



پنج: لیست پیوندی

ساختمان داده ها و الگوریتم

مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان

ذخیره سازی در حافظه اصلی

به طور کلی برای ذخیره سازی انواع داده ها در حافظه اصلی (RAM) از دو نوع ساختار می توان استفاده کرد:

۱. آرایه ها

۲. لیست های پیوندی



آرایه

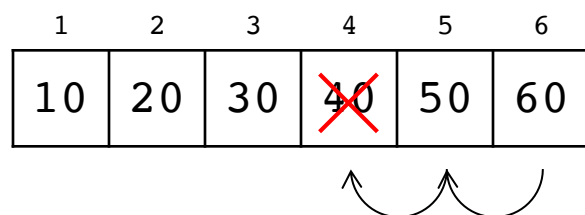
هر آرایه لیست پشت سرهم از داده‌ها است که همگی داده‌ها از یک نوع بوده و ناحیه‌ای پیوسته از حافظه را در اختیار دارند. به طور کلی آرایه‌ها مشکلاتی داشتند:

۱. تعداد عناصر هر آرایه همیشه **محدود و مشخص و ایستا** است و در تمام طول برنامه ثابت می‌باشد.
۲. عملیات حذف و درج یک داده دلخواه در خانه‌ای از آرایه n عضوی نیاز به **شیفت** دارد.

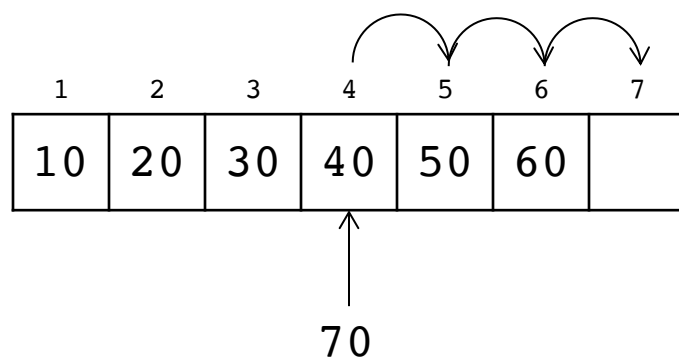
دقت کنید که عملیات همراه با شیفت عناصر در شرایطی که n بزرگ باشد همواره هزینه بالایی دارد.

زمان اجرا	متوسط تعداد شیفت	تعداد شیفت مورد نیاز	عملیات
$O(n)$	$\frac{n + 1}{2}$	$n - k + 1$	درج داده در خانه k ام
$O(n)$	$\frac{n - 1}{2}$	$n - k$	حذف داده از خانه k ام

مثال



حذف 40 در محل $k=4$ ام از آرایه $n=6$ عضو نیاز به $n-k=6-4=2$ شیفت دارد.



درج 70 در محل $k=4$ ام از آرایه $n=6$ عضو نیاز به $n-k+1=6-4+1=3$ شیفت دارد.

مزایا آرایه

۱. عملیات جستجوی یک داده دلخواه در آرایه‌ها با یکی از دو روش زیر انجام می‌گیرد:

الف) جستجوی خطی (ترتیبی) در آرایه‌های مرتب یا نامرتب با زمان $O(n)$.

ب) جستجوی دودویی (باینری) در آرایه‌های **مرتب** با زمان $O(\log n)$.

مزیت در اینجا امکان استفاده از جستجوی دودویی با زمان کمتر و سرعت بیشتر می‌باشد.

۲. دسترسی به داده‌های هر آرایه به صورت تصادفی و دلخواه به راحتی توسط اندیس آرایه انجام شده و دسترسی مستقیم و با زمان $O(1)$ خواهد بود.

لیست پیوندی (Linked List)

لیست پیوندی، لیستی دارای عناصر (رکوردها) مختلفی از حافظه پویا (Heap) بوده که لزوماً عناصر آن کنار هم قرار نگرفته‌اند.



```
class Node:
    def initialize(self, data):
        self.item = data
        self.next = None
```

عملیات‌ها:

- پیمایش لیست
- جستجو گره
- درج گره
- حذف گره

پیمایش لیست پیوندی

پیمایش گره‌های یک لیست پیوندی از ابتدای لیست انجام شده و به دو صورت بازگشتی و غیربازگشتی می‌تواند انجام گیرد.

```
if (start!=none) :  
    p=start  
    while (p!=none) :  
        check (p.data)  
        p=p.next
```

```
def traverse (p) :  
    if (p!=none) :  
        check (p.data)  
        traverse (p.data)
```

پیمایش لیست پیوندی با n گره از مرتبه اجرایی $O(n)$ است.

جستجو در لیست پیوندی

جستجو در هر لیست پیوندی خطی فقط و فقط بصورت ترتیبی امکان پذیر بوده و از ابتدا لیست آغاز می شود. مرتبه اجرایی جستجوی کلید دلخواه k در لیست پیوندی با n گره که اشاره گر $start$ به ابتدای آن اشاره می کند برابر $O(n)$ است.

```
def search(start,k):  
    if(start!=None):  
        p=start  
        while(p!=None and p.data!=k):  
            p=p.next  
        return p
```


درج در لیست پیوندی

برای درج یک گره جدید در موقعیتی مشخص از یک لیست پیوندی، باید ابتدا با پیمایش لیست به موقعیت قبل از محل درج رسیده و سپس گره مورد نظر را درج کرد.

می‌خواهیم گره جدید با اشاره گر new را بعد از گره معلوم با اشاره گر pos از لیستی که اشاره گر start به ابتدای آن اشاره می‌کند، درج کنیم، عملیات لازم عبارتند از:

i. آماده‌سازی گره جدید $O(1)$

ii. پیمایش لیست $O(n)$

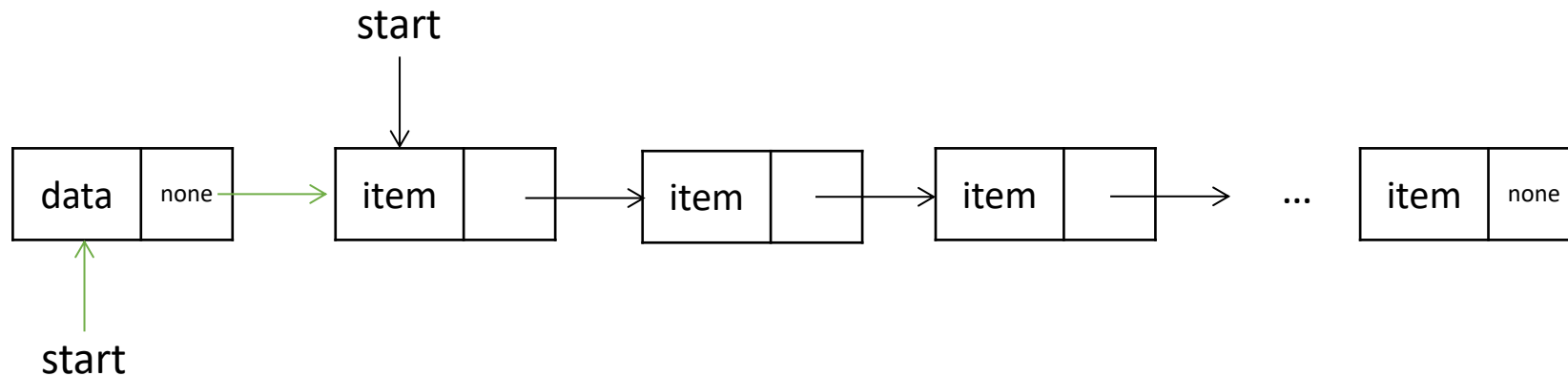
iii. درج گره $O(1)$

بنابراین درج در یک لیست پیوندی از زمان $O(n)$ اجرا می‌شود.

