داده‌ساختارهای ساده

- لیست‌ها
  - -- لیست‌ها یک‌طرفه، دوطرفه دوار
  - -- پشته
  - -- صف
  - -- لیست‌های کلی
  - -- کاربردهای لیست‌ها

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

لیست‌ها

دنباله‌ای از عناصر، که ترتیب آن‌ها مهم است

اعمال

- 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 لیست.

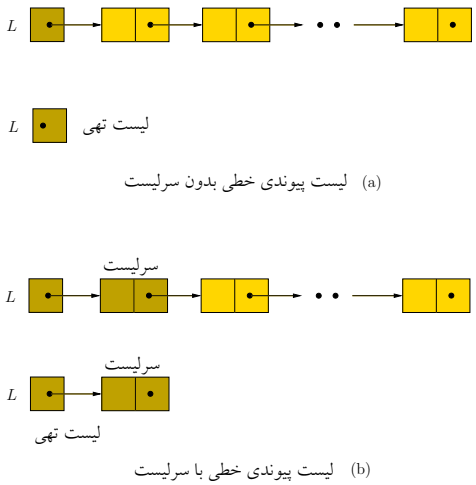
لیست‌ها (ادامه)

بسته به مکان درج یا حذف یک عنصر، لیست به اسامی زیر شناخته می‌شود:

- پُشته (stack): درج و حذف فقط در یک طرف لیست
  Last-In-First-Out (FILO) یا First-In-Last-Out (FILO)
- صف (queue): درج فقط در انتها و حذف از ابتدای
  First-In-First-Out (FIFO).

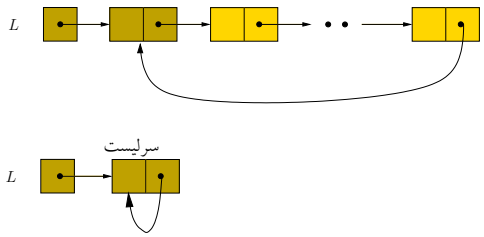
لیست‌های پیوندی یک‌طرفه

```
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;}
}
```

(a) لیست پیوندی خطی بدون سرلیست

(b) لیست پیوندی خطی با سرلیست



لیست دوار با سرلیست.

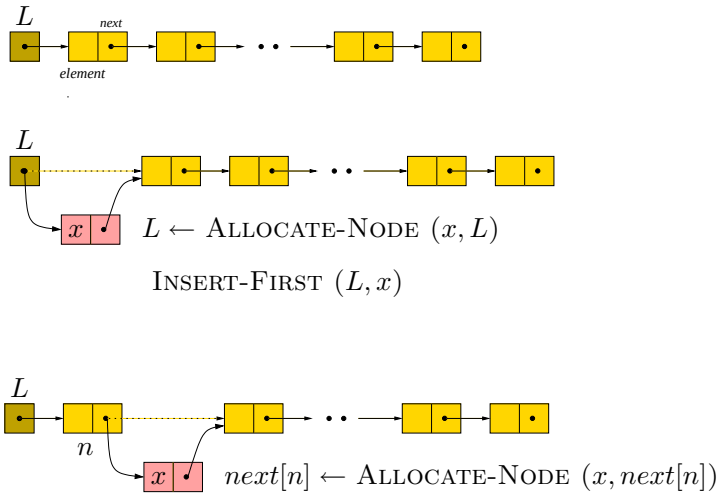
پیاده‌سازی با CLRS

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

C++	جاوا (Java)	شبه‌کد CLRS
	`x = new Node()`	$x \leftarrow$ ALLOCATE-NODE()
	`x = new Node(element, next)`	$x \leftarrow$ ALLOCATE-NODE(e, n)
	`null x`	FREE-MODE(x)
	`x.getNext()`	$next[x]$
	`x.setNext(n)`	$next[x] \leftarrow n$

اعمال اصلی بر روی لیست خطی

- CREATE-LIST(L)
- SIZE(L): **تعداد عناصر لیست را بر می‌گرداند.**
- FIRST(L): **عنصر اول را برمی‌گرداند.**
- ISEMPTY(L): **مشخص می‌کند که آیا لیست خالی.**
- INSERT-FIRST(L, x): **یک عنصر با مقدار x را در ابتدای لیست درج می‌کند.**
- INSERT-AFTER(L, x, n): **یک عنصر با مقدار x را پس از عنصر n در L درج می‌کند.**
- DELETE-FIRST(L): **عنصر اول لیست L را حذف می‌کند.**
- DELETE-AFTER(L, n): **عنصر پس از عنصر n در L را حذف می‌کند.**



