



# سه: پیشته

ساختمان داده ها و الگوریتم

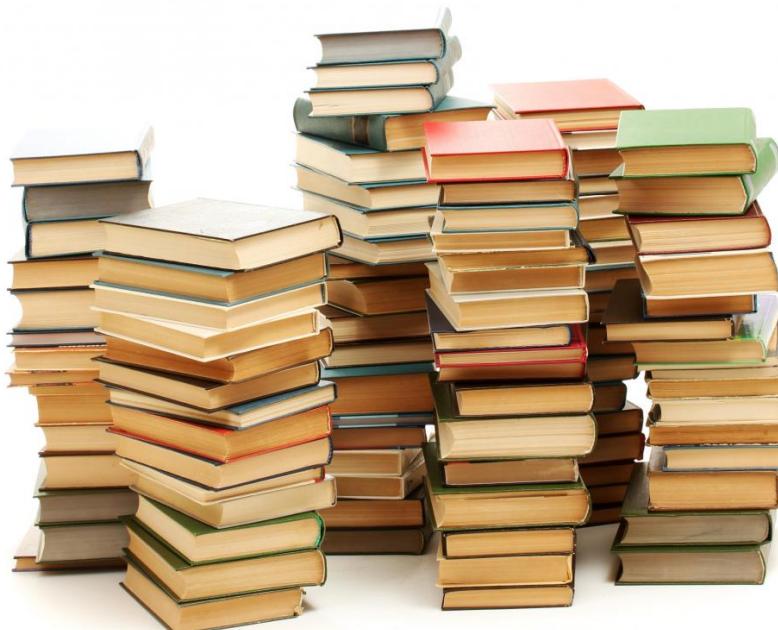
مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان

## پشته

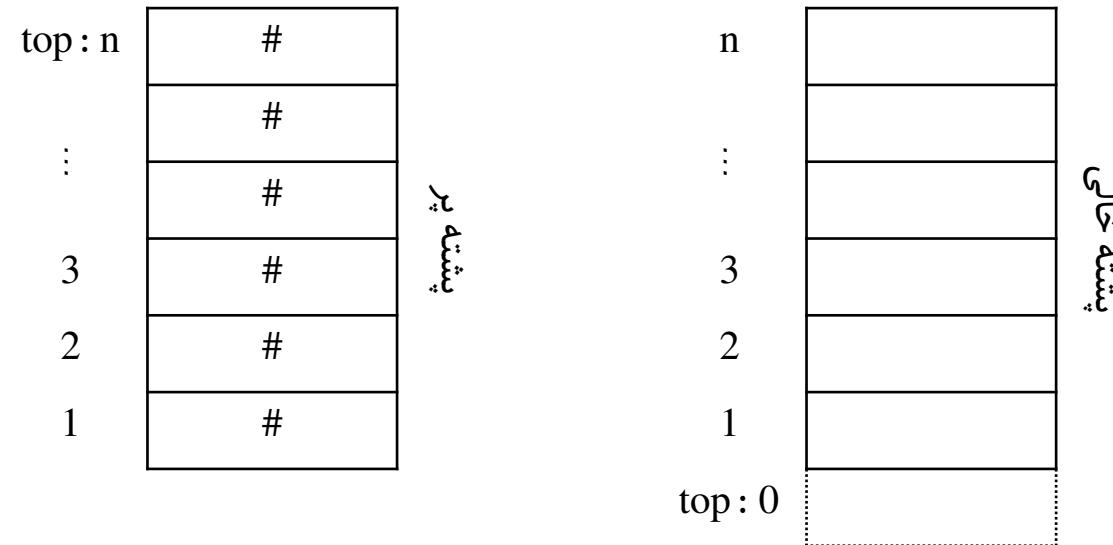
پشته لیستی است که در آن عمل درج و حذف از یک طرف به نام [بالای](#) پشته انجام میشود.

- یعنی عنصری که از همه دیرتر وارد پشته شد، از همه زودتر از پشته حذف میگردد. به همین دلیل گفته میشود که پشته از سیاست خروج به ترتیب عکس ورود پیروی میکند.



# اشاره‌گر بالا

برای نگهداری یک پشته از یک آرایه  $\text{stack}[1 \dots n]$  استفاده می‌کنیم، در اینصورت شرایط مرزی برابر خواهد بود با:



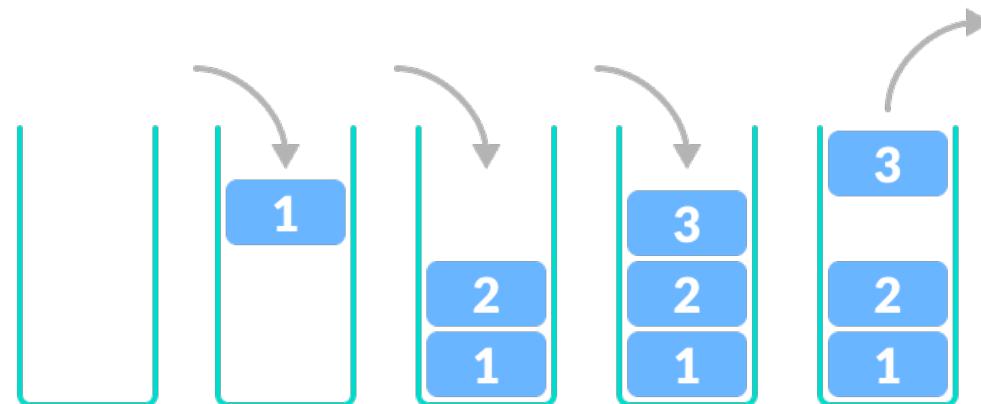
البته اشاره‌گر  $\text{top}$  می‌تواند به آخرین خانه خالی هم اشاره کند!

