

## داده ساختارهای ساده

### • لیست‌ها

-- لیست‌ها یک‌سویه، دوسویه، حلقه‌ای

-- پشته

-- صف

-- لیست‌های کلی

-- کاربردهای لیست‌ها

• درخت‌ها

- درخت‌های کلی، دودویی
- درخت عبارت و کار با آن‌ها
- درخت دودویی جست‌وجو

## لیست‌ها

دنباله‌ای از عناصر، که ترتیب آن‌ها مهم است

### اعمال متداول

- ایجاد یک لیست تهی
- محاسبه‌ی تعداد عناصر موجود در لیست (اندازه‌ی لیست)
- درج یک عنصر در ابتدای یا انتهای لیست
- درج یک عنصر بعد یا قبل از یک عنصر داده‌شده
- حذف یک عنصر از (ابتدا، انتها، یا عنصر بعدی) لیست

## لیست‌ها (ادامه)

بسته به مکان درج یا حذف یک عنصر، لیست‌ها با اسامی زیر شناخته می‌شوند:

- پشته (stack): درج و حذف فقط در یک طرف لیست

First-In-Last-Out (FILO) یا Last-In-First-Out (LIFO)

- صف (queue): درج فقط در انتها و حذف از ابتدای

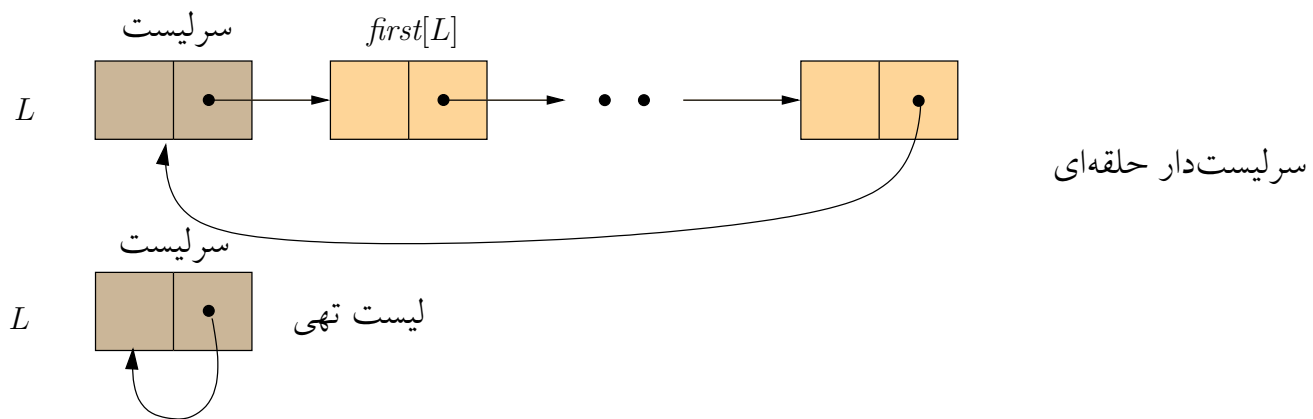
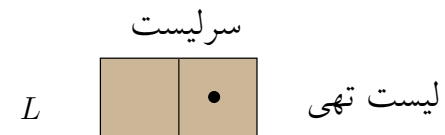
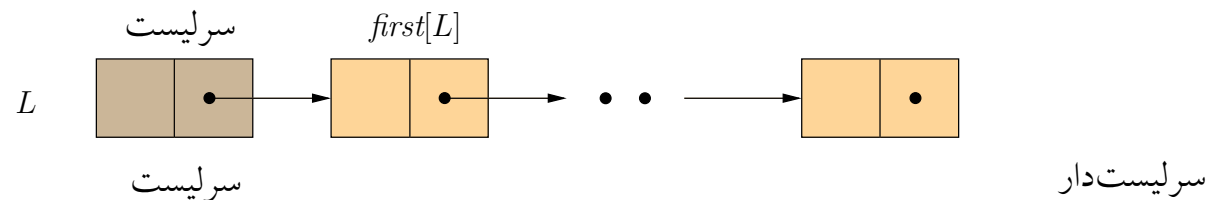
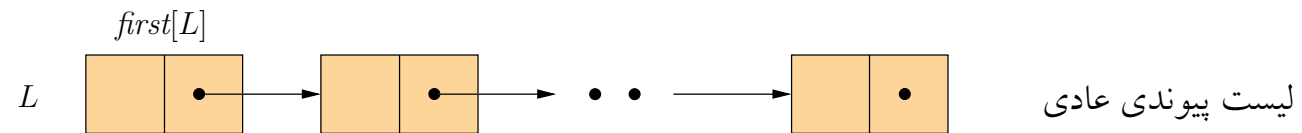
First-In-First-Out (FIFO)

انواع لیست‌های پیوندی: (خطی) یک‌سویه، دوسویه، حلقه‌ای، سلسله‌مراتبی و لیست‌های کلی

## لیست‌های پیوندی یک‌سویه

```
class Node {  
    private Object element;  
    private Node next;  
    // constructors  
    Node(){  
        this(null,null);  
    }  
    public Node(Object e, Node n){  
        element = e  
        next = n;  
    }  
    void setElement(Object newElem){ element = newElem;}  
    void setNext(Node newNext){ next = newNext;}  
    Object getElement(){ return element;}  
    Node getNext() {return next;}  
}
```

## داده‌ساختارها و مبانی الگوریتم‌ها



## پیاده‌سازی با CLRS

مولفه‌ها:  $next$ ،  $element$  و  $size[L]$

شبه کد CLRS	جاوا (Java)
$x \leftarrow \text{ALLOCATE-NODE}()$	<code>x = new Node()</code>
$x \leftarrow \text{ALLOCATE-NODE}(e, n)$	<code>x = new Node(element e, next n)</code>
$\text{FREE-NODE}(x)$	<code>null x</code>
$next[x]$	<code>x.getNext()</code>
$next[x] \leftarrow n$	<code>x.setNext(n)</code>

## اعمال اصلی بر روی یک لیست خطی

- $\text{CREATE-LIST}(L)$ : ایجاد یک لیست تهی  $L$
- $\text{SIZE}(L)$ : تعداد عناصر لیست را برمی‌گرداند
- $\text{FIRST}(L)$ : عنصر اول را برمی‌گرداند
- $\text{ISEMPTY}(L)$ : مشخص می‌کند که آیا لیست خالی
- $\text{INSERT-FIRST}(L, x)$ : درج عنصری با مقدار  $x$  در ابتدای  $L$
- $\text{INSERT-AFTER}(L, x, n)$ : درج عنصری با مقدار  $x$  پس از عنصر  $n$  در  $L$
- $\text{DELETE-FIRST}(L)$ : عنصر اول لیست  $L$  را حذف می‌کند
- $\text{DELETE-AFTER}(L, n)$ : عنصر پس از عنصر  $n$  در  $L$  را حذف می‌کند



## پیاده‌سازی

CREATE ( $L$ )

1  $size[L] \leftarrow 0$

SIZE ( $L$ )

1 **return**  $size[L]$

FIRST ( $L$ )

1 **if**  $SIZE(L) \neq 0$

2     **then return**  $first[L]$

3     **else error** list is empty

ISEMPTY ( $L$ )

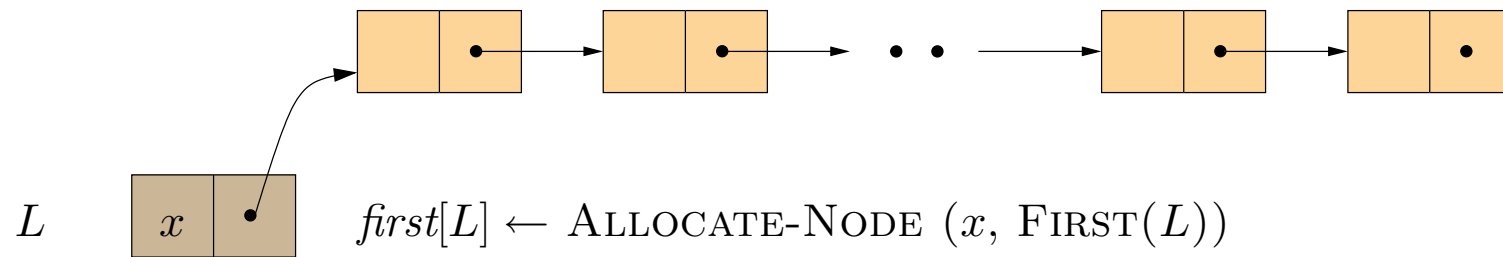
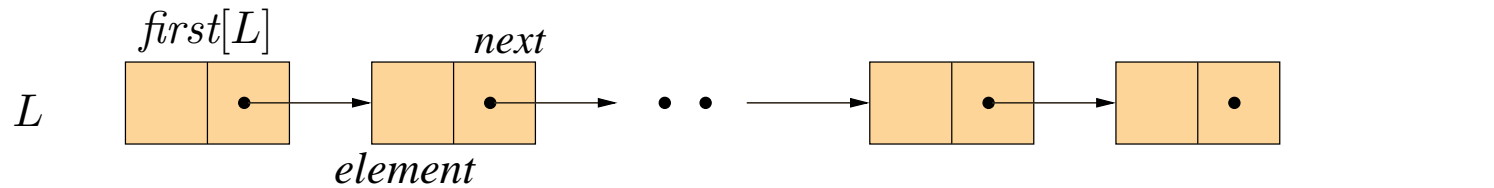
1 **return**  $SIZE(L) = 0$

INSERT-FIRST ( $L, x$ )

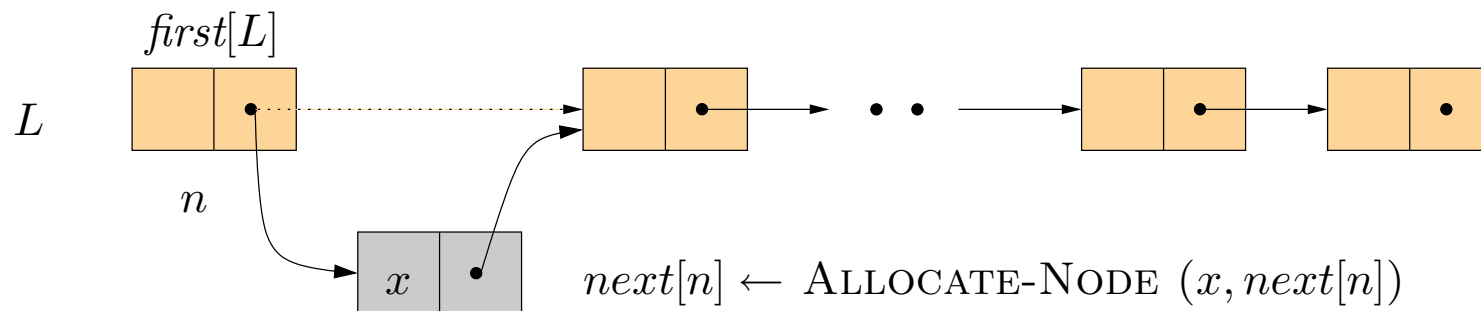
- 1  $first[L] \leftarrow Allocate-Node(x, First(L))$
- 2  $size[L] \leftarrow size[L] + 1$

INSERT-AFTER ( $L, x, n$ )

- 1 **if**  $n = \text{null}$
- 2     **then error** element is empty
- 3  $next[n] \leftarrow ALLOCATE-NODE(x, next[n])$
- 4  $size[L] \leftarrow size[L] + 1$



INSERT-FIRST( $L, x$ )



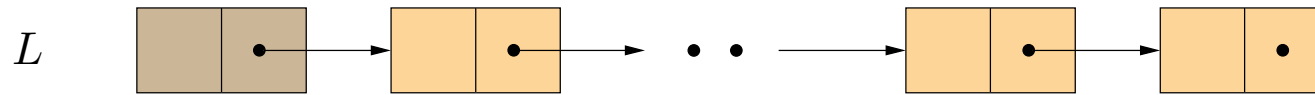
INSERT-AFTER( $L, x, n$ )

DELETE-FIRST ( $L$ )

```
1  if ISEMPTY ( $L$ )  
2    then error list is empty  
3   $n \leftarrow \text{FIRST}(L)$   
4   $first[L] \leftarrow next[n]$   
5  FREE-NODE( $n$ )  
6   $size[L] \leftarrow size[L] - 1$ 
```

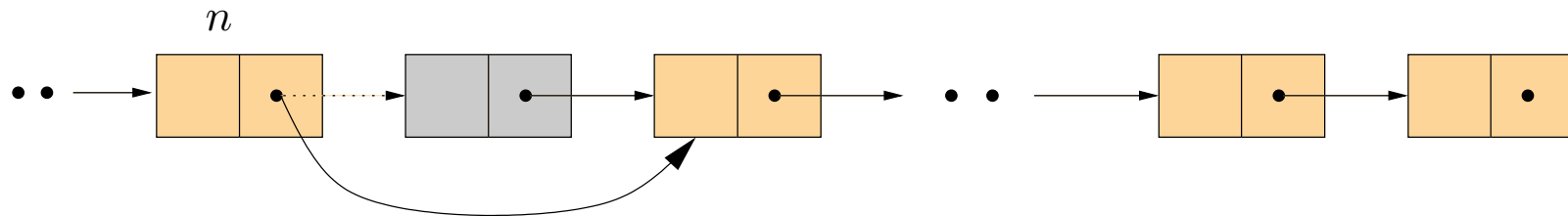
DELETE-AFTER ( $L, n$ )

```
1  if ISEMPTY ( $L$ ) or  $n = \text{null}$  or  $\text{next}[n] = \text{null}$ 
2    then error element does not exist
3   $r \leftarrow \text{next}[n]$ 
4   $\text{next}[n] \leftarrow \text{next}[r]$ 
5  FREE-NODE( $r$ )
6   $\text{size}[L] \leftarrow \text{size}[L] - 1$ 
```



$$first[L] \leftarrow next[first[L]]$$

DELETE-FIRST( $L$ )



$$next[n] \leftarrow next[next[n]]$$

DELETE-AFTER( $L, n$ )

روشن است که هر یک از این اعمال در  $O(1)$  قابل انجام است.