پیاده‌سازی

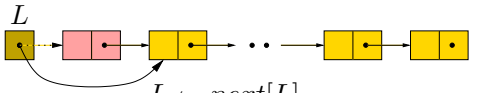
CREATE(L)
1 $size[L] \leftarrow 0$
2 **return null**

IZE(L)
1 **return** $sizeS]$

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

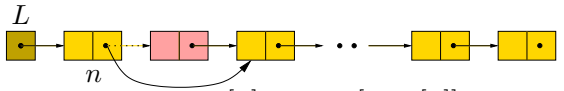
NSERT-FIRST(L, x)
1 $L \leftarrow$ ALLOCATE-NODE (x, L)
2 $size[L] \leftarrow size[L] + 1$

NSERT-AFTER(L, x, n)
1 $next[n] \leftarrow$ Allocate-Node $(x, next[n])$
2 $size[L] \leftarrow size[L] + 1$



$L \leftarrow next[L]$

DELETE-FIRST(L)



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

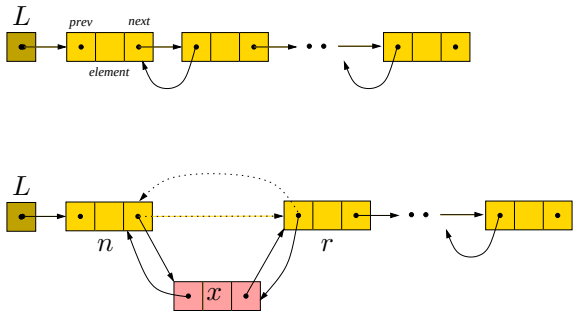
DELETE-NEXT(L, n)

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

DELETE-FIRST(L)
1 **if** ISEMPTY (L)
2 **then error** LIST IS EMPTY
3 $n \leftarrow L$
4 $L \leftarrow next[L]$
5 FREE-NODE(n)
6 $size[L] \leftarrow size[L] - 1$

DELETE-AFTER(L, n)
1 **if** ISEMPTY (L) **or** $n =$ **null**
2 **then error** LIST IS EMPTY
3 $r \leftarrow next[n]$
4 $next[n] \leftarrow next[r]$
5 FREE-NODE(r)
6 $size[L] \leftarrow size[L] - 1$

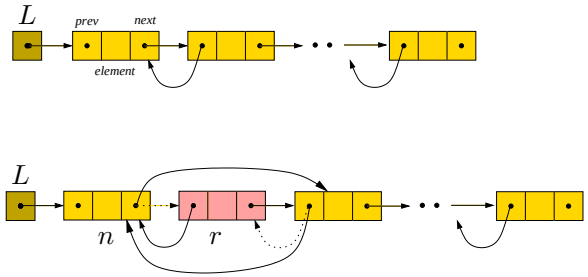
درج و حذف در لیست دوطرفه‌ی خطی



INSERT-AFTER(L, x, n)

INSERT-AFTER(L, x, n)

1 $r \leftarrow next[n]$
2 $next[n] \leftarrow$ Allocate-Node (x, n, r)
3 $prev[r] \leftarrow next[n]$
4 $size[L] \leftarrow size[L] + 1$



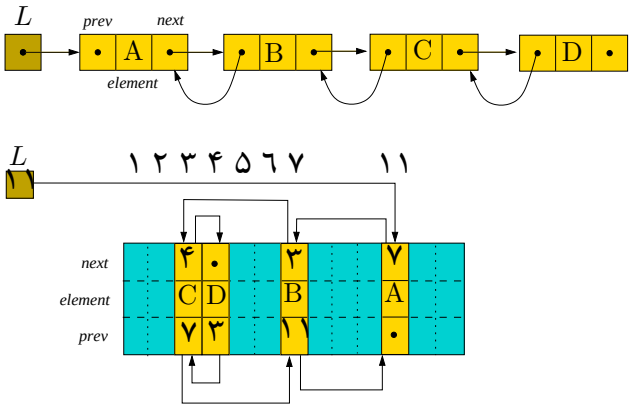
DELETE-AFTER(L, n)

