Last update: June 8, 2022   **Translated**   **From: e-maxx.ru**

Minimum stack / Minimum queue

In this article we will consider three problems: first we will modify a stack in a way that allows us to find the smallest element of the stack in $O(1)$, then we will do the same thing with a queue, and finally we will use these data structures to find the minimum in all subarrays of a fixed length in an array in $O(n)$

Stack modification

We want to modify the stack data structure in such a way, that it possible to find the smallest element in the stack in $O(1)$ time, while maintaining the same asymptotic behavior for adding and removing elements from the stack. Quick reminder, on a stack we only add and remove elements on one end.

To do this, we will not only store the elements in the stack, but we will store them in pairs: the element itself and the minimum in the stack starting from this element and below.

```
stack<pair<int, int>> st;
```

It is clear that finding the minimum in the whole stack consists only of looking at the value `stack.top().second`.

It is also obvious that adding or removing a new element to the stack can be done in constant time.

Implementation:

- Adding an element:

    ```
    int new_min = st.empty() ? new_elem : min(new_elem, st.top().second);
    st.push({new_elem, new_min});
    ```

- Removing an element:

    ```
    int removed_element = st.top().first;
    st.pop();
    ```

- Finding the minimum:

```
int minimum = st.top().second;
```

Queue modification (method 1)

Now we want to achieve the same operations with a queue, i.e. we want to add elements at the end and remove them from the front.

Here we consider a simple method for modifying a queue. It has a big disadvantage though, because the modified queue will actually not store all elements.

The key idea is to only store the items in the queue that are needed to determine the minimum. Namely we will keep the queue in nondecreasing order (i.e. the smallest value will be stored in the head), and of course not in any arbitrary way, the actual minimum has to be always contained in the queue. This way the smallest element will always be in the head of the queue. Before adding a new element to the queue, it is enough to make a "cut": we will remove all trailing elements of the queue that are larger than the new element, and afterwards add the new element to the queue. This way we don't break the order of the queue, and we will also not loose the current element if it is at any subsequent step the minimum. All the elements that we removed can never be a minimum itself, so this operation is allowed. When we want to extract an element from the head, it actually might not be there (because we removed it previously while adding a smaller element). Therefore when deleting an element from a queue we need to know the value of the element. If the head of the queue has the same value, we can safely remove it, otherwise we do nothing.

Consider the implementations of the above operations:

```
deque<int> q;
```

- Finding the minimum:

```
int minimum = q.front();
```

- Adding an element:

```
while (!q.empty() && q.back() > new_element)
    q.pop_back();
q.push_back(new_element);
```

- Removing an element:

```
if (!q.empty() && q.front() == remove_element)
    q.pop_front();
```

It is clear that on average all these operation only take $O(1)$ time (because every element can only be pushed and popped once).

Queue modification (method 2)

This is a modification of method 1. We want to be able to remove elements without knowing which element we have to remove. We can accomplish that by storing the index for each element in the queue. And we also remember how many elements we already have added and removed.

```
deque<pair<int, int>> q;
int cnt_added = 0;
int cnt_removed = 0;
```

- Finding the minimum:

  ```
  int minimum = q.front().first;
  ```

- Adding an element:

  ```
  while (!q.empty() && q.back().first > new_element)
      q.pop_back();
  q.push_back({new_element, cnt_added});
  cnt_added++;
  ```

- Removing an element:

  ```
  if (!q.empty() && q.front().second == cnt_removed)
      q.pop_front();
  cnt_removed++;
  ```

Queue modification (method 3)

Here we consider another way of modifying a queue to find the minimum in $O(1)$. This way is somewhat more complicated to implement, but this time we actually store all elements. And we also can remove an element from the front without knowing its value.

The idea is to reduce the problem to the problem of stacks, which was already solved by us. So we only need to learn how to simulate a queue using two stacks.

We make two stacks, `s1` and `s2`. Of course these stack will be of the modified form, so that we can find the minimum in $O(1)$. We will add new elements to the stack `s1`, and remove elements from the stack `s2`. If at any time the stack `s2` is empty, we move all elements from `s1` to `s2` (which essentially reverses the order of those elements). Finally finding the minimum in a queue involves just finding the minimum of both stacks.

Thus we perform all operations in $O(1)$ on average (each element will be once added to stack `s1`, once transferred to `s2`, and once popped from `s2`)

Implementation:

```
stack<pair<int, int>> s1, s2;
```

- Finding the minimum:

```
if (s1.empty() || s2.empty())
    minimum = s1.empty() ? s2.top().second : s1.top().second;
else
    minimum = min(s1.top().second, s2.top().second);
```

- Add element:

```
int minimum = s1.empty() ? new_element : min(new_element, s1.top().second);
s1.push({new_element, minimum});
```

- Removing an element:

```
if (s2.empty()) {
    while (!s1.empty()) {
        int element = s1.top().first;
        s1.pop();
        int minimum = s2.empty() ? element : min(element, s2.top().second);
        s2.push({element, minimum});
    }
}
int remove_element = s2.top().first;
s2.pop();
```

Finding the minimum for all subarrays of fixed length

Suppose we are given an array A of length N and a given $M \leq N$. We have to find the minimum of each subarray of length M in this array, i.e. we have to find:

$$\min_{0 \leq i \leq M-1} A[i], \min_{1 \leq i \leq M} A[i], \min_{2 \leq i \leq M+1} A[i], \ \dots \ , \min_{N-M \leq i \leq N-1} A[i]$$

We have to solve this problem in linear time, i.e. $O(n)$.

We can use any of the three modified queues to solve the problem. The solutions should be clear: we add the first M element of the array, find and output its minimum, then add the next element to the queue and remove the first element of the array, find and output its minimum, etc. Since all operations with the queue are performed in constant time on average, the complexity of the whole algorithm will be $O(n)$.

Practice Problems

- [Queries with Fixed Length](#)

- [Binary Land](#)

Contributors:

jakobkogler (93.2%)   adamant-pwn (3.14%)   r7919 (1.57%)   prashant-raghu (1.05%)   saumya66 (0.52%)
tanmay-sinha (0.52%)