

# DATA STRUCTURES & ALGORITHMS

- *a fundamental mathematical insight*

Niha N. Shaikh

[niha\\_13@hotmail.com](mailto:niha_13@hotmail.com)

Mohit P. Tahiliani

[tahiliani.nitk@gmail.com](mailto:tahiliani.nitk@gmail.com)

*“A time will come when our descendants will be amazed  
that we had no knowledge of such obvious things”*

- Lucius Annaeus Seneca (c 4 B.C – 65 A.D)

Searching and Sorting are fundamental algorithms or operations (performed on a collection of data = operands, information) to extract some information. Let us look into the underlying mathematical ideas that allow us to perform these operations.

Data is information which may be viewed as an operand. For example:

$$X \text{ apples} + Y \text{ apples} = (X + Y) \text{ apples}$$

Here X and Y are the operands (information) and + is the operation (algorithm). The plus operation performed on the given information X and Y extracts the information of the total number of apples (X + Y). In general the operands may not be so simple. We will need some kind of data structure to store the operand or information. Also the operation may not be so simple. We will need to design an operation or algorithm to extract the required information.

## SEARCH OPERATION

1. Let our basic information be a set of cell phones  $C = \{c_1, c_2, c_3, c_4, c_5\}$ . At the set level we may play around with sets and subsets and extract some information. The operations available to us at this (higher) level are union, intersection, cardinality, difference, symmetric difference, complement and so on. But if we want to know more we have to go deeper into the set or sets. For example, we may have a set of students  $S = \{s_1, s_2, s_3, s_4\}$ .
2. It does not make much sense to take the union, intersection etc of these two sets S and C. We may be interested in knowing which student owns which cell phone. For this we have to go deeper into the sets and see the individual elements of both sets so that

we can identify which student  $s_i$  owns which cell phone  $c_j$ . We may represent this information using the notation of *ordered pairs*:  $(s_i, c_j)$  meaning student  $s_i$  owns cell phone  $c_j$ . Let us say this is the situation:

$$(s_1, c_1), (s_1, c_2), (s_2, c_3), (s_3, c_5), (s_4, c_4)$$

These *ordered pairs* may be viewed as elements of a new set  $R$ .

$$R = \{ (s_1, c_1), (s_1, c_2), (s_2, c_3), (s_3, c_5), (s_4, c_4) \}$$

We use the letter  $R$  to denote the notion of a *relation*. The elements of the set  $S$  and the set  $C$  are *related* by the *relation*  $R = \text{"is the owner of"}$ .

3. Now, where did these ordered pairs  $(s_i, c_j)$  come from in the first place? We may consider the set  $S \times C$  known as the *product set*, the set of all ordered pairs.

$$S \times C = \{ (s_i, c_j) \text{ for } i = 1 \text{ to } 4 \text{ and } j = 1 \text{ to } 5 \}$$

From this *product set*  $S \times C$  only some of the elements belong to the *relation*  $R$ . Hence we may say that a *relation*  $R$  is a *subset* of  $S \times C$ , written as  $R \subseteq S \times C$ .

4. It is not necessary that we always *relate* two different sets. We may define a *relation* between the elements of the same set. For example, on  $C = \{c_1, c_2, c_3, c_4, c_5\}$  we may define the *relation*  $R = \text{"same price range"}$ . We may have three price ranges:

$$\begin{aligned} \text{cell phones} < \text{Rs. } 10,000, \quad & \text{Rs. } 10,000 \leq \text{cell phones} \leq \text{Rs. } 20,000 \\ & \text{and Rs. } 20,000 < \text{cell phones.} \end{aligned}$$

Under the *relation*  $R = \text{"same price range"}$  we may group the elements of  $C$  into subsets. Say

$$\{ c_1, c_2, c_3 \}$$

$$\{ c_4 \}$$

$$\{ c_5 \}$$

cell phones < Rs. 10,000

Rs. 10,000 ≤ cell phones ≤ Rs. 20,000

Rs. 20,000 < cell phones

5. If we observe closely we note that these three subsets under the *relation*  $R = \text{"same price range"}$  are:

(i) Non-empty

(ii) Pair-wise disjoint

(iii) The union of all of them =  $C$

Such a collection of subsets of a set we call a **PARTITION**.