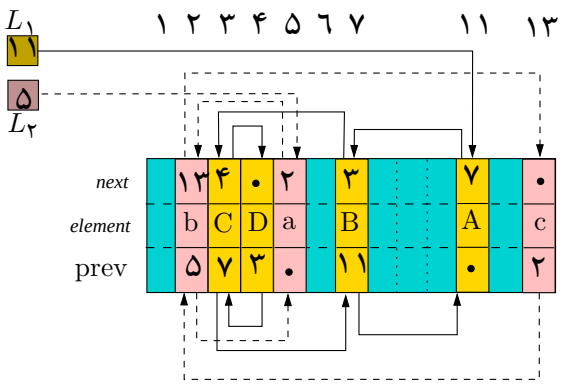
DELETE-AFTER(L, n)

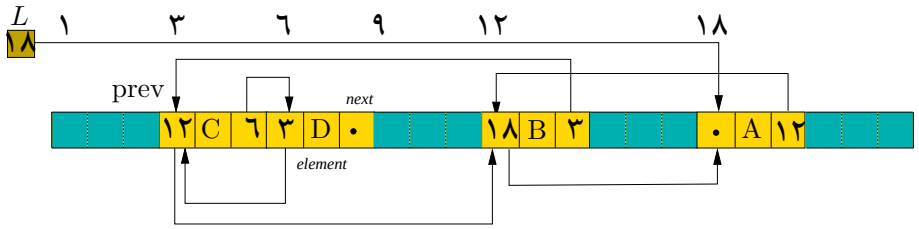
1 **if** ISEMPTY (L) **or** $n =$ **null** **or** $next[n] =$ **null**
2 **then error** THE ELEMENT DOES NOT EXIT
3 $r \leftarrow next[n]$
4 **if** $next[r] \neq$ **null**
5 **then** $prev[next[r]] \leftarrow n$
6 $next[n] \leftarrow next[r]$
7 FREE-NODE(r)
8 $size[L] \leftarrow size[L] - 1$

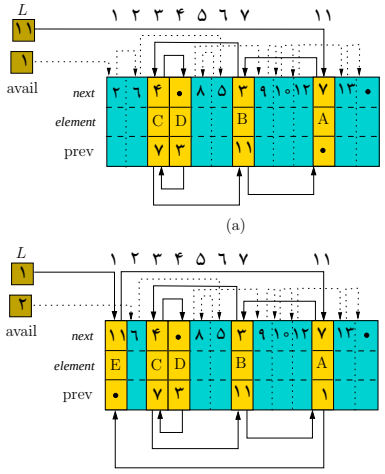
پیاده‌سازی لیست‌ها با اشاره‌گرهای اندیسی

یا زبان فرترن لیست‌ها را چه‌گونه پیاده‌سازی می‌کنیم؟

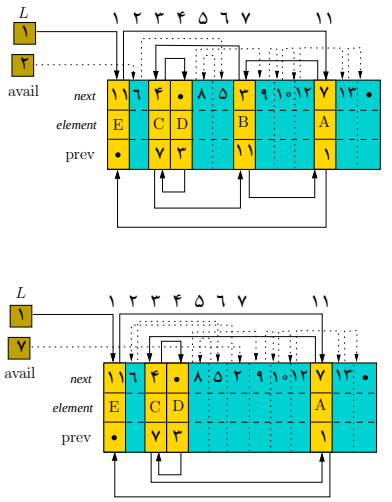








After inserting E at the beginning of L

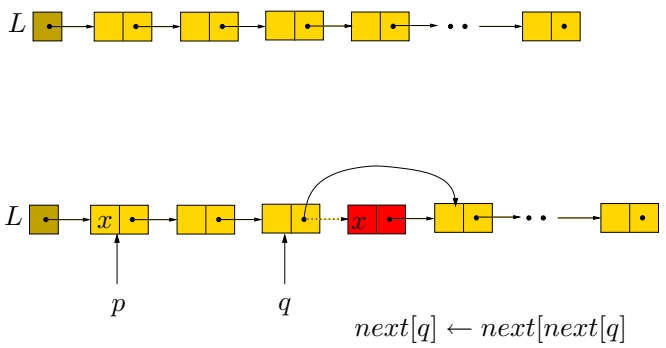


After deleting 3rd element from list (b)