# درج و حذف

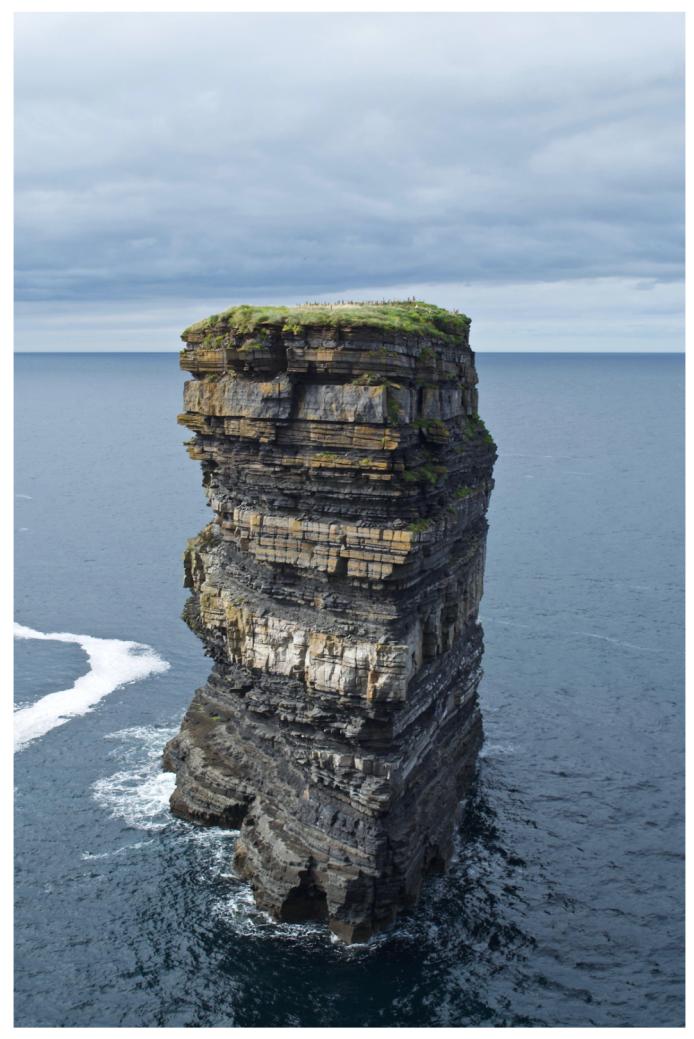
```
def push(x):  
    if(top==n):  
        print('stack is full')  
    else:  
        stack[top]=x  
        top=top+1  
        return top  
  
def pop():  
    if(top == 0):  
        print('stack is empty')  
    else:  
        x = stack[top]  
        top = top-1  
        return x
```

پیچیدگی زمانی اضافه کردن یک عنصر به یک پشته یا برداشتن یک عنصر از روی یک پشته با پیاده‌سازی آرایه‌ای، از  $O(1)$  است.

:Push که عنصری را به بالای پشته اضافه می‌کند.  
:Pop که عنصر بالای پشته را حذف می‌کند.



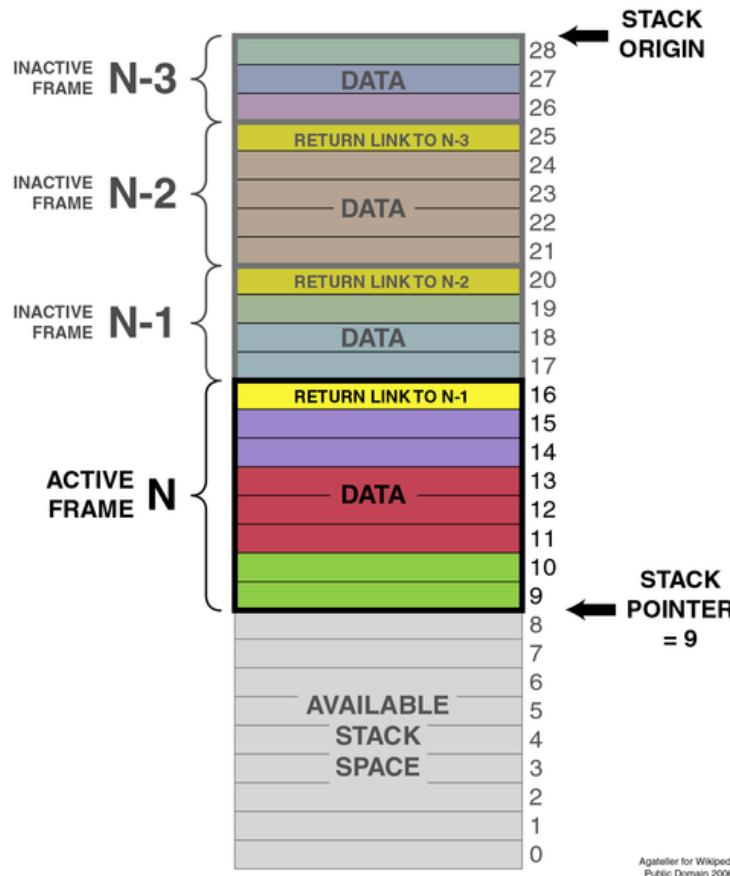
# کاربرد های پشته



Sea stack In Dun Briste, County Mayo, Ireland.

- تخصیص حافظه مبتنی بر پشته
- پیاده‌سازی فراخوانی توابع در یک کامپایلر
- ارزیابی عبارت های ریاضی
- طراحی الگوریتم ها به روش پس‌گرد

# تخصیص حافظه مبتنی بر پشته



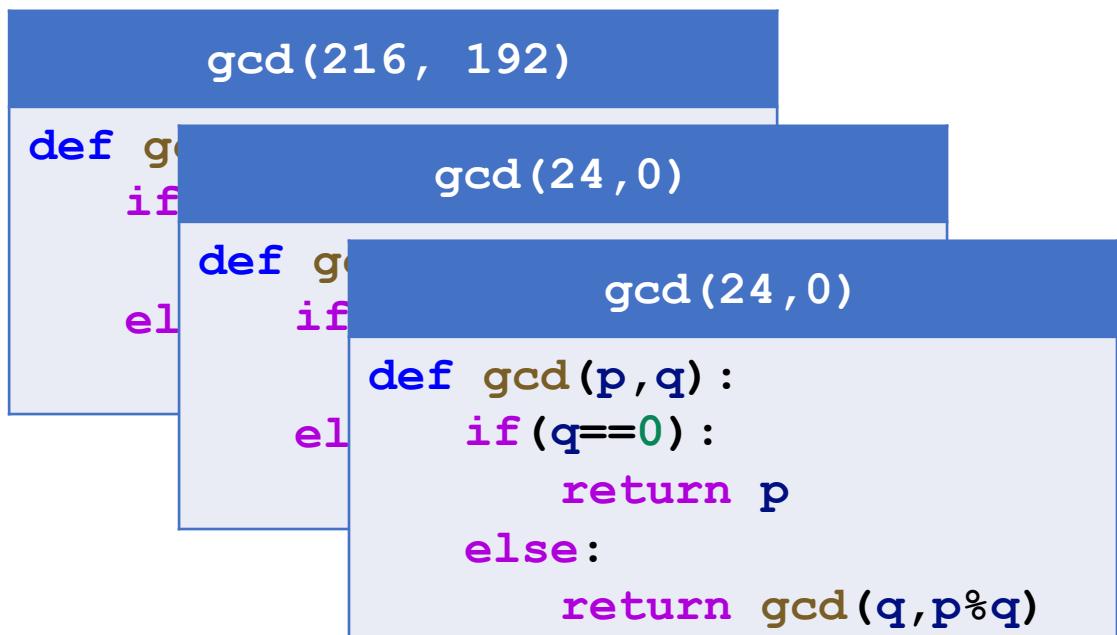
ناحیه stack با ساختار LIFO، به طور رایج در بالاترین بخش از حافظه قرار می‌گیرد. یک (اشاره گر پشته) در بالاترین قسمت قرار می‌گیرد. زمانی که تابعی فراخوانی می‌شود این تابع به همراه تمامی متغیرهای محلی خودش در داخل حافظه استک قرار می‌گیرد و با فراخوانی یک تابع جدید تابع جاری بر روی تابع قبلی قرار می‌گیرد و کار به همین صورت درباره دیگر توابع ادامه پیدا می‌کند.