6. Also we note that a PARTITION helps in a Search operation. If we were to search for a cell phone of a particular price, we do not have to search the entire set C. We need only to search for a cell phone (based on its price) in just the subset of its price range.
7. Let us now look more deeply at the notion of *relation* and try to identify the underlying mathematical ideas that induced a PARTITION on the set C. Let us just list the *relation*  $R \subseteq C \times C$ .
8. The ordered pairs  $(c_1, c_2)$ ,  $(c_2, c_3)$ ,  $(c_1, c_3)$  are elements in the set R because they are in the “same price range”.

If that is the case then the ordered pairs  $(c_2, c_1)$ ,  $(c_3, c_2)$ ,  $(c_3, c_1)$  must also be in the set R because they are related under  $R = \text{“same price range”}$ . What we are saying is: If  $c_1$  is in the “same price range” of  $c_2$  then  $c_2$  is in the “same price range” as  $c_1$ .

Then the ordered pairs  $(c_1, c_1)$ ,  $(c_2, c_2)$ ,  $(c_3, c_3)$ ,  $(c_4, c_4)$ ,  $(c_5, c_5)$  must also be elements in the set R. Because  $c_i$  is in the “same price range” as  $c_i$  for all  $i$ .

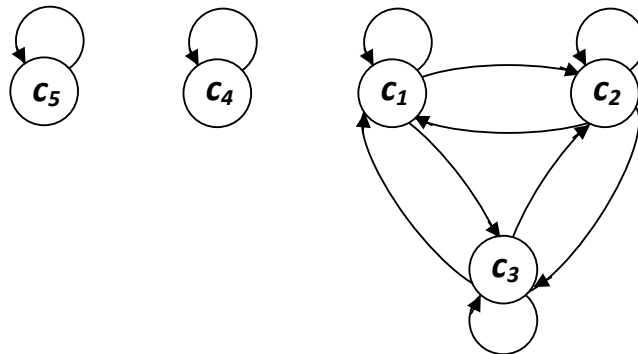
So now we may say

$$R = \{ (c_1, c_2), (c_2, c_1), (c_2, c_3), (c_3, c_2), (c_1, c_3), (c_3, c_1), \\ (c_1, c_1), (c_2, c_2), (c_3, c_3), (c_4, c_4), (c_5, c_5) \}$$

We may also denote R by

$$R = \{ c_1Rc_2, c_2Rc_1, c_2Rc_3, c_3Rc_2, c_1Rc_3, c_3Rc_1, \\ c_1Rc_1, c_2Rc_2, c_3Rc_3, c_4Rc_4, c_5Rc_5 \}$$

We may also depict a relation by digraphs:



Digraph to depict *relation* R

9. Now let us try to identify some special *properties of the relation* R. We note that:

- (i)  $(c_i, c_i)$  for all  $i$
- (ii) if  $(c_i, c_j)$  then  $(c_j, c_i)$  for every  $i, j$
- (iii) if  $\{ (c_i, c_j) \text{ and } (c_j, c_k) \}$  then  $(c_i, c_k)$  for every  $i, j, k$

Hence we can say that if a *relation* has these *three special properties* it will induce a PARTITION on the set. In other words, if the elements of a *set* are *related* in such a way that satisfies these *three special properties* it may make our Search operation (algorithm) easy. We may be able to divide-and-conquer.

10. We have special names for these *three special properties*

- (i) Reflexive
- (ii) Symmetric
- (iii) Transitive

respectively.

We also have a special name for *relations* with these *three special properties*. We call them EQUIVALENCE RELATIONS. In Algebra we differentiate between *equal* or *identical* on the one hand and *equivalence* on the other hand. We are familiar with this kind of distinction with respect to triangles in Geometry: *congruent triangles* on the one hand versus *similar triangles* on the other hand.

