دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی

نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۲۳: ساختمان داده و الگوریتم‌ها

نگارنده: فاطمه علیرضایی

۱۹ آبان ۱۴۰۰

فهرست مطالب

۱ عملیات روی لیست‌های پیوندی

کلیه عملیاتی که در این بخش معرفی می‌شوند بر پایه لیست پیوندی دوطرفه تشریح شده است، با این حال به‌راحتی قابل تبدیل به دیگر نوع‌ها نیز می‌باشد.

- List-Search(L،k): یک پرسمان بازیابی است که برای پیدا کردن اولین عنصری که مقدار کلید آن برابر k است مورد استفاده قرار می‌گیرد. اگر عنصر در لیست موجود باشد، آن عنصر برگردانده می‌شود، در غیر اینصورت مقدار Null خروجی خواهد بود.

Algorithm 1 List-Search(L, k)

```
x = L.head
while x ≠ Null and x.key ≠ k do
    x = x.next
Return x
```

پیچیدگی زمانی الگوریتم فوق در بدترین حالت O(n) است،جایی که n تعداد نودهای لیست پیوندی L است.

- List-Insert(L،x): یک پرسمان بروزرسانی است که عنصر x را به ابتدای لیست L اضافه می‌کند.

Algorithm 2 List-Insert(L, x)

```
1: x.next = L.head
2: if (L.head ≠ Null) then
3:     L.head.prev = x
4: L.head = x
5: x.prev = Null
```

پیچیدگی زمانی الگوریتم فوق O(۱) است.

- List-Delete(L،x): یک پرسمان بروزرسانی است که عنصر x را از لیست L اضافه می‌کند.

Algorithm 3 List-Delete(L, x)

```
1: if (x.prev ≠Null) then
2:     x.prev.next = x.next
3: else
4:     L.head = x.next
5: if (x.next ≠Null) then
6:     x.next.prev = x.next
```

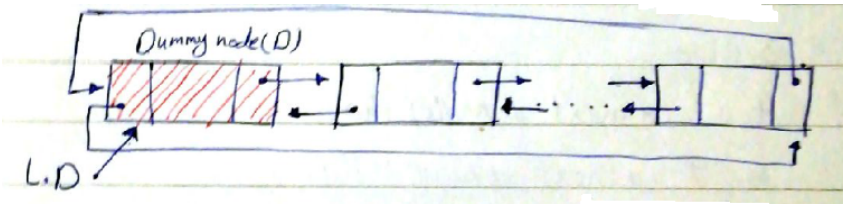
پیچیدگی زمانی الگوریتم فوق O(۱) است. نوشتن خطوط ۷ تا ۹ اجباری نیست و از لحاظ منطقی در خط ۲ بررسی شده است.

نکته: دقت کنید در حذف، چون عنصر x مدنظر بود پیچیدگی زمانی $O(1)$ شد. با این حال اگر مقدار کلید k مدنظر بود، می‌بایست اول آن را پیدا کنیم و سپس تغییرات را اعمال کنیم که در نهایت پیچیدگی زمانی آن $O(n)$ است. مطابق زیر:

Algorithm 4 List-Delete(L, k)

```
1: x = List-Search(L, k)
2: List-Delete(L, x)
```

اطلاعات تکمیلی: اگر یک نود تصنعی (Dummy) در ابتدای لیست پیوندی دو طرفه اضافه کنیم و در عین حال آن را به صورت دایره‌ای نیز در نظر بگیریم، شبه کد مربوط به عملیات حذف ساده‌تر می‌شود.



شبه کد مربوط به بررسی تهی بودن لیست با مشخصات بالا:

$$L.D.next = L.D, \quad L.D.Prev = L.D$$

نکته: همیشه ایجاد نود ساختگی ($Dummy$)ممکن است سودمند نباشد و مثال بالا سبب ساده‌تر شدن الگوریتممان می‌شود. مثلا وقتی تعداد لیست‌های کوچک زیادی داریم، در نظر گرفتن نود ساختگی سبب اتلاف حافظه می‌شود.