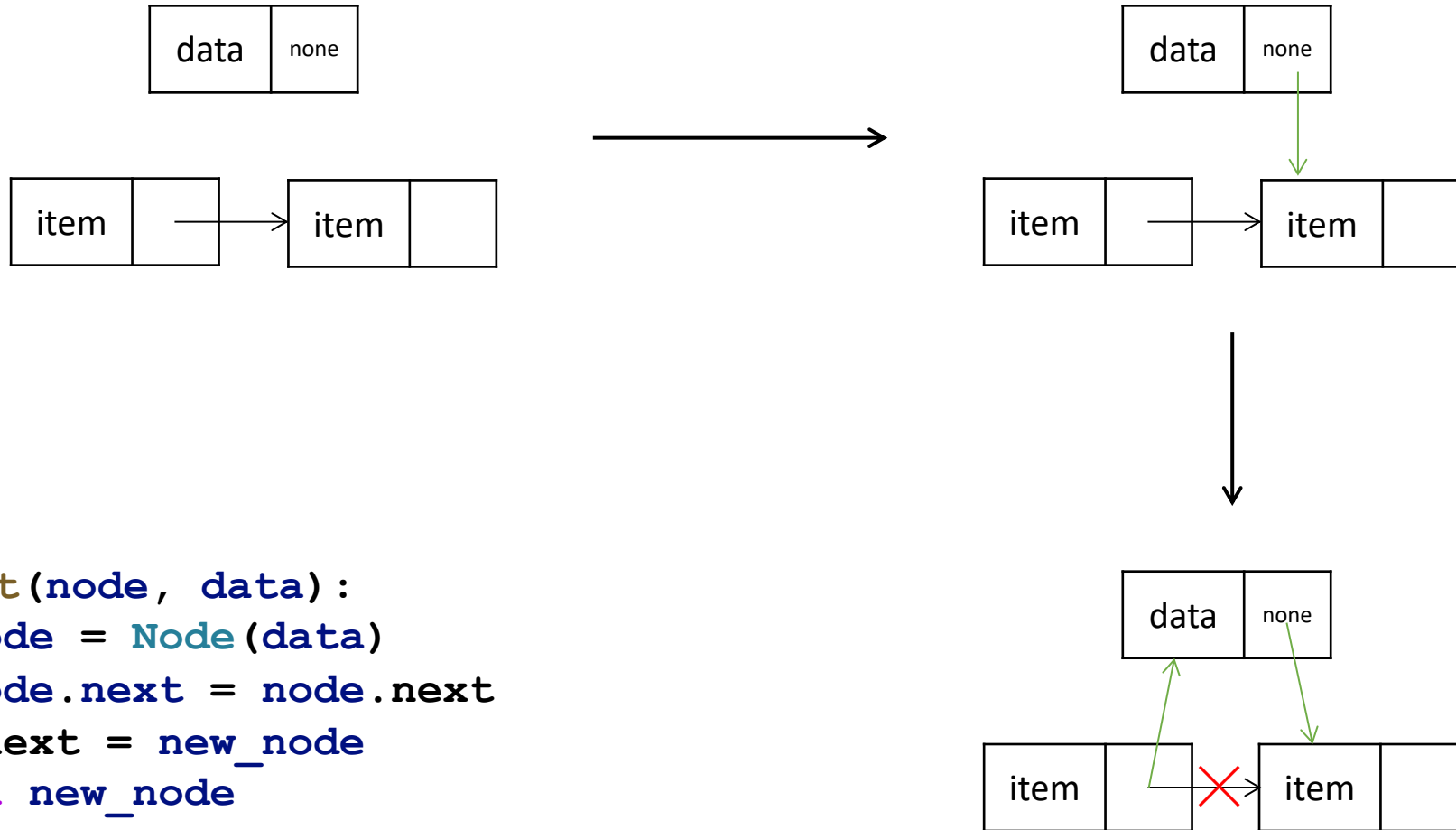
گره جدید

برای آماده سازی یک گره جدید با اشاره گر new داریم:

```
def make_node(start, data):  
    if(heap.available==None):  
        return 'overflow'  
    new = heap.available  
    new.data=data  
    new.next=start  
    start=new  
    return new
```



درج گره

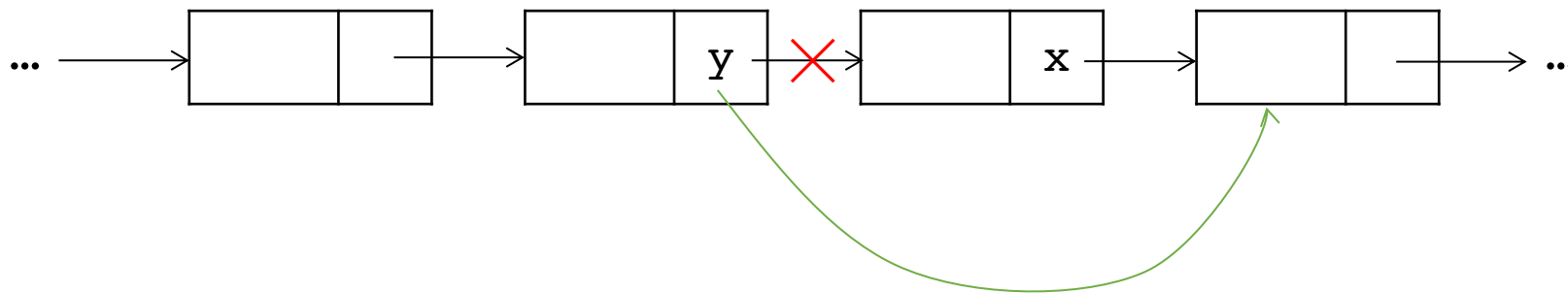


```
def insert(node, data):  
    new_node = Node(data)  
    new_node.next = node.next  
    node.next = new_node  
    return new_node
```

حذف در لیست پیوندی

برای حذف گره‌ای دلخواه از موقعیتی مشخص از یک لیست پیوندی، باید ابتدا با پیمایش لیست به موقعیت قبل از محل حذف رسیده، سپس گره مورد نظر را حذف کرد.

می‌خواهیم گره‌ای دلخواه با اشاره‌گر x را که اشاره‌گر y از گره قبل به آن اشاره می‌کند حذف کنیم.



```
def deleteNode(pos):  
    pos.next = pos.next.next
```

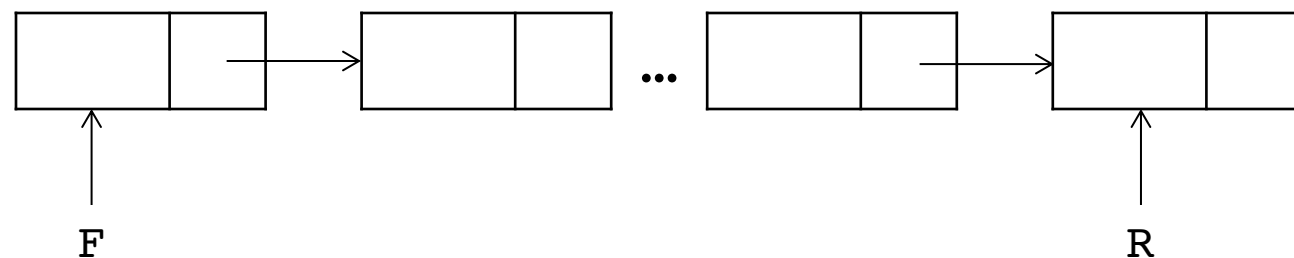
حذف در لیست پیوندی

- پیمایش لیستی با n گره برای رسیدن به موقعیت قبل از حذف زمانی معادل $O(n)$ صرف می کند.
 - برای حذف هر گره نیاز به ۱ عمل جایگزینی است.
- بنابراین در مجموع حذف در لیست پیوندی از مرتبه $O(n)$ خواهد بود.

```
def delete(ls,pos):  
    node=ls.start  
    #find the key to delete  
    repeat pos - 1 times:  
        node = node.next  
  
    next = node.next.next  
    node.next = next  
    return node
```

مثال

یک لیست خطی یک طرفه با دو اشاره گر F و R که به ترتیب به عنصر اول و آخر لیست اشاره می کنند پیاده سازی شده است. هزینه کدامیک از اعمال زیر وابسته به تعداد عناصر لیست است؟