زباله‌روبی (Garbage Collection)


```
INITIALIZE()
1  null ← 0
2  avail ← 1
3  for i ← 1 to M − 1
4       do next[i] ← i + 1
5  next[M] ← null

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
```


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



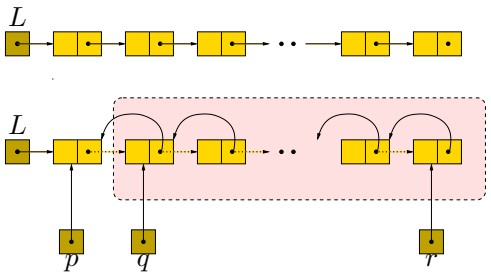
PURGELIST(L)

▷ removes all identical elements but one

```
1  p ← FIRST (L)
2  while p ≠ null
3        do q ← p
4           while next[q] ≠ null
5                 do if element[p] = element[next[q]]
6                       then DELETE-AFTER(L, q)
7                       else   q ← next[q]
8           p ← next[p]
```

آیا می‌توان این کار را در $O(n \lg n)$ حل کرد؟

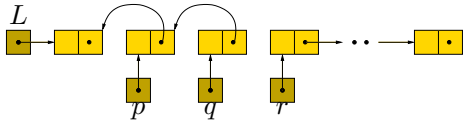
وارون کردن یک لیست تنها با تغییر اشاره‌گرها



وارون کردن یک لیست به‌صورت بازگشتی.

RECURSIVE-REVERSE(L)

```
1  if L = null
2     then return null
3  p ← L
4  q ← next[p]
5  r ← RECURSIVE-REVERSE (q)
6  next[q] ← p
7  next[p] ← null
8  size[r] ← size[L]
9  return r
```

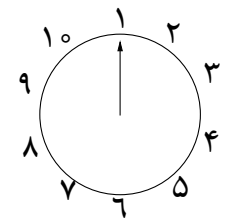



وارون کردن یک لیست به‌صورت غیر بازگشتی.

```
NR-Reverse(L)
 1  if L = null
 2     then return null
 3  p ← null
 4  q ← L
 5  r ← next[q]
 6  while q ≠ null
 7        do next[q] ← p
 8           p ← q
 9           q ← r
10           r ← next[r]
11  size[p] ← size[L]
12  return p
```


مسئله‌ی ژوزفوس

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



مسئله‌ی ژوزفوس با ۱۰ نفر.

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

**جواب این مسئله $J(n)$ به‌صورت ریاضی قابل محاسبه است و می‌توان جواب را ا
رابطه‌ی بازگشتی زیر به‌دست آورد.**

$$\begin{aligned} J(۱) &= ۱ \\ J(۲n) &= ۲J(n) - ۱, \text{ for } n \geq ۱, \\ J(۲n+۱) &= ۲J(n) + ۱ \quad \text{for } n \geq ۱. \end{aligned}$$

**اگر n را به‌صورت عدد دودویی بنویسیم و آن‌را یک بیت شیفت چپ دورانی دهیم $J(n)$
به‌دست می‌آید.**

مثلاً برای $n = ۱۰۰ = (۱۱۰۰۱۰)_۲$، جواب $۷۳ = (۱۰۰۱۰۱)_۲ = J(n)$ است.

حل مسئله‌ی ژوزفوس با لیست پیوندی دوار

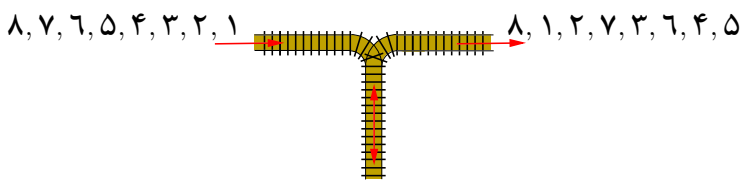
حل آن‌را بنویسید.

حل مسئله‌ی ژوزفوس با لیست پیوندی دوار

```
OESEPHOUS(N)
    ▷ create a cicular link list with N nodes
 1  L ← ALLOCATE-NODE(1, null )
 2  p ← L
 3  for i ← 2 to N
 4      do next[p] ← ALLOCATE-NODE(i, null )
 5         p ← next[p]
 6  next[p] ← L
    ▷ NOW THE SOLUTION
 7  p ← L
 8  while next[p] ≠ p
 9        do DELETE-AFTER( L, p)
10           p ← next[p]
11  return element[p]
```

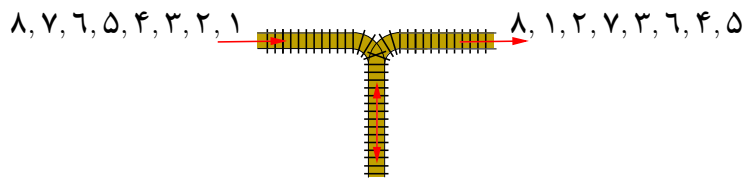

پشته‌ها

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



پشته‌ی قطارها.