- دسترسی بسیار سریع به متغیرها
- نیازی برای باز پس گیری حافظه اختصاص یافته شده ندارید
- فضا در زمان مورد نیاز به اندازه کافی توسط پردازنده مرکزی مدیریت می‌شود، حافظه ای نشت نخواهد کرد
- متغیرها فقط محلی هستند
- محدودیت در حافظه stack بسته به نوع سیستم عامل متفاوت است
- متغیرها نمی‌توانند تغییر اندازه دهند

# فراخوانی توابع

چگونه کامپایلر اجرای توابع را ممکن می‌سازد؟

- فراخوانی تابع: ذخیره مقادیر متغیرهای محلی و آدرس بازگشت در پشته
- برگشت: برداشتن آدرس بازگشت و مقادیر متغیرهای محلی از پشته



# ارزشیابی عبارات محاسباتی

هر عبارت ریاضی با توجه به اولویت عملگرهاش قابل ارزیابی است.

یادآوری ۱. در هر عبارت محاسباتی:

$-$	$a$	$a$	$\times$	$b$
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
عملگر	عملوند	عملوند	عملگر	عملوند
Operator	Operand	Operand	Operator	Operand

یادآوری ۲. با توجه به یادآوری ۱ دیده می‌شود که در عبارت‌های محاسباتی دو نوع عملگر وجود دارد:

۱- یکانی (Unary)      ۲- باینری (Binary)

عملگرهای یکانی مانند:  $-$  (منفی) و  $+$  (ثبت) فقط به یک عملوند نیاز دارند.

عملگرهای دودویی مانند:  $\times$  (ضرب)،  $\div$  ( تقسیم )،  $+$  (جمع) و  $-$  (تفريق) و  $^{\wedge}$  (توان)

# اولویت عملگرها

اولویت	عملگر	توضیحات
۱	( )	
۲	- (منفی)، + (ثبت)	
۳	$^{\wedge}$ (توان)	اولویت توانهای متوالی از راست به چپ
۴	$\times$ (ضرب)، / (تقسیم)	هم اولویت، اولویت از چپ به راست
۵	+ (جمع)، - (تفريق)	هم اولویت، اولویت از چپ به راست

- در هر عبارت محاسباتی اگر تعداد عملوندها ۱ واحد بیشتر از تعداد عملگرها باشد، در آن عبارت همه **عملگرها دودویی** هستند.
- در هر عبارت محاسباتی اگر تعداد عملگرها بیشتر یا مساوی تعداد عملوندها باشد، در آن عبارت **حتماً عملگر یکانی** وجود دارد.

مثال

$$a \times b - c^{\wedge} d^{\wedge} e + f$$

3                            1  
 \_\_\_\_\_  
 2  
 \_\_\_\_\_  
 4  
 \_\_\_\_\_  
 5

# مثال

$$a/(b+c) - e \times (f+g)$$

The expression is grouped into five levels:

- Level 1:  $a/(b+c)$  and  $e \times (f+g)$
- Level 2:  $b+c$  and  $f+g$
- Level 3:  $a$  and  $e$
- Level 4:  $b$ ,  $c$ ,  $f$ , and  $g$
- Level 5: All terms are grouped together.

# درخت عبارات محاسباتی

(Parse Tree)

برای تشکیل درخت هر عبارت محاسباتی به صورت زیر عمل می کنیم:

۱. ابتدا عبارت را برحسب اولویت عملگرهایش اولویتبندی (شماره گذاری) می کنیم.
۲. ریشه درخت، عملگر با کمترین اولویت (بیشترین شماره) است.
۳. ریشه زیر درختان چپ و راست نیز به همین ترتیب عملگر با کمترین اولویت (بیشترین شماره) در چپ و راست ریشه درخت خواهد بود.
۴. برگ‌های درخت عملوند و سایر گره‌ها (۱ فرزندی یا ۲ فرزندی)، عملگر خواهند بود.

