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** $(L.head \neq$ Null$)$ **then**
3: $\quad L.head.prev = x$
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** $(x.prev \neq$Null$)$ **then**
2: $\quad x.prev.next = x.next$
3: **else**
4: $\quad L.head = x.next$
5: **if** $(x.next \neq$Null$)$ **then**
6: $\quad x.next.prev = x.next$

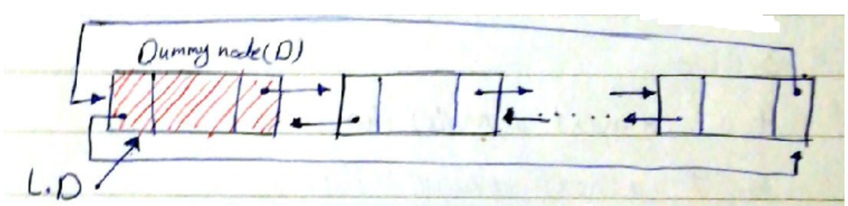
پیچیدگی زمانی الگوریتم فوق O(۱) است. نوشتن خطوط ۷ تا ۹ اجباری نیست و از لحاظ منطقی در خط ۲ بررسی شده است.

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)

$x.prev.next = x.next$
$x.next.prev = x.prev$

- جستجوی نود با کلید k در لیست L:

Algorithm 6 DList-Search(L, k)

$x = L.D.next$
while $x \neq L.D$ and $x.key \neq k$ **do**
    $x = x.next$
Return x

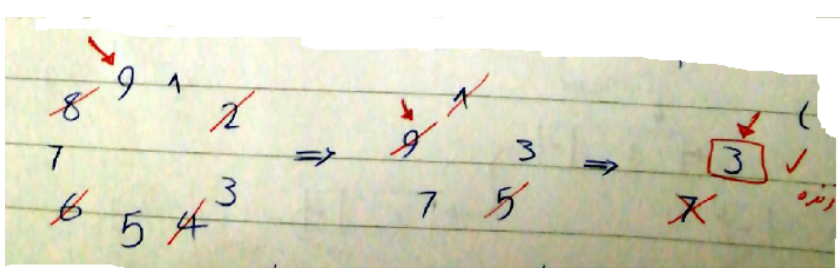
- درج نود x در ابتدای لیست L:

Algorithm 7 DList-Insert(L, x)

$x.next = L.D.next$
$L.D.next.prev = x$
$L.D.next = x$
$x.prev = L.D$

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.



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:

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

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

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

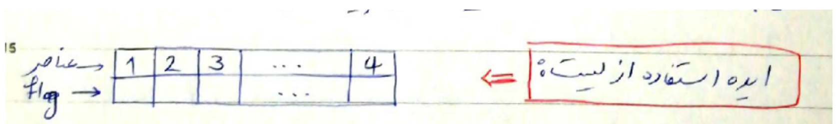
یک راه‌حل ساده برای $k=2$:

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

حل مثال بالا با روش مذکور به صورت زیر است:

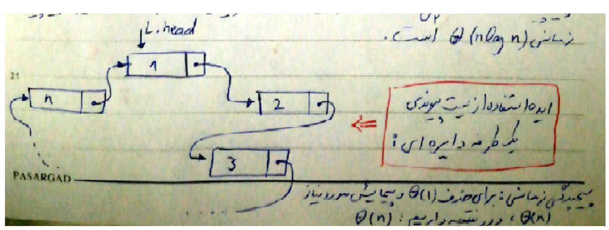
- **گام ۱:** $(n=9)_{10} = (1001)_2$
- **گام ۲:** $(0011)_2 = (3)_{10}$

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: $\theta(n)$.