$< ۸, ۱, ۲, ۷, ۳, ۶, ۴, ۵ >$ قابل تولید است. چه‌طور؟



پشته‌ی قطارها.

$< ۸, ۱, ۲, ۷, ۳, ٦, ۴, ٥ >$ قابل تولید است. چه‌طور؟

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

$< ۱, ۸, ۳, ٦, ۲, ۷, ۴, ٥ >$ چه‌طور؟

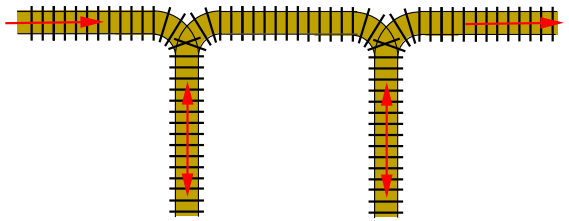
چند مسئله

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.

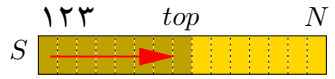
۴) مسئله‌ی اصلی را برای سیستم دو پشته‌ای مطابق شکل صفحه‌ی بعد را حل کنید.



پیاده‌سازی پشته با آرایه

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

```
ublic 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** $S[top[S]]$

پیاده‌سازی چند پشته با آرایه.

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

تحلیل

همه‌ی اعمال در $O(۱)$ 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]
```



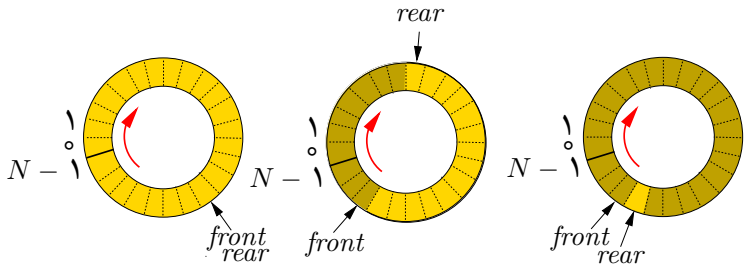
```
PUSH(S, x)
  1 top[S] ← ALLOCATE-NODE (x, top[S])
  2 size[S] ← size[S] + 1

POP(S)
  1 if ISEMPTY(S)
  2    then error ("STACK IS EMPTY")
  3 n ← top[S]
  4 temp ← element[n]
  5 top[S] ← next[n]
  6 size[S] ← size[S] − 1
  7 FREE-OBJECT(n)
  8 return temp
```


صف

درج در انتها و حذف در ابتدای لیست

- **ENQUEUE(Q, x): درج یک عنصر در انتهای صف**
  ورودی: عنصر (شیئ)، خروجی: هیچ
- **DEQUEUE(Q): حذف عنصر از ابتدای صف**
  ورودی: هیچ، خروجی: عنصر (شیئ)، خطا: اگر خالی باشد
- **SIZE(Q): تعداد عناصر موجود در صف**
  ورودی: هیچ، خروجی: یک عدد صحیح
- **ISEMPTY(Q): مشخص می‌کند که آیا صف خالی است**
  ورودی: هیچ، خروجی: درست یا نادرست
- **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 ۰ تا $max - ۱$
- عناصر به‌صورت دوار و در جهت ساعت‌گرد ذخیره می‌شوند.
- $Q[(i + ۱) \bmod max]$ عنصر بعدیِ $Q[i]$ است.
- مولفه‌ی $front[Q]$ اندیس عنصر ابتدایی صف
- $rear[Q]$ اندیس عنصر بعدی آخرین عنصر صف.
- بنابراین حداکثر تعداد عناصر $max - ۱$ است.
- می‌خواهیم دو حالت «کاملاً پر» و «کاملاً خالی» را بتوانیم از هم تمیز دهیم.