(۱) حذف اولین عنصر

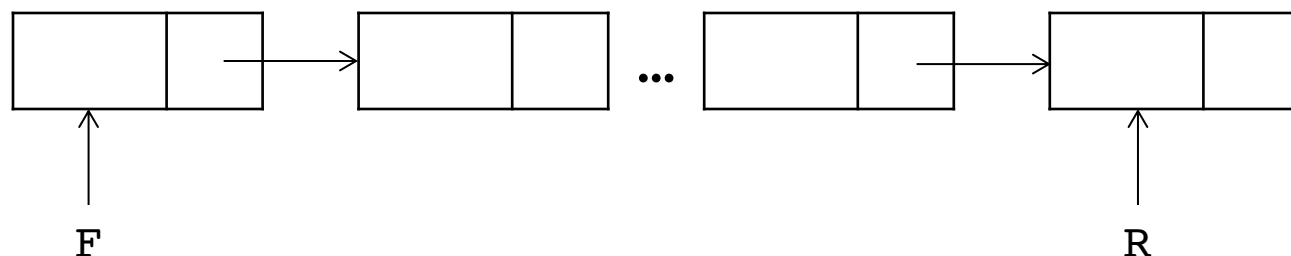
(۲) حذف آخرین عنصر

(۳) درج یک عنصر در انتهای لیست

(۴) درج یک عنصر در ابتدای لیست

مثال

یک لیست خطی یک طرفه با دو اشاره گر F و R که به ترتیب به عنصر اول و آخر لیست اشاره می کنند پیاده سازی شده است. هزینه کدامیک از اعمال زیر وابسته به تعداد عناصر لیست است؟



(۱) حذف اولین عنصر

(۲) حذف آخرین عنصر

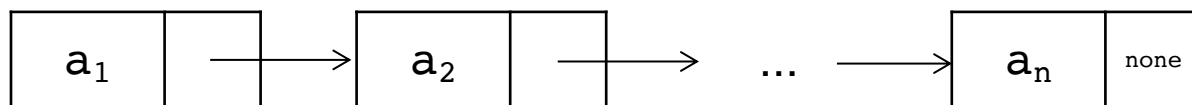
(۳) درج یک عنصر در انتهای لیست

(۴) درج یک عنصر در ابتدای لیست

برای درج و حذف گره از ابتدا (با استفاده از اشاره گر F) و درج گره در انتها (با استفاده از اشاره گر R) نیاز به پیمایش و وابسته به تعداد گره های لیست نیست. اما برای حذف گره آخر ابتدا باید به کمک پیمایش در زمان $O(n)$ به گره ما قبل آخر رسیده تا بتوان عمل حذف را انجام داد.

انواع لیست پیوندی (لیست های خطی)

- لیست پیوندی خطی یک طرفه: در هر گره فقط یک فیلد اشاره گر وجود دارد که آن هم آدرس گره بعدی را دارد.

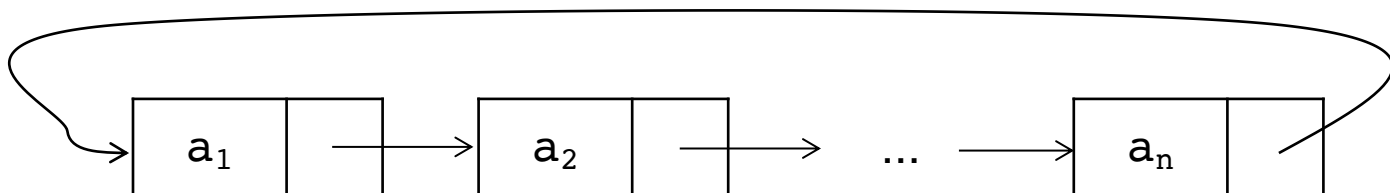


- لیست پیوندی خطی دو طرفه: در هر گره دو فیلد اشاره گر وجود دارد که یکی به گره بعدی و دیگری به گره قبلی اشاره می کند.

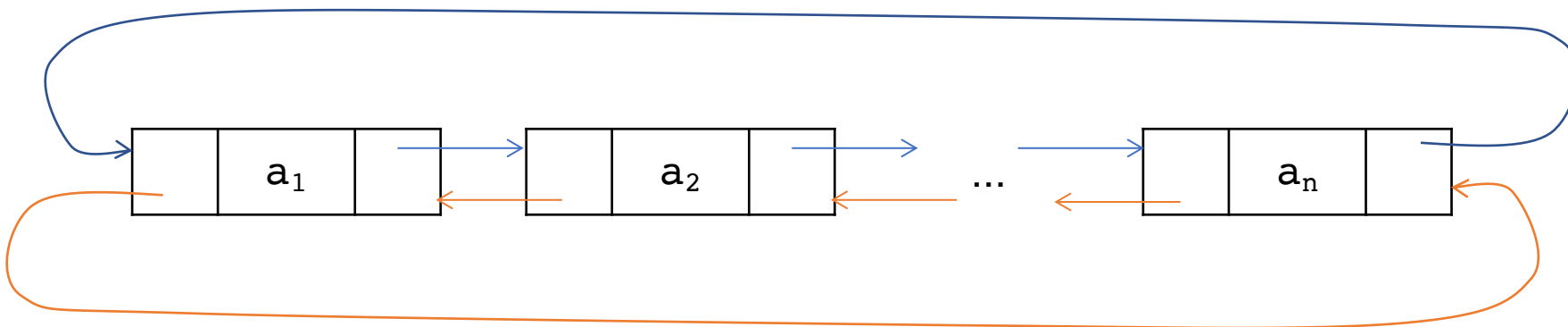


انواع لیست پیوندی (لیست های دوری)

- لیست پیوندی خطی یک طرفه: اشاره گر در گره آخر آدرس گره اول را داشته و به آن اشاره می کند.



- لیست پیوندی خطی دو طرفه: اشاره گر سمت راست در گره آخر به گره اول و اشاره گر سمت چپ در گره اول به گره آخر اشاره می کند.



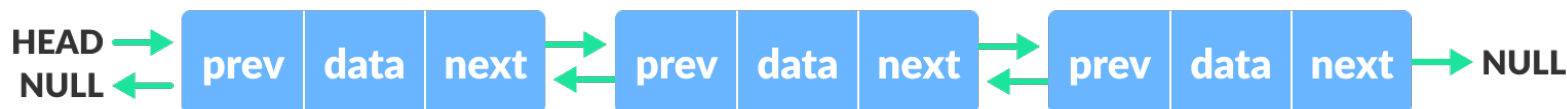
مزیت لیست حلقوی بر لیست خطی

مزیت لیست حلقوی (دوری) بر لیست خطی یک طرفه این است که در لیست حلقوی بدون نیاز به هیچ حافظه اضافی با داشتن آدرس یک گره دلخواه امکان دسترسی به گره قبلی وجود دارد (با پیمایش $n-1$ گره در لیست با n گره از مرتبه $O(n)$ ولی در لیست خطی یک طرفه این امکان وجود نداشته و دسترسی به گره‌های قبل از یک گره فقط از طریق آدرس شروع لیست وجود دارد).



```
struct node {  
    int data;  
    struct node *next;  
};
```

```
struct node {  
    int data;  
    struct node *next;  
    struct node *prev;  
};
```



پیمایش لیست حلقوی

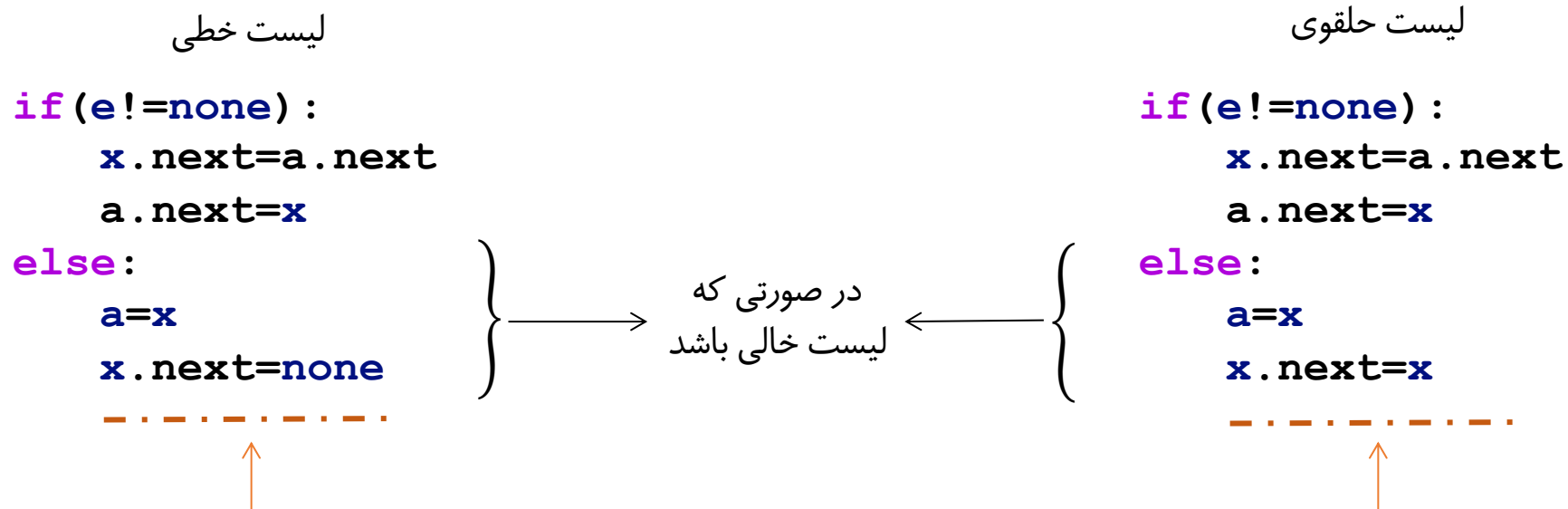
پیمایش یک لیست حلقوی در صورتی که start اشاره‌گری به ابتدای لیست حلقوی باشد به شرح زیر خواهد بود:

```
if (start != none) :  
    p = start  
    repeat:  
        check (p.data)  
        p = p.next  
    until p == start
```

