Top K Elements: Introduction

Let's go over the Top K Elements pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following

Overview

The **top K elements** pattern helps find some specific k number of elements from the given data with optimum time complexity.

Many problems ask us to find the top, the smallest, or the most/least frequent k elements in an unsorted list of elements. To solve such problems, sorting the list takes $O(n \log(n))$ time, then finding the k elements takes $O(k)$ time. However, the top k elements pattern can allow us to solve the problem using $O(n. \log k)$ time without sorting the list first.

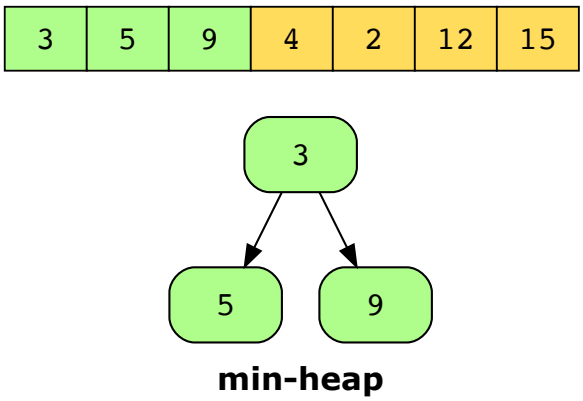
Which data structure can we use to solve such problems? The best data structure to keep track of the smallest or largest k elements is heap. With this pattern, we either use a max-heap or a min-heap to find the smallest or largest k elements, respectively.

For example, let's look at how this pattern takes steps to solve the problem of finding the top k largest elements (using min-heap) or top k smallest elements (using max-heap):

1. Insert the first k elements from the given set of elements to the min-heap or max-heap.
2. Iterate through the rest of the elements.
    I. For min-heap, if you find the larger element, remove the top (smallest number) of the min-heap and insert the new larger element.
    II. For max-heap, if you find the smaller element, remove the top (largest number) of the max-heap and insert the new smaller element.

Iterating the complete list takes $O(n)$ time, and the heap takes $O(\log k)$ time for insertion. However, we get the $O(1)$ access to the k elements using the heap.

Let's look at the following illustration to understand how we can use min-heap to find the top k elements.

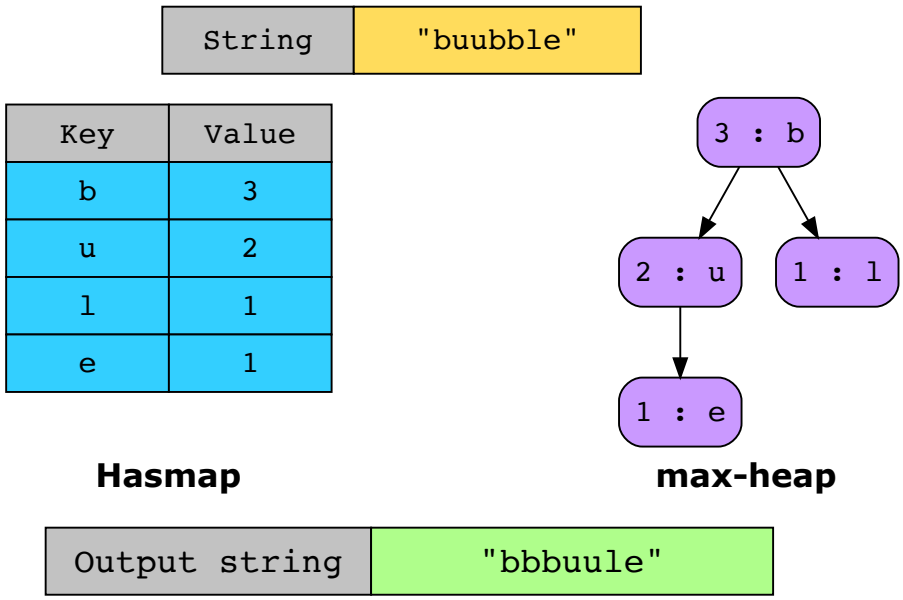


Examples

The following examples illustrate some problems that can be solved with this approach:

Sort characters by frequency

We'll calculate frequencies of each character and store them in the hash map. We'll use the frequencies to maintain the max-heap and sort characters by frequency.