حالت‌های مختلف صف

- در شروع $front[Q] = rear[Q] = ۰$
- 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 $(max - front[Q] + rear[Q]) \bmod max = max - ۱$.


```
SIZE(Q)
1  return  (max − front[Q] + rear[Q]) mod 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)

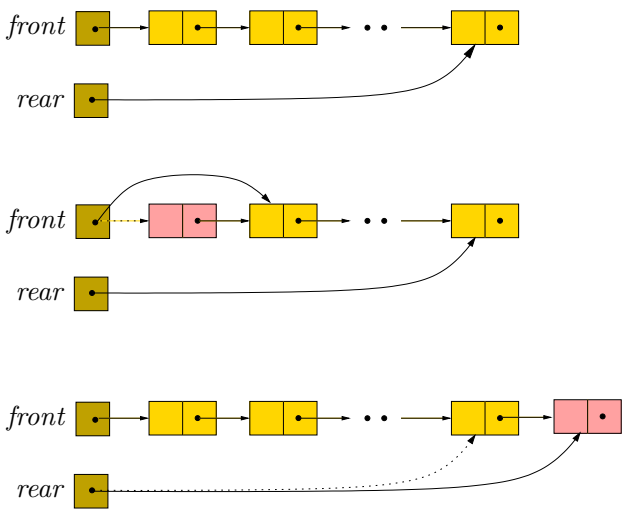
```
1  if Size(Q) = max − 1
2     then  error "Queue is full"
3  Q[rear[Q]] ← x
4  rear[Q] ← (rear[Q] + 1) mod max
```

DEQUEUE(Q)

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


این اعمال همه از $O(1)$ هستند.

پیاده‌سازی صف با لیست پیوندی



```
ISEMPTY(Q)
  1  return size[Q] = 0
```

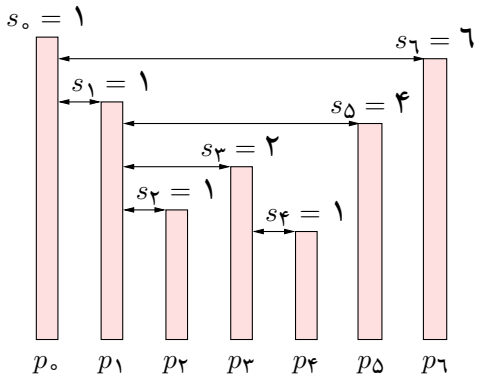
```
ENQUEUE(Q, x)
  1  next[rear[Q]] ← ALLOCATE-NODE (x, null )
  2  rear[Q] ← next[rear[Q]]
  3  size[Q] ← size[Q] + 1
```



```
DEQUEUE(Q)
  1  if ISEMPTY(Q)
  2     then  error 'QUEUE IS EMPTY'
  3  n ← front[Q]
  4  x ← element[n]
  5  front[Q] ← next[front[Q]]
  6  FREE-NODE(n)
  7  size[Q] ← 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 ← 0 to n − 1
 2      do k ← 0
 3         done ← false
 4         repeat if P[i − k] ≤ P[i]
 5                  then k ← k + 1
 6                  else done ← true
 7         until k = i or done
 8         S[i] ← k + 1
 9  return S
```


تحلیل

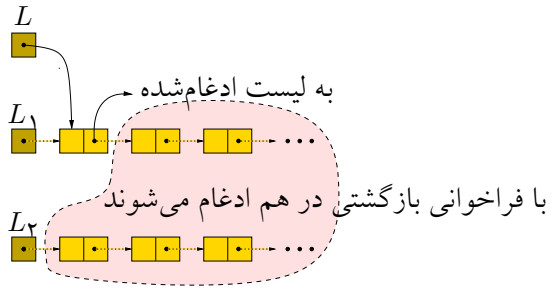
$O(n^2)$

الگوریتم خطی

```
ComputeSpans2(P)
    ▷ we use a stack D
 1  for i ← 0 to n − 1
 2      do done ← false
 3         while ( not isEmpty(D) or done)
 4                do if P[i] ≥ P[Top(D)]
 5                      then Pop(D)
 6                      else done ← true
 7         if isEmpty(D)
 8           then h ← −1
 9           else h ← Top(D)
10         S[i] ← i − h
11         Push(D, i)
12  return S
```

تحلیل

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



با فراخوانی بازگشتی در هم ادغام می‌شوند

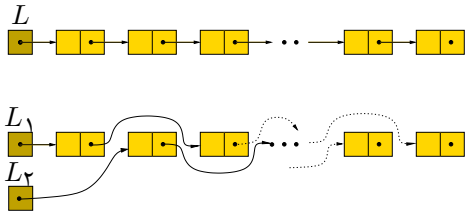
ادغام دو لیست مرتب L_1 و L_2 و تولید لیست مرتب L از عناصر آن‌ها.

مرتب‌سازی ادغامی با لیست