عملیات‌های لیست حلقوی

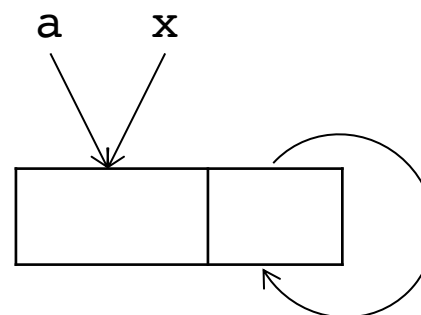
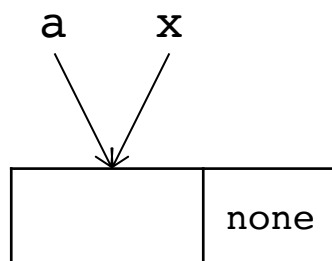
عملیات‌های لیست‌های حلقوی مانند عملیات در لیست‌های خطی است با این تفاوت که باید شرط پایان حلقه‌ها و نحوه اصلاح اشاره‌گر گره آخر باتوجه به دوری بودن لیست تغییر یابد.

فرض کنید e اشاره‌گر به انتهای یک لیست حلقوی و یک لیست غیرحلقوی باشد و بخواهیم گره x را بعد از آن درج کنیم:



عملیات‌های لیست حلقوی

دیده می‌شود برای درج در حالتی که لیست تهی نیست دو الگوریتم شبیه هم عمل می‌کنند اما زمانی که لیست تهی باشد دو وضعیت متفاوت به صورت زیر بعد از درج گره به وجود می‌آید:



اشاره‌گر HEAD در لیست دوری

در صورتی که اشاره‌گر HEAD در لیست‌های دوری به گره اول یا آخر اشاره کند وضعیت‌های متفاوتی از نقطه نظر زمان انجام بعضی از عملیات‌ها بوجود می‌آید که به شرح زیر آن‌ها را بررسی میکنیم.

i. اشاره‌گر HEAD به ابتدای لیست اشاره کند:

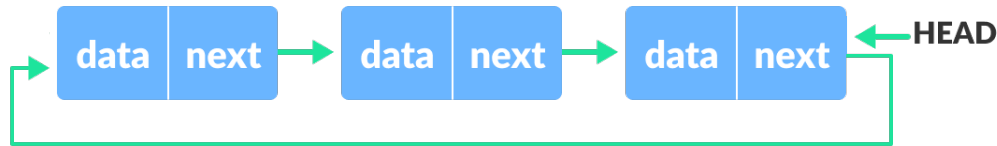


عملیات	مرتبه اجرایی	توضیحات
درج در ابتدا	$O(n)$	باید به انتهای لیست رفته تا گره جدید را در ابتدای لیست درج کنیم.
حذف از ابتدا	$O(n)$	باید به انتهای لیست رفته تا گره ای را از ابتدای لیست حذف کنیم.
درج در انتها	$O(n)$	باید به انتهای لیست رفته تا گره جدید را در انتهای لیست درج کنیم.
حذف از انتها	$O(n)$	باید به گره ماقبل آخر رفته تا گره آخر را حذف کنیم.

اشاره گر HEAD در لیست دوری

در صورتی که اشاره گر HEAD در لیست های دوری به گره اول یا آخر اشاره کند وضعیت های متفاوتی از نقطه نظر زمان انجام بعضی از عملیات ها بوجود می آید که به شرح زیر آن ها را بررسی میکنیم.

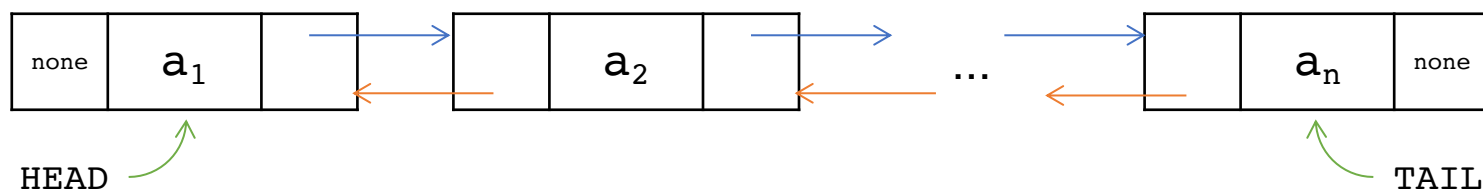
ii. اشاره گر HEAD به انتهای لیست اشاره کند:



عملیات	مرتبه اجرایی	توضیحات
درج در ابتدا	$O(1)$	درج در انتها همان درج در ابتدا خواهد بود.
حذف از ابتدا	$O(1)$	اشاره گر HEAD در انتها به راحتی گره را از ابتدا حذف می کند.
درج در انتها	$O(1)$	درج در انتها به سادگی انجام خواهد شد.
حذف از انتها	$O(n)$	باید به گره ماقبل آخر رفته تا گره آخر را حذف کنیم.

لیست پیوندی دو طرفه (Double linked list)

در لیست‌های پیوندی دو طرفه هر گره به کمک دو اشاره‌گر left (آدرس گره قبلی) و right (آدرس گره بعدی) می‌تواند به گره قبلی و بعدی اشاره کند؛ در نتیجه با داشتن آدرس هر گره به راحتی می‌توان به گره قبلی یا بعدی دسترسی پیدا کرد.



درج در لیست‌های دوپیوندی

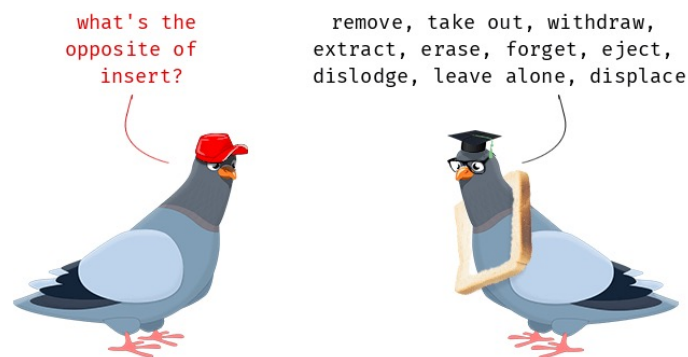
با داشتن آدرس یک گره دلخواه مانند P در یک لیست دو پیوندی می‌توان گره جدید با اشاره گر NEW را در سمت چپ یا راست آن درج کرد؛ برای اینکار به ۴ عمل جایگزینی نیاز داریم:

(۱) درج گره با اشاره گر NEW سمت راست P

(۲) درج گره با اشاره گر NEW سمت چپ P

(۳) درج گره با اشاره گر P در ابتدای لیست

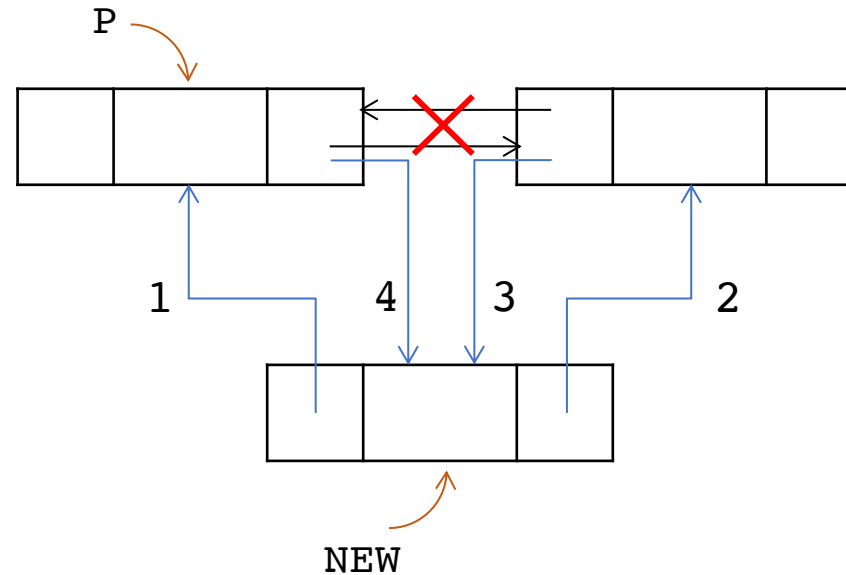
(۴) درج گره با اشاره گر P در انتهای لیست



یک چیز بی مورد، لطفا توجه نکنید!

