پیاده‌سازی پشته با یک صف

مهم‌ترین تفاوت پشته و صف در برگرداندن عنصر است، که در پشته (LIFO) آخرین عنصر و در صف (FIFO) اولین عنصر اضافه شده برگردانده می‌شود. پس برای پیاده‌سازی پشته با استفاده از یک صف ما می‌توانیم عناصر را به ترتیب برعکس در صف نگهداری کنیم. برای اینکار در هر سری که عمل enqueue رخ می‌ده، ما باید با استفاده از یک حلقه تمام عناصر موجود در صف را dequeue و سپس enqueue کنیم. با اینکار ترتیب عناصر به صورت برعکس ذخیره شده و dequeue را می‌توان معادل pop در پشته اضافه کرد.(برگرداندن آخرین عنصر اضافه شده) لازم به ذکر است که در این پیاده‌سازی، پیچیدگی زمانی push به O(n) تبدیل می‌شود و pop از O(1) است و همچنین پیچیدگی حافظه نیز از O(n) می‌باشد.

```
class stack_with_queue{
private:
    queue<int> s;
public:
    void push(int val){
        s.push(val);
        for(int i=0; i<s.size()-1; i++){
            s.push(s.front());
            s.pop();
        }
    }

    void pop(){
        if(s.size() != 0)
            s.pop();
    }

    int top(){
        return (s.size() == 0) ? -1:s.front();
    }
};
```

امیررضا محمدی یگانه

پیاده‌سازی صف با دو پشته

این سؤال برعکس همان سؤال قبل است و همان مشکل ترتیب برگرداندن را داریم. ایده کلی این است که با هر push ما ترتیب عناصر را جوری نگهداری کنیم که با pop کردن اولین عنصر اضافه شده برگردانده شود. برای اینکار می‌توانیم برای هر عمل push ابتدا تمام عناصر را از پشته اول حذف کرده و به پشته دوم اضافه کنیم، سپس عنصر جدید را به پشته دوم اضافه کرده و در آخر تمام عناصر پشته دوم حذف و به پشته اول اضافه می‌کنیم.
پیچیدگی زمانی push از $O(n)$ و pop از $O(1)$ می‌باشد. همچنین پیچیدگی حافظه ای نیز از $O(n)$ است.

```
class stack_with_queue1{
private:
    stack<int> s1;
    stack<int> s2;
public:
    void enqueue(int val){
        for(int i=0; i<s1.size(); i++){
            s2.push(s1.top());
            s1.pop();
        }
        s2.push(val);
        for(int i=0; i<s2.size(); i++){
            s1.push(s2.top());
            s2.pop();
        }
    }
    int dequeue(){
        if(s1.size() == 0)
            return;

        int val = s1.top();
        s1.pop();
        return val;
    }
};
```

امیررضا محمدی یگانه

روش دوم:

ما در روش قبلی عمل push(enqueue) را تغییر دادیم، می‌توانیم عمل pop(dequeue) را نیز به گونه تغییر دهیم که عملیات مانند صف عمل کنند. برای اینکار push مثل قبل می‌ماند و عنصر جدید را به پشته اول اضافه می‌کند ولی با هر بار pop کردن ما چک می‌کنیم که اگر پشته دوم خالی بود، تمام عناصر پشته اول را به پشته دوم اضافه کرده و آخرین عنصرش را برمی‌گردانیم و اگر پشته دوم خالی نبود که ترتیب در پشته دوم درست است و ما می‌توانیم آخرین عنصر پشته دوم را برگردانیم.

این روش از روش قبلی بهتر است زیرا درست است که با تحلیل کلی عملیات pop در بدترین حالت از O(n) است ولی ما همیشه نیاز به انتقال n عنصر به پشته دوم نداریم و اکثر اوقات صرفاً با pop کردن روی پشته دوم با O(1) عنصر را برمی‌گردانیم. پس تحلیل سرشکن آن از O(1) است.

```
};

class queue_with_stacks2{
private:
    stack<int> s1;
    stack<int> s2;
public:
    void enqueue(int val){
        s1.push(val);
    }

    int dequeue(){
        if(s1.size() == 0 && s2.size() == 0)
            return -1;

        if(s2.size() == 0)
            for(int i=0; i<s1.size(); i++){
                s2.push(s1.top());
                s1.pop();
            }
        int val = s2.top();
        s2.pop();
        return val;
    }
};
```