Sorting with Custom Comparators

https://usaco.guide/silver/sorting-custom

CP Initiative
joincpi.org

Sorting with Custom Comparators

If we use custom objects or want to sort elements in an order other than the default, then we'll need to define a custom comparator.

By default, integers sort from least to greatest and characters sort in nondecreasing order. To sort the data in a different way, we would need to use a custom comparator

If we want to store multiple pieces of information about something, we can make a class to store the data and then use a custom comparator to sort.

Classes

First, we need to define a class that stores the data that we want

Here is an example of a class in Java:

```java
class Edge {
    public int a, b;
    public int weight, index;
    public Edge(int a, int b, int weight, int index){
        this.a = a;
        this.b = b;
        this.w = w;
        this.index = index;
    }
}
```

CP Initiative
joincpi.org

Comparable

To use a comparable, we need to implement the Comparable class and the `compareTo` method.

```java
class Edge implements Comparable<Edge> {
    public int a, b, weight, index;
    public Edge(int a, int b, int weight, int index) {
        this.a = a;
        this.b = b;
        this.weight = weight;
        this.index = index;
    }
    public int compareTo(Edge compareEdge) {
       // Put stuff here
    }
}
```

compareTo Method

- The `compareTo` method returns an integer.
- A negative integer is returned if the argument is “greater” than the object that called the method.
- A positive integer is returned if the argument is “less” than the object that called the method.
- `0` is returned if the argument is “equal” to the object that called the method.

Comparable Example

The following code will sort the Edges by weight in ascending order.

```java
class Edge implements Comparable<Edge> {
    public int a, b, weight, index;
    public Edge(int a, int b, int weight, int index) {
        this.a = a;
        this.b = b;
        this.weight = weight;
        this.index = index;
    }
    public int compareTo(Edge compareEdge) {
        return this.weight - compareEdge.weight;  //ascending order
        //return compareEdge.weight - this.weight; -> descending order
    }
}
```

CP Initiative
joincpi.org

Comparators

- To use a Comparator, we need to make a `Comparator` class that implements `Comparable` with a compare method.
- When we were using `Comparable`, we were only able to sort our object in one way. Comparators allow us to sort an object in multiple ways.
- Previously, we sorted our `Edges` by weight. If we use comparators, we can still sort the objects by weight, but now we can also sort them by index.
- We can put `Comparator` classes inside or outside the class of object that the Comparator will compare.

compare Method

- The compare method takes in two objects in as parameters.
- A negative integer should be returned if the second argument is “greater” than the first argument.
- A positive integer should be returned if the second argument is “less” than the first argument.
- `0` should be returned if both arguments are equal.

Comparator Example

```java
class sortByIndex implements Comparator<Edge>{
  public int compare(Edge a, Edge b){
     return a.index - b.index;//ascending order
     //return b.index - a.index; -> descending order
  }
}

class sortByWeight implements Comparator<Edge>{
  public int compare(Edge a, Edge b){
     return a.weight - b.weight;//ascending order
     //return b.weight - a.weight; -> descending order
  }
}
```

CP Initiative
joincpi.org

Sorting Syntax

When sorting using `Comparator`, we can do something like this:

```
Edge[] edges = new Edge[N];
Arrays.sort(edges, new sortByWeight());
```

When sorting using `Comparable`, we can do something like this:

```
Edge[] edges = new Edge[N];

Arrays.sort(edges);
```

CP Initiative
joincpi.org

Sorting by Two Criteria

Suppose we wanted to sort our `Edge` class by weight, from least to greatest, breaking ties by index, with smaller indices coming before larger ones.

We can do this similarly to how we did previously, but instead of returning `0` when weights are equal, we can return the difference in the indices.

```
public int compareTo(Edge compareEdge){
    if(this.weight == compareEdge.weight)
        return this.index - compareEdge.index;
    return this.weight - compareEdge.weight;
}
```

CP Initiative
joincpi.org

Example Problem

CSES - Restaurant Customers

Solution Sketch

- Assign starting points with a value 1 and ending points with a value −1.
- Sort the points by time.
- The problems now becomes: “given an array of values, find the maximum at some prefix in the array.
- Solve by looping through the array and storing a running sum.

Challenge Problem

USACO - Splitting the Field

CP Initiative
joincpi.org