1 of 2

Does my problem match this pattern?

- Yes, if both of these conditions are fulfilled:
  - We need to find the largest, smallest, most frequent, or least frequent subset of elements in an unsorted list.
  - This may be the requirement of the final solution, or it may be necessary as an intermediate step toward the final solution.
- No, if any of these conditions is fulfilled:
  - The input data structure does not support random access.

- The input data is already sorted according to the criteria relevant to solving the problem.
- If only 1 extreme value is required, that is, $k = 1$, as that problem can be solved in $O(n)$ with a simple scan through the input array.

Real-world problems

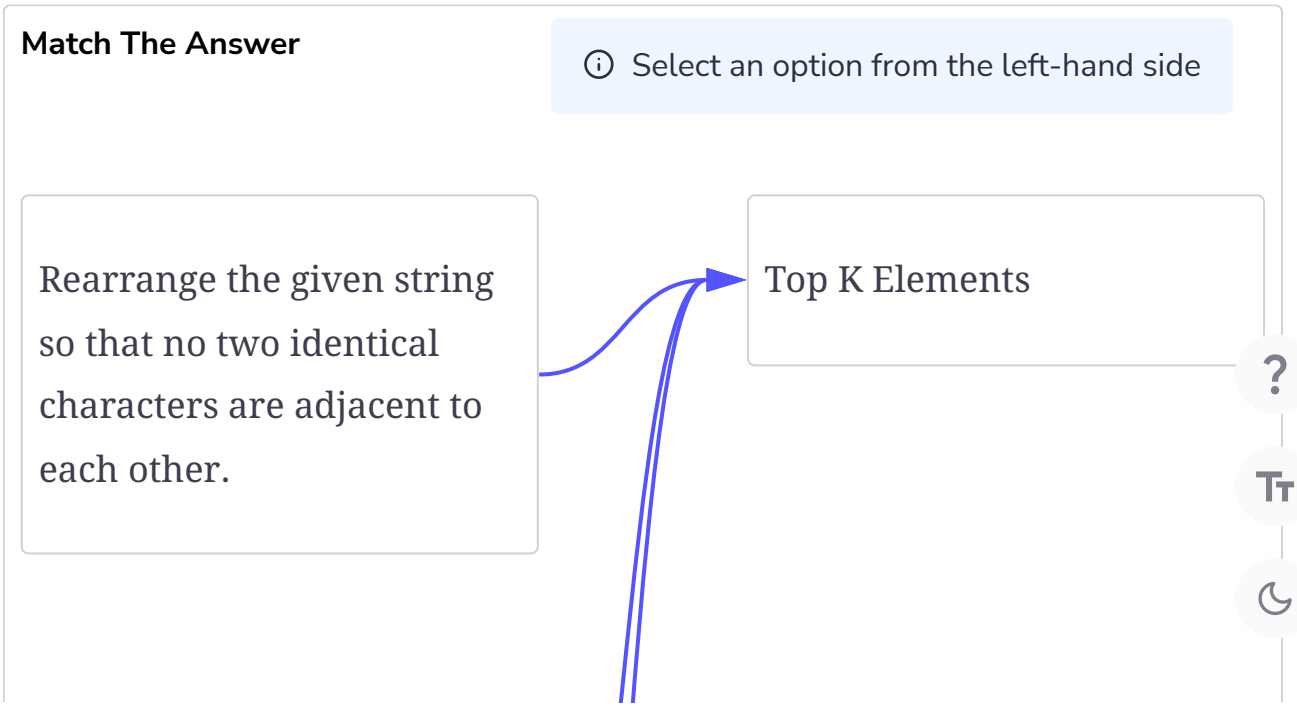
Many problems in the real world use the top K elements pattern. Let's look at some examples.