## درج و حذف در لیست دوسویه‌ی خطی

### INSERT-AFTER ( $L, x, n$ )

▷ عنصری با محتوای  $x$  را پس از عنصر  $n$  در لیست دوسویه‌ی  $L$  درج می‌کند

```
1  if ISEMPTY( $L$ ) or  $n = \text{null}$ 
2    then error element  $n$  does not exist
3   $r \leftarrow \text{next}[n]$ 
4   $\text{next}[n] \leftarrow \text{ALLOCATE-NODE} (x, n, r)$ 
5   $\text{prev}[r] \leftarrow \text{next}[n]$ 
6   $\text{size}[L] \leftarrow \text{size}[L] + 1$ 
```

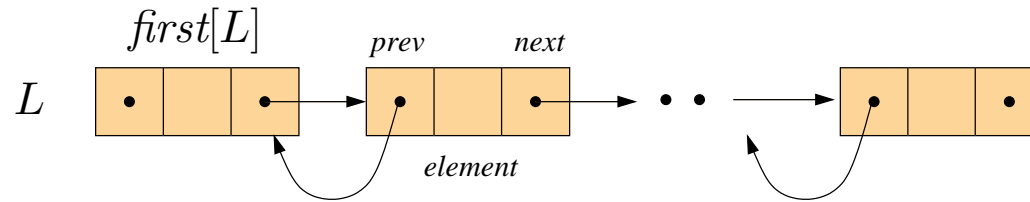


### DELETE-AFTER ( $L, n$ )

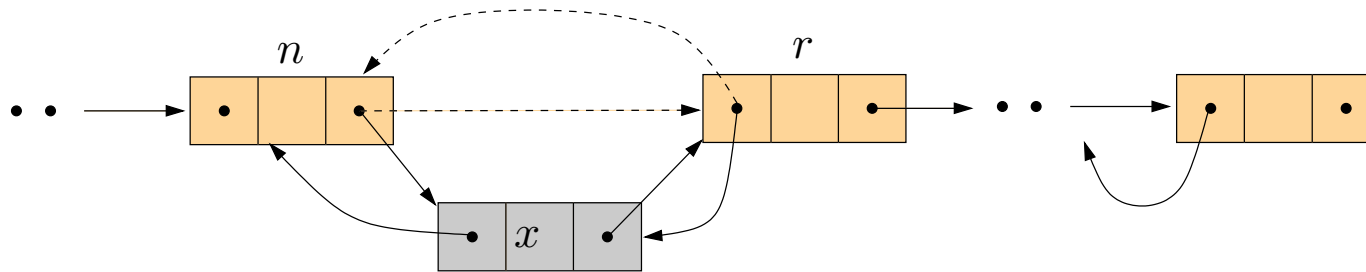
▷ عنصر بعدی  $n$  را در لیست دو سوییجه  $L$  حذف می‌کند

```
1  if ISEMPTY( $L$ ) or  $n = \text{null}$  or  $\text{next}[n] = \text{null}$ 
2    then error element does not exist
3   $r \leftarrow \text{next}[n]$ 
4  if  $\text{next}[r] \neq \text{null}$ 
5    then  $\text{prev}[\text{next}[r]] \leftarrow n$ 
6   $\text{next}[n] \leftarrow \text{next}[r]$ 
7  FREE-NODE( $r$ )
8   $\text{size}[L] \leftarrow \text{size}[L] - 1$ 
```

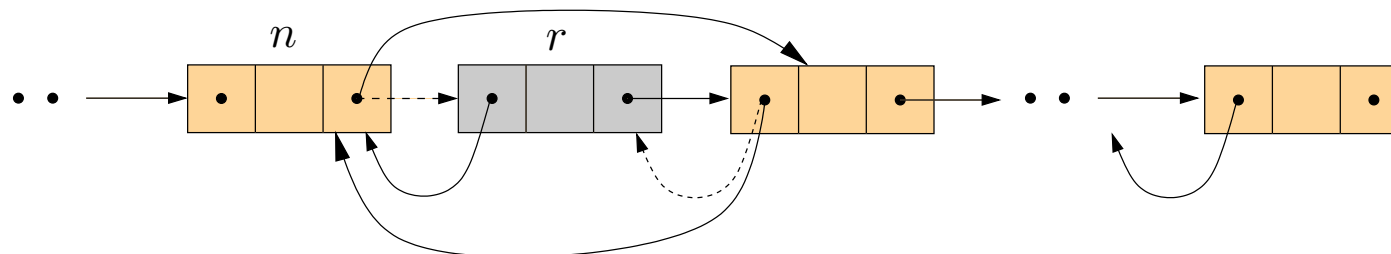
## داده ساختارها و مبانی الگوریتم‌ها



لیست خطی دوسویه‌ای

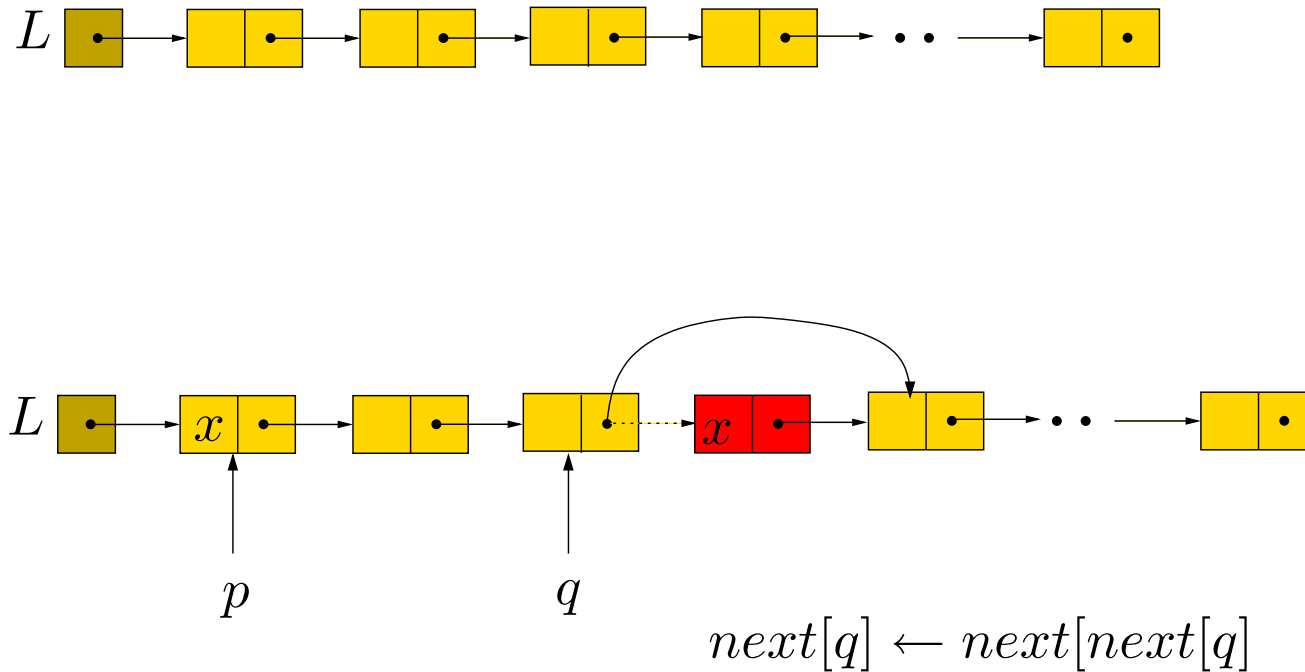


INSERT-AFTER( $L, x, n$ )



DELETE-AFTER( $L, n$ )

## عملیات دیگر بر روی لیست‌ها: حذف عناصر تکراری در یک لیست



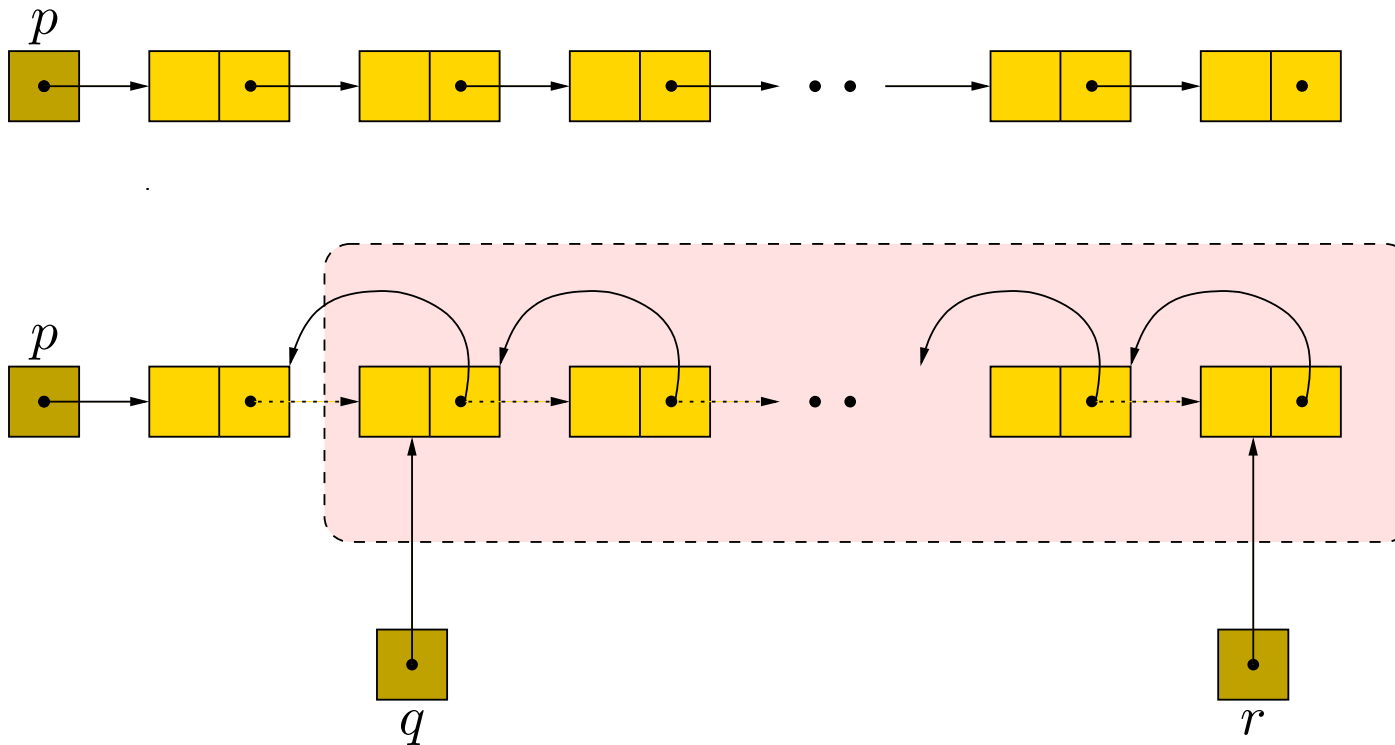
### PURGELIST ( $L$ )