# مثال

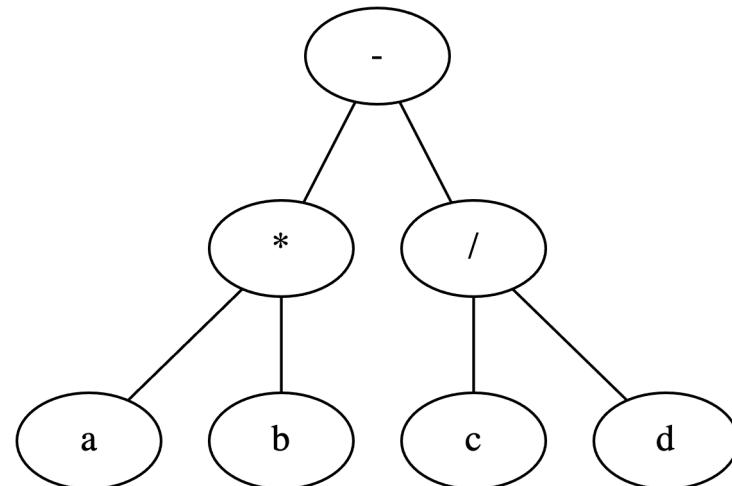
$$a \times b - c/d$$

اولویت بندی عبارت

$$\underbrace{a \times b}_{1} - \underbrace{c/d}_{2}$$

$\underbrace{\hspace{1cm}}_{3}$

درخت عبارت



# مثال

$$-a \times b + c - d^{\wedge}e$$

اولویت بندی عبارت

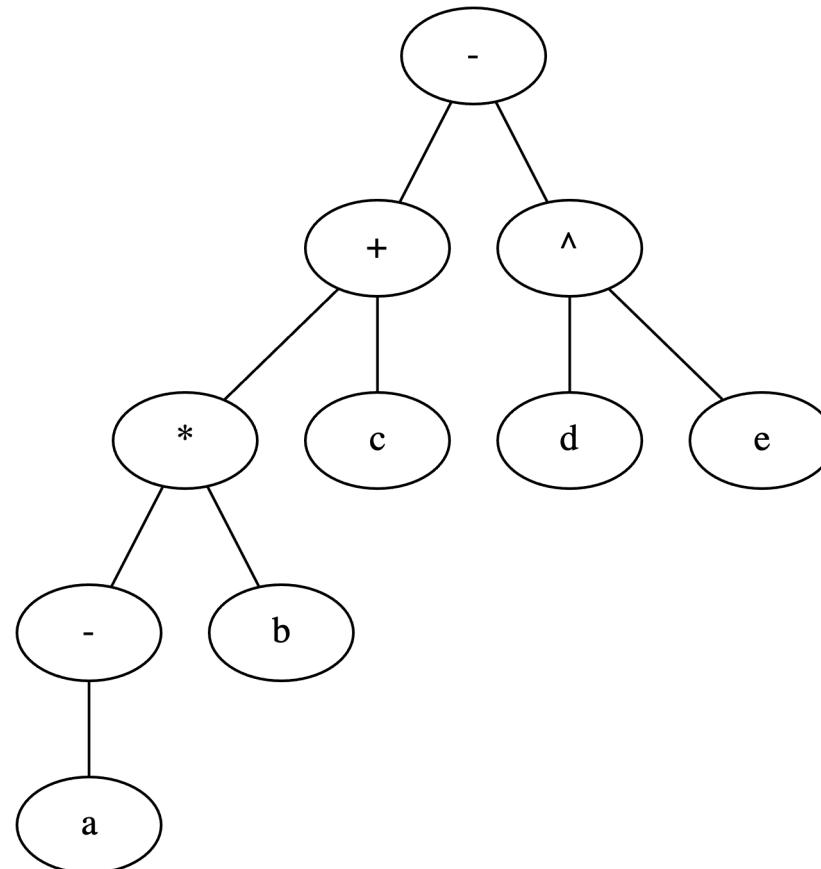
$$\underbrace{-a \times b}_{1} + \underbrace{c - d^{\wedge}e}_{2}$$

$\underbrace{\hspace{1cm}}_{3}$

$\underbrace{\hspace{2cm}}_{4}$

$\underbrace{\hspace{3cm}}_{5}$

درخت عبارت

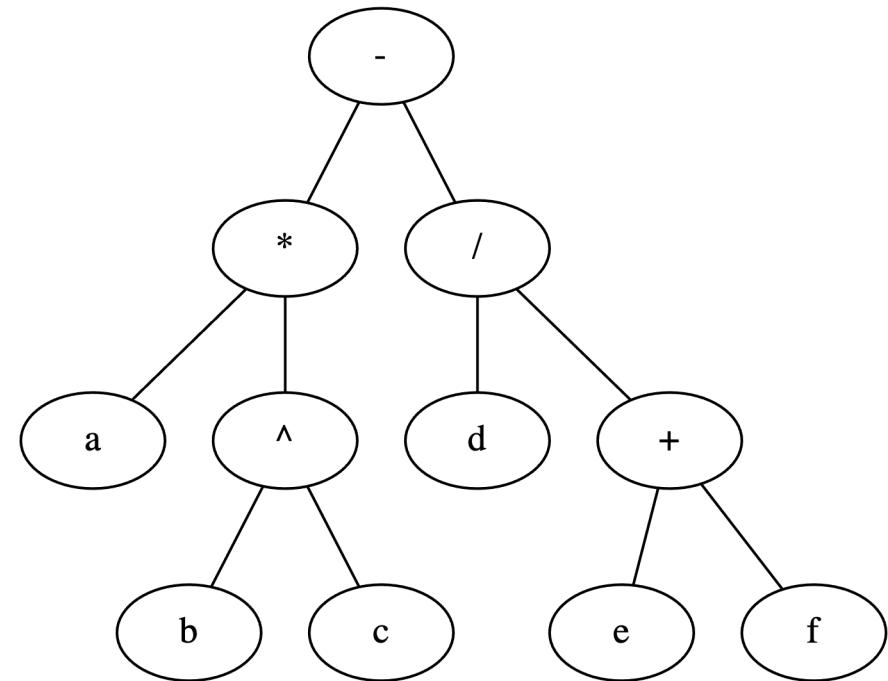


مثال

$$a \times b^\wedge c - d/(e + f)$$

اولویت بندی عبارت

درخت عبارت



# تعداد گره‌ها و عملگرهاي يکاني

عبارت	درخت عبارت	تعداد گره	تعداد عملگر يکاني
$-a \times b$	<pre> graph TD     Root((*)) --- Minus(( ))     Root --- B((b))     Minus --- A((a))   </pre>	۴ گره	۱ عملگر يکاني
$a \times b + c$	<pre> graph TD     Root((+)) --- Mult(( ))     Root --- C((c))     Mult --- A((a))     Mult --- B((b))   </pre>	۵ گره	بدون عملگر يکاني
$-a \times -b$	<pre> graph TD     Root((*)) --- MinusL(( ))     Root --- MinusR(( ))     MinusL --- A((a))     MinusL --- B((b))     MinusR --- A((a))     MinusR --- B((b))   </pre>	۵ گره	۲ عملگر يکاني

# تعداد گره‌ها و عملگرهاي يکاني

نتیجه	تعداد عملگر یکانی	تعداد گره درخت عبارت
حداقل یک عملگر یکانی داریم	فرد	زوج
ممکن است عملگر یکانی نداشته باشیم	زوج	فرد

# مثال

تعداد گرهای یک درخت دودویی که یک عبارت ریاضی را نمایش می‌دهد ۱۴ می‌باشد. عملگرهای این عبارت، دودویی یا یکانی می‌باشند. کدامیک از گزینه‌های زیر درست است؟

- ۱) این عبارت حتماً تعداد فردی عملگر دودویی دارد.
- ۲) این عبارت حتماً تعداد زوجی عملگر یکتایی دارد.
- ۳) این عبارت نمی‌تواند عملگر دودویی داشته باشد.
- ۴) حداقل یک عملگر یکانی در این عبارت وجود دارد.