- حذف نود x از لیست L:

Algorithm 5 DList-Delete(L, x)

1: $x.prev.next = x.next$
2: $x.next.prev = x.prev$

- جستجوی نود با کلید k در لیست L:

Algorithm 6 DList-Search(L, k)

1: $x = L.D.next$
2: **while** $x \neq L.D$ and $x.key \neq k$ **do**
3: $\quad x = x.next$
4: Return x

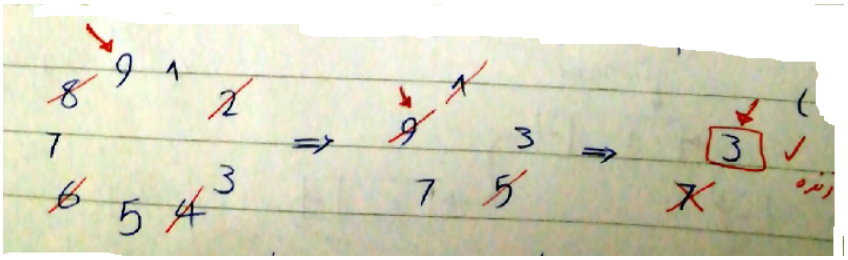
- درج نود x در ابتدای لیست L:

Algorithm 7 DList-Insert(L, x)

1: $x.next = L.D.next$
2: $L.D.next.prev = x$
3: $L.D.next = x$
4: $x.prev = L.D$

۲ کاربرد لیست‌ها - مساله جوزفوس

فرض کنید n نفر به صورت دایره‌وار ایستاده و منتظر اعدام هستند. مساله جوزفوس k (برای اعداد صحیح k ۱<) به این طریق اعمال می‌کند که از نظر اول $k-1$ نفر رد کرده و نفر k را اعدام می‌کند و همین ترتیب تا زنده ماندن تنها 1 نفر کار را ادامه می‌دهد. شکل زیر را به عنوان یک مثال برای این مساله در نظر بگیرید.



mod n

برای مثال بالا، در نهایت نفری که در جایگاه ۳ قرار دارد زنده می‌ماند. به طور کلی، برای k و n فرمول بازگشتی زیر وجود دارد:

$$f(x,k) = (((f(n-1),k) + k - 1) mod n), \quad f(1,k) = 1$$

به طور ساده‌تر، برای $k = 2$ رابطه بازگشتی زیر ارائه شده است:

$$\begin{cases} f(2n) = 2f(n) + 1 & n\ is\ even. \\ f(2+n+1) = 2f(n) - 1 & n\ is\ odd. \\ f(1) = 1 & Base \end{cases}$$

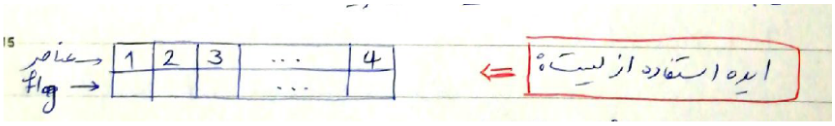
یک راه‌حل ساده برای $k = 2$:

- نوشتن رشته باینری معادل،
- شیفت دورانی به سمت چپ.

حل مثال بالا با روش مذکور به صورت زیر است:

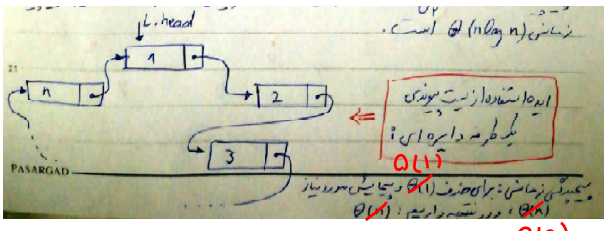
- **گام ۱:** $(n = 9)_{10} = (1001)_2$
- **گام ۲:** $(0011)_2 = (3)_{10}$

حل با استفاده از داده ساختارهای لیست و لیست پیوندی یک طرفه:



- در شروع مقدار flag برای تمام عناصر صفر است.
- هر عنصری که حذف می‌شود، $flag$ آن به ۱ تغییر می‌کند.

پیچیدگی زمانی: $k = \log_2 n$ بار می‌بایست لیست را پیمایش کنیم $O(n)$، در نتیجه پیچیدگی زمانی $\theta(n \log_2 n)$است.



پیچیدگی زمانی : برای حذف $\theta(1)$ و پیمایش مورد نیاز $\theta(n)$، و در نتیجه داریم: $\theta(n)$.