▷ همه‌ی عناصر یکسان را جز یکی حذف می‌کند

```
1  $p \leftarrow \text{FIRST}(L)$ 
2 while  $p \neq \text{null}$ 
3     do  $q \leftarrow p$ 
4         while  $\text{next}[q] \neq \text{null}$ 
5             do if  $\text{element}[p] = \text{element}[\text{next}[q]]$ 
6                 then DELETE-AFTER ( $L, q$ )
7                 else  $q \leftarrow \text{next}[q]$ 
8      $p \leftarrow \text{next}[p]$ 
```

آیا می‌توان این کار را در  $O(n \lg n)$  انجام داد؟

## وارون کردن یک لیست با تغییر اشاره‌گرها

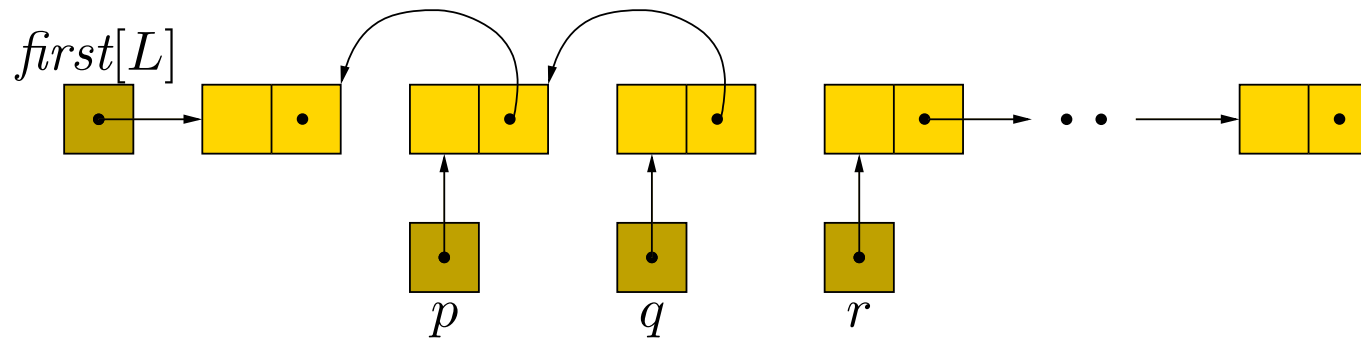


### RECURSIVE-REVERSE ( $L, p$ )

لیست  $L$  را از عنصر  $p$  به بعد وارون می‌کند و حاصل را برمی‌گرداند.  $\triangleright$

```
1  if  $p = \text{null}$  or  $\text{next}[p] = \text{null}$ 
2    then return  $p$ 
3   $q \leftarrow \text{next}[p]$ 
4   $r \leftarrow \text{RECURSIVE-REVERSE}(L, q)$ 
5   $\text{next}[q] \leftarrow p$ 
6   $\text{next}[p] \leftarrow \text{null}$ 
7  return  $r$ 
```

## وارون کردن یک لیست به صورت غیر بازگشتی



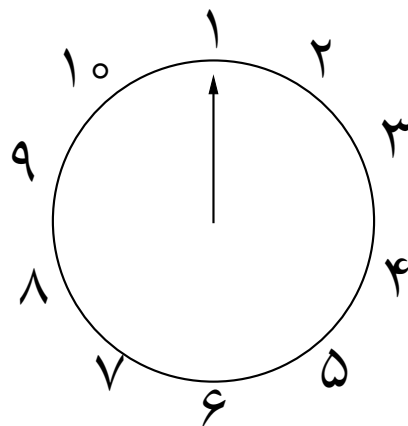
NR-REVERSE ( $L$ )

```
1  if SIZE( $L$ )  $\leq$  1
2    then return  $first[L]$ 
3   $p \leftarrow$  null
4   $q \leftarrow$  FIRST( $L$ )
5   $r \leftarrow next[q]$ 
6  while  $r \neq$  null
7    do  $next[q] \leftarrow p$ 
8       $p \leftarrow q$ 
9       $q \leftarrow r$ 
10      $r \leftarrow next[r]$ 
11   $next[q] \leftarrow p$ 
12  return  $q$ 
```



## مسئله‌ی ژوزفوس

اگر  $n$  نفر با شماره‌های ۱ تا  $n$  دور دایره‌ای قرار بگیرند و با شروع از شماره‌ی ۱ و در جهت ساعت‌گرد هر بار دومین (یا  $k$  امین) نفر خودش را بکشد، آخرین نفر چه شماره‌ای دارد؟



مسئله‌ی ژوزفوس با ۱۰ نفر.

برای  $n = 10$  به ترتیب افراد ۲، ۴، ۶، ۸، ۱۰، ۳، ۷، ۱، ۹ خودکشی می‌کنند و ۵ زنده می‌ماند.

جواب این مسئله  $J(n)$  به صورت ریاضی قابل محاسبه است و می توان جواب را از رابطه‌ی بازگشتی زیر به دست آورد.

$$\begin{aligned} J(1) &= 1 \\ J(2n) &= 2J(n) - 1, \text{ for } n \geq 1, \\ J(2n + 1) &= 2J(n) + 1 \text{ for } n \geq 1. \end{aligned}$$

اگر  $n$  را به صورت عدد دودویی بنویسیم و آن را یک بیت شیفت چپ دورانی  $J(n)$  به دست می آید.

مثلاً برای  $n = 100 = (1100100)_2$ ، جواب  $J(n) = (100101)_2 = 73$  است.

## حل مسئله‌ی ژوزفوس با لیست پیوندی حلقه‌ای

# JOSEPHUS ( $n$ )

▷ در ابتدا یک لیست پیوندی حلقوی با  $n$  گره ایجاد می‌کند

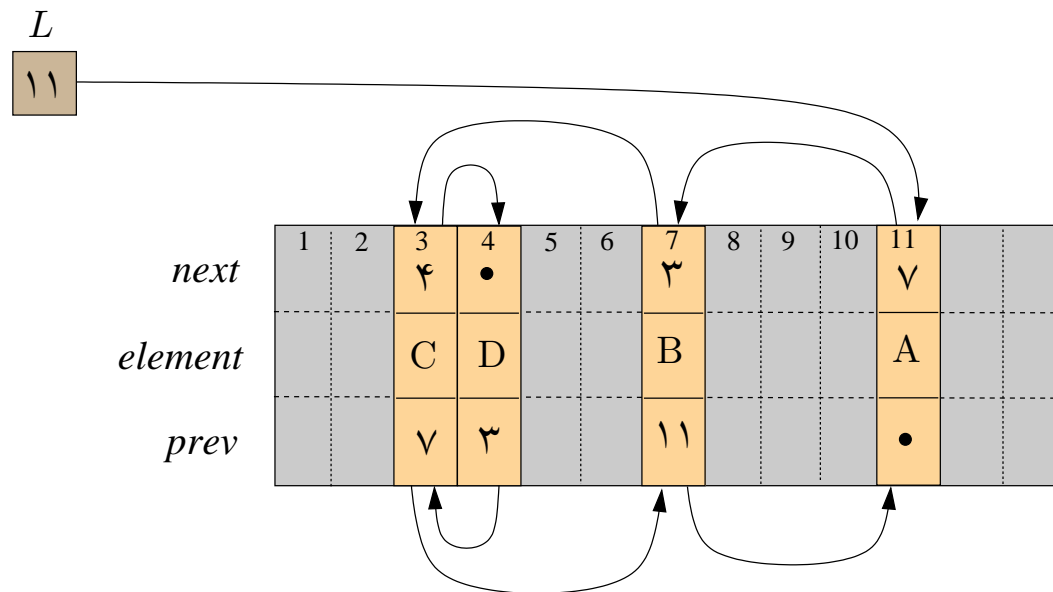
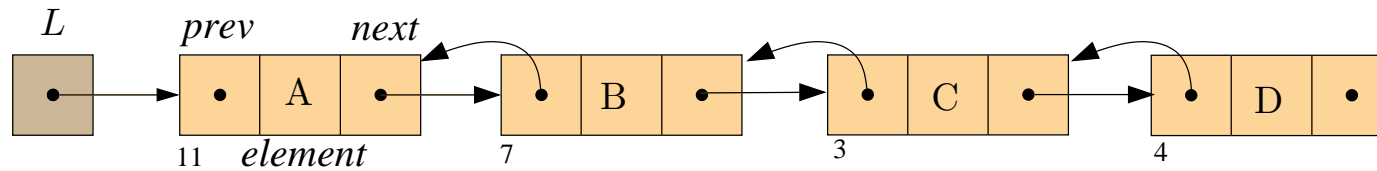
```

1  CREATE( $L$ )
2   $first[L] \leftarrow q \leftarrow \text{ALLOCATE-NODE}(1, \text{null})$ 
3   $p \leftarrow q$ 
4  for  $i \leftarrow 2$  to  $n$ 
5      do  $next[p] \leftarrow \text{ALLOCATE-NODE}(i, next[p])$ 
6       $p \leftarrow next[p]$ 
7   $next[p] \leftarrow q; \quad size[L] \leftarrow n$ 
    ▷ و حالا راه حل گند مسئله‌ی ژوزفوس
8   $p \leftarrow \text{FIRST}(L)$ 
9  while  $next[p] \neq p$ 
10     do  $\text{DELETE-AFTER}(L, p)$ 
11      $p \leftarrow next[p]$ 
12  PRINT  $element[p]$ 
    
```

## پیاده‌سازی لیست‌ها با اشاره‌گرهای اندیسی

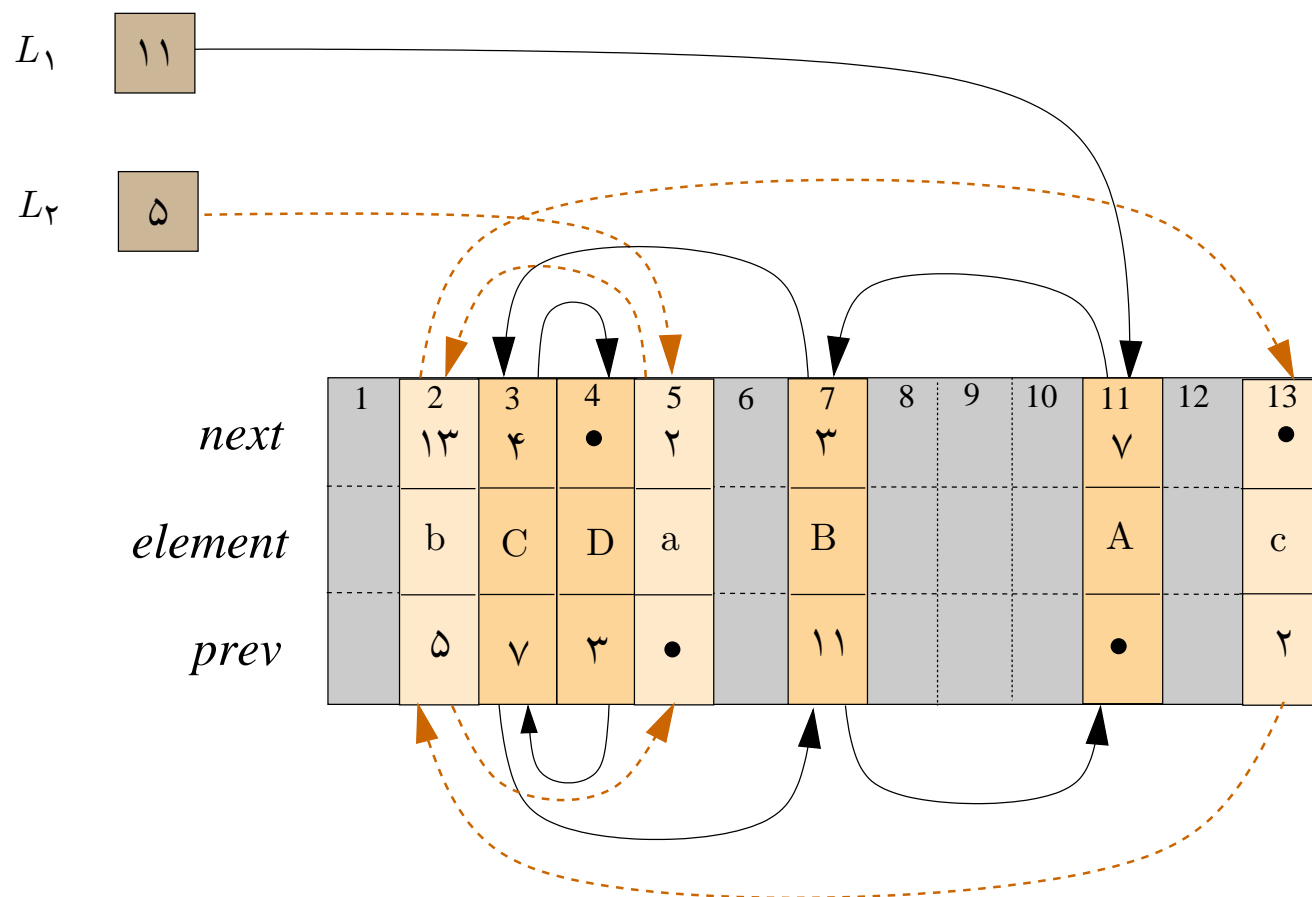
یا زبان فرترن لیست‌ها را چه گونه پیاده‌سازی می‌کنیم؟

## داده ساختارها و مبانی الگوریتم‌ها



سه آرایه‌ی  $prev$ ،  $element$  و  $next$ . اشاره‌گرها اندیس هستند.

