedia for their blackout today!)
  - But the idea being that a weak ordering is a linear or total ordering of equivalence classes.

Relationships

- Weak ordering:
  - The mathematical definition is somewhat tricky/involved (though it is quite neat! I invite you to look it up on, say, Wikipedia — not today, though! Kudos to Wikipedia for their blackout today!)
  - But the idea being that a weak ordering is a linear or total ordering of equivalence classes.
  - An example being, ordering persons by age; we can arrange *the groups* in a linear fashion.

Relationships

- Given a weak ordering, typical operations are the same as those for linear orderings and equivalence classes.
- However:
  - The smallest and largest elements are not necessarily unique (leading to the common notion that a smallest element is one such that no other element is smaller than it)
  - Next or previous elements may be equivalent.

Relationships

- Adjacency relations:
  - Adjacency relations can be defined in an abstract way as a relationship between the elements of a set where the only possible (though not necessary) constraints would be reflexivity and symmetry:
    - No transitivity, no totality.

Relationships

- Typical examples are relationships given by some form of “connections”:
  - $x \sim y$ if x and y are friends (presumably this is an example where the relation is symmetric)
    - For Social networks systems, this seems like the most fundamental notion to consider!
  - Physical adjacency is another obvious example (i.e., “being neighbour”)

Relationships

- Concrete examples of physical adjacency are:
  - Intersections in a city: streets represent the relationship between two intersections — i.e., two intersections are related (adjacent) if they are directly connected by a street.
  - Circuit elements — points in a circuit are “adjacent” if there is a trace (a connection) between them.

Relationships

- Adjacency relations are typically represented by Graphs — elements are only aware of their neighbouring, or adjacent elements, without any global relationship being directly present.

  (we'll see a lot of this later on during the course!)

Relationships

- Typical operations on a container aware of adjacency relations (a graph):
  - Are two elements adjacent?
  - Iterate through all the elements adjacent to a given element.
  - Given two elements *x* and *y*, is there a sequence of adjacent elements that connect *x* to *y*?