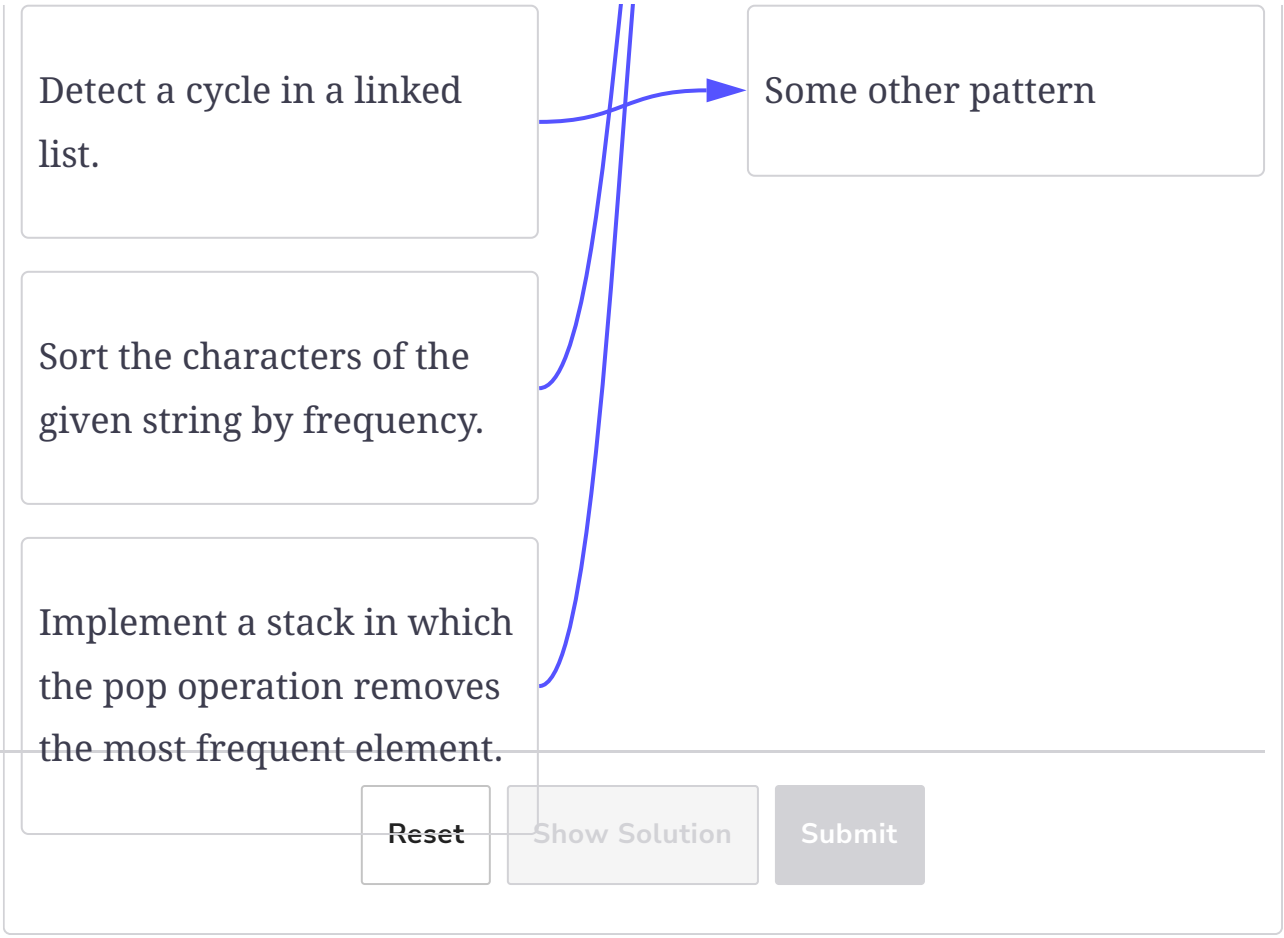
- **Uber:** Select at least the n nearest drivers within the user's vicinity, avoiding the drivers that are too far away.
- **Stocks:** Given the set of IDs of brokers, determine the top K broker's

Strategy time!

Match the problems that can be solved using the top K elements pattern.

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.





Back
Median of Two Sorted...
Next
Kth Largest Element i...
Mark as Completed