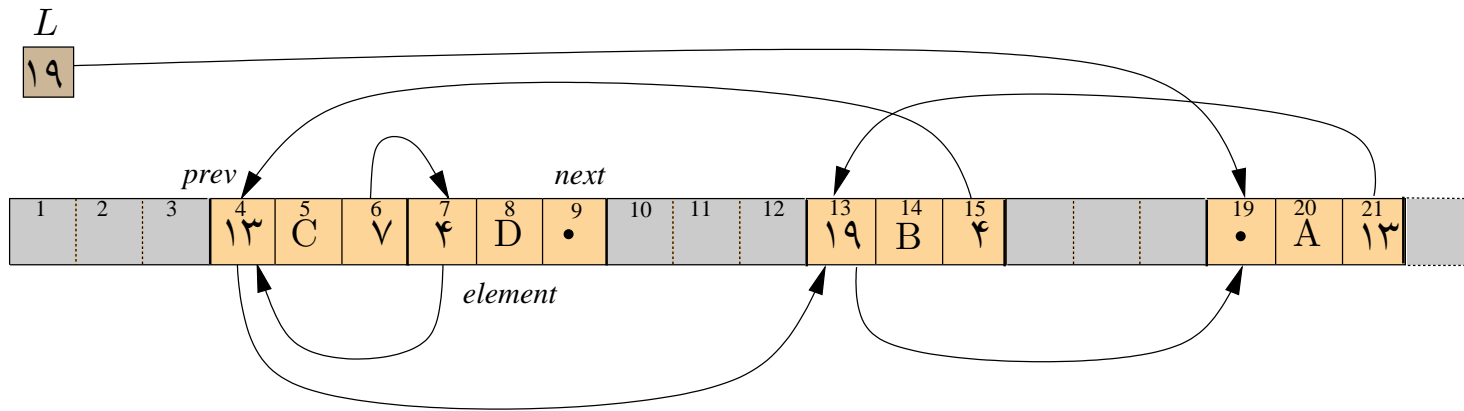
## داده ساختارها و مبانی الگوریتم‌ها



بیش از یک لیست (مثلاً  $L_1$  و  $L_2$ ) را می‌توان در همان حافظه پیاده‌سازی کرد.



## داده ساختارها و مبانی الگوریتم‌ها



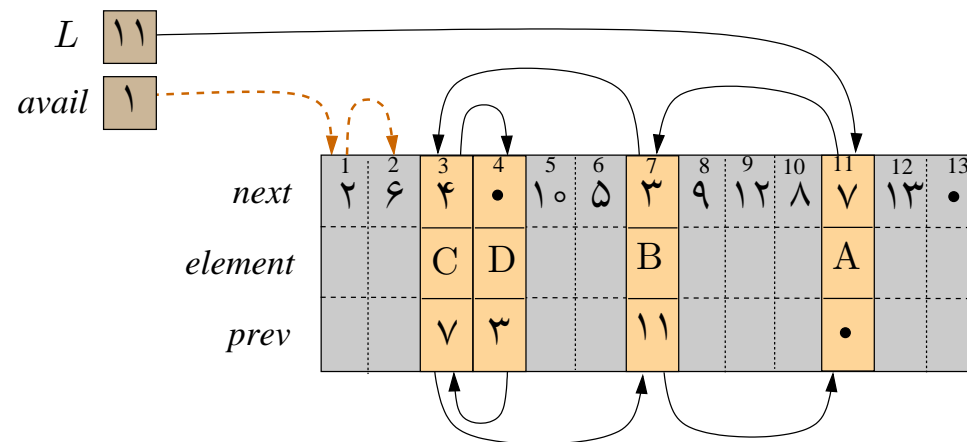
می‌توان فقط از یک آرایه برای پیاده‌سازی یک لیست استفاده کرد.

## مدیریت فضای آزاد

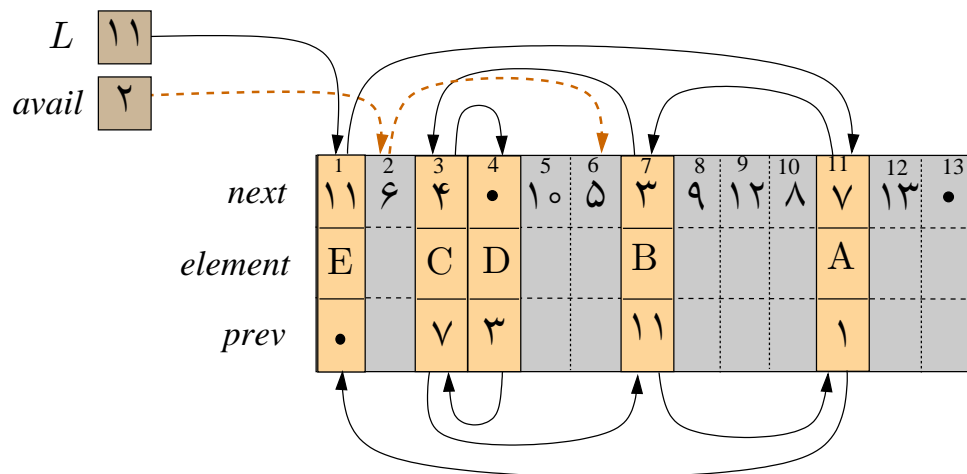
- همه‌ی عناصر آزاد در لیستی به‌نام *avail* قرار می‌گیرند.
- در ابتدا همه‌ی عناصر آرایه‌ها آزادند.
- (عمل Allocate-Object) برای درج، اولین عنصر لیست آزاد از *avail* حذف شده در اختیار قرار می‌گیرد
- (عمل Free-Object) عنصر با درج در ابتدای لیست *avail* آزاد می‌شود.

## مثالی از درج و حذف در یک لیست با اشاره‌گر اندیسی

## داده‌ساختارها و مبانی الگوریتم‌ها

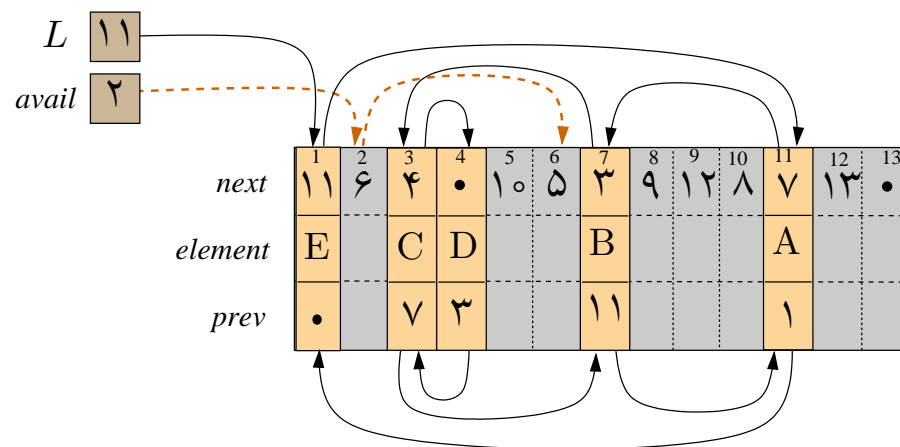


(الف) لیست اصلی

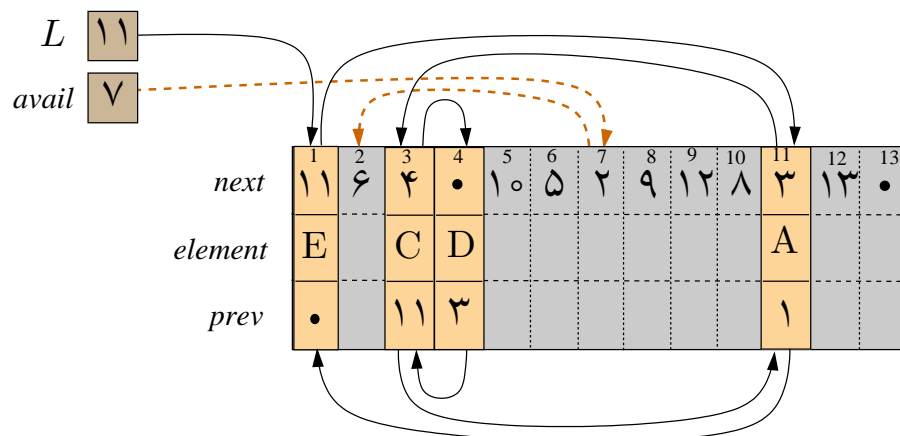


(ب) پس از درج **E** به عنوان عنصر اول لیست (الف)

## داده ساختارها و مبانی الگوریتم‌ها



(ب) پس از درج **E** به عنوان عنصر اول لیست (الف)



(پ) پس از حذف عنصر **B** در لیست (ب)

## پیاده سازی

INITIALIZE ()

```
1  null  $\leftarrow$  0
2  avail  $\leftarrow$  1
3  for  $i \leftarrow 1$  to  $M - 1$ 
4      do  $next[i] \leftarrow i + 1$ 
5   $next[M] \leftarrow$  null
```

## پیاده‌سازی (ادامه)

### ALLOCATE-OBJECT ()

```
1  if avail = null
2    then error out of space
3  x ← avail
4  avail ← next[avail]
5  return x
```