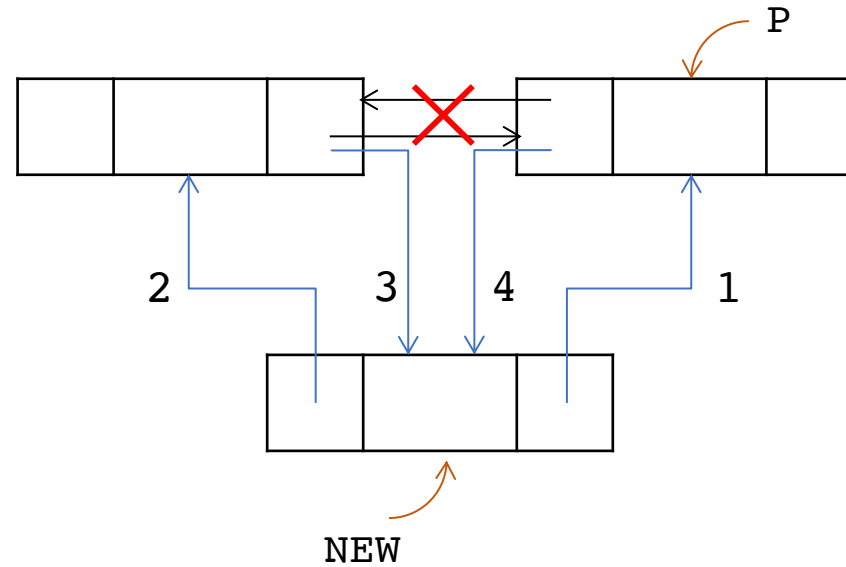
درج گره با اشاره گر NEW سمت راست P

1. `new.left=p`
2. `new.right=p.right`
3. `(p.right).left=new`
4. `p.right=new`

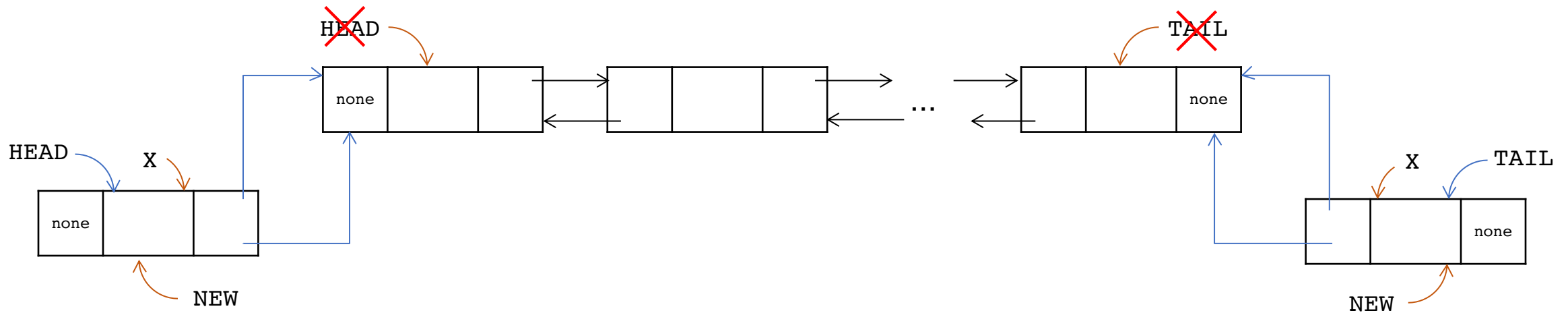


درج گره با اشاره گر NEW سمت چپ P

1. `new.right=p`
2. `new.left=p.left`
3. `(p.left).right=new`
4. `p.left=new`



درج گره با اشاره گر x در ابتدا یا انتها لیست



درج در ابتدا

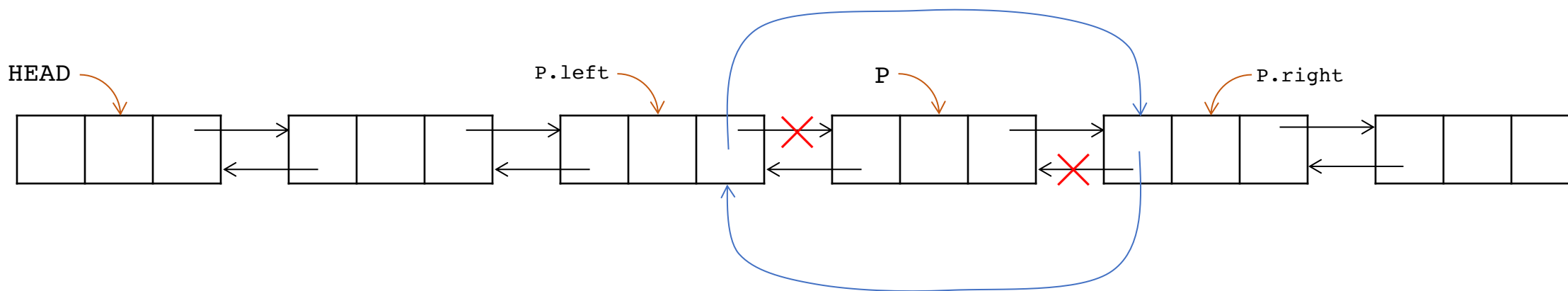
```
1. x.right=head
2. if (head!=none) :
    head.left=x
3. head=x
4. x.left=none
```

درج در انتها

```
1. x.left=tail
2. if (tail!=none) :
    tail.right=x
3. tail=x
4. x.right=none
```

حذف در لیست های دویپوندی

برای حذف گره ای دلخواه از یک لیست دویپوندی (دو طرفه) در صورتی که اشاره گر P به گره مورد نظر اشاره کند با جایگزینی ۲ آدرس بصورت زیر می توان گره مورد نظر را حذف کرد:



1. `(p.left).right=p.right`
2. `(p.right).left=p.left`

لیست‌های پیوندی

مزایا:

۱. تخصیص پویا و متناسب برای داده‌ها برخلاف آرایه‌ها که تخصیص ایستا و محدود دارند.
۲. عملیات درج و حذف بدون نیاز به شیفت با $O(1)$ انجام می‌شود برخلاف آرایه‌ها که نیاز به شیفت دارند با $O(n)$.

معایب:

۱. اتلاف حافظه نسبت به آرایه‌ها به علت استفاده از فیلد اشاره‌گر بیشتر است.
۲. تنها روش جستجو جستجوی خطی است و با زمان $O(n)$ ؛ برخلاف آرایه‌ها که جستجو دودویی را نیز دارند.
۳. دسترسی به هر داده ترتیبی و از ابتدای لیست است با زمان $O(n)$ ؛ برخلاف آرایه‌ها که امکان دسترسی تصادفی با زمان $O(1)$ را دارند.