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی

نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۲۴ ساختمان داده و الگوریتم ها

نگارنده: فرزانه مولایی

۷ آذر ۱۴۰۰

فهرست مطالب

۱ شهود پیاده سازی لیست پیوندی با استفاده از آرایه

- یادآوری : لیست پیوندی یک ترتیب خطی منطبق با یک سری اشاره گر است. زمانی که یک آرایه تعریف می کنیم منظور یک ترتیب خطی منطبق با یک سری اندیس می باشد.

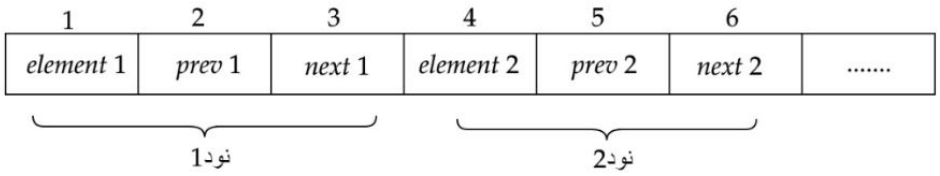
شکل ۱: قالب لیست پیوندی دو طرفه

در ادامه سعی داریم پیاده سازی لیست پیوندی را با یک آرایه و چند آرایه شرح دهیم.

۱.۱ پیاده‌سازی با یک آرایه

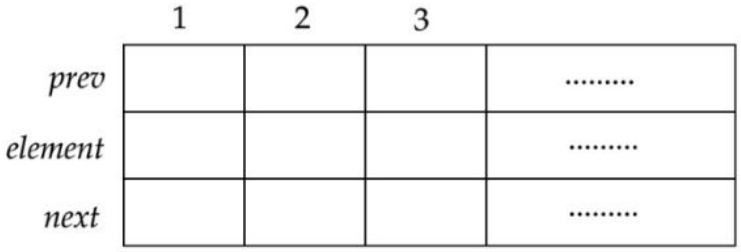
۱- نمایش null با مقدار صفر است.

۲-$next_i$ و $perv_i$ حاوی اندیس های مناسبی از آرایه است.



شکل ۲: پیاده‌سازی لیست پیوندی دوطرفه با یک آرایه .

۲.۱ پیاده سازی با چند آرایه



شکل ۳: پیاده‌سازی لیست پیوندی دوطرفه با چند آرایه

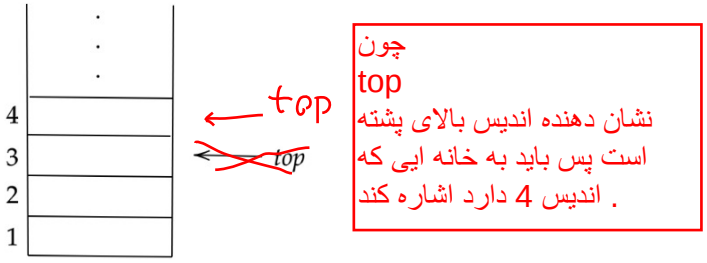
چالش‌ها: برای درج (نیازمند پیدا کردن خانه خالی برای درج)/ حذف (نیازمند بروزرسانی خانه‌ها برای خالی بودن) / جستجو (نیازمند اشاره به نود اول) را بررسی کنید.

۲ داده ساختار پشته(Stack)

یک داده ساختار برای نگهداری مجموعه های پویاست که عناصر آن دریک ترتیب خطی قرارگرفته اند و برای حذف عناصر در آن از سیاست"آخرین ورودی، اولین خروجی"استفاده می شود. اشاره گر (top) نشان دهنده اندیس بالای پشته می باشد.

نکته

اگر top=۰ باشد یعنی پشته خالی است.



شکل ۴: نمایش گرافیکی پشته

۱.۲ عملیات روی پشته

۱. **Stack-Push(S،x)**: یک پرسمان بروزرسانی است که عنصر x را به بالای پشته S اضافه می کند .

$$Stack\_Push(S, x)$$
$$1.S.top = S.top + 1$$
$$2.S[S.top] = x$$

پچیدگی زمانی الگوریتم فوق O(۱) است .

۲. **Stack-Pop(S)**: یک پرسمان بروزرسانی است که عنصر بالای پشته را حذف و برمیگرداند . لازم به ذکر است که اگر پشته تهی باشد پیام "underflow" برگردانده می شود .

```
1: if (S.top == 0) then
2:     return "underflow"
3: else
4:     return S.top = S.top - 1
5:     return S[S.top + 1]
```

پیچیدگی زمانی الگوریتم فوق O(۱) است .