### FREE-OBJECT (*x*)

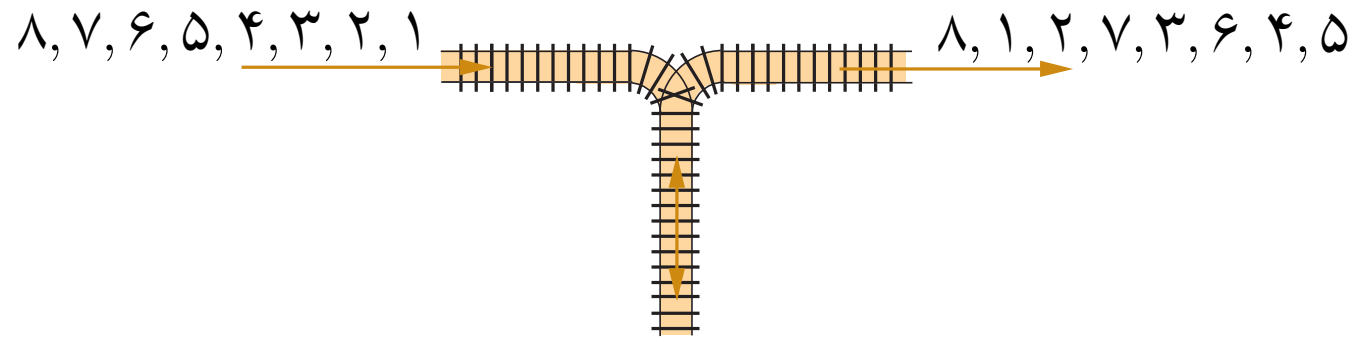
```
1  next[x] ← avail
2  avail ← x
```

## زباله‌روبی (Garbage Collection)



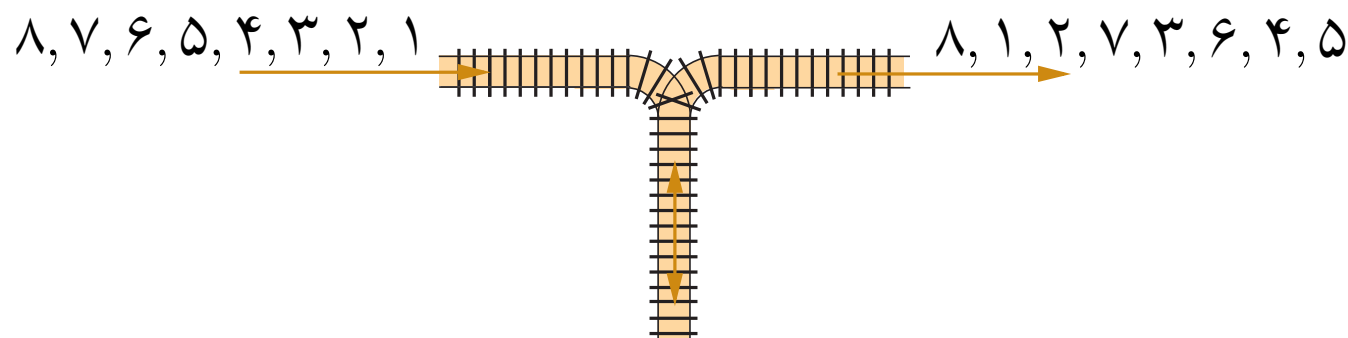
## پشته‌ها

- $\text{PUSH}(S, x)$ : درج یک عنصر  $x$  در بالای پشته‌ی  $S$   
ورودی: عنصر (شیء)، خروجی: هیچ
- $\text{POP}(S)$ : حذف و بازگرداندن عنصر بالای پشته  
ورودی: هیچ، خروجی: عنصر (شیء)، خطا: اگر پشته خالی باشد
- $\text{SIZE}(S)$ : تعداد عناصر موجود در پشته  
ورودی: هیچ، خروجی: یک عدد صحیح
- $\text{ISEMPTY}(S)$ : مشخص می‌کند که آیا پشته خالی است  
ورودی: هیچ، خروجی: درست یا نادرست
- $\text{TOP}(S)$ : عنصر بالای پشته را برمی‌گرداند  
ورودی: هیچ، خروجی: عنصر (شیء)، خطا: اگر پشته خالی باشد



پشته‌ی قطارها.

$\langle 8, 1, 2, 7, 3, 6, 4, 5 \rangle$  قابل تولید است. چه طور؟



پشته‌ی قطارها.

۸, ۱, ۲, ۷, ۳, ۶, ۴, ۵ قابل تولید است. چه طور؟

Push, Push, Push, Push, Push, Pop, Pop, Push, Pop, Pop, Push, Pop, Pop, Push, Pop

۱, ۸, ۳, ۶, ۲, ۷, ۴, ۵ چه طور؟

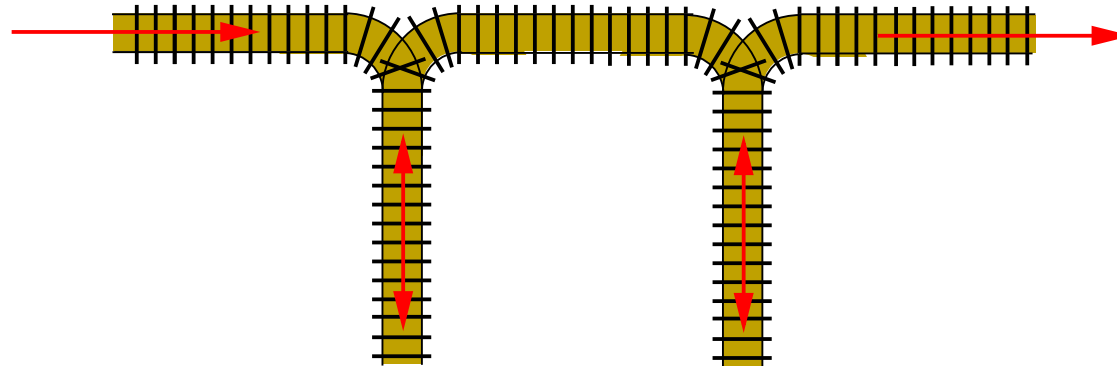
## چند مسئله

(۱) شرط لازم و کافی برای یک دنباله که قابل تولید باشد چیست؟ آنرا اثبات کنید.

(۲) الگوریتمی از  $O(n)$  ارائه دهید تا قابل تولید بودن دنباله‌ای را تشخیص دهد.

(۳) فرض کنید یک ریل مستقیم هم بین قطارهای ورودی و خروجی وجود دارد؛ یعنی اولین قطار ورودی می‌تواند یا به داخل پشته رود و یا مستقیماً به ریل خروجی منتقل شود، و یا برعکس از خروجی به ورودی. در این صورت الگوریتم تشخیص یک دنباله‌ی قابل تولید را ارائه دهید.

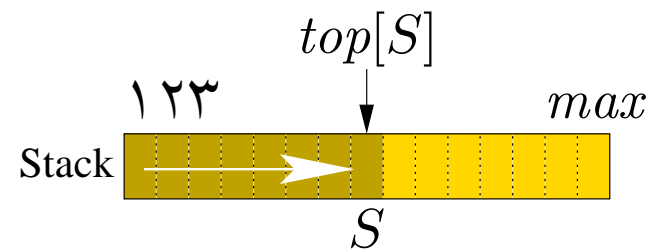
(۴) مسئله‌ی اصلی را برای سیستم دو پشته‌ای مطابق شکل حل کنید.



## پیاده‌سازی پشته با آرایه

آرایه‌ی  $S$  با اندازه‌ی حداکثر  $max$  و مؤلفه‌ی  $top[S]$  که اندیس بالاترین عنصر موجود در  $S$  است.

```
public class ArrayStack implements Stack {  
    public static final int CAPACITY=1000;  
    private int capacity;  
    private object S[];  
    private int top = -1;  
    public ArrayStack(){  
        this(CAPACITY);  
    }  
    public ArrayStack(int cap){  
        capacity = cap;  
        S = new object[capacity];  
    }  
}
```



پیاده‌سازی پشته با آرایه.

SIZE( $S$ )

1 **return**  $top[S]$    ▷ assuming that initially  $top[S] = 0$

ISEMPTY( $S$ )

1 **return**  $SIZE(S) = 0$

TOP( $S$ )

1 **if**  $ISEMPTY(S)$

2   **then error** ("STACK IS EMPTY")

3 **return**  $S[top[S]]$



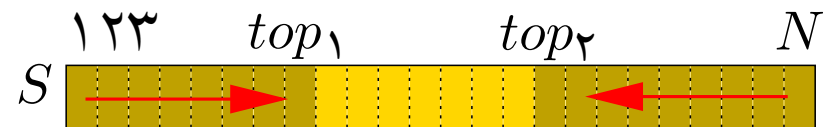
PUSH ( $S, x$ )

```
1  if SIZE( $S$ ) =  $max$ 
2    then error ("stack is full")
3   $top[S] \leftarrow top[S] + 1$ 
4   $S[top[S]] \leftarrow x$ 
```

POP ( $S$ )

```
1  if isEmpty()
2    then error ("stack is empty")
3   $e \leftarrow S[top[S]]$ 
4   $top[S] \leftarrow top[S] - 1$ 
5  return  $e$ 
```

## داده ساختارها و مبانی الگوریتم‌ها



پیاده‌سازی چند پشته با آرایه.

## تحلیل

همه‌ی اعمال در  $O(1)$  انجام می‌شوند.

## پیاده‌سازی پشته با لیست پیوندی

SIZE( $S$ )

1 **return**  $size[S]$

ISEMPTY( $S$ )

1 **return** ( $size[S] = 0$ )

TOP( $S$ )

1 **if** isEmpty( $S$ )

2   **then error** ("stack is empty")

3 **return**  $top[S]$

PUSH ( $S, x$ )

- 1  $top[S] \leftarrow \text{ALLOCATE-NODE}(x, top[S])$
- 2  $size[S] \leftarrow size[S] + 1$

POP ( $S$ )

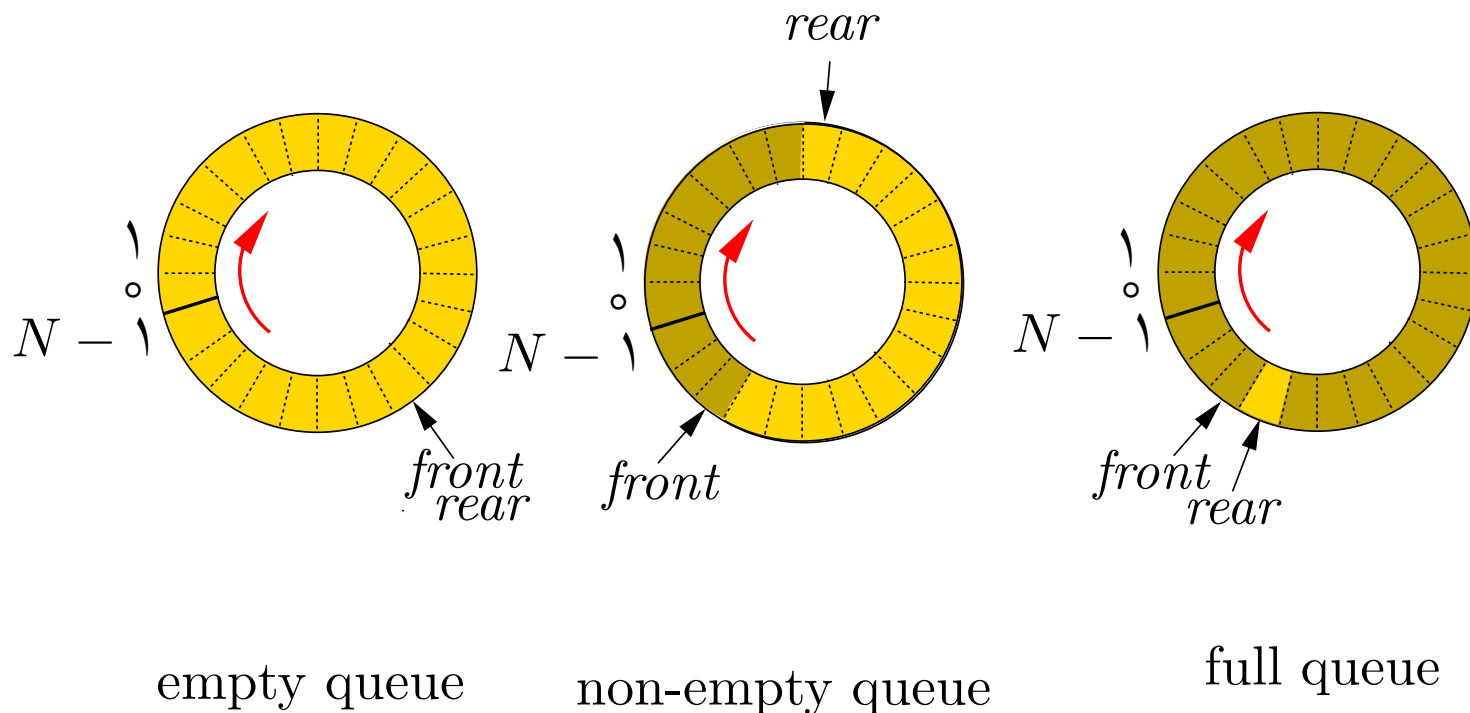
- 1 **if**  $\text{ISEMPTY}(S)$
- 2     **then error** ("STACK IS EMPTY")
- 3  $n \leftarrow top[S]$
- 4  $temp \leftarrow element[n]$
- 5  $top[S] \leftarrow next[n]$
- 6  $size[S] \leftarrow size[S] - 1$
- 7  $\text{FREE-OBJECT}(n)$
- 8 **return**  $temp$



درج در انتها و حذف در ابتدای لیست

- $\text{ENQUEUE}(Q, x)$ : درج یک عنصر در انتهای صف  
ورودی: عنصر (شیء)، خروجی: هیچ
- $\text{DEQUEUE}(Q)$ : حذف عنصر از ابتدای صف  
ورودی: هیچ، خروجی: عنصر (شیء)، خطا: اگر خالی باشد
- $\text{SIZE}(Q)$ : تعداد عناصر موجود در صف  
ورودی: هیچ، خروجی: یک عدد صحیح
- $\text{ISEMPTY}(Q)$ : مشخص می‌کند که آیا صف خالی است  
ورودی: هیچ، خروجی: درست یا نادرست
- $\text{FRONT-ELEMENT}(Q)$ : عنصر ابتدای صف را برمی‌گرداند  
ورودی: هیچ، خروجی: عنصر، خطا: اگر خالی باشد

## پیاده‌سازی با آرایه‌ی دوار





## یک صف $Q$

- یک آرایه با اندازه‌ی  $max$  و اندیس‌های  $0$  تا  $max - 1$
- عناصر به صورت دوار و در جهت ساعت گرد ذخیره می‌شوند.
- $Q[(i + 1) \bmod max]$  عنصر بعدی  $Q[i]$  است.
- مولفه‌ی  $front[Q]$  اندیس عنصر ابتدایی صف
- $rear[Q]$  اندیس عنصر بعدی آخرین عنصر صف.
- بنابراین حداکثر تعداد عناصر  $max - 1$  است.
- می‌خواهیم دو حالت «کاملاً پر» و «کاملاً خالی» را بتوانیم از هم تمیز دهیم.