حل: گزینه ۴ درست است.

# عبارت میان‌وندی (infix)

در هر عبارت عمومی محاسباتی هر عملگر دودویی بین عملوند‌هایش قرار می‌گیرد:

$$a \times b, x + y, \dots$$

در هر عبارت میان‌وندی اولویت عملگرها مطرح بوده و از () برای تغییر اولویت عملگرها استفاده می‌شود.

در هر عبارت میان‌وندی اولویت عملگرها با توجه به جدول اولویت عملگرها مشخص می‌شود.

# عبارت پسوندی (postfix)

در این روش هر عملگر بعد از عملوندهایش قرار می‌گیرد:

$ab \times, xy +, \dots$

در هر عبارت پسوندی، اولویت عملگرها مطرح نبوده و () تعریف نمی‌شود.  
به روش پسوندی، روش لهستانی معکوس یا (Reverse Polish Notation) PRN نیز می‌گویند.

# عبارت پیشوندی (prefix)

در این روش هر عملگر قبل از عملوند هایش قرار می گیرد:

$\times ab$ ,  $+xy$ , ...

در هر عبارت پسوندی، اولویت عملگرها مطرح نبوده و () تعریف نمی شود.

به روش پسوندی، روش لهستانی نیز می گویند چون اولین بار این روش توسط ریاضیدان لهستانی معرفی شد.

# مثال

infix:  $a \times b + c \rightarrow \left( (a \times b)^\times + c \right)^+ \Rightarrow$  postfix:  $a, b, \times, c, +$

infix:  $a \times (b+c) - g/d \rightarrow \left( \left( a \times (b+c)^+ \right)^\times - (g/d)'/ \right)^- \Rightarrow$  postfix:  $a, b, c, +, \times, g, d, /, -$

infix:  $\sqrt{b^2 - 4ac} \rightarrow \left( \left( (b^2)^{\wedge} - ((4 \times a)^\times \times c)^\times \right)^- \wedge (1/2)'/ \right)^{\wedge} \Rightarrow$   
postfix:  $b, 2, ^\wedge, 4, a, \times, c, \times, -, 1, 2, /, ^\wedge$

infix:  $a \times -b^{\wedge}c^{\wedge}d - e/f \rightarrow \left[ \left( a \times \left( (-b)^- \wedge (c^{\wedge}d)^{\wedge} \right)^{\wedge} \right)^\times - (e/f)'/ \right]^- \Rightarrow$   
postfix:  $a, b, -, c, d, ^\wedge, ^\wedge, \times, e, f, /, -$

# مثال

infix:  $a \times b + c \rightarrow \left( (a \times b)^\times + c \right)^+ \Rightarrow$  prefix:  $+, \times, a, b, c$

infix:  $a \times (b+c) - g/d \rightarrow \left( \left( a \times (b+c)^+ \right)^\times - (g/d)'/ \right)^- \Rightarrow$  prefix:  $- , \times , a , + , b , c , / , g , d$

infix:  $\sqrt{b^2 - 4ac} \rightarrow \left( \left( (b^2)^- - ((4 \times a)^\times \times c)^\times \right)^- \wedge (1/2)'/ \right)^\wedge \Rightarrow$   
 prefix:  $\wedge , - , ^\wedge , b , 2 , \times , \times , 4 , a , c , / , 1 , 2$

infix:  $a \times -b^c d - e/f \rightarrow \left[ \left( a \times \left( (-b)^- \wedge (c^d)^- \right)^\wedge \right)^\times - (e/f)'/ \right]^- \Rightarrow$   
 prefix:  $- , \times , a , ^\wedge , - , b , ^\wedge , c , d , / , e , f$

# تبدیل infix به postfix

برای این تبدیل به ترتیب زیر عمل می‌کنیم:

الف) عبارت را از سمت چپ پیمایش می‌کنیم.

ب) عملوندها را در خروجی می‌نویسیم.

ج) به هر پرانتز (" ) که رسیدیم آن را داخل stack قرار می‌دهیم.

د) به هر عملگر که رسیدیم به شرطی که اولویت آن عملگر از عملگر بالای stack بیشتر باشد، آن را داخل stack قرار می‌دهیم. در غیر این صورت آنقدر از بالای stack عملگر خارج کرده و در خروجی می‌نویسیم تا یا stack خالی شده و یا به عملگری بررسیم که بتوانیم عملگر مورد نظر را روی آن در stack قرار دهیم.

تذکر. عملگر + نمی‌تواند روی + یا - قرار گیرد. همین طور - روی + یا - و / روی × یا / و × روی × یا / نمی‌توانند قرار گیرند. اما توان ^ روی توان ^ می‌تواند قرار گیرد. چون اولویت توان‌های پشت سرهم از راست به چپ بررسی می‌شود.

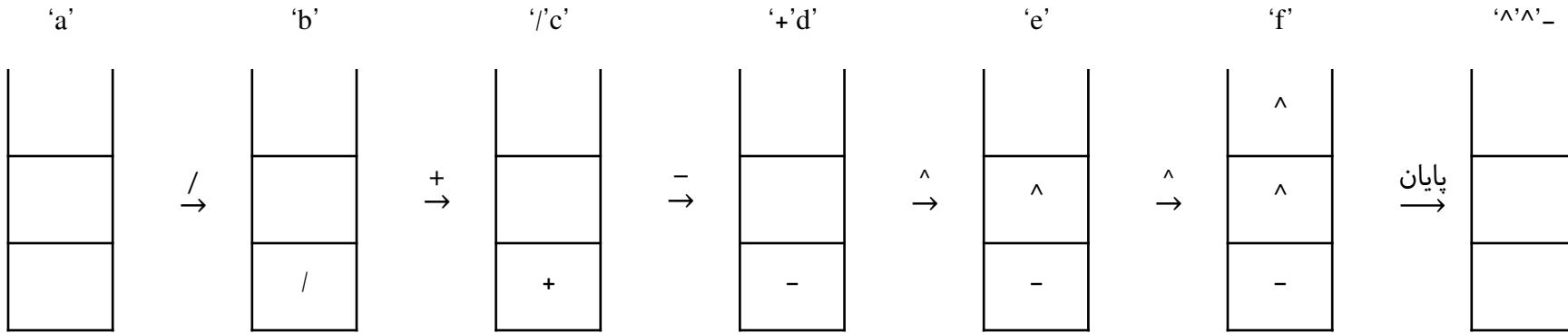
۵) اگر بالای stack پرانتز (" ) باشد، هر عملگری به راحتی روی آن قرار می‌گیرد.