```
MergeSort(L)
    ▷ Will sort a link list L by only
    changing the pointes
 1  if Size(L) > 1
 2    then L1, L2 ← Split(L)
 3         MergeSort (L1)
 4         MergeSort (L2)
 5         return Merge(L1, L2)
```



```
Merge(L1, L2)
 1  if isEmpty(L1)
 2    then return L2
 3  if isEmpty(L2)
 4    then return L1
 5  if element[L1] ≤ element[L2]
 6    then next[L1] ← Merge(next[L1], L2)
 7         size[L1] ← size[L1] + size[L2]
 8         return L1
 9    else next[L2] ← Merge(L1, next[L2])
10         size[L2] ← size[L1] + size[L2]
11         return L2
```



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


```
SPLIT(L)
1  if ISEMPTY(L)  or SIZE(L) = 1
2     then return null
3  L1 ← L
4  L2 ← next[L]
5  next[L1], next[L2] ← SPLIT(next[L2])
6  size[L1] ← ⌈n/2⌉
7  size[L2] ← ⌊n/2⌋
8  return L1, L2
```


لیست‌های کلی

- **یک چندجمله‌ای در حالت کلی**
- **جمع عبارت‌های از نوع** $cx^{e_x}y^{e_y}z^{e_z}\ldots$ **است که**
- c **ضریب این عبارت و** e_x، e_y، e_z، ... **به‌ترتیب ضرایب توان متغیرهای** x، y، z **و ... هستند. مثلاً**

$$P = x^{۱۰}y^{۳}z^{۲} + ۲x^{۸}y^{۳}z^{۲} + ۳x^{۸}y^{۲}z^{۲} + x^{۴}y^{۴}z + ۶x^{۳}y^{۴}z + ۶x^{۳}y^{۴}z + ۲yz \quad (۱)$$

هدف طراحی داده‌ساختار مناسب با اعمال زیر:

-- چاپ عبارت

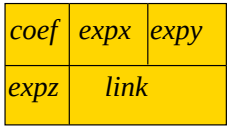
-- تعیین بیش‌ترین عمق آن

-- کپی کردن یک عبارت

-- جمع یا تفریق دو عبارت

-- مشتق‌گیری از عبارت برحسب یکی از متغیرها

روش اول: یک لیست با عناصر زیر



لیست کلی با ساختار بازگشتی

اگر $P(z,y,x,n_z,n_y,n_x)$ یک چند جمله‌ای برحسب z,y,x,n_z,n_y,n_x،

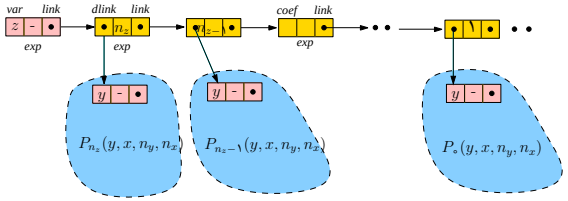
- متغیرها به‌ترتیب z، y و x و
- درجه‌ی آن‌ها به‌ترتیب برابر n_z، n_y و n_x باشند،

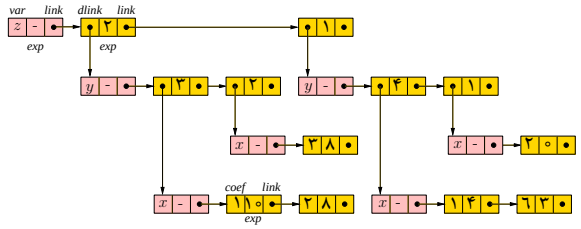
آن‌را به‌صورت زیر تعریف می‌کنیم:

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

مثال

$$((x^{10}+2x^8)y^3+3x^8y^2)z^2+((x^4+6x^3)y^4+2y)z$$





$((x^{10}+2x^8)y^3+3x^8y^2)z^2+((x^4+6x^3)y^4+2y)z$.

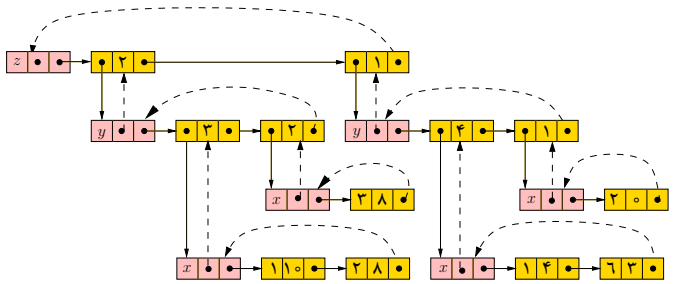
اعمال