## حالت‌های مختلف صف

- در شروع  $front[Q] = rear[Q] = 0$
- تعداد عناصر همیشه برابر  $(max - front[Q] + rear[Q]) \bmod max$ .
- اگر صف کاملاً خالی باشد داریم  $front[Q] = rear[Q]$ .
- اگر کاملاً پر باشد داریم  $(max - front[Q] + rear[Q]) \bmod max = max - 1$ .

SIZE ( $Q$ )

1 **return**  $(max - front[Q] + rear[Q]) \bmod max$

ISEMPTY ( $Q$ )

1 **return**  $(front[Q] = rear[Q])$

FRONT-ELEMENT ( $Q$ )

1 **if** ISEMPTY( $Q$ )  
2     **then error** "Queue is empty"  
3 **return**  $Q[front[Q]]$

ENQUEUE ( $Q, x$ )

```
1  if Size( $Q$ ) =  $max - 1$ 
2    then error "Queue is full"
3   $Q[rear[Q]] \leftarrow x$ 
4   $rear[Q] \leftarrow (rear[Q] + 1) \bmod max$ 
```

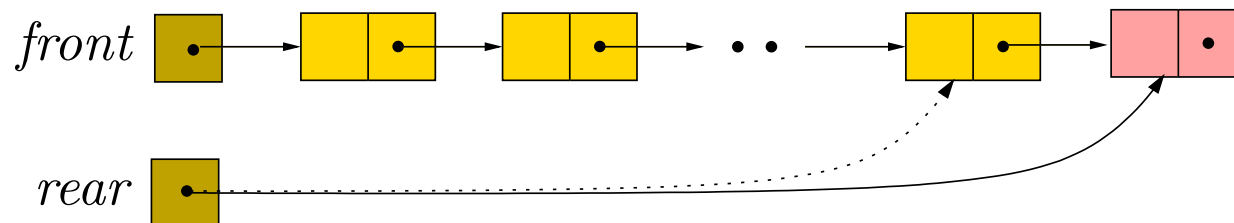
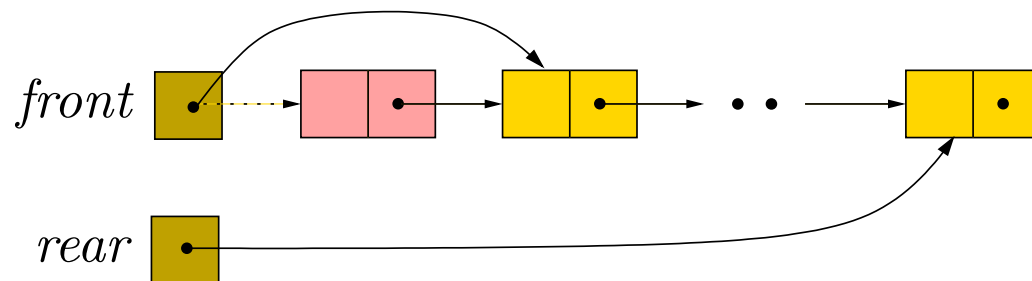
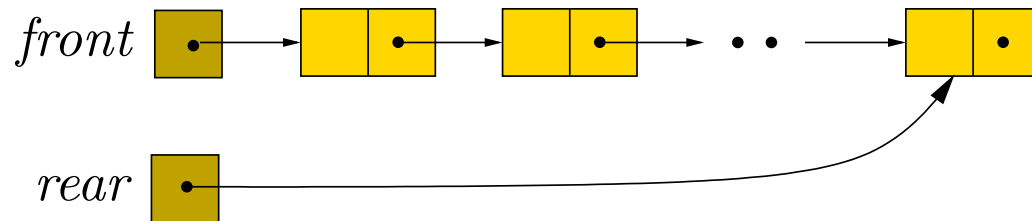
DEQUEUE ( $Q$ )

```
1  if isEmpty()
2    then error "Queue is empty"
3   $temp \leftarrow Q[front[Q]]$ 
4   $front[Q] \leftarrow (front[Q] + 1) \bmod max$ 
5  return  $temp$ 
```

این اعمال همه از  $O(1)$  هستند.

## پیاده‌سازی صف با لیست پیوندی

## داده ساختارها و مبانی الگوریتم‌ها



isEmpty ( $Q$ )

1 **return**  $size[Q] = 0$

ENQUEUE ( $Q, x$ )

1  $next[rear[Q]] \leftarrow \text{ALLOCATE-NODE}(x, \text{null})$

2  $rear[Q] \leftarrow next[rear[Q]]$

3  $size[Q] \leftarrow size[Q] + 1$

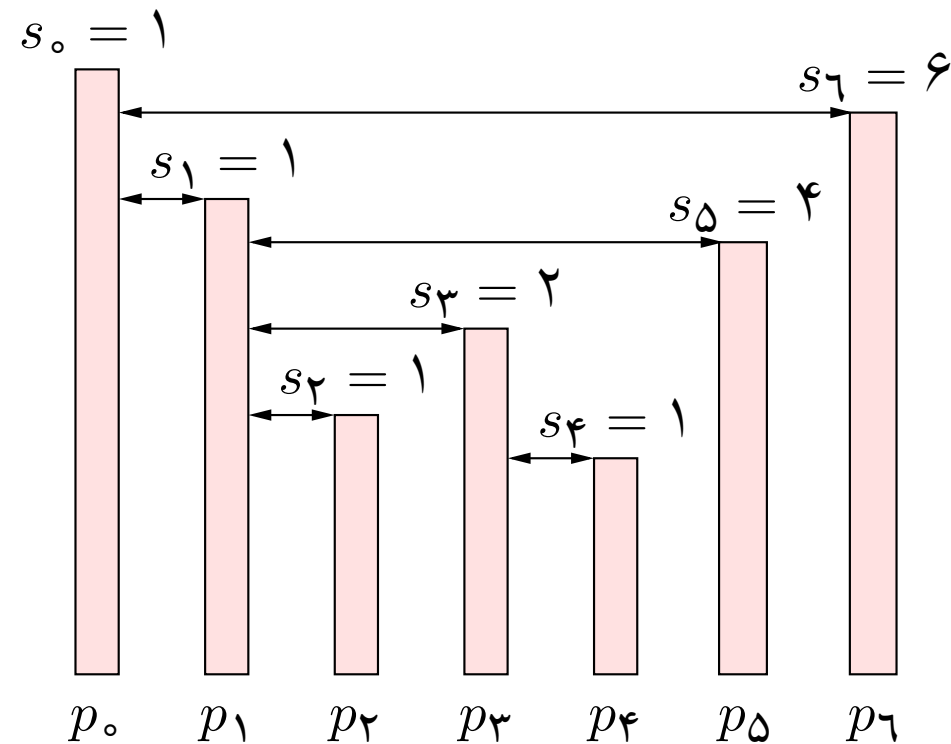


DEQUEUE ( $Q$ )

```
1  if ISEMPTY( $Q$ )  
2    then error 'QUEUE IS EMPTY'  
3   $n \leftarrow front[Q]$   
4   $x \leftarrow element[n]$   
5   $front[Q] \leftarrow next[front[Q]]$   
6  FREE-NODE( $n$ )  
7   $size[Q] \leftarrow size[Q] - 1$   
8  return  $x$ 
```

## کاربردهایی از لیست‌ها

### مسئله‌ی ارزیابی بازار بورس



قیمت روزانه یک سهم در بازار بورس ( $p_i$ ) و «دوره»ی آن در هر روز ( $s_i$ ).

قیمت سهام یک شرکت در روزهای مختلف تهیه می‌شود  
می‌خواهیم در هر روز «دوره‌ی آن سهام» را پیدا کنیم:

اگر قیمت سهام در روز  $i$  برابر  $p_i$  باشد، دوره در روز  $i$  برابر است با تعداد روزهای  
بلافاصله قبل از  $i$  (شامل  $i$ ) که قیمت سهام کم‌تر یا مساوی  $p_i$  باشد.

ورودی: آرایه‌ی  $n$  تایی  $P$

خروجی: آرایه‌ی  $n$  تایی  $S$  که  $S[i]$  «دوره‌ی سهام» در روز  $i$  باشد.

## الگوریتم کند

### COMPUTESPANS ( $P$ )

▷ Input:  $n$ -element array  $P$

▷ Output:  $n$ -element array  $S$

```
1  for  $i \leftarrow 0$  to  $n - 1$ 
2      do  $k \leftarrow 0$ 
3           $done \leftarrow \text{false}$ 
4          repeat if  $P[i - k] \leq P[i]$ 
5              then  $k \leftarrow k + 1$ 
6              else  $done \leftarrow \text{true}$ 
7          until  $k = i$  or  $done$ 
8           $S[i] \leftarrow k + 1$ 
9  return  $S$ 
```

## تحلیل

$$O(n^2)$$

## الگوریتم خطی

### COMPUTESPANS2( $P$ )

▷ we use a stack  $D$

```
1  for  $i \leftarrow 0$  to  $n - 1$ 
2      do  $done \leftarrow \text{false}$ 
3          while ( not  $\text{ISEMPTY}(D)$  or  $done$  )
4              do if  $P[i] \geq P[\text{TOP}(D)]$ 
5                  then  $\text{POP}(D)$ 
6                  else  $done \leftarrow \text{true}$ 
7          if  $\text{ISEMPTY}(D)$ 
8              then  $h \leftarrow -1$ 
9              else  $h \leftarrow \text{TOP}(D)$ 
10          $S[i] \leftarrow i - h$ 
11          $\text{PUSH}(D, i)$ 
12  return  $S$ 
```

## تحلیل

$O(n)$  چون هر عنصر دقیقاً یک بار در پشته درج و حداکثر یک بار از پشته حذف می‌شود.



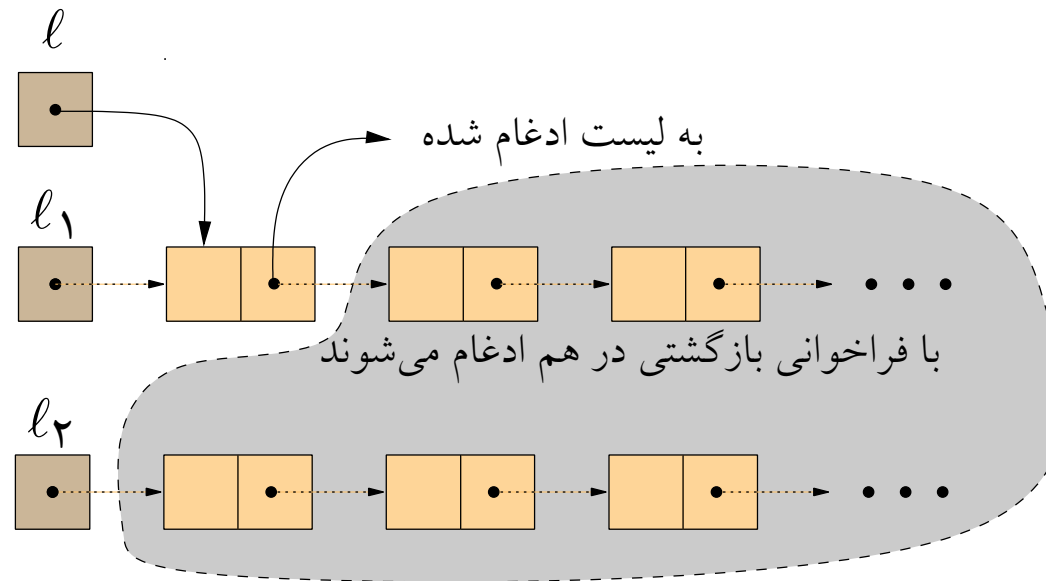
## مرتب‌سازی ادغامی با لیست

### MERGESORT ( $L$ )

▷ یک لیست پیوندی را فقط با تغییر اشاره‌گرها مرتب می‌کند

```
1 if SIZE( $L$ ) > 1
2   then CREATE( $L_1$ ); CREATE( $L_2$ )
3      $first[L_1], first[L_2] \leftarrow$  SPLIT( $first[L]$ )
4     MERGESORT ( $L_1$ )
5     MERGESORT ( $L_2$ )
6      $first[L] \leftarrow$  MERGE( $first[L_1], first[L_2]$ )
7   return  $L$ 
```