۳. **Stack-Empty(S)**: یک پرسمان بازیابی است که تعیین می کند که آیا پشته خالی است یا نه .

```
1: if (S.top == 0) then
2:     return True
3: else
4:     return False
```

۴. پیچیدگی الگوریتم فوق O(۱) است .

۵. **Stack-Top(S)**: یک پرسمان بازیابی است که عنصر بالایی پشته را بدون تغییر در پشته بر میگرداند . لازم به ذکر است که اگر پشته تهی باشد پیام underflow" " را برمی گرداند .

```
1: if (S.top == 0) then
2:     return "underflow"
3: else
4:     return S[S.top]
```

پیچیدگی زمانی الگوریتم فوق O(۱) است .

دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی

نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۲۵ ساختمان داده و الگوریتم ها

نگارنده: زهرا رشیدی

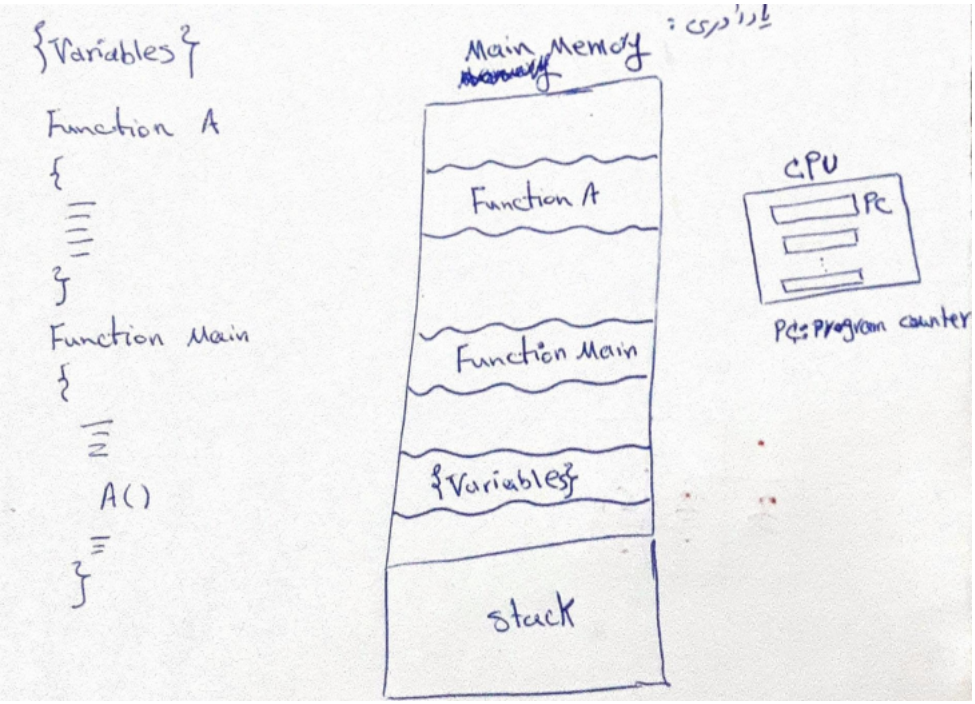
۷ آذر ۱۴۰۰

فهرست مطالب

۱ کاربردهای پشته

۱.۱ فراخوانی تابع

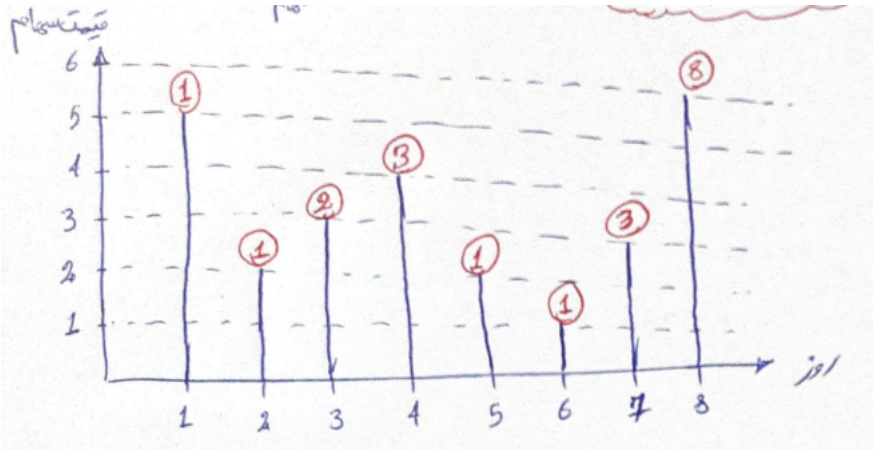
یکی از مهم ترین کاربردهای پشته است که در بخش های قبلی (الگوریتم های بازگشتی) تشریح کردیم.