```
PRINT-PLIST(P)
 1  p ← P
 2  X = var[p]
 3  while p ≠ null
 4        do if NODE-TYPE(p) = poly
 5              then PRINT 'coef[p]+ ('
 6                   PRINT-PLIST (dlink[p])
 7                   PRINT ') +X^exp[p],'
 8              else
 9                   ▷ it is of term type
10                   Print '+coef[p] X^exp[p],'
11           p ← link[p]
```

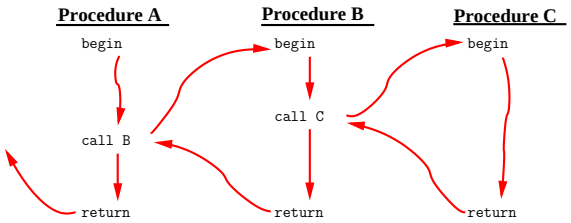


```
DEPTH-PLIST(P)
 1  p ← P
 2  depth ← 0
 3  while p ≠ null
 4        do if NODE-TYPE(p) = poly
 5              then depth ← max{depth, DEPTH-PLIST (dlink[p]) + 1}
 6           p ← link[p]
 7  return depth
```


نخ (thread) و اشاره‌گر



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 می شود (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).

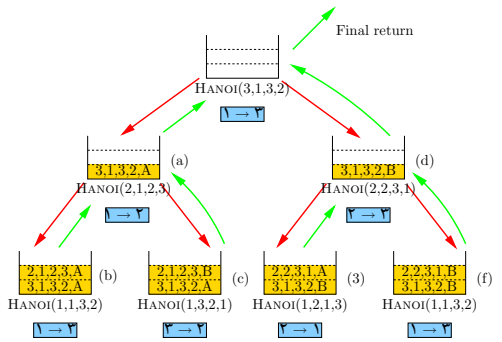
۴) کنترل برنامه را از آدرس بازگشت (بند ۲) ادامه می‌دهیم.

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 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:
```



مراحل مختلف فراخوانی‌های بازگشتی و مقادیر پشته‌ی سیستم در Hanoi(3,1,3,2).


```
NonRecursive-Honoi(n, f, t)
 1  h ← the other peg
 2  ▷ to make a recursive call start from here
 3  if n = 1
 4     then Move the top coin from leg f to leg t
 5          goto 9
 6     else Push (S, StackRec(n, f, t, h, 'A'))
 7          n, f, t, h ← n − 1, f, h, t      ▷ parameter passing
 8          goto 3
```



```
▷ to end a recursive call start from here
 9  if not isEmpty(S)
10     then return-address, n, f, t, h ← Pop(S)
11          switch
12               case return-address = 'A'
13                    do Move the top coin from leg f to leg t
14                       Push(S, StackRec(n, f, t, h,,'B'))
15                       n, f, t, h ← n − 1, h, t, f   ▷ parameter passing
16                       goto 3
17               case return-address = 'B'
18                    do goto 9
```


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 واجرای برنامه از نقطه‌ی (x) دنبال می‌شود. ولی (x) تنها یک بازگشت است.

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 ← n − 1, h, f      ▷ parameter passing
 6          goto 1
```


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:
```


NonRecursive-Honoi2(n, f, t)

```
 1  h ← 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 Push(S, StackRec(n, f, t, h)
 8         n, f, t, h ← n − 1, f, h, t      ▷ parameter passing
 9          goto 3
10  ▷ end recursive call
11  if not isEmpty(S)
12    then n, f, t, h ← Pop(S)
13         Move the top coin from leg f to leg t
14         n, f, t, h ← n − 1, h, t, f      ▷ parameter passing
15          goto 3
```