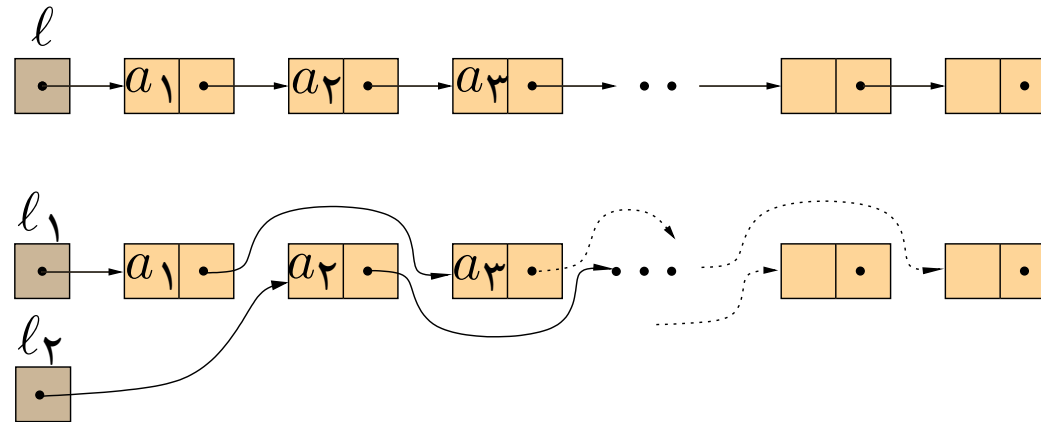
## داده ساختارها و مبانی الگوریتم‌ها



ادغام دو لیست مرتب با عناصر اول  $l_1$  و  $l_2$  و تولید یک لیست مرتب با عنصر اول  $l$ .

MERGE ( $\ell_1, \ell_2$ )

```
1  if  $\ell_1 = \text{null}$ 
2    then return  $\ell_2$ 
3  if  $\ell_2 = \text{null}$ 
4    then return  $\ell_1$ 
5  if  $\text{key}[\ell_1] \leq \text{key}[\ell_2]$ 
6    then  $\text{next}[\ell_1] \leftarrow \text{MERGE}(\text{next}[\ell_1], \ell_2)$ 
7         return  $\ell_1$ 
8  else  $\text{next}[\ell_2] \leftarrow \text{MERGE}(\ell_1, \text{next}[\ell_2])$ 
9         return  $\ell_2$ 
```



تقسیم یک لیست  $n$  عضوی به دو لیست  $\lceil \frac{n}{2} \rceil$  و  $\lfloor \frac{n}{2} \rfloor$  عضوی.

SPLIT ( $\ell$ )

```
1  if  $\ell = \text{null}$ 
2    then return null , null
3  if  $\text{next}[\ell] = \text{null}$ 
4    then return null ,  $\ell$ 
5   $\ell_1 \leftarrow \ell$ 
6   $\ell_2 \leftarrow \text{next}[\ell]$ 
7   $\text{next}[\ell], \text{next}[\ell_2] \leftarrow \text{SPLIT}(\text{next}[\ell_2])$ 
8  return  $\ell_1, \ell_2$ 
```

## لیست‌های کلی

- یک چندجمله‌ای در حالت کلی
- جمع عبارت‌های از نوع  $cx^{e_x}y^{e_y}z^{e_z} \dots$  است که
- ضریب این عبارت و  $e_x, e_y, e_z, \dots$  به ترتیب ضرایب توان متغیرهای  $x, y, z$  و  $\dots$  هستند. مثلاً

$$P = x^1 y^3 z^2 + 2x^4 y^3 z^2 + 3x^4 y^2 z^2 + x^4 y^4 z + 6x^3 y^4 z + 6x^3 y^4 z + 2yz \quad (1)$$

هدف طراحی داده‌ساختار مناسب با اعمال زیر:

- چاپ عبارت
- تعیین بیش‌ترین عمق آن
- کپی کردن یک عبارت
- جمع یا تفریق دو عبارت
- مشتق‌گیری از عبارت بر حسب یکی از متغیرها

روش اول: یک لیست با عناصر زیر

<i>coef</i>	<i>expx</i>	<i>expy</i>
<i>expz</i>	<i>link</i>	



## لیست کلی با ساختار بازگشتی

اگر  $P(z, y, x, n_z, n_y, n_x)$  یک چند جمله‌ای بر حسب  $z, y, x, n_z, n_y, n_x$

• متغیرها به ترتیب  $z, y$  و  $x$  و

• درجه‌ی آن‌ها به ترتیب برابر  $n_z, n_y$  و  $n_x$  باشند،

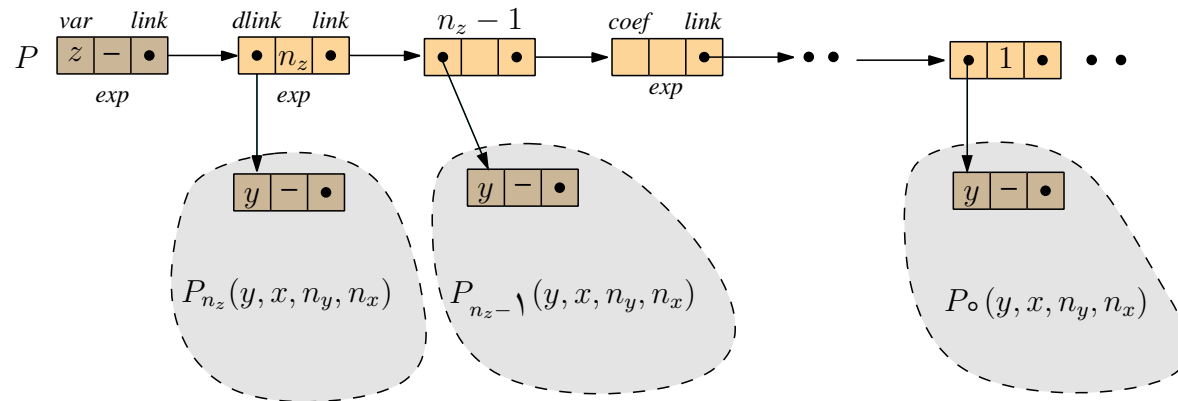
آن‌را به صورت زیر تعریف می‌کنیم:

$$P(z, y, x, n_z, n_y, n_x) = \\ c_{n_z}(P_{n_z}(y, x, n_y, n_x)z^{n_z} + c_{n_z-1}(P_{n_z-1}(y, x, n_y, n_x)z^{n_z-1} + \\ \dots + c_0(P_0(y, x, n_y, n_x)))$$

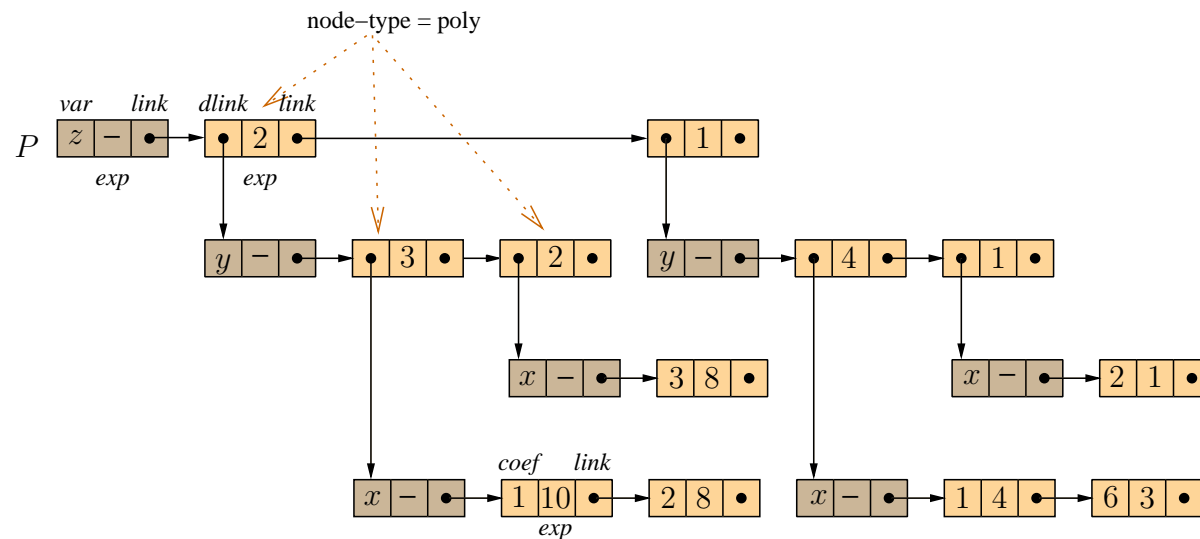
مثال

$$(((x^1 + 2x^8)y^3 + 3x^8y^2)z^2 + ((x^4 + 6x^3)y^4 + 2y)z)$$

## داده ساختارها و مبانی الگوریتم‌ها



$$P(z, y, x, n_z, n_y, n_x) = P_{n_z}(y, x, n_y, n_x)z^{n_z} + P_{n_z-1}(y, x, n_y, n_x)z^{n_z-1} + \dots + P_0(y, x, n_y, n_x)$$



$$((x^1 + 2x^8)y^3 + 3x^8y^2)z^2 + ((x^4 + 6x^3)y^4 + 2y)z$$

## اعمال

### PRINT-PLIST ( $P$ )

▷ فرض می‌کنیم که لیست درست ساخته شده است

```

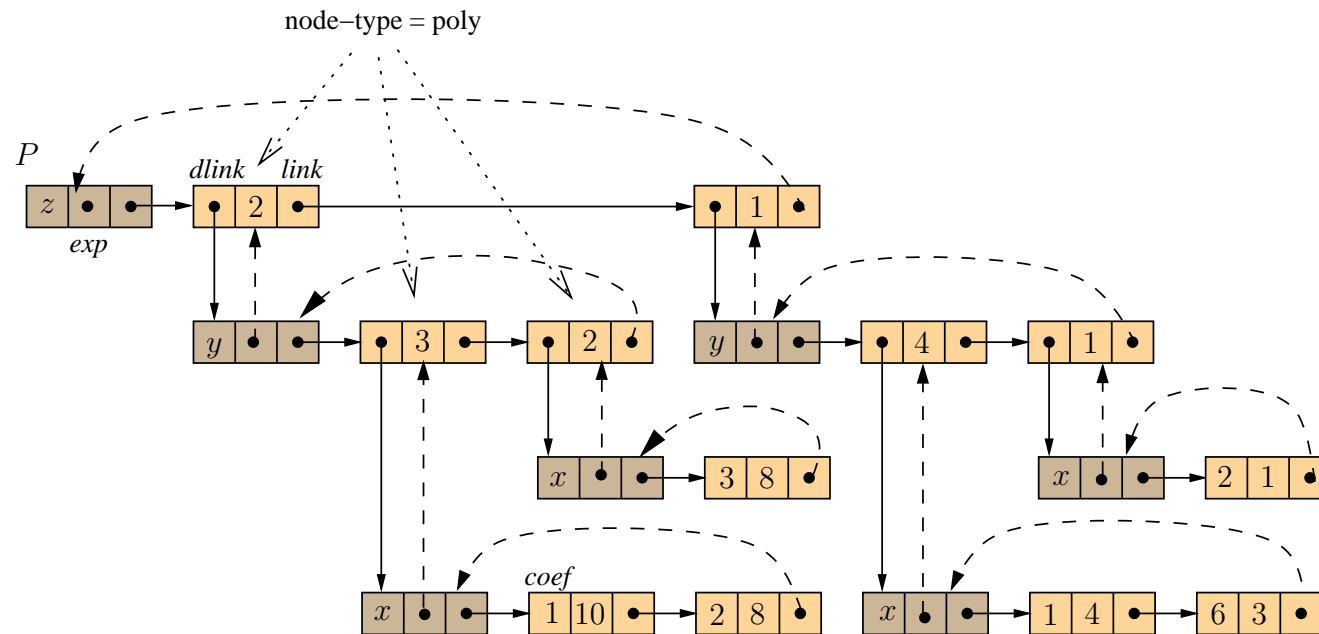
1   $p \leftarrow P$ 
2   $X = var[p]$  ▷  $P$  چند جمله‌ای
3  while  $p \neq \text{null}$ 
4      do if  $\text{NODE-TYPE}(p) = \text{poly}$ 
5          then  $\text{PRINT '('}$ 
6               $\text{PRINT-PLIST}(dlink[p])$ 
7               $\text{PRINT ')} ^{exp[p]}$ 
8          else
9              ▷ با فرض درستی داده‌گونه‌ها
10              $\text{PRINT '+coef}[p] X^{exp[p]}$ 
11              $p \leftarrow link[p]$ 
    
```