و) به هر پرانتز (" ) که رسیدیم آنقدر از stack عملگر خارج کرده و در خروجی می‌نویسیم تا به (" ) بررسیم در این وضعیت (" ) با هم خنثی می‌شوند.  
را) زمانی که به انتهای عبارت رسیدیم در صورتی که stack خالی نباشد. آن را به طور کامل در خروجی می‌نویسیم.

# مثال

$$a/b + c - d^{\wedge}e^{\wedge}f$$

حداقل اندازه پشته برای تبدیل عبارت میان‌وندی فوق به عبارت پسوندی را محاسبه کنید:



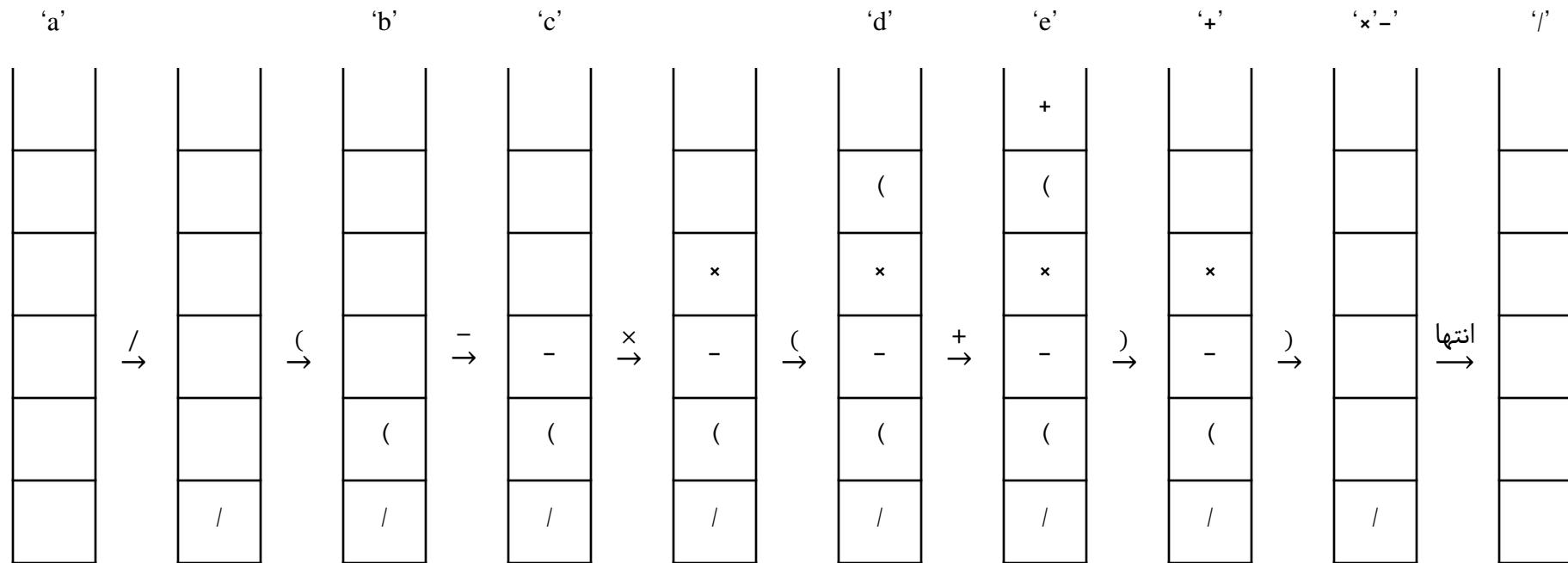
نتیجه از چپ به راست: a,b,/,c,+,d,e,f,^,^ خواهد بود.

حداقل فضای مورد نیاز: ۳

تعداد حذف = تعداد درج: تعداد ("") + تعداد عملگر ها = ۰ + ۵ = ۵

# مثال

$$a / \left( b - c \times (d + e) \right)$$



نتیجه از چپ به راست: /, -, \*, +, a, b, c, d, e خواهد بود.

تعداد حذف = تعداد درج: تعداد ("()") + تعداد عملگر ها = ۶ = ۲ + ۴

حداقل فضای مورد نیاز: ۶

# تبديل infix به prefix

برای این تبدیل به ترتیب زیر عمل می‌کنیم:

الف) عبارت را از سمت راست پیمایش می‌کنیم.

ب) عملوندها را در خروجی می‌نویسیم.

ج) به هر پرانتز "(" که رسیدیم آن را داخل stack قرار می‌دهیم.

د) به هر عملگر که رسیدیم به شرطی که اولویت آن عملگر از عملگر بالای stack بیشتر یا مساوی باشد، آن را داخل stack قرار می‌دهیم. در غیر این صورت آنقدر از بالای stack عملگر خارج کرده و در خروجی می‌نویسیم تا یا stack خالی شده و یا به عملگری بررسیم که بتوانیم عملگر مورد نظر را روی آن در stack قرار دهیم.

تذکر. عملگر + می‌تواند روی + یا - قرار گیرد. همین‌طور - روی + یا / و روی × یا / می‌توانند قرار گیرند. اما توان ^ روی توان ^ نمی‌توانند قرار گیرد. چون اولویت توان‌های پشت سرهم از راست به چپ بررسی می‌شود.

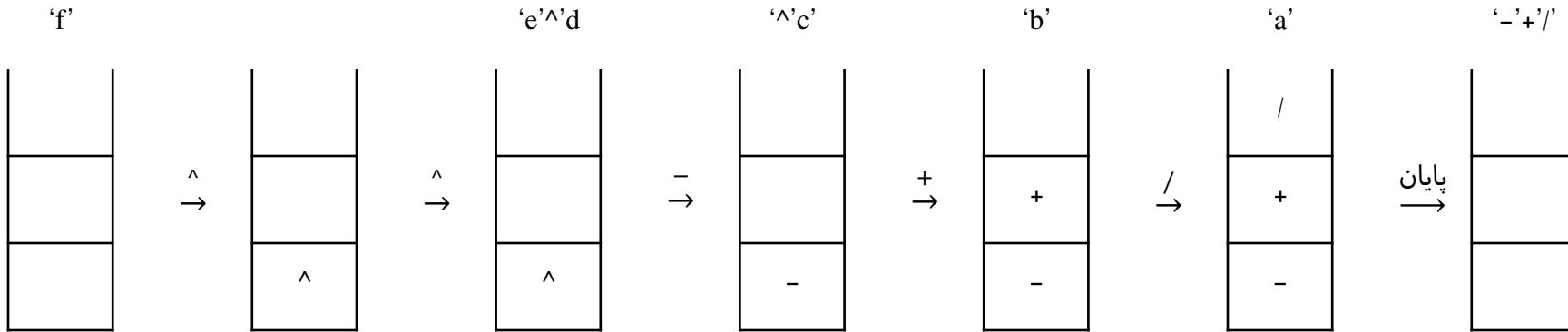
۵) اگر بالای stack پرانتز "(" باشد، هر عملگری به راحتی روی آن قرار می‌گیرد.