شکل ۱: نمای کلی از برنامه، حافظه اصلی و واحد پردازش مرکزی

۲.۱ محاسبه دوره سهام روز i –ام

یک لیست از (روز و قیمت) سهام داده شده است. برای نمونه شکل زیر یک نمونه از این لیست را نشان می‌دهد.

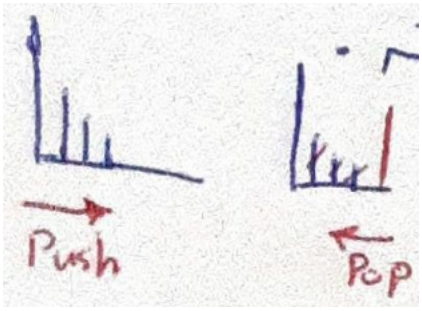


شکل ۲: لیست روز–قیمت سهام

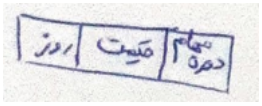
تعریف دوره سهام روز i –ام : تعداد روزهای متوالی قبل از روز i –ام که قیمت سهام آنها در بازه کوچکتر یا مساوی روز i –ام هستند. لازم به ذکر است که خود روز i –ام را هم به این تعداد اضافه میکنیم. دوره سهام هر یک از روزها در شکل بالا مشخص شده است.

سوال: یک لیست به طول n از (روز و قیمت سهام)داده شده است. دوره سهام هر یک از روزهای لیست را محاسبه کنید.

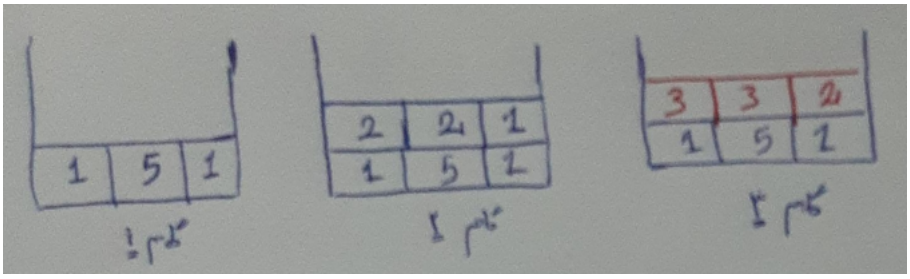
- **راه حل ۱:** از روز مد نظر به عقب برمیگردیم و شرایط دوره سهام را چک میکنیم.در این راه حل، اگر قیمت ها به صورت صعودی باشد بدترین حالت اتفاق می افتد و پیچیدگی این راه حل $\mathcal{O}(n^2)$ است.
- **راه حل ۲:** استفاده از پشته و نگه داشتن عناصر به ترتیب نزولی در پشته.



شکل ۳: زمان اجرای عمل push و pop



شکل ۴: قالب هر عنصر در پشته



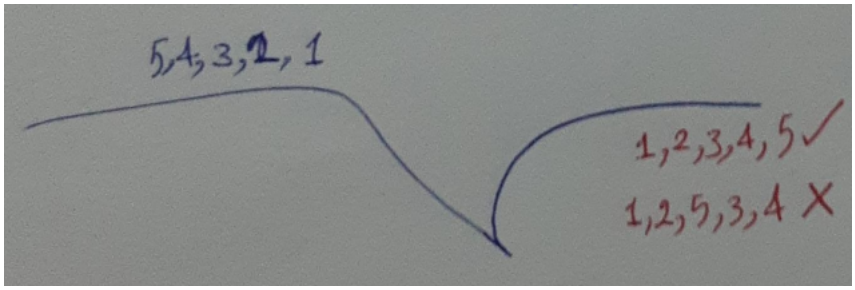
شکل ۵: سه گام از اجرای راه‌حل ۲

گام ۳: روز ۲ با دوره سهام ۱ از پشت خارج میشود ودوره سهام این روز به طور نهایی تعیین میشود و تاثیری هم در روزهای اتی ندارد. با این حال ۱ واحد برای روز سوم نیز لحاظ میشود (به خاطر pop).