### DEPTH-PLIST ( $P$ )

▷ فرض می‌کنیم که لیست درست ساخته شده است

```
1  $p \leftarrow P$ 
2  $depth \leftarrow 0$ 
3 while  $p \neq \text{null}$ 
4     do if  $\text{NODE-TYPE}(p) = \text{poly}$ 
5         then  $dp \leftarrow \text{DEPTH-PLIST}(dlink[p]) + 1$ 
6              $depth \leftarrow \max\{depth, dp\}$ 
7          $p \leftarrow link[p]$ 
8 return DEPTH
```

## نخ (thread) و اشاره‌گر



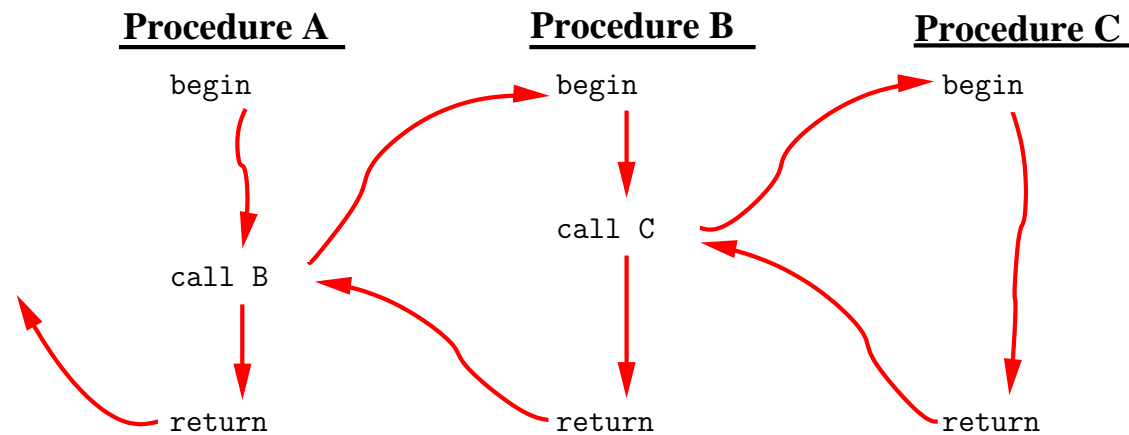
### DEPTH-THREADED-PLIST ( $P$ )

```

1   $p \leftarrow P$ 
2   $depth \leftarrow 0$ 
3   $maxdepth \leftarrow 0$ 
4  while true
5      do if  $Node-Type[p] = \text{poly}$ 
6          then  $depth \leftarrow depth + 1$ 
7               $p \leftarrow dlink[p]$ 
8          else while  $link[p]$  is a thread
9              do  $p \leftarrow link[p]$ 
10              $maxdepth = \max\{maxdepth, depth\}$ 
11             if  $exp[p] \neq \text{null}$ 
12                 then  $p \leftarrow exp[p]$ 
13                      $depth \leftarrow depth - 1$ 
14             else return  $maxdepth$ 
15     if  $link[p]$  is pointer
16         then  $p \leftarrow link[p]$ 
    
```



## تبدیل الگوریتم‌های بازگشتی به غیربازگشتی



انتقال کنترل برنامه در فراخوانی و بازگشت

## مراحل

(۱) عمل فراخوانی

(۲) بازگشت از یک فراخوانی

## هر فراخوانی (Call)

- ۱) ذخیره‌ی کلیه‌ی متغیرهای محلی (در حالت کلی کلیه متغیرهای دسترس‌پذیر) و مقدارهایشان در پشته‌ی سیستم (Push).
- ۲) آدرس بازگشت به پشته منتقل می‌شود (Push).
- ۳) عمل انتقال پارامترها (Parameter Passing) صورت می‌گیرد. پارامترها ممکن است از نوع ارزشی (Val) یا آدرسی (Variable) باشند.
- ۴) کنترل برنامه (ثبات شمارنده‌ی برنامه، Program Counter) به ابتدای رویه‌ی جدید اشاره می‌کند.

## عمل بازگشت (Return)

### عکس عملیات فوق

۱) مقدارهای متغیرهای محلی را از رکورد بالای پشته برداشته و در خودشان قرار می‌دهیم.

۲) آدرس بازگشت را از بالای پشته به دست می‌آوریم.

۳) آخرین رکورد را از پشته برمی‌داریم (Pop).

۴) کنترل برنامه را از آدرس بازگشت (بند ۲) ادامه می‌دهیم.

## مثال

HONOI ( $n, f, t, h$ )

▷ moving  $n$  coins from leg  $f$  to leg  $t$  with the help of leg  $h$

1 **if**  $n = 1$

2     **then** Move the coin from leg  $f$  to leg  $t$

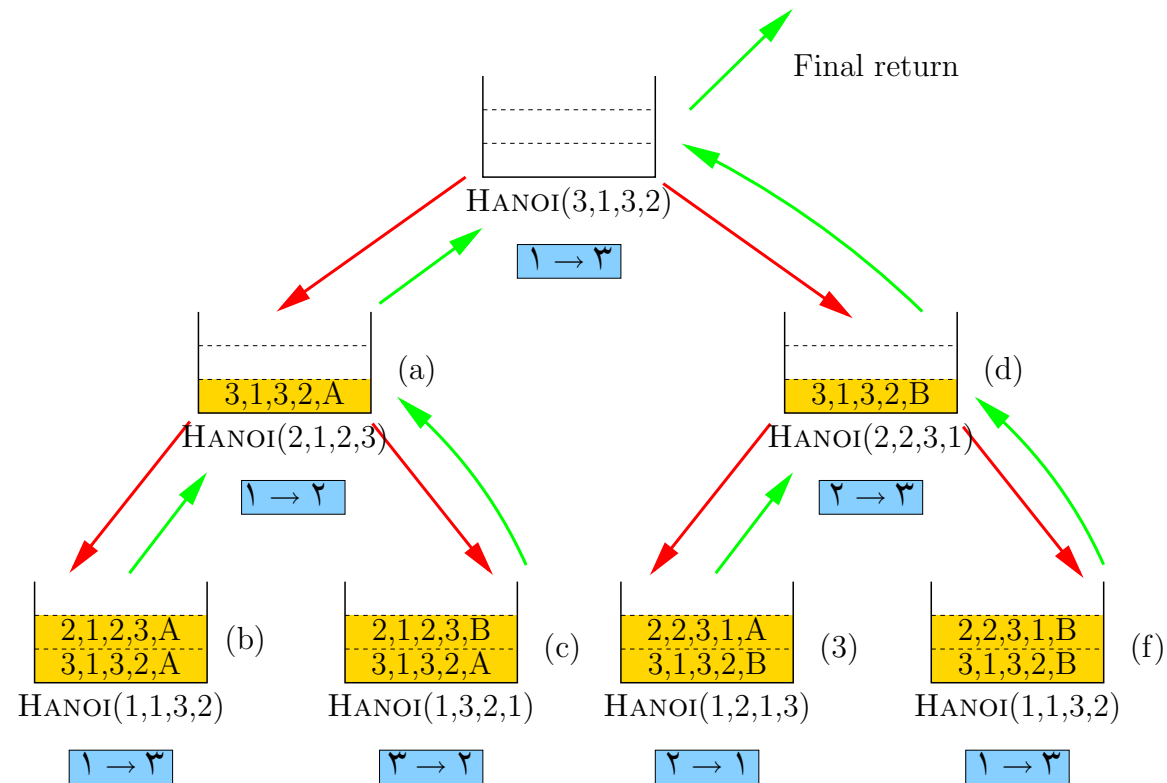
3     **else**  $\text{HONOI}(n - 1, f, h, t)$

4         A: Move the coin from leg  $f$  to leg  $t$

5          $\text{HONOI}(n - 1, h, t, f)$

6         B:

## داده ساختارها و مبانی الگوریتم‌ها



مراحل مختلف فراخوانی‌های بازگشتی و مقادیر پشت‌بندی سیستم در  $HANOI(3,1,3,2)$ .

### NONRECURSIVE-HONOI ( $n, f, t$ )

```

1  CREATE-STACK( $S$ ) ▷ پشته‌ی  $S$  شامل آدرس بازگشت و مقادیر همه‌ی متغیرهای محلی است
2   $h \leftarrow$  the other peg
3  Rec-Call: ▷ (آغاز یک فراخوانی بازگشتی) if  $n = 1$ 
4      then Print  $f \rightarrow t$  ▷ سکه‌ی بالایی را از میله‌ی  $f$  به میله‌ی  $t$  منتقل کن
5          goto Return-Label
6      else PUSH ( $S$ , STACKREC( $n, f, t, h, 'A'$ )))
7           $n, f, t, h \leftarrow n - 1, f, h, t$  ▷ انتقال پارامترها با فرض ارزشی بودن
8          goto Rec-Call
9  Return-Label: if not ISEEMPTY( $S$ )
10     then ▷ از این دستور عمل بازگشت شبیه‌سازی می‌شود
11          $n, f, t, h$ , return-addr  $\leftarrow$  POP( $S$ )
12     switch
13         case return-addr = 'A'
14             do Print  $f \rightarrow t$ ; PUSH( $S$ , SREC( $n, f, t, h$ , 'B'))
15                  $n, f, t, h \leftarrow n - 1, h, t, f$  ▷ انتقال پارامترها
16                 goto Rec-Call
17         case return-addr = 'B'
18             do goto Return-Label
    
```

## حذف آخرین بازگشت (Tail Recursion)

آخرین فراخوانی بازگشتی که بعد از آن در هیچ شرایطی دستوری که از مقدارهای متغیرها استفاده کند، اجرا نشود را آخرین بازگشت می‌گوییم.



این بازگشت را می‌توان بدون استفاده از پشته حذف کرد.

```
RECURSIVEPROC(...)
```

```
...
```

```
...
```

```
A: RECURSIVEPROC(...)    ▷ this is the last line
```

```
x:
```

در بازگشت از این فراخوانی (A) متغیرهای محلی مقدارهایشان تغییر می‌کند و اجرای برنامه از نقطه‌ی (x) دنبال می‌شود. ولی (x) تنها یک بازگشت است.

## مثال

HONOI ( $n, f, t, h$ )

▷ moving  $n$  coins from leg  $f$  to leg  $t$  with the help of leg  $h$

1 **if**  $n = 1$

2     **then** Move the coin from leg  $f$  to leg  $t$

3     **else** HONOI( $n - 1, f, h, t$ )

4         A: Move the coin from leg  $f$  to leg  $t$

5         HONOI( $n - 1, f, h, t$ )

6         B:

## حذف آخرین فراخوان

TOWER-OF-HONOI2 ( $n, f, t, h$ )

▷ eliminating the last recursion

```
1  if  $n = 1$ 
2    then Move the coin from leg  $f$  to leg  $t$ 
3    else TOWER-OF-HONOI( $n - 1, f, h, t$ )
4          Move the coin from leg  $f$  to leg  $t$ 
5           $n, f, h \leftarrow n - 1, h, f$     ▷ parameter passing
6          goto 1
```

### NONRECURSIVE-HONOI2 ( $n, f, t$ )

```

1   $h \leftarrow$  the other peg
2  ▷ make recursive call
3  if  $n = 1$ 
4    then
5      Move the top coin from leg  $f$  to leg  $t$ 
6      goto 10
7    else  $\text{PUSH}(S, \text{STACKREC}(n, f, t, h))$ 
8       $n, f, t, h \leftarrow n - 1, f, h, t$     ▷ parameter passing
9      goto 3
10 ▷ end recursive call
11 if not  $\text{ISEMPTY}(S)$ 
12   then  $n, f, t, h \leftarrow \text{POP}(S)$ 
13     Move the top coin from leg  $f$  to leg  $t$ 
14      $n, f, t, h \leftarrow n - 1, h, t, f$     ▷ parameter passing
15     goto 3
    
```