معکوس کردن لیست پیوندی

برای معکوس کردن یک لیست پیوندی با هر تعداد گره، تنها به ۳ اشاره گر نیاز داریم:

```
p=head  
q=None  
while (p!=None) :  
    1. r=q  
    2. q=p  
    3. p=p.next  
    4. q.next=r
```

مرتبه اجرایی:

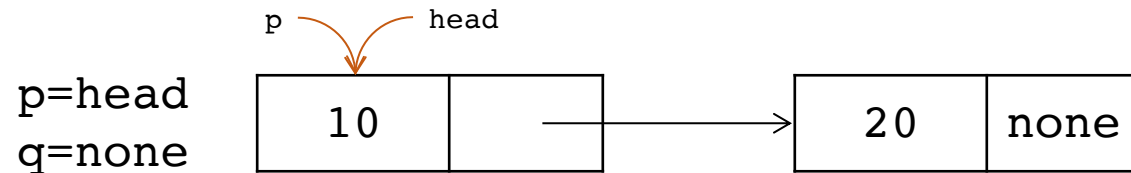
الگوریتم فوق، لیست پیوندی را یک بار پیمایش می کند، بنابراین زمان لازم برای اجرای الگوریتم فوق برای معکوس کردن یک لیست پیوندی با n گره، برابر با $O(n)$ خواهد بود.

شروع الگوریتم

p، به گره اول لیست اشاره می کند

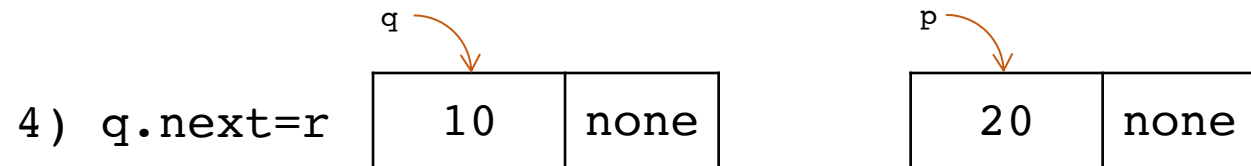
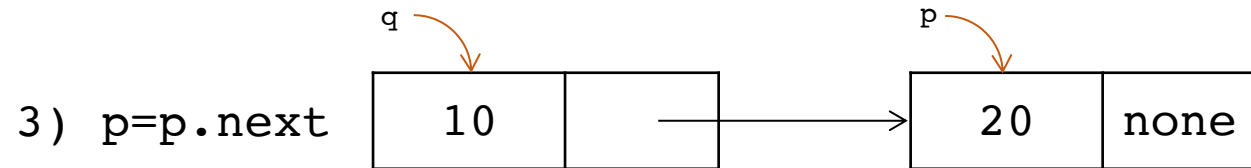
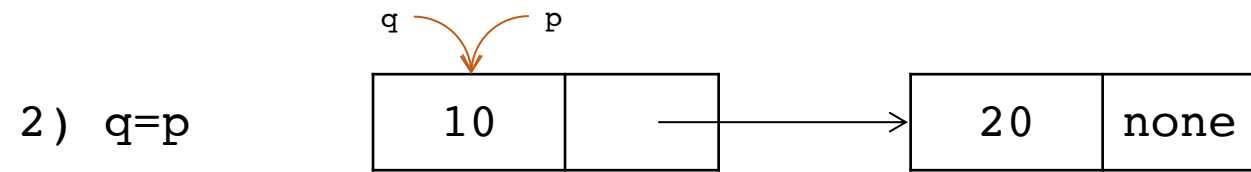
q، none می شود

r، none می شود

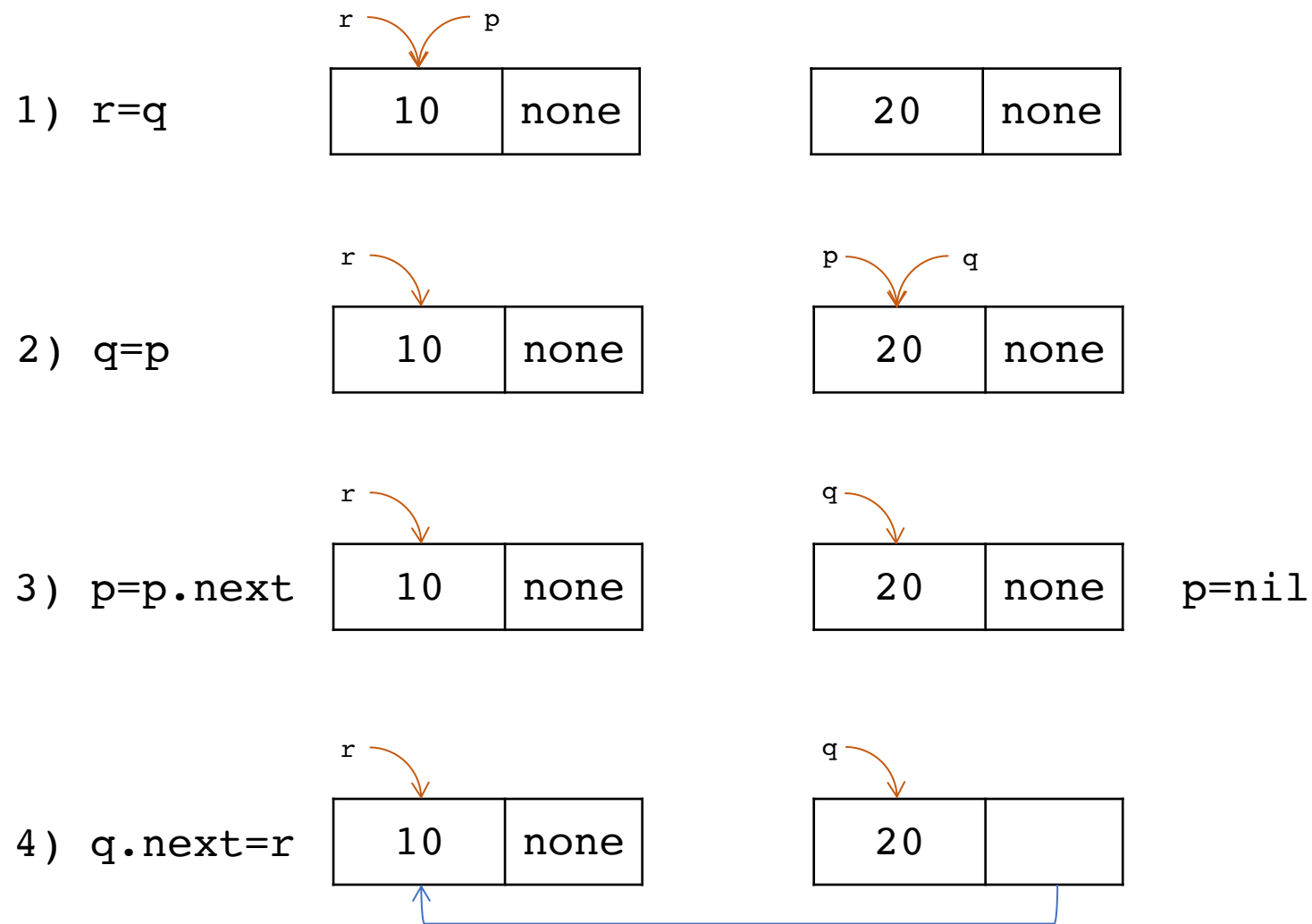


تکرار اول حلقه

1) $r=q$ $r=nil$



تکرار دوم حلقه

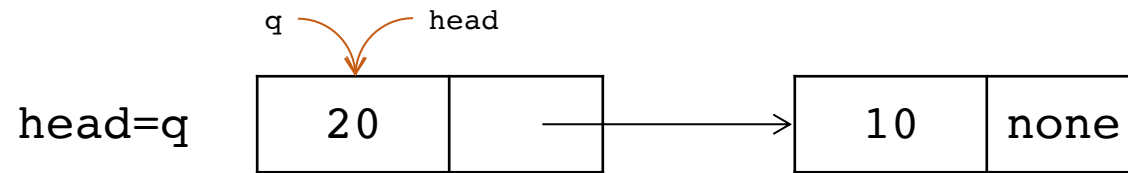


پایان الگوریتم

p، none می شود

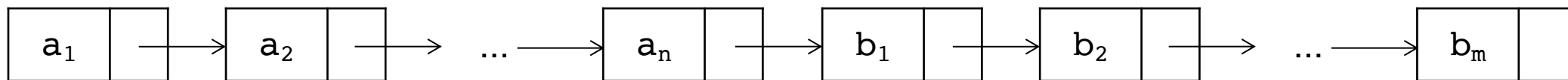
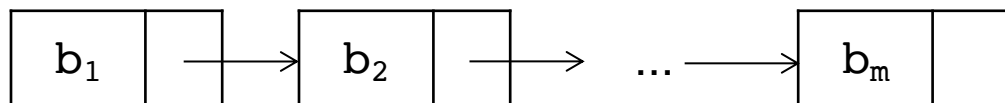
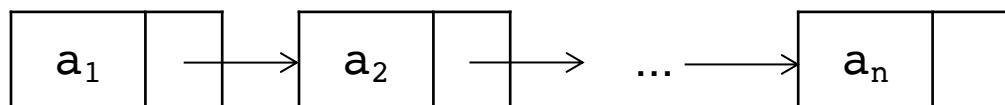
q، به ابتدای لیست معکوس شده اشاره می کند

r، به گره بعد از q اشاره می کند



اتصال دو لیست پیوندی

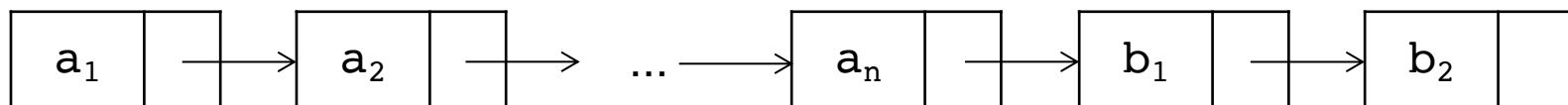
فرض کنید دو لیست پیوندی A و B موجود باشند، می‌خواهیم B را به انتهای A بچسبانیم.



