We can model this problem as follows:

- A *box* is defined as a triple (b1, b2, b3), where bi is the dimension of the box in direction i.
- We assume without loss of generality that the dimensions are sorted in ascending order, i.e. b1 <= b2 <= b3. So our algorithm needs to 3-sort the dimensions of each box in the input before further processing.
- Then, a box (a1, a2, a3) fits inside another box (b1, b2, b3) iff. a1 < b1, a2 < b2 and a3 < b3. This is, because if we can rotate a box (i.e. permute the dimensions) such that ax < b1, ay < b2, az < b3, then a1 < b1, a2 < b2, a3 < b3, since if not, well for sure a1 <= ax < b1 since a1 is the minimum of ax, ay, az, so either a2 >= b2 or a3 >= b3. But if a2 >= b2, then a3 >= a2 >= b2 >= b1, so ax must be a1 but then we cannot fit a3 nor a2 in b2, hence we cannot fit the boxes. And if a3 >= b3, then a3 >= b3 >= b2 >= b1, so we cannot fit a3 in any of b1, b2, b3, hence we cannot fit the boxes. So (a1, a2, a3) fits inside (b1, b2, b3) by a suitable permutation, if and only if (a1, a2, a3) fits componentwise (b1, b2, b3). Given that the dimensions are sorted.

This gives a strict partial order "(a1, a2, a3) < (b1, b2, b3)", denoted "<", on the set of n boxes: because

- < is transitive: if a < b and b < c, then ai < bi < ci for each i, hence a < c.
- < is antisymmetric: if a < b, then ai < bi for each i, hence a /= b and not b < a.

This makes the set of n boxes ordered by < into a partially ordered set (poset). A storing strategy is simply a partition of this set into chains: a chain is a nonempty subset of elements a1, a2, ... , ak, that are each comparable, i.e. we can say a1 < a2 < ... < ak. So we can put a1 into a2, a2 into a3, etc. The maximum element in this chain, say ak, is the element such that ai < ak for all i = 1, ... , k-1, and this is referred to as a final box in the assignment. Since a chain has only one maximum element (the outermost box), the number of chains in the partition is precisely the number of final boxes of the strategy. We seek to minimize this number.

Note that chains can be singleton sets: this simply means that they consist of a box that is both final and not contained in any other box.

Some terminology regarding posets:

- An element m is maximal if there is no element x such that m < x
- An element M is maximum if for all x, x < M
- An element l is minimal if there is no element x such that x < l
- An element L is minimum if for all x, L < x

Notice that maximum implies maximal and minimum implies minimal, by antisymmetry. In chains, which are totally ordered subsets, the two are equivalent since all elements are comparable, so for x, y unequal, we have either x < y or y < x.

The mimizing storing strategy amounts to finding a chain partition of the set of n boxes (call this set P, for brevity) that has a minimum number of chains (a minimum partition).

## Solution

Define the volume of a box (x,y,z) to be vol(x,y,z) = x * y * z.

- If a box (x,y,z) fits inside a box (x',y',z'), then it follows that vol(x,y,z) = x * y * z < x' * y' * z' < vol(x',y',z'). Denote this (x,y,z) <_vol (x',y',z')

- The first step of the algorithm is to sort all boxes in P on their volumes. This is a topological sort extending the partial order < on P, because <_vol is total order extension of <.

## Wrong solution (Greedy)

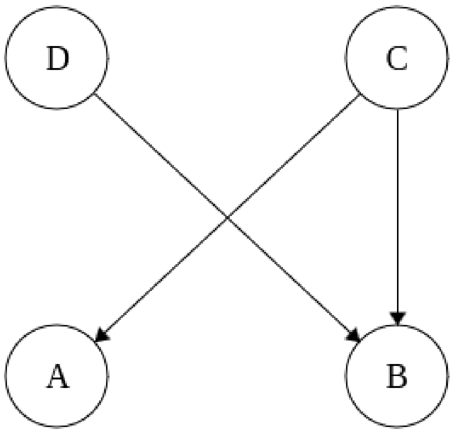
The front element of a topological sorting of the poset is clearly minimal, otherwise it would have been preceded by the element that was smaller. A first idea is to make an algorithm that builds a chain partition inductively:

- Let P' be the poset when we remove a minimal element a (i.e. the front of the topologically sorted list).
- Find a minimum chain partition of P', call it B'
- Try to fit a in front of a chain in B'. Two cases:
  1. If this can be done, then let B be the partition arising from putting a in that chain in B'. If B' consists of w elements, then B too. So B is obviously minimum, otherwise B' cannot have been minimum, because if there is a partition M of P into less than w chains, then removing a from the chain in M where it occurs will also give a partition M' into less than w chains, thereby contradicting minimality of B'.
  2. If it cannot be done, put a in a separate singleton chain and append this to B'.

```python
def findMinChainPartition(rel, topologically_sorted_poset):
    if not topologically_sorted_poset:
        return []
    else:
        minimal_element = topologically_sorted_poset[0]
        inductive_partition = findMinChainPartition(rel,
topologically_sorted_poset[1:])

        minimal_element_fits_somewhere = False

        for chain in inductive_partition:
            if rel(minimal_element, chain[0]):
                chain.insert(0,minimal_element)
                minimal_element_fits_somewhere = True
                break

        if not minimal_element_fits_somewhere:
            inductive_partition.append([minimal_element])

        return inductive_partition
```

This solution seems right at first glance, and it even seems that you can prove correctness simply based on an inductive argument, which seems to hold in case 1.

In case 2, however, the argument fails to hold. Why? In this case, we can have a minimum chain B' for P', without the chain constructed from B' by appending a singleton {a} being minimum. Yes, if B is minimum for P, then B' should be minimum for P'. But what if B' is not a unique minimum partition? Then, if we find another minimum partition B'' for P' in the inductive step, it might not hold that B constructed from B'' is actually minimum. Consider the following counterexampling poset (we draw its Hasse diagram):



A topological sort of this poset would be {A,B,C,D}. The algorithm will consecutively find partitions of {D}, {C,D}, {B,C,D}, {A,B,C,D}, by each time trying to push the front element to a chain partition yet made, and else giving it a new singleton chain. Running it in this manner will give the intermediate paritions:

1. {}
2. {{D}}
3. {{C},{D}}
4. {{B,C}, {D}}

And here we see the algorithm fail: partitions 1-4 are all fine minimal chain partitions. The problem is that {B,C,D} admits multiple minimum partitions. One of these, {{C}, {B,D}}, will indeed lead to a minimum partition {{B,C},{A,D}} for {A,B,C,D} (to argue that this chain is indeed minimum, we need at least 2 chains because the given poset is obviously not totally ordered. Alternatively, use Dilworth's theorem and the antichain {A,B} consisting of 2 elements). But the found minimum chain partition {{B,C}, {D}} will lead to {{B,C}, {D}, {A}}, because we cannot fit A in any chain anymore.

In other words, this algorithm is only correct if for any subposet P' of n elements of the given poset P, a minimum chain partition for P' is unique. For many small trivial posets such as the one given above, this already does not hold. Thereby the algorithm is not correct...