So we are not saying that the cell phones in the “same price range” are *equal* or *identical*. We say that they are *equivalent* because they are in the “same price range”.

We know that the *relation* “same price range” (which has these *three special properties*) induces a PARTITION on the set C. Hence we say: EQUIVALENCE RELATION  $\Leftrightarrow$  PARTITION. The subsets in the PARTITION are called *equivalence classes*, *cells*, *blocks*.

11. Before we proceed further let us make these *three special properties* more clear by a few examples.

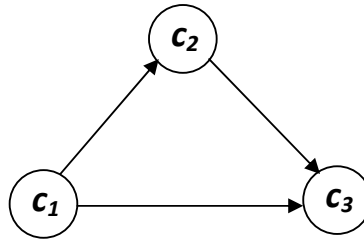
**Example 1:**  $R = \{ (c_1, c_1), (c_2, c_2), (c_3, c_3), (c_4, c_4), (c_5, c_5) \}$  is *reflexive* because it satisfies the *reflexive property*:  $(c_i, c_i)$  for all  $i$ .

**Example 2:**  $R = \{ (c_1, c_1), (c_2, c_2), (c_4, c_4), (c_5, c_5) \}$  is not *reflexive* because  $(c_3, c_3)$  is not present. It violates the *reflexive property*.

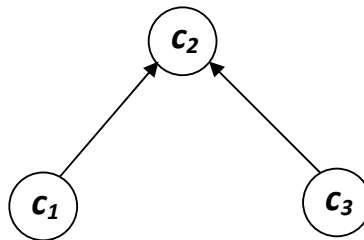
**Example 3:**  $R = \{ (c_1, c_2), (c_2, c_1) \}$  is *symmetric* because it satisfies the *symmetric property*.

**Example 4:**  $R = \{ (c_4, c_4) \}$  is also *symmetric* because it does not violate the *symmetric property*. So by default it satisfies the *symmetric property*. So with this in mind, the null relation (the relation without any ordered pairs) is also *symmetric*.

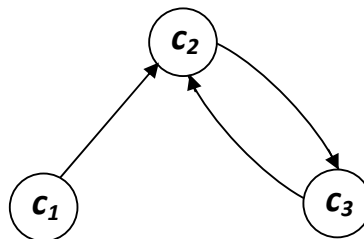
**Example 5:**  $R = \{ (c_1, c_2), (c_2, c_3), (c_1, c_3) \}$  is *transitive* because it satisfies the *transitive property*.



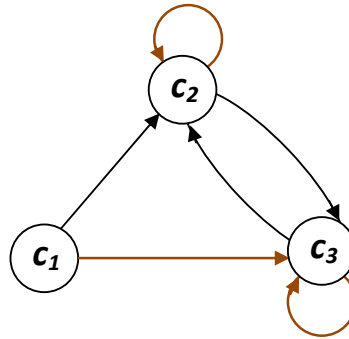
**Example 6:**  $R = \{ (c_1, c_2), (c_3, c_2) \}$  is *transitive* because it does not violate the *transitive property*. So by default it satisfies the *transitive property*.



**Example 7:**  $R = \{ (c_1, c_2), (c_2, c_3), (c_3, c_2) \}$  is not *transitive* because it does not satisfy the *transitive property*.



We may make it *transitive* by adding ordered pairs or edges.



Now we come to a very important point. Let us look at these three properties *reflexive*, *symmetric* and *transitive* and ask what is so special about these three properties? Or better still let us ask what is so special about the following three questions:

- (i) Reflexive question: is  $c_i R c_i$  for all  $i$  ?
- (ii) Symmetric question: if  $c_i R c_j$  does  $c_j R c_i$  ?
- (iii) Transitive question: if  $\{ c_i R c_j \text{ and } c_j R c_k \}$  does  $c_i R c_k$  ?