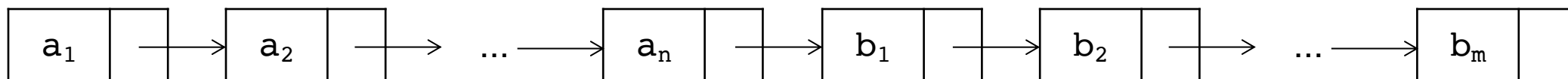
اتصال دو لیست پیوندی

می‌توانیم با استفاده از همان عمل درج در لیست A، تک به تک عناصر را به انتهای B اضافه کنیم.

- برای این کار از m عمل درج استفاده می‌کنیم که برای هر کدام باید یک درج در انتهای A انجام دهیم که برای این کار به $O(n)$ عملیات نیاز است، این یعنی در مجموع پیچیدگی الگوریتم ما $O(nm)$ خواهد بود.



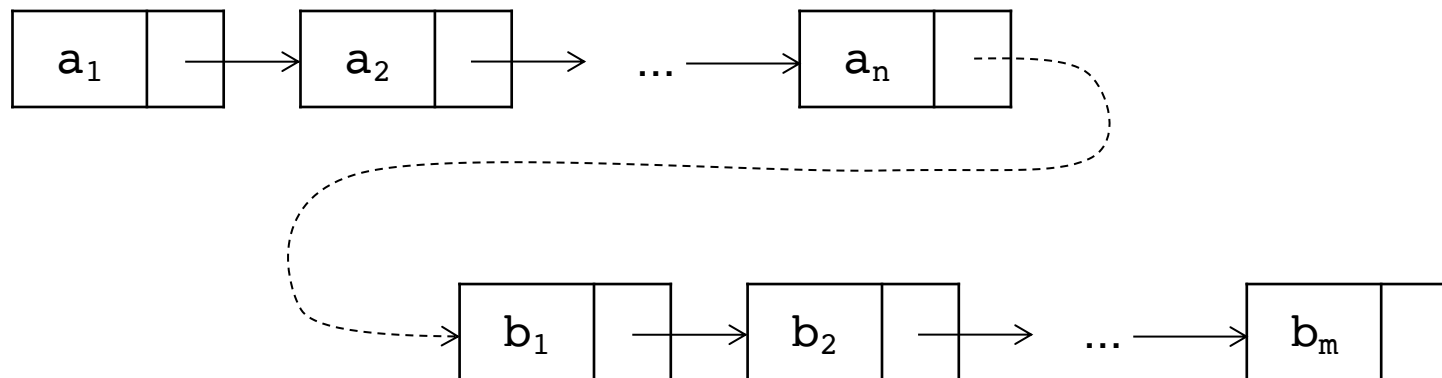
⋮



اتصال دو لیست پیوندی

```
def concate(A,B):  
    if (A==None):  
        A.head=B.head  
    else:  
        if (B!=None):  
            p=A.head  
            while (p.next!=None):  
                p=p.next  
            p=B.head  
    return
```

اما میتوانیم لیست A را تا به انتها پیمایش کنیم؛
و سپس اشاره گر عنصر نهایی را به عنصر ابتدایی B وصل کنیم.

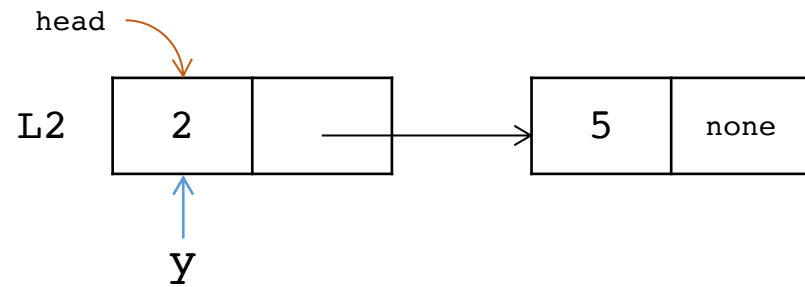
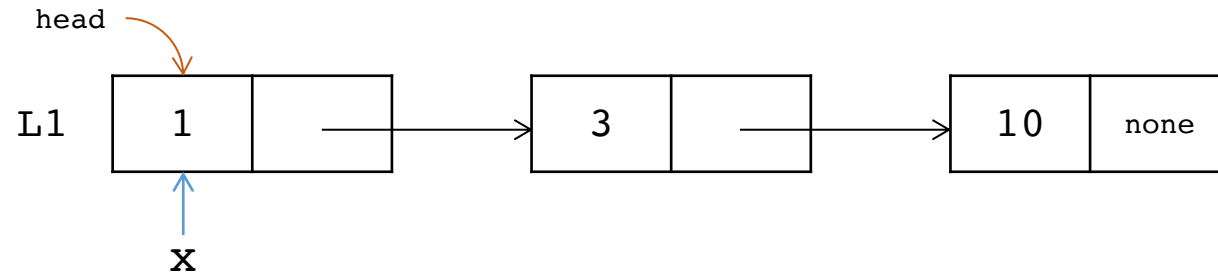


ادغام دو لیست پیوندی مرتب در یک لیست

```
def merge(L1,L2):  
    L= new empty linked list  
    x=L1.head  
    y=L2.head  
    while(x!=None or y!=None):  
        item=None  
        if (x==None):  
            item=y.data, y=y.next  
        else if (y==None):  
            item=x.data, x=x.next  
        else if (y.data<x.data):  
            item=y.data, y=y.next  
        else:  
            item=x.data, x=x.next  
        L.append(item)  
    return L
```

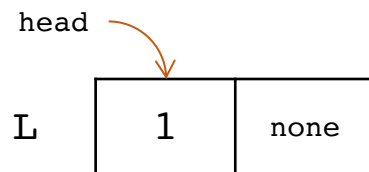
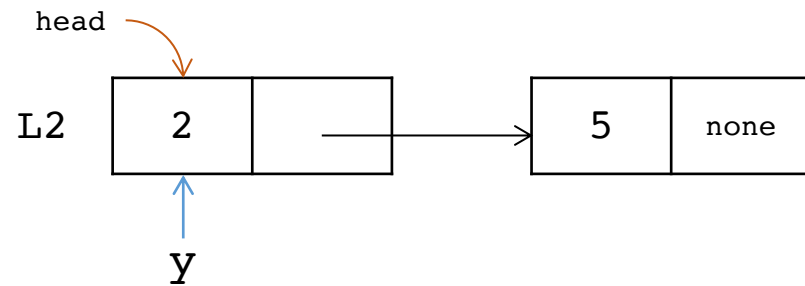
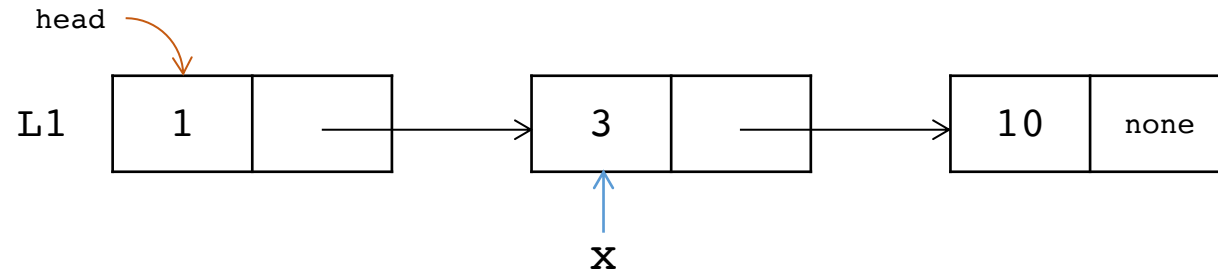
پیچیدگی مرتبه زمانی؟