پیچیدگی زمانی راه حل دوم: از انجایی که هر عنصر تنها یکبار در پشته اضافه میشود و یکبار از پشته حذف میشود و دوره سهام ان روز تعیین میشود، هزینه کلی $\mathcal{O}(n)$ است.

۳.۱ مساله پارکینگ قطارها

یک لیست از ترتیب ورود قطارها به پارکینگ داریم هر قطاری که وارد پارکینگ میشود امکان برگشت به ورودی پارکینگ را ندارد. از بین کسانی که در پارکینگ قرار دارند، کسی که زودتر وارد پارکینگ شده، امکان خروج دارد.



شکل ۶: یک نمونه از خروج مجاز/غیر مجاز

صورت مساله پارکینگ قطارها: با توجه به قواعد و لیست وارد شده برای ترتیب ورود قطارها، ایا لیست تعیین شده برای خروجی قطار ها از پارکینگ معتبر است؟

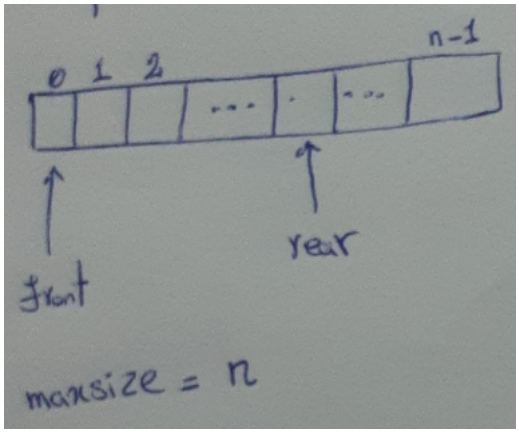
نکته: داده‌ساختار پشته را می‌توان با استفاده از لیست ساده و یا لیست‌پیوندی پیاده‌سازی کرد.

۲ داده ساختار صف (Queue)

یک داده ساختار برای نگهداری مجموعه‌های پویاست که عناصر ان در یک ترتیب خطی قرار گرفته اند و برای حذف عناصر در ان از سیاست "اولین ورودی، اولین خروجی" (FIFO=First In First Out) استفاده می‌شود. در ادامه از دو نمایش گرافیکی برای داده ساختار صف معرفی می‌کنیم:

۱.۲ صف ساده

شکل زیر نمایش گرافیکی این ساختار پیشنهادی را نشان می‌دهد:



شکل ۷: نمایش گرافیکی صف ساده

نکات زیر برای این نحو نمایش قابل بیان است:

- front به ابتدای صف اشاره می‌کند.
- rear به جایی اشاره می‌کند که داده جدید باید درج شود (انتهای صف).
- اگر front=rear باشد به این معناست که صف Q خالی است .

- اگر rear=maxsize باشد به این معناست که صف Q پر است.

عملیات قابل تعریف روی صف ساده به قرار زیر است:

- **Queue-EnQueue(Q،x):** یک پرسمان بروزرسانی است که عنصر x را به انتهای صف اضافه میکند. لازم به ذکر است اگر صف پر باشد، پیام بر"overflow" گرداننده میشود.

```
1: if (Q.rear == Q.maxsize) then
2:     return "overflow"
3: else
4:     Q[Q.rear]=x
5:     Q.rear=Q.rear+1
```

- **Queue-DeQueue(Q):** یک پرسمان بروز رسانی است که عنصر ابتدایی صف را حذف و برمیگرداند، لازم به ذکر است که اگر صف خالی باشد، پیام برگرداننده"underflow" می‌شود.

```
1: if (front == rear) then
2:     return "underflow"
3: else
4:     x=Q[Q.front]
5:     Q.front = Q.front+1
6:     Return x
```

چون در قسمت * گفته است"برمیگرداند" پس باید دستور return x در انتها اضافه کنیم.

مشکل صف ساده: پس از انجام تعدادی عمل حذف و درج در صف، با اینکه آرایه فضای آزاد دارد، امکان درج عنصر جدید را نخوا هیم داشت.

۲.۲ صف حلقوی

این نوع صف در جلسه بعد تشریح می‌شود.