To understand the significance of the three questions above let us ask:

Why is it we did not ask the question: if  $\{ c_i R c_j \text{ and } c_j R c_k \}$  does  $c_k R c_i$  ?

Why is it we did not ask the question: if  $\{ c_i R c_j \text{ and } c_j R c_k \text{ and } c_k R c_l \}$  does  $c_i R c_l$  ?

With some thinking and observation we can see that the above two questions can be answered by asking the same three questions in different combinations.

So we can say that the *reflexive*, *symmetric* and *transitive* questions are very **basic**. Once we get the answers to these questions (concerning how the elements of a set relate) we can get the answer to any big question.

We also observe that these three basic questions are **independent**. What do we mean by that?

We cannot get the answer to the *reflexive* question by the answers from the *symmetric* and *transitive* questions.

Likewise, we cannot get the answer to the *symmetric* question by the answers from the *reflexive* and *transitive* questions.

Also, we cannot get the answer to the *transitive* question by the answers from the *reflexive* and *symmetric* questions.

**We make one more crucial observation. We observe that:**

- (i) The *reflexive* question deals with *one element at a time*.
- (ii) The *symmetric* question deals with *two elements at a time*.
- (iii) The *transitive* question deals with *three elements at a time*.

**What can we infer from this?**

When asked to find the answer to some big question we need to think and formulate a set of **basic** and **independent** questions that when asked in some combinations will “*extract the information*” (algorithm) and give us the answer to the big question.

**Did you notice a similar pattern in Formal Language Theory?**

Finite State Automats (FSA) can recognize  $a^n$ , one symbol at a time.

Push Down Automats (PDA) can recognize  $a^n b^n$ , two symbols at a time.

Turing Machines (TM) can recognize  $a^n b^n c^n$ , three symbols at a time.




*One may say that because a Turing Machine can recognize relations on a given set of symbols and even form new relations and new sets of symbols it is the most powerful machine we have.*

When it comes to Relational Database design we may need to ask more than three basic questions working with maybe more than three elements at a time (maybe from the same set of data or from different sets of data). This is a thought process that is required in *normalizing* a Relational Database to the required normal form.

## SORT OPERATION

1. Let us now look at the Sort operation from a *relation* and its *properties* point of view.

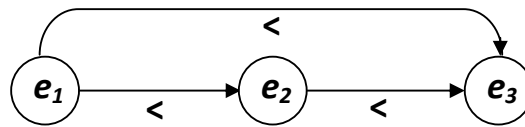
$$S = \{ \text{apple}, \text{book}, \text{bat} \}$$

		
$e_1 = \text{apple}$	$e_2 = \text{book}$	$e_3 = \text{bat}$
\$1	\$10	\$100

We may *relate* the elements of the set S by their price and Sort them.

$$\text{relation } R = e_i R e_j \Leftrightarrow \text{price of } e_i < \text{price of } e_j$$

$$R = \{ (e_1, e_2), (e_2, e_3), (e_1, e_3) \}$$



We can see that price of apple < price of book < price of bat

Now let us look at the properties of R. We can see that R is:

(i) *Not reflexive*: because there is at least one element  $e_i$  in S such that the price of  $e_i$  is not less than the price of  $e_i$ . E.g.: \$1 is not less than \$1. We cannot say *price of apple* < *price of apple*. So the ordered pair  $(e_1, e_1)$  is not in R. In fact  $(e_i, e_i)$  is not in R for all i. Hence we say R is *irreflexive*.

(ii) *Anti-symmetric*: if  $e_i R e_j$  then  $e_j$  “does not R”  $e_i$ , which means if the price of  $e_i$  is less than the price of  $e_j$ , then certainly the price of  $e_j$  cannot be less than the price of  $e_i$ .

(iii) *Transitive*: because it satisfies the *transitive property*:

$$\$1 < \$10 \text{ and } \$10 < \$100 \text{ implies } \$1 < \$100$$

Since we cannot say *price of apple* < *price of apple* our Sort algorithm will hang when it encounters this situation. So this relation is not good enough for us to perform a Sort. Why do we face this difficulty? Because  $(e_i, e_i)$  is not in *relation*  $R = \text{“less than”}$ .