ادغام دو لیست پیوندی مرتب در یک لیست

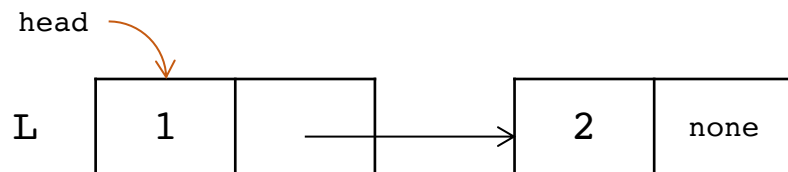
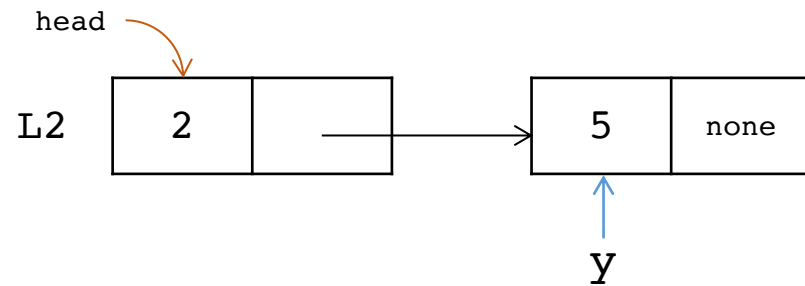
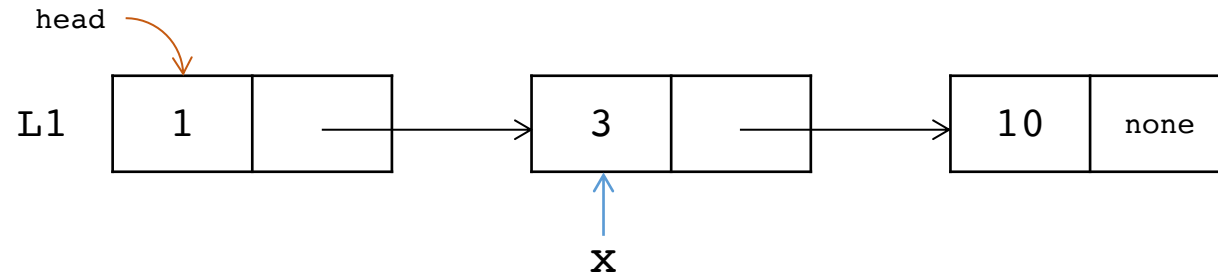


L

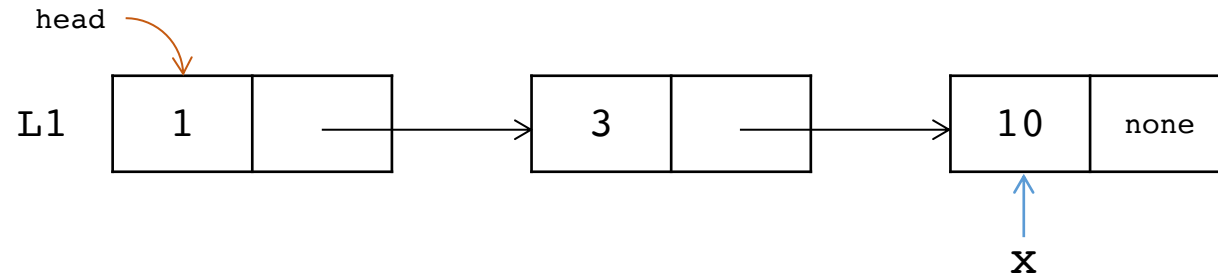
ادغام دو لیست پیوندی مرتب در یک لیست



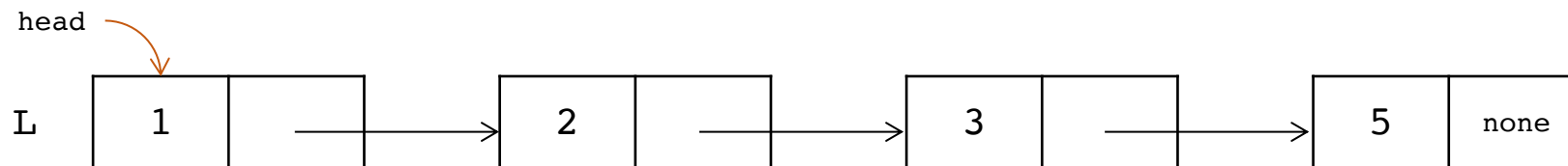
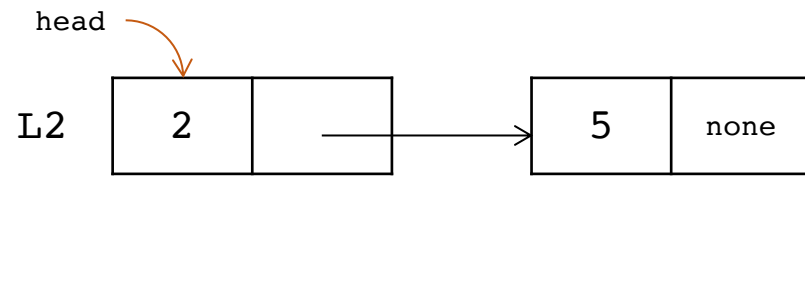
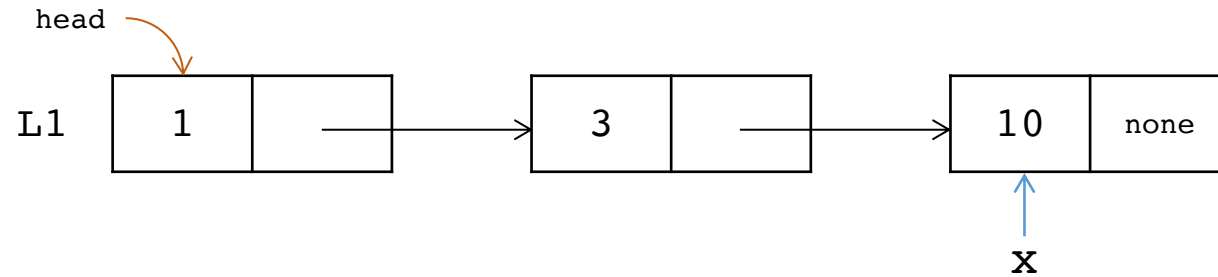
ادغام دو لیست پیوندی مرتب در یک لیست



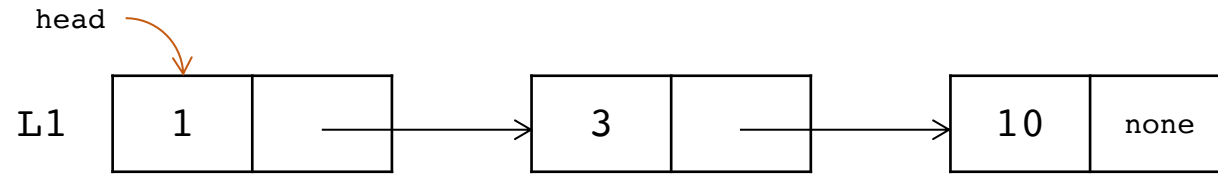
ادغام دو لیست پیوندی مرتب در یک لیست



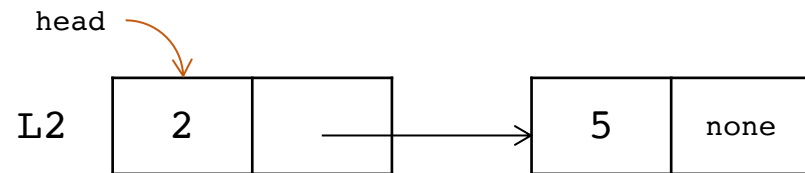
ادغام دو لیست پیوندی مرتب در یک لیست



ادغام دو لیست پیوندی مرتب در یک لیست



↑
x



↑
y