و) به هر پرانتز ")" که رسیدیم آنقدر از stack عملگر خارج کرده و در خروجی می‌نویسیم تا به "(" بررسیم در این وضعیت "(" ")" با هم خنثی می‌شوند.  
را) زمانی که به انتهای عبارت رسیدیم در صورتی که stack خالی نباشد. آن را به طور کامل در خروجی می‌نویسیم.

مثال

$$a/b + c - d^\wedge e^\wedge f$$

حداقل اندازه پشته برای تبدیل عبارت میان‌وندی فوق به عبارت پیشوندی را محاسبه کنید:



نتیجه از چپ به راست: +, /, a, b, c, ^, d, ^, e, f -خواهد بود.

### حداقل فضای مورد نیاز:

**تعداد حذف = تعداد درج: تعداد ("(") + تعداد عملگر ها = ۵ + ۵**

# تبدیل عبارات infix و postfix و به prefix

الف) عبارت را از سمتی بررسی می‌کنیم که با عملوند شروع می‌شود. در postfix از چپ و در prefix از راست عبارت را بررسی می‌کنیم.

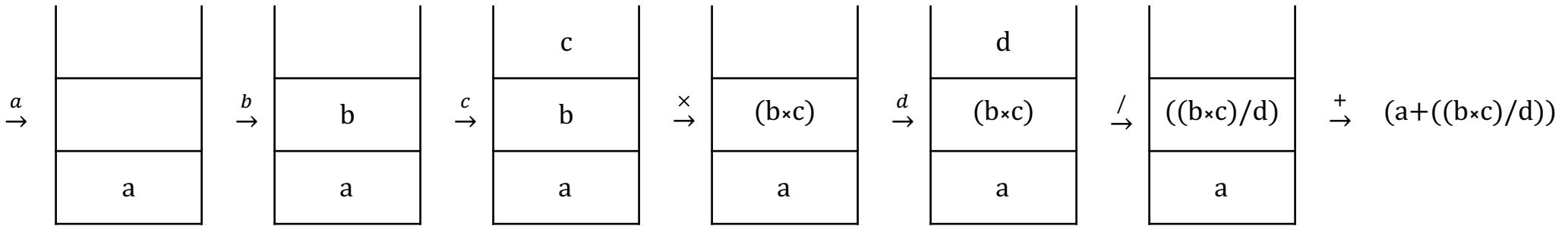
ب) عملوند را در stack قرار می‌دهیم و با رسیدن به هر عملگر (به جز عملگر آخر) دو عملوند خارج کرده و بعد از محاسبه دوباره حاصل عبارت را داخل stack قرار می‌دهیم. (دقت کنید که دو عملوندی که از stack خارج می‌کنیم و از سمت چپ به راست عمل می‌کنیم)

ج) به عملگر آخر که رسیدیم بعد از خارج کردن دو عملوند آخر و محاسبه آن را دیگر در stack قرار نمی‌دهیم و حاصل را در خروجی می‌نویسیم.

# مثال

postfix:  $a, b, c, \times, d, /, +$

حداقل اندازه پشته برای تبدیل عبارت پسوندی فوق به عبارت میان‌وندی را محاسبه کنید:



حداقل فضای مورد نیاز: ۳

تعداد حذف = تعداد درج: تعداد عملگرها  $\times$   $2 = 3 \times 2 = 2 = 6$

# خروجی های مجاز برای ورودی های پشته

در یک پشته  $n$ -تایی در صورتی که داده های  $a_1, a_2, \dots, a_n$  به ترتیب وارد stack شوند. با رعایت قواعد زیر می توانیم خروجی های مجاز را تعیین کنیم:

- الف) هر داده به محض ورود می تواند از پشته خارج شود.
- ب) در هر مرحله هرگاه بخواهیم داده ای را در خروجی ببینیم که هنوز وارد stack نشده است می بایست داده ها را پشت سرهم وارد پشته کنیم تا به داده مورد نظر برسیم سپس داده را وارد پشته کرده و سپس خارج می کنیم.
- ج) در هر مرحله هرگاه بخواهیم داده ای را در خروجی ببینیم که پایین stack است و عنصر بالای پشته نیست این عمل غیرمجاز است.

# مثال

ورودی  $\langle A,B,C,D,E,F \rangle$  به یک پشته را در نظر بگیرید، کدام یک از ترتیب های زیر در پشته ممکن است؟

E
C
A

مجاز

ابتدا A وارد شده، سپس B خارج شده و D، C وارد شده، سپس D خارج شده و E وارد می شود.

F
B

مجاز

ابتدا A وارد، سپس خارج می شود؛ سپس E، D، C، B، F خارج شده سپس E، D، C وارد می شود.

A
D
B

غیر مجاز

یا باید پایین B قرار گیرد یا قبل از پشته حذف شده باشد.

# مثال

کدام یک از گزینه های زیر را نمی توان با هیچ ترتیبی از ورودی  $\langle 1, 3, 4, 5, 6, 7 \rangle$  بدست آورد؟