To resolve this difficulty we need to augment the relation R by redefining it as  $R = \text{“less than or equal to”}$  denoted by the symbol  $\leq$ .

Now  $e_i \leq e_i$  or  $e_i R e_i$  or  $(e_i, e_i)$  in R for all i. Thus we see that R is now *reflexive*.



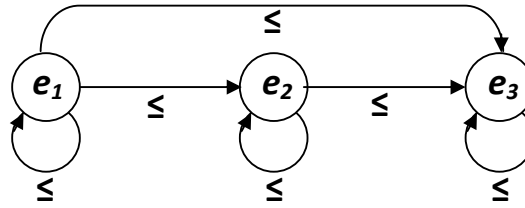
2. Let us now look at the properties of the “less than or equal to” relation

It is:

(i) *Reflexive*

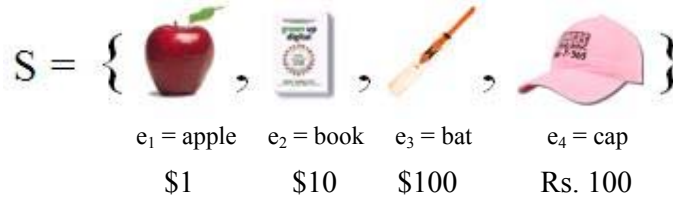
(ii) *Anti-symmetric*

(iii) *Transitive*

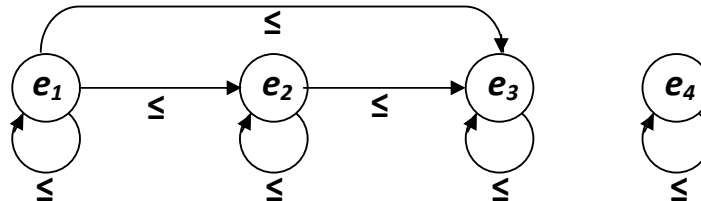


We have a special name for *relations* with these *three special properties*. We call them PARTIAL ORDER. To Sort we need one extra property. We need all pairs of elements to be *comparable*. So a *relation* that is a PARTIAL ORDER with *all pairs comparable* is required to perform a Sort.

3. Let us augment the set S by adding element  $e_4 = \text{cap}$ . The price of cap is Rs. 100



The relations is still a PARTIAL ORDER on S: The digraph is



However, we cannot Sort the set S because we cannot *compare* the price of elements in \$ with the price of elements in Rs. To Sort the set S we need *all pairs comparable*.

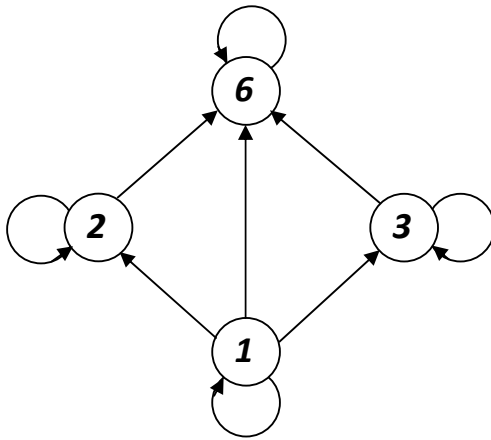
4. Let us now define what we mean by a pair of elements being *comparable*. We say a pair of elements  $e_i$  and  $e_j$  ( $i \neq j$ ) are comparable if  $(e_i, e_j)$  is in R or  $(e_j, e_i)$  is in R but not both.

