Stack ve Queue'nun Minimum Bulmak İcin Kullanimi

Stack'in Modifikasyonu

Stack veri yapısını calışma zamanını etkilemeden minimum değeri sorgulayabilecek bir hale getirmek istiiyoruz.

Bunu yapababilmek için stack'e eklediğimiz her bir elemanın yanı sıra son eklenen elemanın gerisinin minimumunu da tutmamız gerekecek. Daha farklı bir şekilde ifade edecek olursak, stack veri yarısını pair'lerden oluşan bir array olarak hayal edin:

```
stack [i] .second = min {stack [j] .first}
                    j = 0..i
```

Açıkca görülüyor ki, herhangi bir zamanda stack'te bulunan elemanların minimumu stack'in en üstündeki pair'in second değerine eşit.

Implementation:

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

- Adding an element:
- `int minima = st.empty ()? new_element: min (new_element, st.top (). second);`
  `st.push (make_pair (new_element, minima));`
- Removing elements:
- `int result = st.top (). first;`
  `st.pop ();`
- Finding a minimum:
- `minima = st.top (). second;`

Queue'nun Modifikasyonu Yöntem 1

Burada queue'yu işimize yarayacak hale getirmek için basit bir yolu ele alacağız. Bildiğiniz gibi queue'nun başından elemanlar çıkarılacağı için eleman çıkardıktan sonra yeni minimumun ne olduğuna sabit zamanda cevap verebilmeliyiz. Ayrıca bu yöntem altdizilerin minimumunu bulmakta da kullanılabilecek.

Yöntem temelde queue'nun sürekli olarak monoton artan kalmasını sağlamaya dayalı. Dolayısıyla herhangi bir zamanda queue'nun içindeki minimum değer başına eşit olacak. Yeni bir eleman queue'nun sonuna eklerken monotonluk bozulduğu sürece queue'nun son elemanı

atılacak. Baştan eleman silerken ise eğer queue'nun en başındaki eleman silmek istediğimiz sayıya eşit ise queue'nun başındaki elemanı queue'dan atmak yeterli olacak.

Yukarıda anlatılan işlemlerin implementasyonunu ele alalım:

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

- Finding a minimum:

  ```
  current_minimum = q.front ();
  ```

- Adding an element:

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

- Removing elements:

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

Böylece yapılan işlemlerin ortalama zaman karmaşıklığının O(1) olduğu anlaşıldı.

Queue'nun Modifikasyonu Yöntem 2

Burada yukarıdaki yöntemle aynı işi yapan fakat çok daha karmaşık bir implementasyona sahip yeni bir yöntemden bahsedeceğiz. Tabi ki implementasyona gelen bu zahmeti spesifik durumlarda bize fayda sağlayacağı için tercih edeceğiz. Kısaca kendine göre daha komplex problemlerde diğer yönteme nazaran artılara sahip.

Öncelikle problemi stackle kullanacak hale indirgemeliyiz. Bunun için 2 tane stack kullanılacak. Bunlara s1 ve s2 diyelim. Yeni eklenecek elemanlar her zaman s1'e eklenecek ve çıkartılacaklar her zaman s2'den cıkarılacak. Bu sırada s2'den eleman çıkarmak istediğimiz fakat s2'nin boş olduğu durumda s1'deki bütün elemanları ters sırada olacak şekilde s2'ye aktaracağız. Sonuçta bütün elemanların minimumu s1 ve s2'nin minimumu olacak. Bütün bu işlemler sonucunda her elaman en fazla 1 defa s1'e eklenecek ve en fazla 1 defa s1'den s2'ye aktarılacak ve yine en fazla 1 defa s2'den çıkarılacak.

Implementation:

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

Finding a minimum:

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

        current_minimum = min (s1.top (). second, s2.top (). second);
```

Adding an element:

```
int minima = s1.empty ()? new_element: min (new_element, s1.top (). second);
s1.push (make_pair (new_element, minima));
```

Removing elements:

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

t_minimum = min (s1.top (). second, s2.top (). second);
```

Sol ve Sağ İndisleri Kendi İçinde Sıralı Olan Altdizilerin Minimum Sorgularını Cevaplama

Yukarıda bahsedilen 2 yöntem de bize mevcut aralığın başından eleman silme ve sonuna eleman eklemeyi sağlıyor. Tek yapmamız gereken minimumları bulunacak altdizilerin sol indilerine göre sıralanıp işleme alınması ve her adımda queue'muzda veya stack'lerimizde o anki sorgu aralığında bulunan sayıların bulunmasının sağlanması.