۱) ۱، ۲، ۳، ۵، ۶، ۴

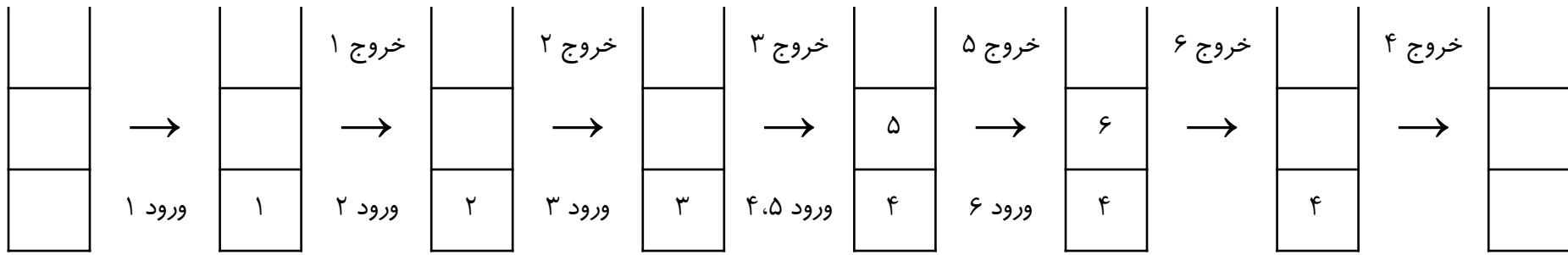
۲) ۳، ۲، ۴، ۶، ۵، ۱

۳) ۴، ۳، ۲، ۱، ۶، ۵

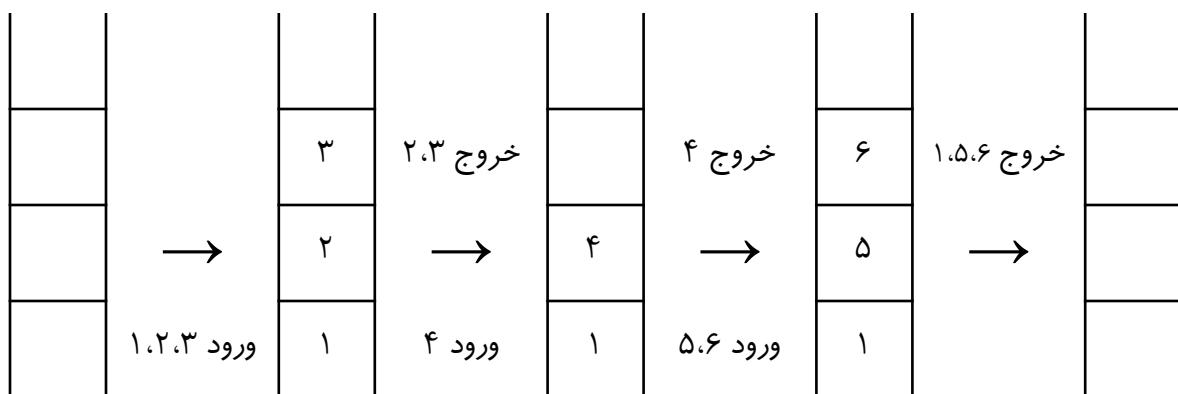
۴) ۲، ۱، ۵، ۳، ۶، ۴

# مثال

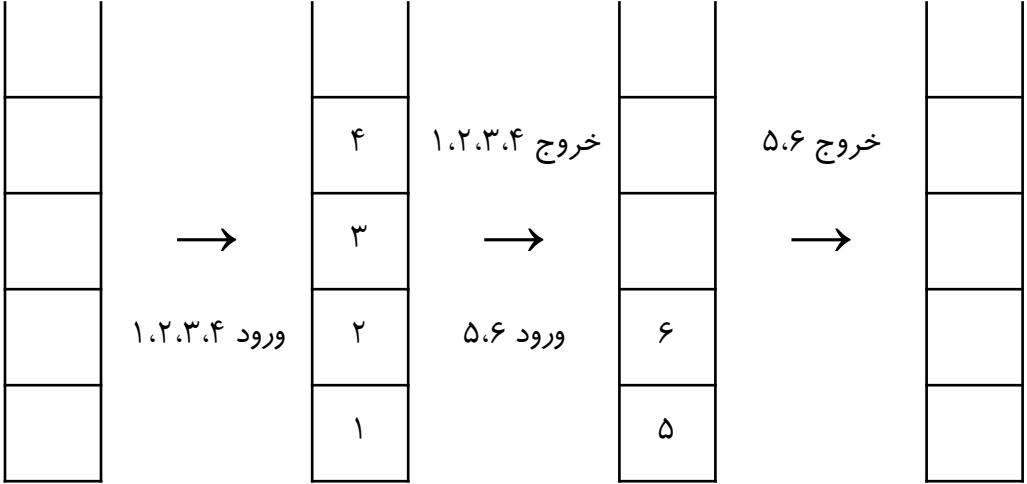
گزینه ۱ (۱، ۲، ۳، ۵، ۶، ۴) مجاز است:



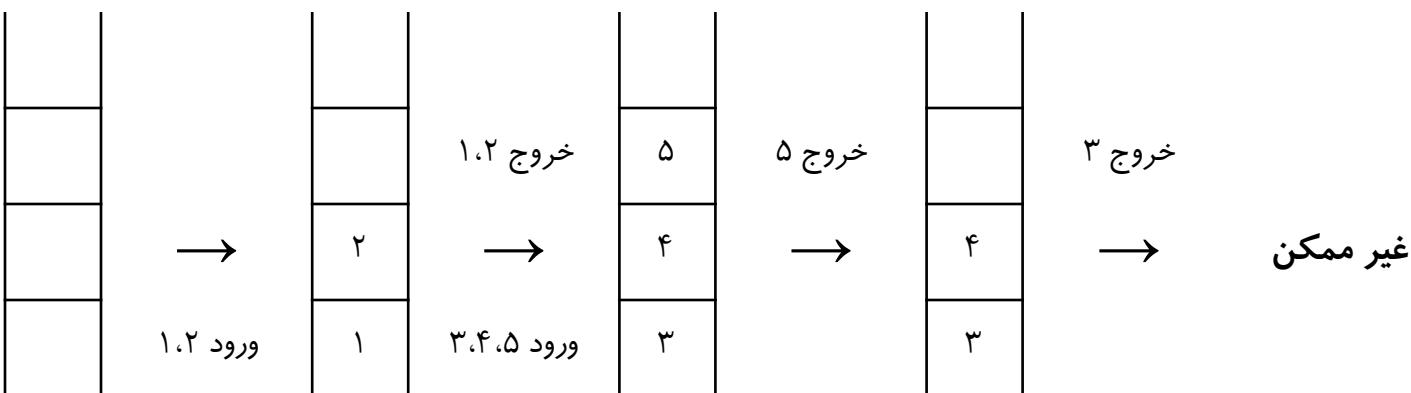
گزینه ۲ (۱، ۲، ۴، ۶، ۵، ۳) مجاز است:



# مثال



گزینه ۳ (۴, ۳, ۲, ۱, ۵) مجاز است:



گزینه ۴ (۴, ۳, ۲, ۱, ۵) مجاز نیست:

# تعداد خروجی های مجاز

$$C_n = \frac{\binom{2n}{n}}{n+1}$$

تعداد خروجی های مجاز در یک پشته  $n$ -تایی برابر با همان اعداد کاتالان است.