Number of n-element binary relations of different types								
n	all	Transitive	Reflexive	Preorder	Partial order	Total preorder	Total order	Equivalence relation
0	1	1	1	1	1	1	1	1
1	2	2	1	1	1	1	1	1
2	16	13	4	4	3	3	2	2
3	512	171	64	29	19	13	6	5
4	65536	3994	4096	355	219	75	24	15

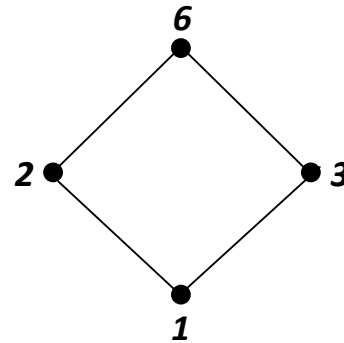
# LATTICES

There is also a mathematical entity called Lattice. A Lattice can be created by a *relation* that is a **PARTIAL ORDER** on a set with the additional condition that every pair of elements must have a *greatest lower bound* and *least upper bound*. Lattices are beautiful structures with applications in the design of Privacy and Security systems, and robust Computer Networks (Ref.: Nigel Day, “**Lattices for Computer Security** – An Introduction and some useful properties”, TXSG-87-8, 30 March 1987. TOPEXPRESS Computer Security Group, Cambridge, United Kingdom). Let us see a simple example:

Let set  $S = \{ 1, 2, 3, 6 \}$  and let  $R$  be a relation on  $S$  such that  $e_i R e_j \Leftrightarrow e_i$  “divides”  $e_j$



Digraph of  $R$



Transitive Reduction of  $R$

$R$  is a partial order on  $S$ . Note that  $(2, 3)$  is not in  $R$ . So 2 cannot be compared with 3. Likewise  $(3, 2)$  is not in  $R$ . So 3 cannot be compared with 2. In other words 2 and 3 are not comparable. So  $R$  is a partial order where all pairs are not comparable.

Every pair of elements in the **Transitive Reduction** of  $R$  has a *greatest lower bound* and a *least upper bound*. So the **Transitive Reduction** of  $R$  is a simple example of a Lattice.

Pairs of elements	(1, 2)	(1, 3)	(1, 6)	(2, 3)	(2, 6)	(3, 6)
<b>Greatest Lower Bound</b>	1	1	1	1	2	3
<b>Least Upper Bound</b>	2	3	6	6	6	6

## CONCLUSION

So we can see that questions concerning the *reflexive*, *symmetric* and *transitive* properties of a *relation*, *comparable*, *greatest lower bound* and *least upper bound* for every pair of elements (in the **Transitive Reduction**) will help us to decide whether we can Search efficiently or Sort or create a Lattice.

Let us now apply this way of thinking to Warshall's algorithm. Let  $R$  be a relation on a set such that  $v_i R v_j \Leftrightarrow$  there is a *direct edge* from vertex  $v_i$  to vertex  $v_j$ . The **Transitive Closure** relation  $R^+$  (also called the *connectivity relation*  $R^\infty$ ) tells us whether there is a *path* or not between every pair  $(v_i, v_j)$ . Warshall's algorithm finds the **Transitive Closure** in  $O(n^3)$  time. Can we improve on Warshall's algorithm? The answer is "NO". Because we have to ask the *transitive question* for every combination of three vertices (more precisely combinations with repetition) and this is  $nC_3$  which is  $O(n^3)$ .

**Let us now try to summarize what we just did:**

Anytime there is some *information* (*operand* stored in a **Data Structure**) and a big question (i.e. the need to *extract some information*), we try to formulate the necessary and sufficient **basic** and **independent** questions that will answer the big question by performing an *operation* (**Algorithm**).

**Algorithm** (*operation*) is a procedure to ask these **basic** and **independent** questions in different combinations as many times as required to *extract the information*.

The real world exists in SPACE and TIME. In a computer (Turing Machine) *information* about the real world can be represented in **Data Structures** (SPACE) and *information is extracted* by **Algorithms** in TIME. Designing efficient **Data Structures** (use of SPACE) and efficient **Algorithms** (use of TIME) is the art in Computer Science and Engineering.

*"A time will come when our descendants will be amazed  
that we had so much understanding of such subtle things"*

- Stephen Vadakkan